



華南師範大學

本科学生实验（实践）报告

院 系： 计算机学院

实验课程：编译原理

实验项目：实验 4 LALR(1) 分析生成器

指导老师：黄煜廉

开课时间：2023~2024 年度第二学期

专 业：计算机科学与技术

班 级：计科 2 班

华南师范大学教务处

华南师范大学实验报告

学生姓名 林泽勋 学 号 20212821020
专 业 计算机科学与技术 年级班级 22 级计科 2 班
课程名称 编译原理 实验项目 实验 4 LALR(1) 分析生成器
实验类型 ☐ 验证 ☐ 设计 ☐ 综合 实验时间 2024 年 6 月 5 日
实验指导老师 黄煜廉 实验评分 _____

一、实验内容

设计一个应用软件，以实现 LALR(1) 分析生成器。

二、实验要求：

1. 必做功能：

(1) 要提供一个文法输入编辑界面，让用户输入文法规则（可保存、打开存有文法规则的文件）

(2) 求出文法各非终结符号的 first 集合与 follow 集合，并提供窗口以便用户可以查看这些集合结果。【可以采用表格的形式呈现】

(3) 需要提供窗口以便用户可以查看文法对应的 LR(1)DFA 图。（可以用画图的方式呈现，也可用表格方式呈现该图点与边数据）

(4) 需要提供窗口以便用户可以查看文法对应的 LALR(1)DFA 图。（可以用画图的方式呈现，也可用表格方式呈现该图点与边数据）

(5) 需要提供窗口以便用户可以查看文法对应的 LALR(1) 分析表。（如果该文法为 LALR(1) 文法时）【LALR(1) 分析表采用表格的形式呈现】

(6) 应该书写完善的软件文档

(7) 应用程序应为 Windows 界面。

2. 选做功能。

(1) 需要提供窗口以便用户输入需要分析的句子。

(2) 需要提供窗口以便用户查看使用 LALR(1) 分析该句子的过程。【可以使用表格的形式逐行显示分析过程】

二、实验目的

本实验的目的是设计并实现一个 LALR(1) 分析生成器应用软件，旨在

加深对编译原理中词法分析、语法分析等基础概念的理解，并通过实践掌握 LALR(1)分析算法的实现原理和过程。通过该软件，用户能够输入和编辑上下文无关文法规则，自动计算并展示文法中非终结符号的 first 集合和 follow 集合，以及生成文法对应的 LR(1)和 LALR(1) DFA 图和分析表。此外，实验还包括了选做功能的实现，即允许用户输入待分析的句子，并展示使用 LALR(1)分析器进行语法分析的详细过程。通过本实验，我们将能够提升编程能力、算法实现能力以及解决复杂问题的能力，同时，实验也要求遵循软件开发的规范，编写完善的软件文档，确保软件的可读性、可维护性和用户友好性。最终，我们需要独立完成实验任务，并按照指定的格式提交源代码、设计文档、测试数据和可执行程序。通过本实验，我们将获得宝贵的实践经验，为将来在软件开发和编译原理领域的进一步学习和研究打下坚实的基础。

本实验旨在通过设计和实现一个 LALR(1)分析生成器应用软件，使我们深入理解编译原理中的高级概念，特别是 LALR(1)分析器的构造和使用。实验的主要目的如下：

1. 理论与实践结合：通过将编译原理中的理论知识应用到实际编程中，帮助我们更好地理解 LALR(1)分析器的设计原理和工作机制。
2. 编程技能提升：通过使用 C++程序设计语言来开发软件，锻炼我们的编程能力和对复杂算法的实现能力。
3. 算法实现：我们需要实现自动计算文法的 first 集合和 follow 集合的算法，以及 LR(1)和 LALR(1) DFA 的构造算法，从而加深对这些算法的理解。
4. 软件工程实践：我们将学习如何规划、设计、编码、测试和文档化一个完整的软件项目，这是软件工程实践的重要环节。
5. 可视化展示：通过图形化的方式展示 LR(1)和 LALR(1) DFA 图，帮助我们直观理解文法的自动机表示。
6. 文法规则的自动化处理：通过自动化处理文法规则，减少人工操作的复杂性，提高效率。

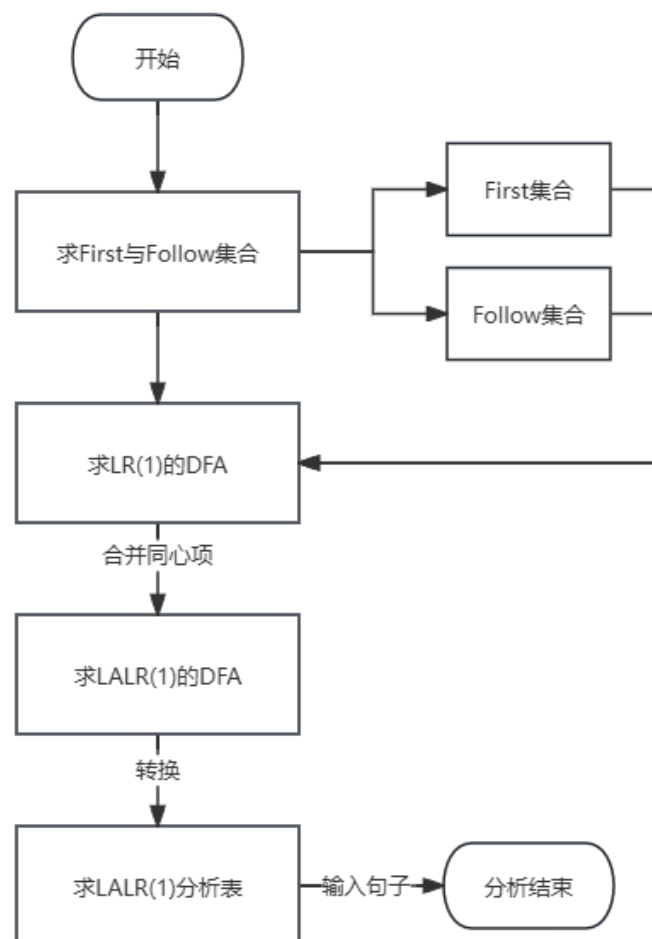
通过本实验，我们不仅能够获得宝贵的编程和算法实现经验，还能够

学习到如何将理论知识转化为实际应用，为将来在计算机科学和软件工程领域的职业生涯打下坚实的基础。

三、实验文档：

1 系统总体设计

1.1 总体流程图



2 项目所用数据结构

2.1 存储 first 与 follow 集合的数据结构

1. `QHash<QString, QSet<QString>> firstSet;`
2. `QHash<QString, QSet<QString>> followSet;`

存储文法的 first 集合和 follow 集合通常需要一种能够快速访问和更新数据的数据结构。哈希表（散列表）是一种非常适合这类需求的数据结构，因为它提供了平均常数时间复杂度的查找、插入和删除操作，使用哈希表（散列表）存储，表示某一非终结符能够转移到各个

终结符的集合。

- (1) **firstSet**: 键是非终结符（以 `QString` 形式），值是该非终结符可以推导出的终结符集合（以 `QSet<QString>` 形式）。例如，如果非终结符 `A` 可以推导出终结符集合 `{"a", "b"}`，则 `firstSet["A"]` 将包含这两个终结符。
- (2) **followSet**: 类似于 **firstSet**，键是非终结符，值是该非终结符的 **follow** 集合，即出现在该非终结符后面的概率更高的终结符集合。

2.2 存储 LR(1) 与 LALR(1) 的 DFA 数据结构

首先我们需要存储这些 DFA 的状态，每一个状态中包含多个文法规则，下面我们先介绍一下单条文法规则的存储方法。

```
1. // 单条规则
2. typedef struct Item {
3.     QString name;
4.     QStringList rule;
5.     QSet<QString> next;
6.     int pos;
7.
8.     static bool haveSameCore(Item i, Item j);
9.
10.    friend bool operator==(const Item a, const Item b) {
11.        return a.name == b.name && a.rule == b.rule && a.next == b.next && a.pos == b.pos;
12.    }
13.
14.    friend bool operator<(const Item a, const Item b) {
15.        if (a.name == b.name) {
16.            if (a.rule == b.rule) {
17.                if (a.pos == b.pos) {
18.                    return a.pos < b.pos;
19.                }
20.                return a.pos < b.pos;
21.            }
22.            return a.rule < b.rule;
23.        }
24.        return a.name < b.name;
25.    }
26. }Item;
```

该数据结构存储的是 DFA 状态之中的一条文法规则。

- **name:** 表示项目左侧的非终结符名称。
- **rule:** 表示整个产生式的右侧，包括左侧非终结符和产生式符号。
- **next:** 表示可以跟随当前产生式符号的终结符集合。
- **pos:** 表示当前分析位置在产生式中的位置。

Item 结构体还包含了两个静态成员函数和一个友元函数：

- **haveSameCore:** 用于比较两个项目的核心是否相同，即它们的名字、规则和位置是否相同。
- **operator==** 和 **operator<:** 分别用于比较两个项目是否相等和定义项目的小于关系，这对于将项目存储在 QHash 中并进行排序非常有用。

```

1. // 单个状态
2. typedef struct State {
3.     QSet<Item> st;
4.     static State closure(State I, QHash<QString, QSet<QStringList>> grammars, QVector<QString> nonFinalizers, QHash<QString, QSet<QString>> firstSet);
5.     static State change(State I, QString X, QHash<QString, QSet<QStringList>> grammars, QVector<QString> nonFinalizers, QHash<QString, QSet<QString>> firstSet);
6.
7.     static bool haveSameCore(State i, State j);
8.
9.     friend bool operator==(const State a, const State b) {
10.         return a.st == b.st;
11.     }
12.
13. }State;

```

State 结构体表示 LR 分析器中的一个状态，它由一组文法规则组成。

- **st:** 是一个包含 Item 的集合，表示当前状态中的所有项目。

State 结构体包含两个静态成员函数和一个友元函数：

- **closure:** 用于计算状态的闭包，即从当前状态出发，考虑所有可能的移动，得到所有可达的项目。
- **change:** 用于计算从一个状态到另一个状态的转移，基于特定的终结符。

- `haveSameCore`: 用于比较两个状态是否相同，即它们包含的项目集合是否相同。
- `operator==`: 用于比较两个状态是否相等。

```

1. // LALR 结构
2. class LALR
3. {
4. public:
5.     LALR();
6.
7.     // 成员变量
8.     int size;
9.     QHash<State, int> stateHash;
10.    QHash<int, QHash<QString, int>> changeHash;
11.
12.    // 构建 LR1
13.    void buildLR1(State faState, QHash<QString, QSet<QStringList>> grammars, QVector<QString> nonFinalizers,
14.                  QHash<QString, QSet<QString>> firstSet,
15.                  QHash<QString, QSet<QString>> followSet);
16.
17.    // 构建 LALR1
18.    void buildLALR1(LALR lr1);
19.};

