实现语法分析器

杨侯哲 李煦阳
杨科迪 孙一丁

2020 年 11 月—2021 年 10 月

目录

1 实验描述

学期已过半，我们实现编译器的征程也终于来到最有趣最关键的地方。

如果你还记得本学期初探索编译器的时候，我们曾使用`-fdump-tree-original-raw`获得 gcc 构建的语法树。对于`void main(){}`，它的输出如下。

```
;; Function main (null)
;; enabled by -tree-original

@1      bind_expr       type: @2       body: @3
@2      void_type       name: @4       algn: 8
@3      statement_list
@4      type_decl       name: @5       type: @2
@5      identifier_node strg: void     lngt: 4
```

我们知道，输出的每一行可以理解为语法树上的一个结点。每一个结点有其自身的类型、属性，以及数个子结点。本次作业便是要求构建这样一棵树并输出。

可以想象，gcc 采取了更复杂的语法定义去构建这棵树，并使用一些压缩算法处理这棵树。本次实验，我们只要求以最简洁最直观的方式将这棵树构建出来、展示结果。

构建出树后，我们之后的所有操作，比如树上各结点信息的获取与流动、类型检查、翻译至中间代码，都可以理解为对该树进行一次遍历。同时值得一提的是，若一些操作需要考虑语法，比如构建作用域树，那么通过语法树上一次遍历，便可以很容易完成。

2 实验要求

1. 完善预备工作 2 中你所设计的 SysY 语言的上下文无关文法，借助 Yacc 工具实现语法分析器：
   - 语法树数据结构的设计：结点类型的设计，不同类型的节点应保存的信息。
   - 扩展上下文无关文法，设计翻译模式。
   - 设计 Yacc 程序，实现能构造语法树的分析器。
   - 以文本方式输出语法树结构，验证语法分析器实现的正确性。
2. 无需撰写完整研究报告，但需要在雨课堂上提交本次实验的 gitlab 链接。
3. 上机课时，以小组为单位，向助教讲解程序。

3 实验流程

3.1 目录结构

本次实验框架代码的目录结构如下：

```
./
├── include
│   ├── Ast.h
│   ├── SymbolTable.h
│   └── Type.h
├── src
│   ├── Ast.cpp ..... 抽象语法树
│   ├── lexer.l ..... 词法分析器
│   ├── parser.y ..... 语法分析器
│   ├── main.cpp
│   ├── SymbolTable.cpp ..... 符号表
│   └── Type.h ..... 类型系统
├── sysyruntimelibrary ..... SysY 运行时库
├── test ..... 测试用例
├── .gitignore
├── example.sy ..... SysY 语言样例程序
└── Makefile
```

3.2 类型系统

变量的类型，仿佛只是简单作为变量结点的一个属性而已，但仔细考虑会发现它可以极其复杂。直观上，我们有 `struct`、`union` 构造复合类型，函数本身作为变量，它也有其自身的特殊类型。类型系统是编程语言理论的一个重要一部分。很有趣的一点是，类型系统与数理逻辑紧密相关，类型的检查可以视为定理的证明，这一关系被称为Curry-Howard Correspondence。比如 `struct` 可以视为合取，`union` 可以视为析取，函数的输入类型与输出类型可以视为蕴含[1] 。为了进行静态类型检查，你需要根据你的目标语言设计好与你需要的类型系统有关的数据结构。你可能还要考虑如何插入“类型转换”。

在框架代码中，我们只实现了 int、void 和函数类型，如果你想要实现 const、数组等其他类型，你需要设计相关的数据结构。

3.3 符号表

符号表是编译器用于保存源程序符号信息的数据结构，这些信息在词法分析、语法分析阶段被存入符号表中，最终用于生成中间代码和目标代码。符号表条目可以包含标识符的词素、类型、作用域、行号等信息。

符号表主要用于作用域的管理，我们为每个语句块创建一个符号表，块中声明的每一个变量都在该符号表中对应着一个符号表条目。在词法分析阶段，我们只能识别出标识符，不能区分这个标识符是用于声明还是使用。而在语法分析阶段，我们能清楚的知道一个程序的语法结构，如果该标识符用

[1] 这一部分是私货。

于声明，那么语法分析器将创建相应的符号表条目，并将该条目存入当前作用域对应的符号表中，如果是使用该标识符，将从当前作用域对应的符号表开始沿着符号表链搜索符号表项。

