Report for Compiler Final Project

陈子豪[1]

Spring 2015

[1]5130309305 上海交通大学 2013 级 ACM 班

Contents

Chapter 1

导言

本文是关于作者 2014-2015 学年春季学期课程《编译原理》的课程设计。作者用 Java 实现了一个 C 语言的编译器，包括了 C 的大部分常用功能。接下来将分别介绍编译器实现过程中的几个阶段：语法分析，语义检查，生成中间表示，生成汇编代码。

Chapter 2

语法分析

语法分析主要分为两个部分，分别为 Lexer 和 Parser。Lexer 大致是将原来的 C 程序转换成一个 token 流，Parser 做的主要是讲 token 流转换为抽象语法树(Abstract Syntax Tree)。

除此之外，这里将会介绍作者做的一项额外工作，即实现代码美化器。可以将缩进、换行、空格格式混乱的 C 语言代码美化。

2.1 Lexer

由于 Lexer 和 Parser 的广泛用途，网上有开源的工具可以帮助人们更加方便地实现 Lexer 和 Parser，考虑到编译器初始阶段的程序的可靠性要求，我选择实用工具 Jflex 来实现 Lexer，通过给定对 token 的定义以及不同种类字符的处理，Jflex 可以帮助使用者较为轻松地实现 Lexer。

2.2 Parser

对于实现 Parser 的工具，和 Jflex 兼容性较好的是 JCup。使用者可以通过给定终结字符和语言所对应的上下文无关文法 (Context Free Grammar) 来做 parse 的工作，不符合给定上下文无关文法的程序会在这里被识别并报出错误信息。此外，使用者可以在每条上下无关分法处填写代码以满足自己需要，作者在这里添加代码以生成抽象语法树。(注：抽象语法树并未丢失能影响程序运行结果的信息)

下面举一个寻找两个数最大公约数函数的例子，源代码为：

```
        int gcd(int x, int y) {
            if (x%y == 0) return y;
            else return gcd(y, x%y);
        }
```

其抽象语法树的结构为

```
    <FunctionDef>
            <IntType>
            <Id: gcd>
                <VarDecl>
                    <IntType>
                    <Id: x>
                <VarDecl>
                    <IntType>
                    <Id: y>
                    <If>
                        <BinaryExpr>
                            <EQ>
                            <BinaryExpr>
                                <MOD>
                                <Id: x>
                                <Id: y>
                            <IntConst: 0>
                        <ReturnStmt>
                            <Id: y>
                        <ReturnStmt>
                            <FunctionCall>
                                <Id: gcd>
                                <ExprList>
                                    <Id: y>
                                    <BinaryExpr>
                                        <MOD>
                                        <Id: x>
                                        <Id: y>
```

2.3 代码美化器

代码美化器 (Pretty Printer) 的实现主要是利用 Lexer 给出的 Token 流，模拟整个程序的缩进过程。值得一提的是，这个代码美化器支持注释的原位置保留。效果如下。

美化前：

```
#include <stdio.h>
int a[10] = {1,2,3}, b[5] = {2,3,4};
char c = 'c';
char s[10] = "345asfdfa\n";
//asfsa
struct yy{int a, b;} qwe,ads;

int main()
{
char op;
    int a;
    int b = -c;
op=cin.get();
    for (i = 0;                          i<n;i++)
        while (1)
            asfddsfghjk();
if ((fabs(x1*y2-x2*y1)<16)&&(fabs(x2*y3-x3*y2)<16) ) { flag=1;}
    if () {
        if (){
            if (){
            }
            else ;
            haha();
        }
        haha();
    }
    for (;;)
        for (;;)
            for (;;)
                ;

    return 'a';
    }
```

美化后：

```
#include <stdio.h>
int a[10] = {1, 2, 3}, b[5] = {2, 3, 4};
char c = 'c';
char s[10] = "345asfdfa\n";
//asfsa
struct yy{
    int a, b;
}qwe, ads;
int main(){
    char op;
    int a;
    int b = -c;
    op = cin.get();
    for (i = 0; i < n; i++)
        while (1)
            asfddsfghjk();
    if ((fabs(x1*y2 - x2*y1) < 16) && (fabs(x2*y3 - x3*y2) < 16)){
        flag = 1;
    }
    if (){
        if (){
            if (){
            }
            else
                ;
            haha();
        }
        haha();
    }
    for (; ; )
        for (; ; )
            for (; ; )
                ;
    return 'a';
}
```