```

- `size`: 表示分析表的大小。
- `stateHash`: 哈希表，用于存储状态和唯一标识符之间的映射。
- `changeHash`: 哈希表，用于存储从一个状态到另一个状态的转移字符串。
- `buildLR1`: 成员函数，用于构建 LR(1)分析器。
- `buildLALR1`: 成员函数，用于基于 LR(1)分析器构建 LALR(1)分析器。

2.3 LALR(1) 分析表的数据结构

```

1. // 分析表结点
2. typedef struct LALR1TableItem {
3.     int kind;    // 1 表示移进、2 表示规约、3 表示非终结符移进、
4.                 4 表示接受
5.     int idx;

```

```
5. }LALR1TableItem;  
6.  
7. // LALR1_table  
8. QHash<int, QHash<QString, LALR1TableItem>> LALR1_table;
```

LALR1TableItem 结构体定义了分析表中的一个条目，每个条目包含以下信息：

- **kind**: 一个整数，表示该条目的类型。不同类型的条目对应不同的解析动作：
 - 1 表示移进（Shift），即将分析器的状态向前移动到下一个输入符号。
 - 2 表示规约（Reduce），即根据当前状态和输入符号，使用某个产生式规则来规约输入。
 - 3 表示非终结符移进（Non-terminal Shift），这在 LALR(1)分析中不常见，通常与移进相同。
 - 4 表示接受（Accept），即解析成功完成。
- **idx**: 一个整数，表示与该条目相关联的具体动作的索引。例如，在规约动作中，idx 可以表示被规约的产生式的索引。

LALR1_table 数据结构

LALR1_table 是一个二维哈希表，用于存储分析表的所有条目。它的结构如下：

- 外层哈希表的键: 一个整数，表示分析器的状态编号。
- 外层哈希表的值: 另一个哈希表，表示在该状态下对各个输入符号的动作。
- 内层哈希表的键: 一个 QString，表示当前输入符号。
- 内层哈希表的值: 一个 LALR1TableItem 结构体，表示在当前状态和输入符号下应执行的动作。

3 文法预处理

- 获取文法规则:

从用户界面的文本编辑框（grammarEdit）中获取所有文法规则，这些规则以文本形式输入，每条规则占一行。

使用 split("\n") 方法将文本按行分割成 QStringList，每个元素代

表一条文法规则。

- 存储文法规则:

遍历每条文法规则，对每条规则进行进一步的处理。

- 获取非终结符:

对每条规则使用 `split(" ")` 分割空白字符，得到一个包含该规则所有单词的 `QStringList`。

移除空字符串元素，确保列表中只包含有效的单词。

检查分割后的列表是否至少包含 3 个元素（非终结符、"`->`"、至少一个终结符或另一个非终结符），并且第二个元素是否为 "`->`"。

如果不符合要求，则弹出警告消息框提示用户文法规则格式错误，并终止程序。

如果符合要求，则将非终结符（列表的第一个元素）添加到 `nonFinalizers` 列表中。

- 获取起始符:

从第一条文法规则中提取起始符，假设起始符是第一条规则的第一个单词。

- 分词:

再次遍历每条文法规则，对每条规则进行分词处理。

对于每个单词，如果它不是非终结符且不是 "`->`"，则将其添加到 `firstSet` 的对应集合中，表示该单词是某个终结符的直接 `first` 集合的一部分。

移除非终结符和 "`->`"，然后处理剩余的单词列表，以处理文法中的选择结构（由 "`|`" 分隔的产生式）。

对于每个选择结构，将其插入到 `grammars` 哈希表中，以非终结符作为键，产生式列表作为值。

```
1. // 获取文法规则
2. QString content = ui->grammarEdit->toPlainText();
3. QStringList grammarList = content.split("\n");
4.
5. // 存储文法规则
6. // 获取非终结符
7. for (const auto &grammar: qAsConst(grammarList)) {
```

```

8.     QStringList wordList = grammar.split(" ");
9.     for (int i = 0; i < wordList.size(); i++) {
10.         if (wordList[i] == "") {
11.             wordList.removeAt(i);
12.         }
13.     }
14.     if (wordList.size() < 3 || wordList[1] != "->") {
15.         QMessageBox::warning(this, "提示", "文法分析失败，左侧非终结符只能出现单个词，非终结符后需要有空格以及 -> 符号", QMessageBox::Yes);
16.         return;
17.     } else {
18.         nonFinalizers.push_back(wordList[0]);
19.     }
20. }
21. qDebug() << nonFinalizers;
22.
23. // 获取起始符
24. QStringList firstGrammar = grammarList[0].split(" ");
25. if (!firstGrammar.empty()) startString = firstGrammar[0];
26. qDebug() << startString;
27.
28. // 分词
29. for (const auto &grammar: qAsConst(grammarList)) {
30.     QStringList wordList = grammar.split(" ");
31.     for (int i = 0; i < wordList.size(); i++) {
32.         if (wordList[i] == "") {
33.             wordList.removeAt(i);
34.         } else {
35.             if (!nonFinalizers.contains(wordList[i]) && wordList[i] != "->") {
36.                 firstSet[wordList[i]].insert(wordList[i]);
37.             }
38.         }
39.     }
40.     // 删除非终结符与->
41.     QString nonfinal = wordList[0];
42.     wordList.removeAt(0);
43.     wordList.removeAt(0);
44.     int lastIdx = 0;
45.     for (int i = 1; i < wordList.size() - 1; i++) {
46.         if (wordList[i] == "|") {
47.             grammars[nonfinal].insert(wordList.mid(lastIdx, i - lastIdx));

```

```

48.         lastIdx = i + 1;
49.     }
50. }
51. grammars[nonfinal].insert(wordList.mid(lastIdx, wordList.size() - lastIdx));
52.}
53.qDebug() << grammars;

```

4 求解 first 集合与 follow 集合实现

4.1 求解 first 集合

```

1. // 计算first集合
2. std::function<QSet<QString>(QString)> getFirst = [&](QString s) {
3.     if (firstSet.contains(s)) {
4.         return firstSet[s];
5.     } else if (!nonFinalizers.contains(s)) {
6.         return firstSet[s] = QSet<QString>({s});
7.     } else {
8.         for (auto grammar: grammars[s]) {
9.             int k = 0;
10.            while (k < grammar.size()) {
11.                firstSet[grammar[k]] = getFirst(grammar[k]);
12.                bool mark = false;
13.                if (firstSet[grammar[k]].contains("@")) mark = true;
14.                firstSet[s].unite(firstSet[grammar[k]].subtract(QSet<QString>({"@"})));
15.                if (mark) firstSet[grammar[k]].insert("@");
16.                else break;
17.                k++;
18.            }
19.            if (k == grammar.size()) firstSet[s].insert("@");
20.        }
21.        return firstSet[s];
22.    }
23.};
24.for (int i = 0; i < nonFinalizers.size(); i++) {
25.    firstSet[nonFinalizers[i]] = getFirst(nonFinalizers[i]);
26.}

```

这段代码实现了计算文法中每个非终结符的 first 集合的功能。

`first` 集合包含一个非终结符可以推导出的所有终结符的集合，以及在某些条件下的空串（用 `@` 表示）。下面给出详细的解释：

- 定义 `getFirst` 函数：

使用 `std::function` 和 `lambda` 表达式定义了一个函数对象 `getFirst`，它接受一个 `QString` 类型的参数 `s` 并返回一个 `QSet<QString>` 类型的 `first` 集合。

- 检查 `firstSet` 中是否已存在：

如果 `firstSet` 已经包含了非终结符 `s` 的 `first` 集合，则直接返回这个集合。

- 处理终结符：

如果 `s` 不是终结符（即不在 `nonFinalizers` 集合中），并且 `firstSet` 中没有 `s` 的 `first` 集合，则创建一个新的集合，只包含 `s` 本身，并将这个集合存入 `firstSet`。

- 递归计算 `first` 集合：

如果 `s` 是非终结符，则对 `grammars[s]` 中的每个产生式进行遍历。对于每个产生式：

- 初始化索引 `k` 为 0。
- 使用 `while` 循环和递归调用 `getFirst` 来计算产生式中每个符号的 `first` 集合。
- 使用一个标记 `mark` 来检查当前符号的 `first` 集合中是否包含空串 `@`。
- 将当前符号的 `first` 集合（除去空串 `@`）合并到非终结符 `s` 的 `first` 集合中。
- 如果 `mark` 为 `true`，则在当前符号的 `first` 集合中插入空串 `@`。
- 如果 `mark` 为 `false`，则退出循环。
- 如果索引 `k` 遍历完整个产生式，则在非终结符 `s` 的 `first` 集合中插入空串 `@`。

- 返回计算结果：

在处理完所有产生式后，返回非终结符 s 的 first 集合。

- 遍历所有非终结符:

通过一个 for 循环，对 nonFinalizers 集合中的每个非终结符调用 getFirst 函数，计算其 first 集合，并将结果存储在 firstSet 中。

4.2 求解 follow 集合

```
1. // 计算 follow 集合
2. bool flag = true;    // 判断循环结束的标志
3. followSet[startString].insert("$");
4. while (flag) {
5.     flag = false;
6.     for (auto key: grammars.keys()) {
7.         for (auto grammar: grammars[key]) {
8.             // 对每条文法进行分析
9.             for (int i = 0; i < grammar.size(); i++) {
10.                if (!nonFinalizers.contains(grammar[i])) c
ontinue;
11.                int k = i + 1;
12.                QSet<QString> tmpSet;
13.                while (k < grammar.size()) {
14.                    bool sign = false;
15.                    if (firstSet[grammar[k]].contains("@")
) sign = true;
16.                    tmpSet.unite(firstSet[grammar[k]].subt
ract(QSet<QString>({"@"})));
17.                    if (sign) firstSet[grammar[k]].insert(
"@");
18.                    else break;
19.                    k++;
20.                }
21.                if (k == grammar.size()) tmpSet.insert("@"
);
22.                bool mark = false;
23.                if (tmpSet.contains("@")) mark = true;
24.                if (!followSet[grammar[i]].contains(tmpSet
.subtract(QSet<QString>({"@"})))) {
25.                    // 将 first( $X_{i+1} \dots X_n$ ) 加入
follow( $X_i$ )
26.                    for (auto tmp: tmpSet) {
27.                        followSet[grammar[i]].insert(tmp);
28.                    }
29.                    flag = true;
30.                }
```

```

31.         if (mark && !followSet[grammar[i]].contains(followSet[key])) {
32.             for (auto tmp: followSet[key]) {
33.                 followSet[grammar[i]].insert(tmp);
34.             }
35.             flag = true;
36.         }
37.     }
38. }
39. }
40.}

```

这段代码实现了计算文法中每个非终结符的 **follow** 集合的功能。**follow** 集合包含了在文法中非终结符可能出现的位置之后可能出现的终结符集合，以及在特定条件下的结束符号（通常是一个特殊的结束符，用 **\$** 表示）。以下是代码的详细解释：