框架代码中，我们定义了三种类型的符号表项：用于保存字面值常量属性值的符号表项、用于保存编译器生成的中间变量信息的符号表项以及保存源程序中标识符相关信息的符号表项。代码已经实现了符号表的插入函数，你需要在 SymbolTable.cpp 中实现符号表的查找函数。

3.4 抽象语法树

语法分析的目的是构建出一棵抽象语法树（AST），因此我们需要设计语法树的结点。结点分为许多类，除了一些共用属性外，不同类结点有着各自的属性、各自的子树结构、各自的函数实现。我们可以简单用 struct 去涵盖所有需要的内容，也可以设计复杂的继承结构。结点的类型大体上可以分为表达式和语句，每种类型又可以分为许多子类型，如表达式结点可以分为词法分析得到的叶结点、二元运算表达式的结点等；语句还可以分为 if 语句、while 语句和块语句等。

以框架代码为例：

```
class Node
{
private:
    static int counter;
    int seq;
public:
    Node();
    int getSeq() const {return seq;};
    virtual void output(int level) = 0;
};

class ExprNode : public Node
{
protected:
    SymbolEntry *symbolEntry;
public:
    ExprNode(SymbolEntry *symbolEntry) : symbolEntry(symbolEntry){};
};

class Id : public ExprNode
{
public:
    Id(SymbolEntry *se) : ExprNode(se){};
    void output(int level);
};
```

Node 为 AST 结点的抽象基类，ExprNode 为表达式结点的抽象基类，从 ExprNode 中派生出 Id。Node 类中声明了纯虚函数 output，用于输出语法树信息，派生出的具体子类均需要对其进行实现。

你需要根据在预备工作 2 中定义的 SysY 语言特性设计其他结点类型，如 while 语句、函数调用等。

3.5 语法分析与语法树的创建

词法分析得到的，实质是语法树的叶子结点的属性值，语法树所有结点均由语法分析器创建。在自底向上构建语法树时（与预测分析法相对），我们使用孩子结点构造父结点。在 yacc 每次确定一个产生式发生归约时，我们会创建出父结点、根据子结点正确设置父结点的属性、记录继承关系：

```
IfStmt
    : IF LPAREN Cond RPAREN Stmt %prec THEN {
        $$ = new IfStmt($3, $5);
    }
    | IF LPAREN Cond RPAREN Stmt ELSE Stmt {
        $$ = new IfElseStmt($3, $5, $7);
    }
    ;
```

你可能会对代码中的%prec THEN 感到疑惑，这是为了解决悬空-else 文法的二义性问题，考虑下面的 if 语句文法：

$stmt \rightarrow$ **if** $expr$ **then** $stmt$

$stmt \rightarrow$ **if** $expr$ **then** $stmt$ **else** $stmt$

在语法分析处于如下状态时:

if $expr$ **then if** $expr$ **then** $stmt \cdot$ **else** $stmt$

我们可以将终结符 **else** 移入，也可以使用产生式 $stmt \rightarrow$ **if** $expr$ **then** $stmt$ 进行归约，这时发生了移入/归约冲突，而正确的做法是将 **else** 移入。

在 yacc 中我们可以给终结符声明优先级：

%precedence **then**

%precedence **else**

这样终结符 **else** 的优先级高于终结符 **then**。产生式的优先级和右部最后一个终结符的优先级相同，即产生式 $stmt \rightarrow$ **if** $expr$ **then** $stmt$ 的优先级和终结符 **then** 的优先级相同。在发生移入/归约冲突时，通过比较向前看符号和产生式的优先级来解决冲突，若向前看符号的优先级更高，则进行移入，若产生式的优先级更高，则进行归约。这里 **else** 的优先级更高，因此会将 **else** 移入。

SysY 语言中的 if 语句并没有终结符 **then**，在 yacc 中我们可以使用%prec 关键字，将终结符 **then** 的优先级赋给产生式。

3.6 实验效果

以下面的 SysY 语言源程序为例：

```
int a;

int main()
{
    int a;
    a = 1 + 2;
    if(a < 5)
        return 1;
    return 0;
}
```

