第一阶段实验报告：基于 ANTLR 的真实语法分析设计

姓名：闫悦斌　学号：3022244121

班级：计算机科学与技术 1 班　小组：第 9 小组

2025 年 11 月 28 日

摘要

本实验旨在利用 ANTLR 4 工具，针对五种具有代表性的真实语言（CSV, JSON, DOT, Cymbol, R）构建语法分析器。实验过程中，不仅根据语言规范编写了 '.g4' 语法文件，还针对教材中原有语法的不足（如 R 语言的换行符处理、浮点数支持等）进行了改进和优化。通过编写两组不同维度的原创测试用例，并生成可视化的语法分析树，验证了语法的正确性与鲁棒性。

目录

1 实验环境 **3**

2 实验内容与设计 **3**

2.1 CSV (逗号分隔值) 语法设计 3

2.1.1 实现逻辑详解 3

2.1.2 语法定义 (CSV.g4) 3

2.1.3 测试验证 4

2.2 JSON (数据交换格式) 语法设计 4

2.2.1 实现逻辑详解 4

2.2.2 语法定义 (JSON.g4) 5

2.2.3 测试验证 5

2.3 DOT (图形描述语言) 语法设计 6

2.3.1 实现逻辑详解 6

2.3.2 语法定义 (DOT.g4) 6

2.3.3 测试验证 7

2.4 Cymbol (类 C 语言) 语法设计 7

2.4.1 实现逻辑详解 7

2.4.2 语法定义 (Cymbol.g4) 8

2.4.3 测试验证 8

2.5 R (统计语言) 语法设计与改进 9

2.5.1 实现逻辑详解 9

2.5.2 最终语法定义 (R.g4) 9

2.5.3 测试验证 10

3 实验总结 **11**

3.1 遇到的挑战 11

3.2 结论 11

1 实验环境

- **操作系统**: Windows 11
- **开发语言**: Java (JDK 21)
- **工具库**: ANTLR 4.13.2
- **IDE/编辑器**: VS Code / Command Line

2 实验内容与设计

2.1 CSV (逗号分隔值) 语法设计

CSV 是最常见的数据格式。设计重点在于处理标题行（Header）以及包含特殊字符（如逗号、双引号）的字符串字段。

2.1.1 实现逻辑详解

CSV 的核心难点在于区分“普通文本”和“带引号的字符串”。在 `CSV.g4` 中设计了如下逻辑：

- **文件结构**：顶层规则 `file` 被定义为“一行标题 (`hdr`) + 多行数据 (`row+`)”，确保了文件至少包含数据，符合标准 CSV 规范。
- **复杂字段处理**：最关键的是 `STRING` 的词法规则。定义正则 `'"' ( '""' | ~'"' )* '"'`。该正则允许字符串被双引号包裹，且内部如果出现双引号 `""`，则视为对引号的转义（Escape）；否则匹配任意非引号字符。这完美解决了“逗号在引号内不应作为分隔符”的问题。

2.1.2 语法定义 (CSV.g4)

```
grammar CSV;
file : hdr row+ ;
row : field (',' field)* '\r'? '\n' ;
field : TEXT | STRING | ;
TEXT : [a-zA-Z0-9 ]+ ;
STRING : '"' ( '""' | ~'"' )* '"' ;
```

Listing 1: CSV.g4 核心代码

2.1.3 测试验证

为了验证语法的鲁棒性，设计了包含常规字段和特殊字符转义的测试用例。

图 1: CSV 测试用例 1 (基础数据) 语法分析树

图 2: CSV 测试用例 2 (特殊字符转义) 语法分析树

2.2 JSON (数据交换格式) 语法设计

JSON 的难点在于递归结构（对象嵌套对象）以及字符串的转义序列处理。

2.2.1 实现逻辑详解

JSON 的核心特征是递归嵌套，在 `JSON.g4` 中通过自引用的语法规则实现了这一点：

- **递归闭环**：定义 `value` 为核心规则，它可以是 `object` 或 `array`。而 `object` 包含 `pair`，`pair` 的值又可以是 `value`。这种 `value -> object -> pair -> value` 的闭环设计使得解析器能够处理无限深度的嵌套结构。