- 初始化 **follow** 集合:

首先，将起始符号的 **follow** 集合初始化为包含结束符号 **\$**。

- 循环计算 **follow** 集合:

使用一个 **while** 循环进行迭代计算，直到没有新的 **follow** 集合可以添加，即 **flag** 为 **false**。

- 遍历所有文法规则:

外层 **for** 循环遍历 **grammars** 中的所有键（非终结符）。

- 分析每条文法规则:

内层 **for** 循环遍历每个非终结符对应的所有产生式。

- 处理产生式中的每个符号:

for 循环遍历产生式中的每个符号。

如果符号是终结符，则跳过当前迭代。

- 递归计算 **first** 集合:

从当前符号后的第一个符号开始，使用 **while** 循环和递归调用 **getFirst** 函数（未在代码片段中显示，但之前定义过）来计算 **first** 集合。使用一个临时集合 **tmpSet** 来存储计算结果。

- 处理空串:

如果 **tmpSet** 中包含结束符号 **\$**，则将其添加到 **tmpSet** 中。

- 更新 follow 集合:

如果 tmpSet 中的元素（除去 @）尚未全部包含在当前非终结符的 follow 集合中，则将它们添加进去，并将 flag 设置为 true 以继续循环。

- 处理产生式末尾的非终结符:

如果当前非终结符是产生式的最后一个符号，并且 tmpSet 包含结束符号 \$，则将 key（当前非终结符）的 follow 集合中的元素添加到当前非终结符的 follow 集合中，并将 flag 设置为 true。

- 结束条件:

如果在一次完整迭代中没有对任何非终结符的 follow 集合进行更新，则 flag 保持为 false，循环终止。

这段代码使用了迭代的方法来计算 follow 集合，确保了在存在递归产生式时能够正确地计算出 follow 集合。代码中的 \$ 符号用于表示输入的结束符号，这是在计算 follow 集合时需要特别考虑的一个特殊情况。通过不断地迭代更新，直到没有新的元素可以添加到任何非终结符的 follow 集合中，从而完成 follow 集合的计算。

5 LR(1)的 DFA 实现

在构建 DFA 之前，我们需要介绍两个非常必要的函数。

5.1 闭包函数:

CLOSURE(I):

$J = I$

while CLOSURE(I) still changing:

foreach LR(1) item $A \rightarrow \alpha \cdot B\beta, a$ **in** CLOSURE(I):

foreach $B \rightarrow \gamma$:

foreach terminal b **in** FIRST(βa):

 add $B \rightarrow \cdot \gamma, b$ to J

return J

闭包函数 closure 接收一个状态 I 作为输入，并返回这个状态的闭包 J。闭包是指从某个状态出发，考虑所有可能的输入符号和相应的转移，能够得到的所有项目集合。下面给出我的思路以及具体代码：

- 初始化:

State J 初始化为输入状态 I 的副本。
- 迭代计算闭包:

使用 while 循环来迭代地将新项目添加到状态 J 中，直到没有新项目可以添加。
- 处理每个项目:

遍历状态 J 中的每个项目 item。
- 转移方法:

如果项目 item 的当前位置 item.pos 是非终结符（通过 nonFinalizers 检查），则需要考虑从这个非终结符开始的所有可能的产生式。
- 计算产生式:

对于每个非终结符的产生式，创建新的项目 newItem，其初始位置为产生式的开始。
- 计算 first 集合:

对于产生式中 item.pos 之后的每个符号，计算其 first 集合，并将这些集合合并到临时集合 tmpSet 中。
- 添加新项目:

将计算得到的 tmpSet 中的每个元素作为新项目 newItem 的 next 集合的一部分，并尝试将其添加到状态 J 中。
- 去重:

如果状态 J 中已存在具有相同核心的项目，则更新该项目的 next 集合而不是添加新项目。
- 结束条件:

当连续两次迭代没有添加新项目时，闭包计算完成。

上述思路的具体实现：

```
1. State State::closure(State I, QHash<QString, QSet<QStringList> > grammars, QVector<QString> nonFinalizers, QHash<QString, QSet<QString>> firstSet)
2. {
3.     State J = I;
```



```

4.     while (true) {
5.         for (Item item: J.st) {
6.             //          qDebug() << item.name << ' ' << item.rule <<
              ' ' << item.next << ' ' << item.pos;
7.             // 下一个字符是非终结符
8.             if (item.pos < item.rule.size() && nonFinalize
rs.contains(item.rule[item.pos])) {
9.                 for (auto grammar: grammars[item.rule[item
.pos]]) {
10.                //          qDebug() << grammar;
11.
12.                for (auto nextItem: item.next) {
13.                //          qDebug() << nextItem;
14.
15.                QStringList betaRule = item.rule.m
id(item.pos + 1);
16.                //          qDebug() << betaRule;
17.
18.                if (nextItem != "@") betaRule.push
_back(nextItem);
19.
20.                // 计算first 集合
21.                int k = 0;
22.                QSet<QString> tmpSet;
23.                while (k < betaRule.size()) {
24.                    bool sign = false;
25.                    if (firstSet[betaRule[k]].cont
ains("@")) sign = true;
26.                    tmpSet.unite(firstSet[betaRule
[k]].subtract(QSet<QString>({"@"})));
27.                    if (sign) firstSet[betaRule[k]
].insert("@");
28.                    else break;
29.                    k++;
30.                }
31.                if (k == betaRule.size()) tmpSet.i
nsert("@");
32.
33.                //          qDebug() << k << ' ' << tmpSet;
34.
35.                // 遍历first 集合中的每个元素
36.                for (auto b: tmpSet) {
37.
38.                    Item newItem;

```

```

39.             newItem.name = item.rule[item.
    pos];
40.             newItem.rule = grammar;
41.             newItem.next.insert(b);
42.             newItem.pos = 0;
43.
44.             if (!J.st.contains(newItem)) {
45.                 bool mark = true;
46.                 for (auto it: J.st) {
47.                     if (Item::haveSameCore
    (it, newItem)) {
48.                         J.st.remove(it);
49.                         it.next.unite(newI
    tem.next);
50.                         J.st.insert(it);
51.                         mark = false;
52.                         break;
53.                     }
54.                 }
55.                 if (mark) {
56.                     J.st.insert(newItem);
57.                 }
58.             }
59.         }
60.
61.     }
62.
63. }
64. }
65. }
66.     if (I == J) break;
67.     I = J;
68. }
69.     qDebug();
70.     for (auto item: J.st) {
71.         qDebug() << item.name << ' ' << item.rule << ' ' <
    < item.next << ' ' << item.pos;
72.     }
73.     return J;
74.}

```

5.2 GOTO 函数:

GOTO(I, X):

foreach item in I :

if the item is of the form $A \rightarrow \alpha \cdot X\beta$:

add $A \rightarrow \alpha X \cdot \beta$ to the collection I_{new}

return CLOSURE(I_{new})

GOTO 函数 `change` 接收一个状态 I 和一个输入符号 X ，返回从状态 I 在输入符号 X 下的转移所得到的状态。

- 初始化新状态:

创建一个新的状态 I_{new} 。

- 处理每个项目:

遍历状态 I 中的每个项目 `item`。

- 转移条件:

如果项目 `item` 的当前位置 `item.pos` 等于产生式的长度（即项目已经完成），或者当前位置的符号不等于输入符号 X ，则跳过该项目。

创建新项目:

对于符合条件的项目，创建一个新的项目 `newItem`，其产生式与原项目相同，但位置向前移动一位。

- 去重:

与闭包函数类似，检查 I_{new} 中是否已存在具有相同核心的新项目，如果有，则更新其 `next` 集合；如果没有，则添加新项目。

- 闭包计算:

如果 I_{new} 不为空，则对 I_{new} 调用闭包函数 `closure` 来计算其闭包。

- 返回结果:

返回计算得到的新状态 I_{new} 。

上述思路的具体实现:

```
1. State State::change(State I, QString X, QHash<QString, QSet<QStringList> > grammars, QVector<QString> nonFinalizers, QHash<QString, QSet<QString>> firstSet)
2. {
```

```

3.     State I_new;
4.     for (auto item: I.st) {
5.         if (item.pos >= item.rule.size() || item.rule[item
        .pos] != X) continue;
6.
7.         Item newItem;
8.         newItem.name = item.name;
9.         newItem.rule = item.rule;
10.        newItem.next = item.next;
11.        newItem.pos = item.pos + 1;
12.
13.        // 判断I_new 中是否已经出现了newItem
14.        if (!I_new.st.contains(newItem)) {
15.            bool mark = true;
16.            for (auto it: I_new.st) {
17.                if (Item::haveSameCore(it, newItem)) {
18.                    I_new.st.remove(it);
19.                    it.next.unite(newItem.next);
20.                    I_new.st.insert(it);
21.                    mark = false;
22.                    break;
23.                }
24.            }
25.            if (mark) {
26.                I_new.st.insert(newItem);
27.            }
28.        }
29.    }
30.    if (I_new.st.empty()) return I_new;
31.    else return closure(I_new, grammars, nonFinalizers, fi
        rstSet);
32.}

```

5.3 LR(1)的 DFA 实现

闭包函数与 GOTO 函数共同用于构建 LR 分析器的状态机。closure 函数用于计算任何给定状态的所有可能转移，而 change 函数用于确定在给定输入符号下从一个状态转移到另一个状态所得到的具体状态。通过递归地应用这两个函数，可以逐步构建出完整的 LR 分析器的状态图。下面介绍 LR(1)的 DFA 图构建过程。

```

1. void LALR::buildLR1(State faState, QHash<QString, QSet<QSt
    ringList>> grammars, QVector<QString> nonFinalizers,
2.     QHash<QString, QSet<QString> > firstSe

```

```

    t, QHash<QString, QSet<QString> > followSet)
3. {
4.     int faStateId = stateHash[faState];
5.
6.     // 找到可能的转移方法
7.     QStringList changeMethods;
8.     for (auto faItem: faState.st) {
9.         if (faItem.pos < faItem.rule.size()) {
10.             changeMethods.append(faItem.rule[faItem.pos]);
11.         }
12.     }
13.
14.     // 深搜每个转移
15.     for (QString changeMethod: changeMethods) {
16.         State sonState = State::change(faState, changeMethod, grammars, nonFinalizers, firstSet);
17.         if (sonState.st.empty()) continue;
18.         if (stateHash.contains(sonState)) {
19.             // 该状态已经存在
20.             int sonStateId = stateHash[sonState];
21.             changeHash[faStateId].insert(changeMethod, sonStateId);
22.         } else {
23.             // 该状态不存在
24.             stateHash[sonState] = size++;
25.             changeHash[faStateId].insert(changeMethod, stateHash[sonState]);
26.             buildLR1(sonState, grammars, nonFinalizers, firstSet, followSet);
27.         }
28.
29.         qDebug() << faStateId << ' ' << changeMethod << ' ' << stateHash[sonState];
30.     }
31. }