如果你已经正确实现了符号表的查找函数，通过 make run 命令，会输出如下的语法树：

```
program
    Sequence
        DeclStmt
            Id    name: a    scope: 0    type: int
        FunctionDefine function name: main, type: int()
            CompoundStmt
                Sequence
                    Sequence
                        Sequence
                            DeclStmt
                                Id    name: a    scope: 2    type: int
                            AssignStmt
                                Id    name: a    scope: 2    type: int
                                BinaryExpr    op: add
                                    IntegerLiteral    value: 1    type: int
                                    IntegerLiteral    value: 2    type: int
                        IfStmt
                            BinaryExpr    op: less
                                Id    name: a    scope: 2    type: int
                                IntegerLiteral    value: 5    type: int
                            ReturnStmt
                                IntegerLiteral    value: 1    type: int
                    ReturnStmt
                        IntegerLiteral    value: 0    type: int
```

3.7 Makefile 使用

框架代码 makefile 的使用如下：

- 编译：

  ```
  make
  ```

  编译出我们的编译器。

- 运行：

  ```
  make run
  ```

  以 example.sy 文件为输入，输出相应的语法树到 example.ast 文件中。

- 调试：

  ```
  make gdb
  ```

  使用 gdb 调试我们的编译器。

- 测试：

  ```
  make testlab5
  ```

  该命令会默认搜索 test 目录下所有的.sy 文件，逐个输入到编译器中，生成相应的抽象语法树.ast 文件到 test 目录中。你还可以指定测试目录：

  ```
  make testlab5 TEST_PATH=dirpath
  ```

- 清理：

  ```
  make clean
  ```

  清理所有可执行文件和测试输出。

4 作业的分级要求，测试样例及大作业评分标准

本次作业与最终作业紧密相关，不同级别选择造成的实现难度的区分会在这三次作业都有体现。**但注意，在本次作业和下次作业中，只要实现最基本要求的全部功能就可以得到满分**。但若你的最终大作业要支持更多的功能，也必然需要补充这两次作业。这一部分你可以在未来逐渐补充。

评价最终大作业功能的完成程度依照 OJ 上的评测结果给分。每个样例对应的分数比例将在本指导书的附录部分给出。最终拿到功能部分的所有分数，需要通过的所有功能样例[2]有 151 个，根据样例在仓库中的已有命名，下面简称为 000-117（共 118 个），1066-1091（部分中间编号缺失，共 22 个），浮点部分包括 196-202（197 缺失，共 6 个）和 35-39（共 5 个）。

本次作业（实现语法分析器）满分 3 分，下次作业（实现类型检查及中间代码生成）满分 6 分，大作业（实现中间代码优化及完成编译器构造）满分 12 分。

下面简单阐述你可能要完成的工作。

[2]浮点部分样例可以在这里查看

4.1 级别一（基本要求）及测试样例

要求：

1. 数据类型：`int`，函数返回值类型：`int`、`void`。
2. 变量、常量的声明和初始化。
3. 语句：赋值（=）、表达式语句、语句块、`if`、`while`、`return`。
4. 表达式：算术运算（+、-、*、/、%，其中 +、-都可以是单目运算符）、关系运算（==，>，<，>=，<=，!=）和逻辑运算（&&、||、!）。
5. 注释（行注释、块注释）。
6. 输入输出（实现连接 SysY 运行时库，参见文档《SysY 运行时库》）。
7. 叶函数。

下次类型检查作业中，你需要支持变量未声明、重声明错误检查，类型错误检查（比如字符串不能参与某些运算、输入输出函数也有参数类型要求）、并在进行隐式类型转换时给出提示。你还需要检查常量的未初始化错误，对常量的重赋值错误。对于非基本类型，实现关于它们的类型检查。

对应要求样例：

本部分要求样例共 60 个，**平分 4 分，每个样例 0.067 分**。

000、001、003、008-010、014-027、043-051、054-057、095、099、1068、1070、1072、1073，共 39 个样例，这部分样例中，叶函数只有 main 函数。

012、013、028-032、035-036、039-041、069、100、102、1074-1076、1078、1080、1083，共 21 个样例，这部分样例中，除 main 函数外的函数均为叶函数，main 函数可能会调用这些叶函数。