- **转义字符**：在 Lexer 中，`STRING` 规则不仅要匹配普通字符，还必须处理反斜杠转义。由于 ANTLR 的正则限制，手动展开了 Unicode 转义规则 `'u' [0-9a-fA-F]...`，以确保能正确识别如 `\u54c8` 这样的字符。

2.2.2 语法定义 (JSON.g4)

针对 ANTLR 在处理正则表达式时的限制，手动展开了 unicode 转义序列的匹配规则。

```
value : STRING | NUMBER | object | array | 'true' | 'false' | 'null' ;
STRING : '"' (ESC | ~["\\])* '"' ;
fragment ESC : '\\' (["\\/bfnrt] | 'u' [0-9a-fA-F][0-9a-fA-F][0-9a-fA-F
    ][0-9a-fA-F]) ;
```

Listing 2: JSON.g4 核心代码

2.2.3 测试验证

测试了嵌套对象、数组、布尔值以及科学计数法数值。

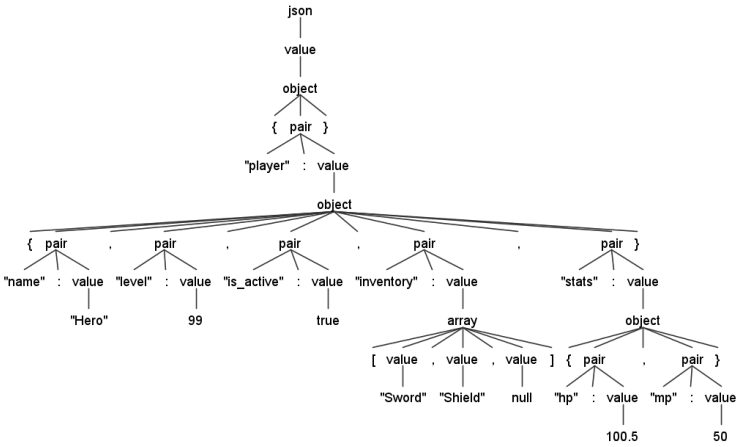

图 3: JSON 测试用例 1 (嵌套对象与数组) 语法分析树

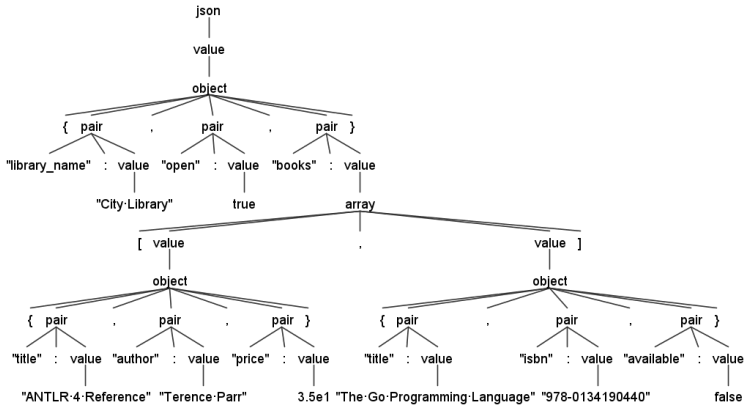

图 4: JSON 测试用例 2 (科学计数法) 语法分析树

2.3 DOT (图形描述语言) 语法设计

DOT 语言挑战在于关键字大小写不敏感以及 HTML 字符串的处理。

2.3.1 实现逻辑详解

DOT 语言的语法相对宽松，在设计时着重解决了以下两点：

- **大小写不敏感**：ANTLR 默认是大小写敏感的。为了兼容 DOT 规范（如 `graph`, `GRAPH`, `Graph` 均合法），在 Lexer 规则中使用了字符类组合，例如 `[Gg][Rr][Aa][Pp][Hh]`，从而在词法层面实现了忽略大小写的匹配。
- **ID 的多态性**：在 DOT 中，标识符（ID）的使用非常灵活。在 `id` 规则中将其定义为 `ID | STRING | HTML_STRING | NUMBER` 的集合。这意味着无论是普通单词、带引号的字符串、HTML 标签还是数字，都可以被作为节点的标识符，大大增强了语法的通用性。