```

- 获取初始状态 ID:

使用 `stateHash` 哈希表获取初始状态 `faState` 的唯一标识符 `faStateId`。

- 找到可能的转移方法:

通过遍历初始状态 `faState` 中的所有项目 (`faState.st`)，提取每个项目当前位置的下一个符号，这些符号表示可能的转移方法，并存储

在 `changeMethods` 列表中。

- 深度优先搜索（DFS）：

对于 `changeMethods` 中的每个转移方法 `changeMethod`，执行深度优先搜索来构建 DFA 的状态转移。

- 计算子状态：

对于每个 `changeMethod`，调用 `State::change` 静态成员函数来计算从当前状态 `faState` 转移得到的子状态 `sonState`。

- 检查子状态是否存在：

如果 `stateHash` 哈希表中已经包含了子状态 `sonState`，则获取其状态 ID `sonStateId`。

如果子状态不存在于 `stateHash` 中，则将该子状态添加到 `stateHash` 中，并分配一个新的唯一 ID（`size++`），然后递归调用 `buildLR1` 来构建这个新状态的 DFA。

- 更新转移哈希表：

无论子状态是新创建的还是已经存在的，都将其 ID 添加到 `changeHash` 哈希表中，以记录从 `faStateId` 状态通过 `changeMethod` 转移到达 `sonStateId`。

6 LALR(1)的 DFA 实现

```
1. void LALR::buildLALR1(LALR lr1)
2. {
3.     // 并查集(合并同心项)
4.     int* p = new int[lr1.size];
5.     for (int i = 0; i < lr1.size; i++) {
6.         p[i] = i;
7.     }
8.     std::function<int(int)> find = [&](int x) {
9.         if (p[x] != x) p[x] = find(p[x]);
10.        return p[x];
11.    };
12.    QHash<int, State> revlr1StateHash;
13.    for (auto state: lr1.stateHash.keys()) {
14.        revlr1StateHash[lr1.stateHash[state]] = state;
15.    }
16.
17.    // 状态合并
```

```

18.     for (int i = 0; i < lr1.size - 1; i++) {
19.         for (int j = i + 1; j < lr1.size; j++) {
20.             if (State::haveSameCore(revlr1StateHash[i], re
vlr1StateHash[j])) {
21.                 p[find(j)] = find(i);
22.             }
23.         }
24.     }
25.
26.     // LR1 到 LALR1 状态映射
27.     QHash<int, int> cntChangeSet;
28.     int idx = 0;
29.     for (int i = 0; i < lr1.size; i++) {
30.         if (!cntChangeSet.contains(find(i))) {
31.             cntChangeSet[find(i)] = idx++;
32.         }
33.     }
34.     size = cntChangeSet.size();
35.
36.     // 建立 LALR 状态集
37.     QHash<int, State> revlalr1StateHash;
38.     for (int i = 0; i < lr1.size; i++) {
39.         if (!revlalr1StateHash.contains(cntChangeSet[find(
i)])) {
40.             revlalr1StateHash[cntChangeSet[find(i)]] = rev
lr1StateHash[i];
41.         } else {
42.             for (auto item: revlr1StateHash[i].st) {
43.                 for (auto lalrItem: revlalr1StateHash[cntC
hangeSet[find(i)]] .st) {
44.                     auto tmpItem = lalrItem;
45.                     if (Item::haveSameCore(item, lalrItem)
) {
46.                         lalrItem.next.unite(item.next);
47.                     }
48.                     revlalr1StateHash[cntChangeSet[find(i)
]].st.remove(tmpItem);
49.                     revlalr1StateHash[cntChangeSet[find(i)
]].st.insert(lalrItem);
50.                 }
51.             }
52.         }
53.     }
54. //     for (auto state: revlalr1StateHash) {

```

```

55.//         for (auto item: state.st) {
56.//             qDebug() << item.name << ' ' << item.rule <<
//             ' ' << item.next << ' ' << item.pos;
57.//         }
58.//         qDebug();
59.//     }
60.     for (auto idx: revl1r1StateHash.keys()) {
61.         stateHash[revl1r1StateHash[idx]] = idx;
62.     }
63.
64.    // 建立 LALR 转移集
65.    for (int i = 0; i < lr1.size; i++) {
66.        for (auto changeMethod: lr1.changeHash[i].keys())
67.        {
68.            if (lr1.changeHash[i].contains(changeMethod))
69.            {
70.                changeHash[cntChangeSet[find(i)]].insert(c
changeMethod, cntChangeSet[find(lr1.changeHash[i][changeMet
hod])]);
71.            }
72.        }
73.    }

```

这段代码是 LALR 类的成员函数 buildLALR1 的实现，其目的是根据已有的 LR(1) 分析器来构建 LALR(1) 分析器。以下是该函数的详细解释：

- 初始化并查集:

使用一个整数数组 p 来实现并查集，用于合并具有相同核心的项目集合。每个状态最初是自己的代表（即 $p[i] = i$ ）。

使用 lambda 表达式定义 find 函数，该函数用于查找并返回状态 x 的根代表。

- 创建 LR(1) 状态的反向哈希映射:

创建 revlr1StateHash 哈希表，以 LR(1) 分析器的状态 ID 作为键，原始状态作为值，方便后续访问。

- 状态合并:

通过双层循环遍历所有状态对，如果两个状态具有相同的核心（使用 State::haveSameCore 判断），则将它们合并（通过将其中一个状态

的根代表指向另一个状态的根代表)。

- LR(1) 到 LALR(1) 状态映射:

创建 `cntChangeSet` 哈希表来存储每个状态的根代表在 LALR(1) 分析器中的新状态 ID。

计算 LALR(1) 分析器的大小 `size`。

- 建立 LALR 状态集:

创建 `revlalr1StateHash` 哈希表来存储新的 LALR(1) 状态。

对于每个 LR(1) 状态, 根据并查集的结果, 将其项目添加到相应的 LALR(1) 状态中, 并合并具有相同核心的项目。

- 状态哈希映射:

将新的 LALR(1) 状态添加到 `stateHash` 哈希表中, 以状态对象作为键, 状态 ID 作为值。

- 建立 LALR 转移集:

遍历 LR(1) 分析器的转移集 `changeHash`, 对于每个转移, 使用 `cntChangeSet` 来更新 LALR(1) 分析器的转移集。

整个 `buildLALR1` 函数的目的是将一个 LR(1) 分析器转换为一个更紧凑的 LALR(1) 分析器。这是通过合并同心项的状态来实现的, 从而减少状态的数量和转移的复杂性。LALR(1) 分析器是编译器设计中用于语法分析的常用算法之一, 它在 LR(1) 分析器的基础上进行了优化, 以减少状态数和提高效率。

但在这之后, 我们需要判断 LALR(1) 的每个状态中是否存在移进-规约冲突和规约-规约冲突, 具体方法如下:

```
1. // 判断 LALR1 文法: 移进-规约冲突 与 规约-规约冲突
2.     for (auto state: lalr1.stateHash.keys()) {
3.         // 找到所有的归约项, 判断前进字符是否有重复
4.         QSet<QString> callback;
5.         for (auto item: state.st) {
6.             // 判断规约项
7.             if (item.pos == item.rule.size()) {
8.                 for (auto next: item.next) {
9.                     if (callback.contains(next)) {
10.                        QMessageBox::warning(this, "警告", "该文法在 LALR(1) 中出现了规约-规约冲突, 出现了向前看符号有
```

```

    相同的规约项，不是 LALR(1) 文法", QMessageBox::Yes);
11.         return;
12.     }
13. }
14.     callback.unite(item.next);
15. }
16. }
17. // 找到所有移进项
18. QSet<QString> footIn;
19. for (auto item: state.st) {
20.     // 判断移进项
21.     if (item.pos != item.rule.size()) {
22.         footIn.insert(item.rule[item.pos]);
23.     }
24. }
25. // 判断是否有移进-规约冲突
26. if (!footIn.intersect(callback).empty()) {
27.     QMessageBox::warning(this, "警告", "该文法
    在 LALR(1) 中出现了移进-规约冲突，出现了规约项的向前看符号与移进
    项的下一个字符相同的情况，不是 LALR(1) 文法，但后续会使用移进项
    进行分析，解决二义性", QMessageBox::Yes);
28. }
29. }

```

- 遍历所有状态:

通过遍历 `lalr1.stateHash` 中的键（状态），来检查每个状态的项目集合。

- 找到所有归约项:

对于每个状态 `state`，遍历其项目集合 `state.st`。

如果项目的位置 `item.pos` 等于产生式的长度（即 `item.pos == item.rule.size()`），则该项目是一个归约项。

- 判断规约项的前进字符:

对于每个归约项，遍历其 `next` 集合（即 `item.next`），这个集合包含了归约时可以跟随的终结符。

使用 `callback` 集合来记录已经出现过的终结符。

如果 `callback` 集合中已包含当前归约项的某个终结符，则表明出现了规约-规约冲突，这时会弹出警告消息框，并结束程序。

- 找到所有移进项:

再次遍历状态的项目集合，找出所有移进项（即 `item.pos != item.rule.size()`）。

将这些移进项的下一个终结符添加到 `footIn` 集合中。

- 判断是否有移进-规约冲突：

使用 `footIn` 集合和 `callback` 集合求交集。

如果交集不为空，说明存在移进-规约冲突。

- 处理移进-规约冲突：

与规约-规约冲突不同，代码中指出如果出现移进-规约冲突，将使用移进项进行分析来解决二义性。

代码中的警告消息框通过 `QMessageBox::warning` 弹出，向用户展示冲突的类型和原因。需要注意的是，`LALR(1)` 分析器要求文法不能有规约-规约冲突，但允许一定条件下的移进-规约冲突，此时分析器会选择移进操作（通过最长串匹配原则来解决，优先移进而非规约）。