4.2 级别二要求及测试样例

要求：

1. 区分常量和变量的作用域。
2. break 和 continue 语句。
3. 非叶函数。
4. 数组的声明及初始化。
5. 浮点数的声明、识别、运算等。
6. 其他特性 (如针对长代码实现文字池等) 和复杂样例。

对于语法分析，你将需要实非叶函数、数组等语法结构的支持。为了支持非基本数据类型，你需要修正你的类型系统。

类型检查作业中,你需要实现对函数调用作参数检查以及对 `break/continue` 语句作 `within loop` 检查。

对应要求样例：

本部分样例共 91 个，共计 4 分。

注意，本部分得分均要求完全完成基本要求才计算得分。

002、060、096、101，共 4 个样例。这部分样例中，需要完成区分常量和变量的作用域，且除作用域之外，仅包括基本要求的语言特性。037、038、052、053、090，共 5 个样例。这部分样例中，需要完成 break 和 continue 语句，并正确实现注释特性。033、034、082、087、088，共 5 个样例。这部分样例中，需要完成非叶函数特性。以上 14 个样例，**平分 1 分，每个样例 0.07 分**。

004-007、011、042、061、062、064、066、068、070、071、073-078、081、083、085、086、089、091-094、1066、1067、1069、1071、1077、1081、1082、1084、1086、1088、1090、1091，共 40 个样例，**平分 1.5 分，每个样例 0.0375 分**。这部分样例中，需要完成数组的相关工作，即数组的声明及初始化。

058、059、063、065、067、072、079、080、084、097、098、103-117，共 26 个样例，**平分 0.5 分，每个样例 0.02 分**。这部分样例中，或样例非常复杂，或样例需要你完成一些特殊的语言特性（如文字池）。

浮点相关样例，即 `196`、`198-202`、`35-39`，共 11 个样例，**平分 1 分，每个样例 0.09 分**。这部分样例中，需要完成浮点数的相关工作。其中也有部分包含数组、复杂函数等要求。

4.3 级别三要求

要求：

1. 实现寄存器分配算法，此项是必需项。
2. 实现 mem2reg 算法，如果要实现优化，此项是必需项。
3. 实现基于数据流分析的强度削弱、代码外提、公共子表达式删除、无用代码删除等，你可以从中选你喜欢的实现，但需要有一定的工作量。
4. 实现其他中间代码优化算法。

本部分无法通过功能测试样例评测，如果你通过了除浮点外的 140 个功能测试样例，你可以使用性能测试样例的运行时间作为评价你中间代码优化算法实现效果的评价标准。

本部分你需要和助教详细讲解你的代码思路。

对应评分标准：

实现寄存器分配算法，1 分。

实现 mem2reg 算法，1 分。

实现基于数据流分析的强度削弱、代码外提、公共子表达式删除、无用代码删除等，1 分。

在完成寄存器分配算法的前提下，实现其他中间代码优化算法，1 分。如果实现多种优化算法，**视性能测试样例的运行时间及完成情况加分，可在总分上叠加**。

5 附录：评测样例、测试特性及赋分对应关系

公开样例共 151 个，共分大样例组两个，样例组 1 和样例组 2。

样例组 1 为基本要求所需通过的所有样例，样例组 1 内分 2 个小样例组；样例组 2 内分 6 个小样例组。每个小样例组内部，各样例赋分相同。

样例组	测试特性	样例名	赋分
1.1	基本要求，叶函数只有 main	000、001、003、008-010、014-027、043-051、054-057、095、099、1068、1070、1072、1073	0.067
1.2	基本要求，main 只调用叶函数	012、013、028-032、035-036、039-041、069、100、102、1074-1076、1078、1080、1083	0.067
2.1	区分常量和变量的作用域，且除作用域之外，仅包括基本要求的语言特性	002、060、096、101	0.07
2.2	完成 break 和 continue 语句，并正确实现注释特性	037、038、052、053、090	0.07
2.3	非叶函数	033、034、082、087、088	0.07
2.4	数组的声明及初始化	004-007、011、042、061、062、064、066、068、070、071、073-078、081、083、085、086、089、091-094、1066、1067、1069、1071、1077、1081、1082、1084、1086、1088、1090、1091	0.0375
2.5	复杂样例，其他特殊语言特性	058、059、063、065、067、072、079、080、084、097、098、103-117	0.02
2.6	浮点相关样例	196、198-202、35-39	0.09