2.3.2 语法定义 (DOT.g4)

利用字符类（Character Class）实现了忽略大小写的关键字匹配。

```
// 词法规则示例：忽略大小写
GRAPH : [Gg][Rr][Aa][Pp][Hh] ;
id : ID | STRING | HTML_STRING | NUMBER ;
HTML_STRING : '<' (TAG | ~[<>])* '>' ;
```

Listing 3: DOT.g4 核心代码

2.3.3 测试验证

测试了有向图、节点属性设置以及复杂标签。

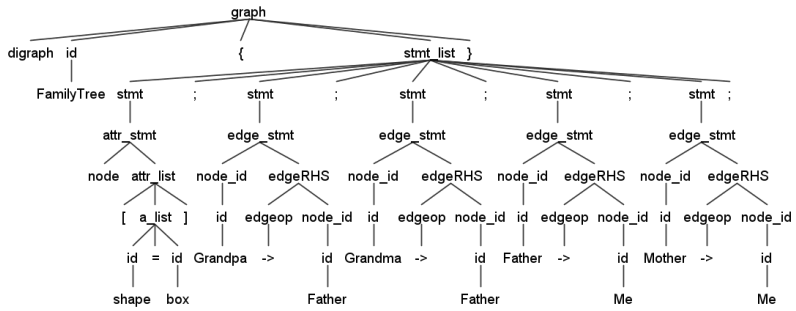

图 5: DOT 测试用例 1 (家庭树) 语法分析树

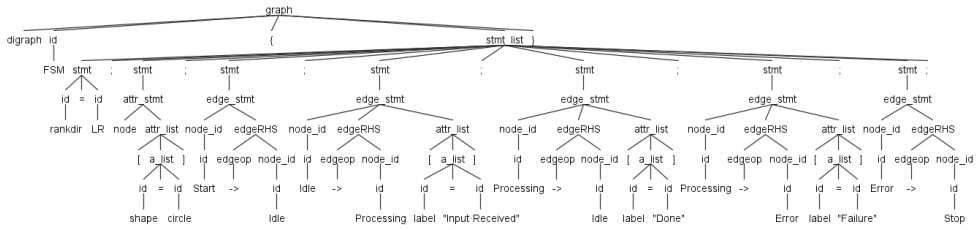

图 6: DOT 测试用例 2 (带属性的状态机) 语法分析树

2.4 Cymbol (类 C 语言) 语法设计

Cymbol 核心在于表达式的运算符优先级处理和函数定义。

2.4.1 实现逻辑详解

Cymbol 是一个典型的命令式语言，其语法设计的难点在于表达式的优先级：

- **隐式优先级**：在 expr 规则中，利用 ANTLR v4 的“备选分支顺序”特性来隐式定义优先级。将乘除法规则 expr ('*'|'/') expr 放置在加减法规则 expr ('+'|'-') expr 之前。
- **效果**：ANTLR 在生成解析器时，会优先匹配列表靠前的规则。因此，这确保了“先乘除后加减”的数学逻辑自动生效，无需像旧版工具（如 YACC）那样维护复杂的优先级表。此外，还补充了 '/' 除法支持，完善了四则运算。

2.4.2 语法定义 (Cymbol.g4)

通过备选分支顺序隐式定义了优先级，并补充了除法支持。

```
expr : ID '(' exprList? ')'       # Call
     | expr ('*'|'/') expr        # Mult (修正：添加了除法)
     | expr ('+'|'-') expr        # AddSub
     | expr '==' expr             # Equal
     ;
```