7 LALR(1)的分析表实现

```
1. // 计算LALR(1)分析表
2.     QStringList header;
3.     QSet<QString> finalString;
4.     for (int i = 0; i < lalr1.size; i++) {
5.         for (auto changeMethod: lalr1.changeHash[i].keys()) {
6.             if (!nonFinalizers.contains(changeMethod))
7.                 finalString.insert(changeMethod);
8.         }
9.     }
10.    }
11.    QList<QString> finalStringList = finalString.toList();
12.    QList<QString> nonFinalStringList = nonFinalizers.toList();
13.    qSort(finalStringList.begin(), finalStringList.end());
14.    finalStringList.append("$");
15.    qSort(nonFinalStringList.begin(), nonFinalStringList.end());
16.    header.append(finalStringList);
17.    header.append(nonFinalStringList);
```

```

18.         ui->lalrAnalysisTableWidget->setColumnCount(header
           .size());
19.         ui->lalrAnalysisTableWidget->setHorizontalHeaderLa
           bels(header);
20.         ui->lalrAnalysisTableWidget->setRowCount(lalr1.siz
           e);
21.
22.         // 可视化 LALR(1) 分析表
23.         QHash<int, State> revHash;
24.         for (auto state: lalr1.stateHash.keys()) {
25.             revHash[lalr1.stateHash[state]] = state;
26.         }
27.         QHash<QString, int> headerHash;
28.         int headerIdx = 0;
29.         for (QString head: header) {
30.             headerHash[head] = headerIdx++;
31.         }
32.         for (int i = 0; i < lalr1.size; i++) {
33.
34.             // 渲染行表头
35.             auto headerItem = new QTableWidgetItem(QString
               ::number(i));
36.             if (i == 0) headerItem->setTextColor(QColor(25
               5, 0, 0));
37.             ui->lalrAnalysisTableWidget->setVerticalHeader
               Item(i, headerItem);
38.
39.             State state = revHash[i];
40.             for (Item item: state.st) {
41.                 if (item.pos == item.rule.size()) {
42.                     // 规约项
43.                     if (item.name == startString) {
44.                         // 接受状态
45.                         for (auto next: item.next) {
46.                             if (next == "$") {
47.                                 LALR1TableItem lalrItem;
48.                                 lalrItem.kind = 4;
49.                                 LALR1_table[i].insert(next
                                   , lalrItem);
50.                                 ui->lalrAnalysisTableWidge
                                   t->setItem(i, headerHash[next], new QTableWidgetItem("接受
                                   "));
51.                                 } else if (recursion.contains(
                                   item)) {

```

```

52.                                LALR1TableItem lalrItem;
53.                                lalrItem.idx = recursion.i
    ndexOf(item);
54.                                lalrItem.kind = 2;
55.                                LALR1_table[i].insert(next
    , lalrItem);
56.                                ui->lalrAnalysisTableWidge
    t->setItem(i, headerHash[next], new QTableWidgetItem("r" +
    QString::number(recursion.indexOf(item))));
57.                                } else {
58.                                    recursion.push_back(item);
59.                                    LALR1TableItem lalrItem;
60.                                    lalrItem.idx = recursion.s
    ize() - 1;
61.                                    lalrItem.kind = 2;
62.                                    LALR1_table[i].insert(next
    , lalrItem);
63.                                    ui->lalrAnalysisTableWidge
    t->setItem(i, headerHash[next], new QTableWidgetItem("r" +
    QString::number(recursion.size() - 1)));
64.                                }
65.                                }
66.                                } else {
67.                                    for (auto next: item.next) {
68.                                        // 判断该规约规则是否已经出现过
69.                                        if (recursion.contains(item))
    {
70.                                            LALR1TableItem lalrItem;
71.                                            lalrItem.idx = recursion.i
    ndexOf(item);
72.                                            lalrItem.kind = 2;
73.                                            LALR1_table[i].insert(next
    , lalrItem);
74.                                            ui->lalrAnalysisTableWidge
    t->setItem(i, headerHash[next], new QTableWidgetItem("r" +
    QString::number(recursion.indexOf(item))));
75.                                            } else {
76.                                                recursion.push_back(item);
77.                                                LALR1TableItem lalrItem;
78.                                                lalrItem.idx = recursion.s
    ize() - 1;
79.                                                lalrItem.kind = 2;
80.                                                LALR1_table[i].insert(next
    , lalrItem);

```

```

81.                ui->lalrAnalysisTableWidget
t->setItem(i, headerHash[next], new QTableWidgetItem("r" +
QString::number(recursion.size() - 1)));
82.                }
83.            }
84.        }
85.    }
86.    }
87.    }
88.    for (int i = 0; i < lalr1.size; i++) {
89.        // 移进项
90.        for (int j = 0; j < finalStringList.size(); j+
+ ) {
91.            auto changeMethod = finalStringList[j];
92.            if (lalr1.changeHash[i].contains(changeMet
hod)) {
93.                LALR1TableItem lalrItem;
94.                lalrItem.idx = lalr1.changeHash[i][cha
ngeMethod];
95.                lalrItem.kind = 1;
96.                LALR1_table[i].insert(changeMethod, la
lrItem);
97.                ui->lalrAnalysisTableWidget->setItem(i
, j, new QTableWidgetItem("s" + QString::number(lalr1.chan
geHash[i][changeMethod])));
98.            }
99.        }
100.        // 规约后的回溯
101.        for (int j = 0; j < nonFinalStringList.size
()); j++) {
102.            auto changeMethod = nonFinalStringList[
j];
103.            if (lalr1.changeHash[i].contains(change
Method)) {
104.                LALR1TableItem lalrItem;
105.                lalrItem.idx = lalr1.changeHash[i][
changeMethod];
106.                lalrItem.kind = 3;
107.                LALR1_table[i].insert(changeMethod,
lalrItem);
108.                ui->lalrAnalysisTableWidget->setIte
m(i, j + finalStringList.size(), new QTableWidgetItem(QStr
ing::number(lalr1.changeHash[i][changeMethod])));
109.            }

```

```
110.         }  
111.     }  
112.  
113.         ui->lalrAnalysisTableWidget->setEditTriggers(QAbstractItemView::NoEditTriggers);
```

- 初始化表头:

通过遍历 LR(1) DFA 的状态和转移, 找出所有用于转移的终结符 (不包含非终结符), 并将它们存储在 `finalString` 集合中。

- 排序和添加特殊符号:

将 `finalString` 集合转换为列表 `finalStringList` 并进行排序。

检查排序后的列表中是否包含结束符号 `$`, 如果没有, 则将其添加到列表中。

类似地, 处理所有非终结符 `nonFinalizers`, 并将它们排序后添加到 `header` 中。

- 构建分析表的行列:

设置分析表的行数为 LR(1) DFA 的状态数 `lalr1.size`。

设置分析表的列数为 `header` 的大小。

- 创建反向哈希映射:

创建一个反向哈希映射 `revHash`, 将 LR(1) DFA 的状态 ID 映射回状态对象。

- 初始化表头哈希:

创建一个 `headerHash` 哈希表, 用于将表头中的每个终结符映射到列索引。

- 填充分析表:

遍历每个状态, 对于每个状态中的每个项目 (`Item`), 执行以下操作:

- 如果项目是一个归约项 (即项目的位置 `item.pos` 等于产生式的长度), 则根据项目的信息填充分析表的相应单元格。
- 如果项目是一个移进项 (即项目的位置 `item.pos` 不等于产生式的长度), 则根据 LR(1) DFA 的转移信息填充分析表的相应单元格。

- 处理归约项:

对于归约项，如果项目名称是起始符号并且归约项的 `next` 集合包含结束符号 `$`，则该状态是接受状态，对应的分析表项设置为“接受”。

对于其他归约项，如果归约项目已经在 `recursion` 列表中，则使用其索引作为规约的产生式索引；如果不在，则添加到 `recursion` 列表中，并使用新索引作为规约的产生式索引。

- 处理移进项:

对于每个终结符，如果存在对应的移进状态，则在分析表中对应的单元格填入移进状态编号。

- 处理非终结符移进:

对于非终结符，如果存在对应的移进状态，则在分析表中对应的单元格填入移进状态编号，并标记为非终结符移进（通常用不同的标记区分）。

8 句子分析的实现

完成 LALR(1)分析表的构建后，可以进行句子的分析，下面先展示我的代码实现，并对代码进行讲解。

```
1. // 获取待分析的句子
2. QString sentence = ui->sentenceEdit->text();
3. ui->resultTableWidget->setRowCount(sentence.size() * 3);
4. if (sentence == "") {
5.     QMessageBox::warning(this, "警告", "待分析的句子不能为空", QMessageBox::Yes);
6.     return;
7. }
8. sentence += "$";
9.
10. // 符号栈栈顶
11. int sentenceTop = 0;
12. // 分析栈
13. QStack<AnalysisItem> analysisStack;
14. AnalysisItem firstItem;
15. firstItem.kind = 1;
16. firstItem.state = 0;
17. analysisStack.push(firstItem);
18. // 分析栈计数器
```



```

19.int cnt = 0;
20.
21.// 打印函数
22.std::function<void(void)> printAnalysis = [&]() {
23.    QString analysisString;
24.    for (auto item: analysisStack) {
25.        if (item.kind == 1) {
26.            analysisString += QString::number(item.state)
+ " ";
27.        } else {
28.            analysisString += item.ch + " ";
29.        }
30.    }
31.    ui->resultTableWidget->setRowCount(cnt + 1);
32.    ui->resultTableWidget->setItem(cnt, 0, new QTableWidgetItem(analysisString));
33.    ui->resultTableWidget->setItem(cnt, 1, new QTableWidgetItem(sentence.mid(sentenceTop)));
34.    cnt++;
35.    qDebug() << analysisString << " " << sentence.mid(sentenceTop);
36.};
37.
38.// 利用分析表分析句子
39.printAnalysis();
40.while (true) {
41.//         if (sentenceTop == sentence.size()) sentence
+= "@";
42.    if (analysisStack.empty()) {
43.        QMessageBox::warning(this, "警告", "待分析的句子与文法不匹配", QMessageBox::Yes);
44.        return;
45.    }
46.    AnalysisItem analysisItem = analysisStack.top();
47.    if (analysisItem.kind == 2) {
48.        QMessageBox::warning(this, "警告", "待分析的句子与文法不匹配", QMessageBox::Yes);
49.        return;
50.    }
51.    if (!LALR1_table[analysisItem.state].contains(QString(sentence[sentenceTop]))) {
52.        sentence.insert(sentenceTop, '@');
53.    }
54.    LALR1TableWidgetItem lalr1TableWidgetItem = LALR1_table[analysisItem

```

```
        em.state][QString(sentence[sentenceTop])]);
55.     if (lalr1TableItem.kind == 1) {
56.
57.         AnalysisItem chAnalysisItem;
58.         chAnalysisItem.kind = 2;
59.         chAnalysisItem.ch = QString(sentence[sentenceTop++
        ]));
60.
61.         AnalysisItem stAnalysisItem;
62.         stAnalysisItem.kind = 1;
63.         stAnalysisItem.state = lalr1TableItem.idx;
64.
65.         analysisStack.push(chAnalysisItem);
66.         analysisStack.push(stAnalysisItem);
67.         printAnalysis();
68.
69.     } else if (lalr1TableItem.kind == 2) {
70.
71.         // 获取规约规则
72.         Item recursionItem = recursion[lalr1TableItem.idx]
        ;
73.
74.         // 删除规约右侧元素
75.         int popCnt = recursionItem.pos * 2;
76.         while (!analysisStack.empty() && popCnt) {
77.             analysisStack.pop();
78.             popCnt--;
79.         }
80.         if (popCnt || analysisStack.empty()) {
81.             QMessageBox::warning(this, "警告", "待分析的句子
            与文法不匹配", QMessageBox::Yes);
82.             return;
83.         }
84.
85.         // 增加规约左侧元素
86.         AnalysisItem nextAnalysisItem = analysisStack.top(
        );
87.         if (nextAnalysisItem.kind != 1) {
88.             QMessageBox::warning(this, "警告", "待分析的句子
            与文法不匹配", QMessageBox::Yes);
89.             return;
90.         }
91.         if (!LALR1_table[nextAnalysisItem.state].contains(
            recursionItem.name)) {
```

```

92.         sentence.insert(sentenceTop, '@');
93.     }
94.     AnalysisItem chAnalysisItem;
95.     chAnalysisItem.kind = 2;
96.     chAnalysisItem.ch = recursionItem.name;
97.     AnalysisItem stAnalysisItem;
98.     stAnalysisItem.kind = 1;
99.     stAnalysisItem.state = LALR1_table[nextAnalysisItem.state][recursionItem.name].idx;
100.    analysisStack.push(chAnalysisItem);
101.    analysisStack.push(stAnalysisItem);
102.    printAnalysis();
103.
104.    } else if (lalr1TableItem.kind == 3) {
105.        QMessageBox::warning(this, "警告", "待分析的句子与文法不匹配", QMessageBox::Yes);
106.        return;
107.    } else if (lalr1TableItem.kind == 4) {
108.        ui->resultTableWidget->resizeColumnsToContents();
109.        QMessageBox::warning(this, "提醒", "分析完毕，" + sentence + "属于该文法的句子", QMessageBox::Yes);
110.        return;
111.    } else {
112.        QMessageBox::warning(this, "警告", "待分析的句子与文法不匹配", QMessageBox::Yes);
113.        return;
114.    }
115. }

```

这段代码实现了使用 LALR(1) 分析表对用户输入的句子进行语法分析的过程。以下是代码的详细解释：

- 获取并预处理句子:
 从用户界面获取待分析的句子，并检查它是否为空。
 如果句子为空，则弹出警告并终止程序。
 将结束符号 \$ 添加到句子末尾，以表示输入的结束。
- 初始化分析环境:
 设置分析栈 `analysisStack` 并将其初始状态压入栈中。
 初始化句子栈顶指针 `sentenceTop` 和分析栈计数器 `cnt`。
- 定义打印函数:

使用 `lambda` 表达式定义 `printAnalysis` 函数,用于打印当前分析栈的状态和句子栈顶的内容。

- 分析过程:

使用 `printAnalysis` 函数打印初始分析状态。

进入 `while` 循环,使用 `LALR(1)` 分析表对句子进行逐步分析。

- 处理移进项:

如果分析表项 `lalr1TableItem.kind` 为 1 (移进),则:

- 创建字符项 `chAnalysisItem` 和状态项 `stAnalysisItem`。
- 将字符项和状态项压入分析栈。
- 打印当前分析状态。

- 处理规约项:

如果分析表项 `lalr1TableItem.kind` 为 2 (规约),则:

- 从 `recursion` 数组中获取对应的规约规则。
- 从分析栈中弹出规约右侧的元素数量。
- 如果栈为空或弹出的元素数量不正确,则弹出警告并终止程序。
- 创建新的字符项和状态项,并将其压入分析栈。
- 打印当前分析状态。

- 处理非终结符移进:

如果分析表项 `lalr1TableItem.kind` 为 3 (非终结符移进),则弹出警告并终止程序。

- 处理接受状态:

如果分析表项 `lalr1TableItem.kind` 为 4 (接受),则:

- 弹出提醒框,告知用户分析完成,并且句子属于文法的句子。
- 调整表格列宽以适应内容,并终止程序。

- 处理错误情况:

如果分析栈为空,或者分析表项不是预期的类型,则弹出警告并终止程序。

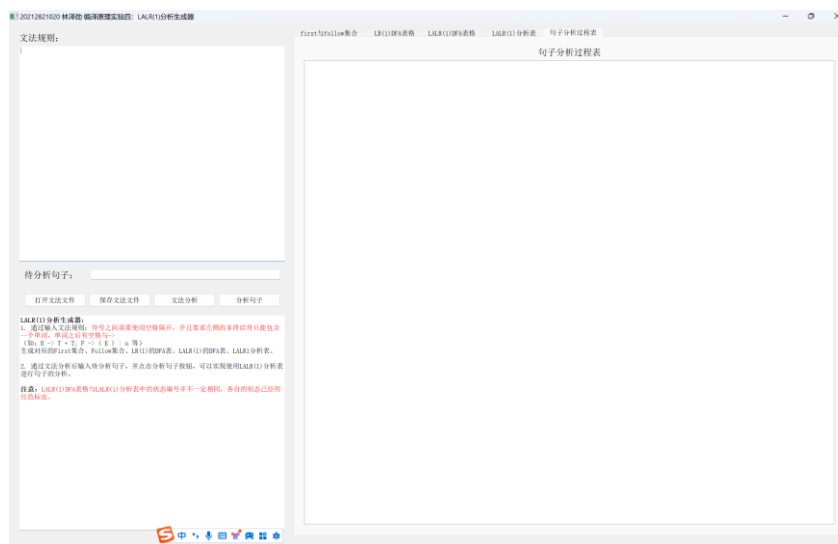
- 循环结束:

当句子被完全分析（即句子栈顶 `sentenceTop` 到达句子末尾），并且分析栈只剩下一个初始状态项时，循环结束。

代码中使用了栈来存储分析过程中的状态信息，包括当前状态、分析动作（移进、规约等）和待分析的输入符号。通过不断查询 `LALR(1)` 分析表并根据当前状态和输入符号来决定分析器的动作，实现了对输入句子的语法分析。如果在分析过程中遇到任何错误（如栈操作错误或分析表项不匹配），程序会弹出警告并终止。如果句子被成功分析，程序会弹出提醒框告知用户分析结果。

9 测试及结果展示

9.1 界面展示



9.2 测试 1（测试网盘样例）：

测试 1 文法：

```
exp -> exp addop term | term  
addop -> + | -  
term -> term mulop factor | factor  
mulop -> * | /  
factor -> ( exp ) | n
```

测试 1 结果：

First集合			follow集合		
非终结符	first集合		非终结符	follow集合	
1 term	(,n		1 term	+),-, \$	
2 term_nonleft	*,/, @		2 term_nonleft	+),-, \$	
3 addop	+,-		3 addop	(,n	
4 exp_nonleft	+,-, @		4 exp_nonleft), \$	
5 factor	(,n		5 factor	+,*),-, \$, /	
6 exp	(,n		6 exp), \$	
7 mulop	*, /		7 mulop	(,n	

LR(1) DFA表格						
	(term	+	term_nonleft	*	addop
0: term_nonleft -> . mulop factor term_nonleft , + ,) , - mulop -> . / , (, n term -> factor . term_nonleft , + ,) , - term_nonleft -> . @ , + ,) , - mulop -> . * , (, n				27	5	
1: term -> factor term_nonleft , + ,) , -						
2: exp_nonleft -> addop term . exp_nonleft ,) exp_nonleft -> . addop term exp_nonleft ,) addop -> . - , (, n exp_nonleft -> . @ ,) addop -> . + , (, n		20				15
3: factor -> (exp .) , + , * , - , \$, /						
4: exp_nonleft -> @ . ,)						
5: factor -> (exp) . , + , * , - , \$, /						
6: exp_nonleft -> addop term exp_nonleft . , \$						
7: factor -> (exp) . , + , * ,) , - , \$, /						
8: mulop -> * . , (, n						
9: factor -> . (exp) , + , * ,) , - , \$, / exp -> . term exp_nonleft ,) term -> . factor term_nonleft , + ,) , - factor -> . (exp) , + , * , - , \$, / factor -> . n , + , * ,) , - , \$, /	11	14				
10: term -> . factor term_nonleft , + , - , \$ exp_nonleft -> addop . term exp_nonleft , \$						

LALR(1) DFA表格

	(term	+	term_nonleft	*	addop
0: mulop -> *, (, n						
1: addop -> -, (, n						
2: factor -> . n, +, *,), -, \$, / term_nonleft -> mulop . factor term_nonleft, +,), -, \$ factor -> . (exp), +, *,), -, \$, /	10					
3: addop -> +, (, n						
4: exp_nonleft -> @,), \$						
5: term -> . factor term_nonleft, +, -, \$ exp -> . term exp_nonleft, \$ factor -> . n, +, *, -, \$, / factor -> . (exp), +, *, -, \$, /	10	13				
6: term_nonleft -> mulop factor term_nonleft, +,), -, \$						
7: factor -> . n, +, *,), -, \$, / factor -> . (exp), +, *,), -, \$, / term -> . factor term_nonleft, +,), -, \$ exp_nonleft -> addop . term exp_nonleft,), \$	10	15				
8: exp_nonleft -> addop term exp_nonleft,), \$						
9: term -> factor term_nonleft, +,), -, \$						
10: mulop -> /, (, n						
11: factor -> (exp), +, *,), -, \$, /						

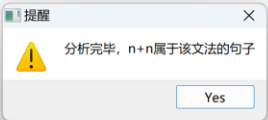
LALR(1) 分析表

	()	*	+	-	/	@	n	\$
0 s10								s9	
1			s5			s4	s3		
2		r0		r0	r0				r0
3		r1		r1	r1				r1
4 r2								r2	
5 r3								r3	
6 s10								s9	
7			s5			s4	s3		
8		r4		r4	r4				r4
9		r5	r5	r5	r5	r5			r5
10 s10								s9	
11		s12							
12		r6	r6	r6	r6	r6			r6
13				s19	s17		s18		14
14 s10				s19	s17		s18	s9	
15				s19	s17		s18		14
16		r7							r7
17 r8								r8	
18		r9							r9
19 r10								r10	
20		r11							接受

测试 1 测试分析句子：

9.2.1 句子 1: $n+n$

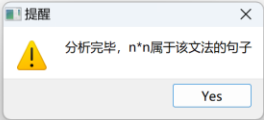
句子分析过程表		
	分析栈	符号栈
1	0	$n+n\$$
2	0 n 9	$+n\$$
3	0 factor 1	$+n\$$
4	0 factor 1 @ 3	$+n\$$
5	0 factor 1 term_nonleft 2	$+n\$$
6	0 term 13	$+n\$$
7	0 term 13 + 19	$n\$$
8	0 term 13 addop 14	$n\$$
9	0 term 13 addop 14 n 9	$\$$
10	0 term 13 addop 14 factor 1	$\$$
11	0 term 13 addop 14 factor 1 @ 3	$\$$
12	0 term 13 addop 14 factor 1 term_nonleft 2	$\$$
13	0 term 13 addop 14 term 15	$\$$
14	0 term 13 addop 14 term 15 @ 18	$\$$
15	0 term 13 addop 14 term 15 exp_nonleft 16	$\$$
16	0 term 13 exp_nonleft 20	$\$$