Listing 4: Cymbol.g4 核心代码

2.4.3 测试验证

测试了函数调用、算术运算、递归算法（GCD 与斐波那契）。

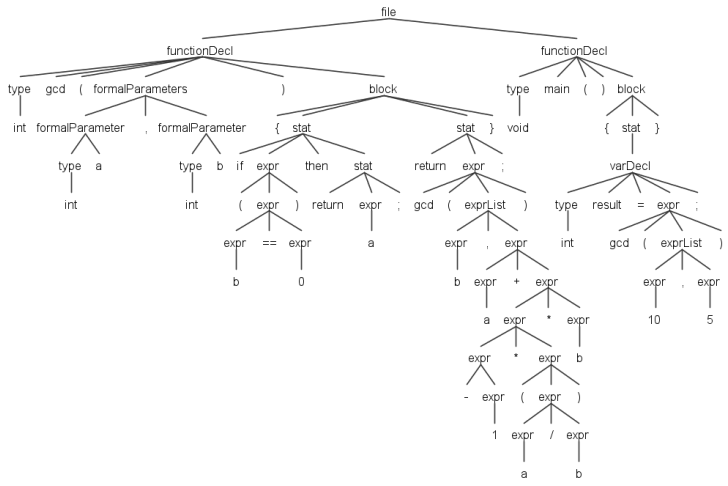

图 7: Cymbol 测试用例 1 (GCD 算法) 语法分析树

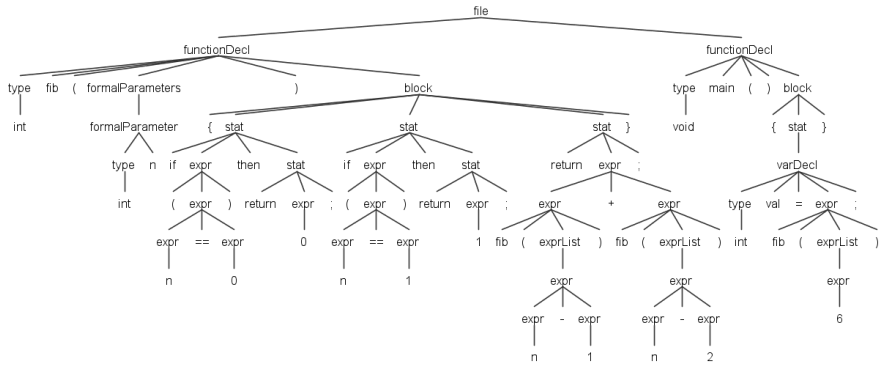

图 8: Cymbol 测试用例 2 (斐波那契递归) 语法分析树

2.5 R (统计语言) 语法设计与改进

针对教材语法的不足（如换行处理、浮点数支持）进行了重大改进。

2.5.1 实现逻辑详解

R 语言的语法非常灵活，教材提供的基础版本存在严重缺陷，在实验中进行了针对性修复：

- **换行符处理 (Newline Hell)**：R 语言允许在表达式中间换行（例如在 `if` 条件后）。原语法要求极其严格，导致稍微格式化代码就会报错。改进方案是在 `exprlist`、`if` 等关键结构的连接处插入 `NL*`（零个或多个换行符），极大地提高了对松散代码风格的兼容性。
- **浮点数歧义**：原语法中 `INT` 规则会优先匹配数字。当输入 `0.5` 时，`0` 被识别为 `INT`，导致剩下的 `.5` 无法解析。引入了优先级更高的 `FLOAT` 规则 `[0-9]+ '.' [0-9]*`，确保浮点数作为一个整体 Token 被识别。

2.5.2 最终语法定义 (R.g4)

```
grammar R;
prog : (expr_or_assign? NL)* EOF ; // 允许空行
expr : ... | FLOAT | '(' expr ')' | '{' exprlist '}'
     // 增强: if 规则允许换行
     | 'if' '(' expr ')' NL* expr (NL* 'else' NL* expr)? ;
exprlist : NL* expr_or_assign ((';'|NL)+ expr_or_assign?)* NL* | NL* ;
FLOAT : [0-9]+ '.' [0-9]* | '.' [0-9]+ ;
```