9.2.2 句子 2: $n*n$

句子分析过程表

	分析栈	符号栈
1	0	n*n\$
2	0 n 9	*n\$
3	0 factor 1	*n\$
4	0 factor 1 * 5	n\$
5	0 factor 1 mulop 6	n\$
6	0 factor 1 mulop 6 n 9	\$
7	0 factor 1 mulop 6 factor 7	\$
8	0 factor 1 mulop 6 factor 7 @ 3	\$
9	0 factor 1 mulop 6 factor 7 term_nonleft 8	\$
10	0 factor 1 term_nonleft 2	\$
11	0 term 13	\$
12	0 term 13 @ 18	\$
13	0 term 13 exp_nonleft 20	\$




9.2.3 句子 3: (n+n)*n

句子分析过程表

	分析栈	符号栈
1	0	(n+n)*n\$
2	0 (10	n+n)*n\$
3	0 (10 n 9	+n)*n\$
4	0 (10 factor 1	+n)*n\$
5	0 (10 factor 1 @ 3	+n)*n\$
6	0 (10 factor 1 term_nonleft 2	+n)*n\$
7	0 (10 term 13	+n)*n\$
8	0 (10 term 13 + 19	n)*n\$
9	0 (10 term 13 addop 14	n)*n\$
10	0 (10 term 13 addop 14 n 9)n\$
11	0 (10 term 13 addop 14 factor 1)n\$
12	0 (10 term 13 addop 14 factor 1 @ 3)n\$
13	0 (10 term 13 addop 14 factor 1 term_nonleft 2)n\$
14	0 (10 term 13 addop 14 term 15)n\$
15	0 (10 term 13 addop 14 term 15 @ 18)n\$
16	0 (10 term 13 addop 14 term 15 exp_nonleft 16)n\$
17	0 (10 term 13 exp_nonleft 20)n\$
18	0 (10 exp 11)n\$
19	0 (10 exp 11) 12	*n\$
20	0 factor 1	*n\$
21	0 factor 1 * 5	n\$
22	0 factor 1 mulop 6	n\$
23	0 factor 1 mulop 6 n 9	\$
24	0 factor 1 mulop 6 factor 7	\$
25	0 factor 1 mulop 6 factor 7 @ 3	\$
26	0 factor 1 mulop 6 factor 7 term_nonleft 8	\$

提醒

 分析完毕, (n+n)*n属于该文法的句子


Yes

9.2.4 句子 4: (n-n)/n

句子分析过程表

	分析栈	符号栈
1	0	(n-n)/n\$
2	0 (10	n-n)/n\$
3	0 (10 n 9	-n)/n\$
4	0 (10 factor 1	-n)/n\$
5	0 (10 factor 1 @ 3	-n)/n\$
6	0 (10 factor 1 term_nonleft 2	-n)/n\$
7	0 (10 term 13	-n)/n\$
8	0 (10 term 13 - 17	n)/n\$
9	0 (10 term 13 addop 14	n)/n\$
10	0 (10 term 13 addop 14 n 9)/n\$
11	0 (10 term 13 addop 14 factor 1)/n\$
12	0 (10 term 13 addop 14 factor 1 @ 3)/n\$
13	0 (10 term 13 addop 14 factor 1 term_nonleft 2)/n\$
14	0 (10 term 13 addop 14 term 15)/n\$
15	0 (10 term 13 addop 14 term 15 @ 18)/n\$
16	0 (10 term 13 addop 14 term 15 exp_nonleft 16)/n\$
17	0 (10 term 13 exp_nonleft 20)/n\$
18	0 (10 exp 11)/n\$
19	0 (10 exp 11) 12	/n\$
20	0 factor 1	/n\$
21	0 factor 1 / 4	n\$
22	0 factor 1 mulop 6	n\$
23	0 factor 1 mulop 6 n 9	\$
24	0 factor 1 mulop 6 factor 7	\$
25	0 factor 1 mulop 6 factor 7 @ 3	\$
26	0 factor 1 mulop 6 factor 7 term_nonleft 8	\$

提醒

 分析完毕, (n-n)/n属于该文法的句子

Yes

9.3 测试 2（测试网盘样例）：

测试 2 文法：

E -> E + T | T

T -> T * F | F

F -> (E) | n

测试 2 结果：

First集合

非终结符	first集合
1 T	(,n
2 E	(,n
3 E_nonleft	+,@
4 F	(,n
5 T_nonleft	*,@

follow集合

非终结符	follow集合
1 T	+,),,\$
2 E),,\$
3 E_nonleft),,\$
4 F	+,*,),,\$
5 T_nonleft	+,),,\$

LR(1) DFA表格

	(+	T	*	n)	E_nonleft
0: T_nonleft -> @ . , + ,)							
1: E -> . T E_nonleft , \$ T -> . F T_nonleft , + , \$ F -> . n , + , * , \$ F -> . (E) , + , * , \$	12		1		11		
2: E_nonleft -> . @ , \$ E -> T . E_nonleft , \$ E_nonleft -> . + T E_nonleft , \$		4					3
3: E_nonleft -> + T . E_nonleft ,) E_nonleft -> . + T E_nonleft ,) E_nonleft -> . @ ,)		23					25
4: T -> F T_nonleft . , + , \$							
5: F -> . n , + , * ,) E_nonleft -> + . T E_nonleft ,) T -> . F T_nonleft , + ,) F -> . (E) , + , * ,)	21		24		13		
6: T_nonleft -> * . F T_nonleft , + , \$ F -> . n , + , * , \$ F -> . (E) , + , * , \$	12				11		
7: E -> T E_nonleft . , \$							
8: F -> . n , + , * ,) F -> (. E) , + , * , \$ T -> . F T_nonleft , + ,) E -> . T E_nonleft ,) F -> . (E) , + , * ,)	21		22		13		14
9: E -> T E_nonleft . ,)							

LALR(1) DFA表格

	(+	T	*	n)	E_nonleft
0: T_nonleft -> @ . , + ,) , \$							
1: T -> F T_nonleft . , + ,) , \$							
2: E_nonleft -> @ . ,) , \$							
3: T_nonleft -> * F T_nonleft . , + ,) , \$							
4: T_nonleft -> * F . T_nonleft , + ,) , \$ T_nonleft -> . * F T_nonleft , + ,) , \$ T_nonleft -> . @ , + ,) , \$				7			
5: E_nonleft -> . @ .) , \$ E_nonleft -> + T . E_nonleft ,) , \$ E_nonleft -> . + T E_nonleft ,) , \$		4					16
6: E_nonleft -> + T E_nonleft . ,) , \$							
7: E_nonleft -> . @ .) , \$ E_nonleft -> . + T E_nonleft ,) , \$ E -> T . E_nonleft ,) , \$		4					3
8: T_nonleft -> * . F T_nonleft , + ,) , \$ F -> . (E) , + , * , \$ F -> . n , + , * , \$	12				11		
9: F -> (. E) , + , * ,) , \$ F -> . n , + , * ,) T -> . F T_nonleft , + ,) E -> . T E_nonleft ,) F -> . (E) , + , * ,)	12		1		11		
10: E -> . T E_nonleft , \$ T -> . F T_nonleft , + ,)							

LALR(1)分析表									
	()	*	+	@	n	\$	E	E_nonleft
0	s12					s11			5
1				s4	s2			3	
2		r0					r0		
3		r1					接受		
4	s12					s11			5
5			s7		s9				
6		r2		r2			r2		
7	s12					s11			8
8			s7		s9				
9		r3		r3			r3		
10		r4		r4			r4		
11		r5	r5	r5			r5		
12	s12					s11		13	5
13		s14							
14		r6	r6	r6			r6		
15				s4	s2			16	
16		r7					r7		

测试 2 测试分析句子：

9.3.1 句子 1： n+n

句子分析过程表

	分析栈	符号栈
1	0	n+n\$
2	0 n 11	+n\$
3	0 F 5	+n\$
4	0 F 5 @ 9	+n\$
5	0 F 5 T_nonleft 6	+n\$
6	0 T 1	+n\$
7	0 T 1 + 4	n\$
8	0 T 1 + 4 n 11	\$
9	0 T 1 + 4 F 5	\$
10	0 T 1 + 4 F 5 @ 9	\$
11	0 T 1 + 4 F 5 T_nonleft 6	\$
12	0 T 1 + 4 T 15	\$
13	0 T 1 + 4 T 15 @ 2	\$
14	0 T 1 + 4 T 15 E_nonleft 16	\$
15	0 T 1 E_nonleft 3	\$

提醒

分析完毕, n+n属于该文法的句子

Yes

9.3.2 句子 2: $n*n$

句子分析过程表

	分析栈	符号栈
1	0	n*n\$
2	0 n 11	*n\$
3	0 F 5	*n\$
4	0 F 5 * 7	n\$
5	0 F 5 * 7 n 11	\$
6	0 F 5 * 7 F 8	\$
7	0 F 5 * 7 F 8 @ 9	\$
8	0 F 5 * 7 F 8 T_nonleft 10	\$
9	0 F 5 T_nonleft 6	\$
10	0 T 1	\$
11	0 T 1 @ 2	\$
12	0 T 1 E_nonleft 3	\$

提醒

分析完毕, n*n属于该文法的句子


Yes

9.3.3 句子 3: $(n+n)*n$

句子分析过程表

	分析栈	符号栈
1	0	$(n+n)*n\$$
2	0 (12	$n+n)*n\$$
3	0 (12 n 11	$+n)*n\$$
4	0 (12 F 5	$+n)*n\$$
5	0 (12 F 5 @ 9	$+n)*n\$$
6	0 (12 F 5 T_nonleft 6	$+n)*n\$$
7	0 (12 T 1	$+n)*n\$$
8	0 (12 T 1 + 4	$n)*n\$$
9	0 (12 T 1 + 4 n 11	$)n\$$
10	0 (12 T 1 + 4 F 5	$)n\$$
11	0 (12 T 1 + 4 F 5 @ 9	$)n\$$
12	0 (12 T 1 + 4 F 5 T_nonleft 6	$)n\$$
13	0 (12 T 1 + 4 T 15	$)n\$$
14	0 (12 T 1 + 4 T 15 @ 2	$)n\$$
15	0 (12 T 1 + 4 T 15 E_nonleft 16	$)n\$$
16	0 (12 T 1 E_nonleft 3	$)n\$$
17	0 (12 E 13	$)n\$$
18	0 (12 E 13) 14	$*n\$$
19	0 F 5	$*n\$$
20	0 F 5 * 7	$n\$$
21	0 F 5 * 7 n 11	$\$$
22	0 F 5 * 7 F 8	$\$$
23	0 F 5 * 7 F 8 @ 9	$\$$
24	0 F 5 * 7 F 8 T_nonleft 10	$\$$
25	0 F 5 T_nonleft 6	$\$$
26	0 T 1	$\$$

提醒

 分析完毕, $(n+n)*n$ 属于该文法的句子

Yes

9.3.4 句子 4 (测试错误句子是否会被捕获): $(n-n)/n$

句子分析过程表

	分析栈	符号栈
1	0	(n-n)/n\$
2	0 (1 2	n-n)/n\$
3	0 (1 2 n 1 1	-n)/n\$

警告

待分析的句子与文法不匹配

Yes

9.4 测试 3：（测试简单 DFA，讲稿的样例）

测试 3 文法：

A -> (A) | a

测试 3 结果：

First集合			follow集合		
	非终结符	first集合		非终结符	follow集合
1	A	a,(1	A), \$

LR(1)DFA表格

	(a)	A
0: A _{new} → . A, \$ A → . (A), \$ A → . a, \$	2	9		1
1: A → . (A),) A → . a,) A → . (A),)	5	6		7
2: A _{new} → A ., \$				
3: A → a ., \$				
4: A → . (A), \$ A → . (A),) A → . a,)	5	6		3
5: A → (A .), \$			4	
6: A → (A) ., \$				
7: A → a .,)				
8: A → (A) .,)				
9: A → (A .),)			8	

LALR(1)DFA表格

	(a)	A
0: A → (A) ., \$				
1: A → a ., \$				
2: A _{new} → . A, \$ A → . (A), \$ A → . a, \$	2	5		1
3: A _{new} → A ., \$				
4: A → (A .), \$			4	
5: A → . (A), \$ A → . (A),) A → . a,)	2	5		3

LALR(1) 分析表					
	()	a	\$	A
0 s2			s5		1
1				接受	
2 s2			s5		3
3		s4			
4		r0		r0	
5		r1		r1	

测试 3 测试分析句子：

9.4.1 句子 1：((a))

句子分析过程表

	分析栈	符号栈
1	0	((a))\$
2	0 (2	(a))\$
3	0 (2 (2	a))\$
4	0 (2 (2 a 5)\$
5	0 (2 (2 A 3)\$
6	0 (2 (2 A 3) 4	\$
7	0 (2 A 3)\$
8	0 (2 A 3) 4	\$
9	0 A 1	\$

提醒

分析完毕, ((a))属于该文法的句子

Yes

9.4.2 句子 2: (a)

句子分析过程表

	分析栈	符号栈
1	0	(a)\$
2	0 (2	a)\$
3	0 (2 a 5)\$
4	0 (2 A 3)\$
5	0 (2 A 3) 4	\$
6	0 A 1	\$

提醒

分析完毕, (a)属于该文法的句子


Yes

9.4.3 句子 3: ((a)

句子分析过程表

	分析栈	符号栈
1	0	((a)\$
2	0 (2	(a)\$
3	0 (2 (2	a)\$
4	0 (2 (2 a 5)\$
5	0 (2 (2 A 3)\$
6	0 (2 (2 A 3) 4	\$
7	0 (2 A 3	\$

警告

待分析的句子与文法不匹配

Yes

9.5 测试 4: (包含规约-规约冲突)

测试 4 文法:

S -> A | B

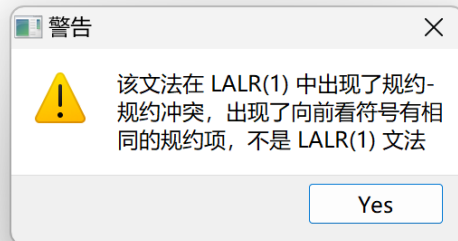
A -> a S | @

B -> b S | @

测试 4 结果:

文法规则:

$S \rightarrow A \mid B$
 $A \rightarrow a S \mid @$
 $B \rightarrow b S \mid @$



9.6 测试 5: (包含移进-规约冲突)

测试 5 文法:

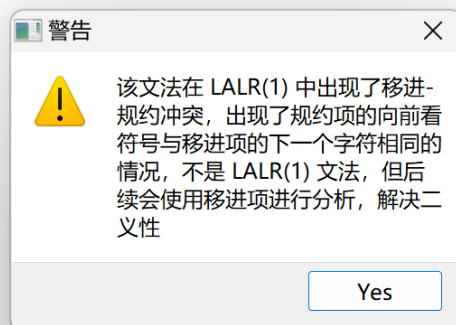
$S \rightarrow T S \mid @$

$T \rightarrow a T b \mid a$

测试 5 结果:

文法规则:

$S \rightarrow T S \mid @$
 $T \rightarrow a T b \mid a$



First集合			follow集合		
非终结符	first集合		非终结符	follow集合	
1 T	a		1 T	a,b,\$	
2 S	a,@		2 S	\$	

LR(1)DFA表格					
	a	T	b	S	@
0: T->aTb.,a,\$					
1: S->@.,,\$					
2: T->.,aTb,b T->.,a,b T->a.Tb,a,\$ T->a.,a,\$	2	5			
3: S_new->S.,,\$					
4: S->TS.,,\$					
5: T->.,aTb,b T->a.Tb,b T->.,a,b T->a.,b	2	3			
6: T->.,aTb,a,\$ S->.,@,\$ S_new->.,S,\$ T->.,a,a,\$ S->.,TS,\$	1	9		8	7
7: T->aT.b,b			4		
8: T->.,aTb,a,\$ S->.,@,\$ S->T.S,\$ T->.,a,a,\$ S->.,TS,\$	1	9		10	7
9: T->aTb.,b					
10: T->aT.h.a,\$			6		

LALR(1) DFA表格

	a	T	b	S	@
0: S->@., \$					
1: T->aTb., a, b, \$					
2: S_new->S., \$					
3: S->TS., \$					
4: T->aT.b, a, b, \$			3		
5: T->.aTb, a, \$ S->.@, \$ S_new->.S, \$ T->.a, a, \$ S->.TS, \$	1	6		5	4
6: T->a., a, b, \$ T->.aTb, b T->.a, b T->a.Tb, a, b, \$	1	2			
7: T->.aTb, a, \$ S->.@, \$ S->T.S, \$ T->.a, a, \$ S->.TS, \$	1	6		7	4

LALR(1) 分析表


	@	a	b	\$	S	T
0 s4	s1				5	6
1	s1	r0	r0			2
2		s3				
3	r1	r1	r1			
4			r2			
5			接受			
6 s4	s1				7	6
7			r3			

测试 5 测试分析句子：

9.6.1 句子 1：a

句子分析过程表		
	分析栈	符号栈
1	0	a\$
2	0 a 1	\$
3	0 T 6	\$
4	0 T 6 @ 4	\$
5	0 T 6 S 7	\$
6	0 S 5	\$

提醒

 分析完毕，a属于该文法的句子

Yes

9.6.2 句子 2：aab

句子分析过程表

分析栈	符号栈
1 0	aab\$
2 0 a 1	ab\$
3 0 a 1 a 1	b\$
4 0 a 1 T 2	b\$
5 0 a 1 T 2 b 3	\$
6 0 T 6	\$
7 0 T 6 @ 4	\$
8 0 T 6 S 7	\$
9 0 S 5	\$

提醒

分析完毕, aab属于该文法的句子

Yes

9.6.3 句子 3: aabb

句子分析过程表

分析栈	符号栈
1 0	aabb\$
2 0 a 1	abb\$
3 0 a 1 a 1	bb\$
4 0 a 1 T 2	bb\$
5 0 a 1 T ...	b\$
6 0 T 6	b\$
7 0 T 6 @ 4	b\$

警告

待分析的句子与文法不匹配

Yes

9.6.4 句子 4: aaabb

句子分析过程表

	分析栈	符号栈
1	0	aaabb\$
2	0 a 1	aabb\$
3	0 a 1 a 1	abb\$
4	0 a 1 a 1 a 1	bb\$
5	0 a 1 a 1 T 2	bb\$
6	0 a 1 a 1 T 2 b 3	b\$
7	0 a 1 T 2	b\$
8	0 a 1 T 2 b 3	\$
9	0 T 6	\$
10	0 T 6 @ 4	\$
11	0 T 6 S 7	\$
12	0 S 5	\$

提醒
分析完毕, aaabb属于该文法的句子
Yes

四、实验总结（心得体会）

通过进行构建 LALR(1) 分析生成器的实验，我获得了深刻的认识和宝贵的经验。在这个过程中，我不仅加深了对编译原理的理解，尤其是在自底向上语法分析和有限状态机方面的知识，而且还锻炼了我的编程技巧和问题解决能力。

实验开始时，我首先需要理解 LALR(1) 分析器的工作原理，包括文法的预备处理、FIRST 和 FOLLOW 集合的计算、项目集的构建、DFA 状态的生成以及冲突的检测和解决。这些理论知识是实现 LALR(1) 分析器的基础，只有深入理解这些概念，才能正确地将它们转化为代码。

在编程实现阶段，我选择了 C++ 语言与 Qt 框架，因为它提供了强大的数据结构和算法支持，非常适合进行编译器相关的开发。我首先搭建了软件的基本框架，包括文法规则的输入界面、数据结构的定义和分析表的生成算法。在实现过程中，我遇到了不少挑战，比如如何高效地存储和检索状态信息，如何准确地实现闭包和转移函数，以及如何检测 and 解决分析中的冲突。通过不断调试和优化代码，我逐步克服了这些困难。

实验的后期，我专注于软件的测试和调试。我设计了多种测试案例，包括具有不同类型冲突的文法和正常文法，以验证分析器的正确性和鲁棒性。测试过程中发现的问题迫使我重新审视和修正代码，这个过程提高了我的调试技巧和逻辑思维能力。

总的来说，这次实验不仅让我对编译原理有了更深刻的理解，而且提升了我的编程实践能力。我学会了如何将复杂的理论问题分解为可管理的编程任务，并逐步实现它们。同时，我也认识到了在软件开发过程中，文档编写、代码规范和测试的重要性。这些经验对我未来的学习和工作都将产生积极的影响。

五、参考文献：

- [1] [自底向上文法分析：LR\(1\) 分析 - 知乎 \(zhihu.com\)](https://www.zhihu.com/question/26666413/answer/111111111)
- [2] [语法分析笔记（四）——LR\(0\) SLR LR\(1\) LALR_lr slr lalr-CSDN 博客](https://blog.csdn.net/qq_34385447/article/details/101411111)
- [3] 编译原理参考书电子书

六、附录一：判断同心项与同心文法

```
1.  bool State::haveSameCore(State i, State j)
2.  {
3.      if (i.st.size() != j.st.size()) return false;
4.      auto iList = i.st.toList();
5.      auto jList = j.st.toList();
6.      qSort(iList.begin(), iList.end());
7.      qSort(jList.begin(), jList.end());
8.      for (int cnt = 0; cnt < i.st.size(); cnt++) {
9.          if (!Item::haveSameCore(iList[cnt], jList[cnt])) {
10.              return false;
11.          }
12.      }
13.      return true;
14.  }
15.
16.  bool Item::haveSameCore(Item i, Item j)
17.  {
18.      if (i.name == j.name && i.rule == j.rule && i.pos == j.
pos) return true;
19.      else return false;
20.  }
```