Listing 5: 改进后的 R.g4

2.5.3 测试验证

验证了浮点数运算、函数定义块结构以及跨行的 ‘if-else‘ 控制流。

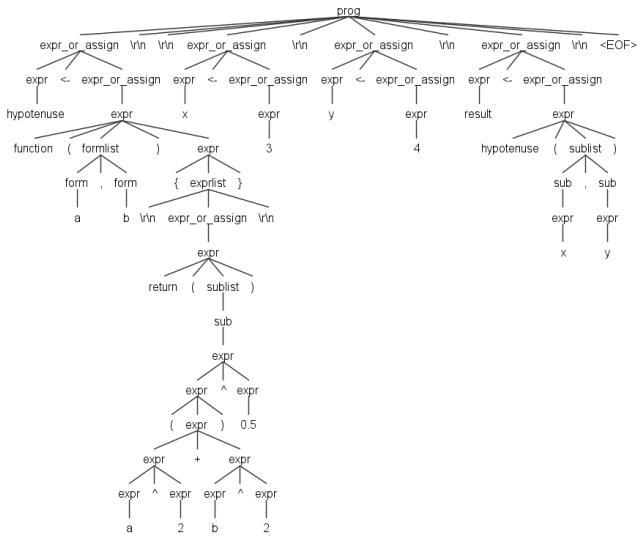

图 9: R 测试用例 1 (勾股定理与浮点数) 语法分析树

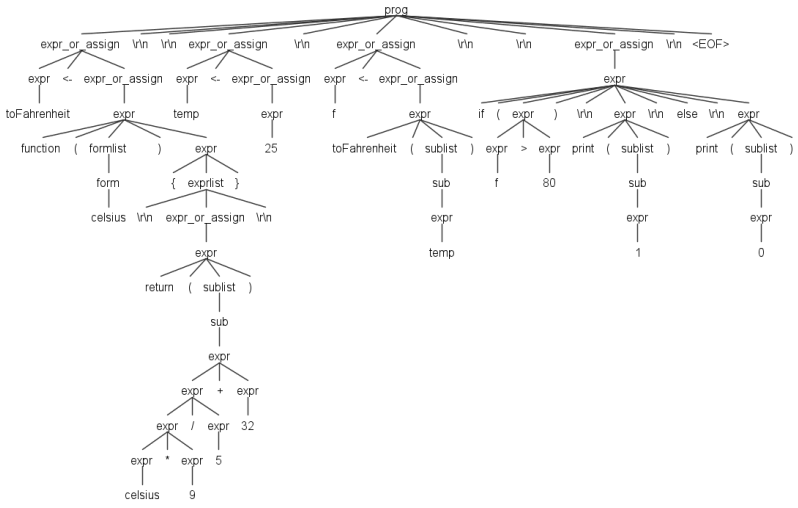

图 10: R 测试用例 2 (多行控制流) 语法分析树

3 实验总结

3.1 遇到的挑战

1. **环境配置：** 初始配置 CLASSPATH 时，未包含当前目录 (.) 和 jar 包全名，导致 ClassNotFoundException。通过显式指定路径解决。

2. **词法冲突：** 在 R 语言中，INT 和 ID 规则最初导致 0.5 被错误拆解。解决方案是引入优先级更高的 FLOAT 规则，并严格限制 ID 不能以“点号 + 数字”开头。

3. **语法严格性：** R 语言参考语法对换行符要求过高。通过在 exprlist 和 if 规则中插入 NL*，成功兼容了松散代码风格。

3.2 结论

本阶段实验成功实现了五种语言的语法分析器。特别是针对 R 语言的重构，不仅完成了作业要求，更深入理解了语法分析中消除歧义的重要性。