如需测试只需运行 PrettyPrinter 包下的 PrettyPrinter 类，可以选择修改所读入文件。

Chapter 3

语义检查

C 语言并不是一个上下文无关语言 (Context Free Language)，不合法的程序也可以通过 Parser，而语义检查的目的就是，处理掉所有的编译错误，并且向抽象语法树节点加入一些信息来方便以后的工作。具体来讲，语义检查主要做了如下事情：

- 类型检查。判断不同变量、语句返回值的类型是否符合要求。
- 未声明后的使用。检查是否有变量使用前未经声明
- 变量重名。检查是否存在在同一个命名空间中变量名重复的情况
- 数组访问检查。检查数组下表访问层数是否正确。
- 参数不匹配。检查是否存在函数调用与定义参数不匹配的情况。
- break, continue 语句位置检查。检查 break, continue 语句是否出现在循环中。

如果源程序存在编译错误，此编译器会跑出相应异常以及错误信息。除此之外，语义检查后，作者将每条表达式的返回类型、空间大小、是否是左值、是否是常量等信息加入了抽象语法树。这将会对后面部分的实现有所帮助。

Chapter 4

生成中间表示

中间表示（Intermediate Representation）是编译器后端的第一个部分，目的是更方便的生成汇编代码和优化。作者基于助教给出的中间表示的架构进行一定改动，例如将结构体和数组全部都看做是指向某一块内存区域的指针。事实上，虽然在某些地方这种处理方式会方便，但是在动态申请内存以及涉及到复杂类型结构体时，这种处理方式就会比较麻烦。

作者采用的中间表示由若干个函数组成，每个函数有参数列表，局部、临时变量列表，函数主体四元组列表组成。这么做的原因是考虑到方便后面运行时环境的处理。函数主体的四元组分为若干类型，比如代数运算，函数参数，函数调用等，具体如下：

- AddressOf，例: a = &b;
- ArithmeticExpr 例：c = a + b ;
- ArrayRead 例: c = a[5];
- ArrayWrite 例：a[5] = c;
- Assgin 例：a = b;
- Label 例: Label 2:
- MemoryRead 例：a = *b;
- MemoryWrite 例: *a = b;
- Goto

- IFEZGoto
- IFNEZGoto
- Param
- Call 例: tmp = f(a)
- Return 例：return 0;

而对于控制流 (Control Flow) 控制，此编译器采用的方法是，给每个表达式两个标记，分别对应真与假，会根据表达式的运算符与内层表达式的标记来确定外层的表达式的标记，例如 $B \rightarrow !(B_1)$, 则有 $B_1.true = B.false, B_1.false = B.true$。

前文提到的最大公约数函数的中间表示为

```
1     _gcd:
2         Temp1 = x MOD y
3         Temp2 = Temp1 EQ 0
4         If Not Equal to Zero Temp2 Goto Label 2
5         Goto Label 3
6         Label 2:    Return BasicParam y
7         Goto Label 1
8         Label 3:    Temp3 = x MOD y
9         BasicParam y
10        BasicParam Temp3
11        Temp4 = call _gcd , 2
12        Return BasicParam Temp4
13        Label 1:
```

Chapter 5

生成汇编代码

生成汇编代码是此编译器的最后一个部分，生成汇编代码需要将原先中间代码中的各种元素对应成 MIPS 指令。对于算术运算、取址、数组访问等操作都有相应的直接对应的指令，难点在于对函数调用的翻译。而注意到函数调用的信息可以完全被栈刻画，所以应该用栈的思想来处理函数调用，这种栈结构被称为运行时环境 (Runntime Environment)，可以通过移动栈指针并读写来进行进栈与退栈的操作。

每个函数在运行时环境中的记录被称为活动记录 (Activation Record)，在此编译器中，每个活动记录保存着参数、返回地址、局部数据、临时数据，栈指针指在返回复制开始的地方。这种活动记录的布局对定长参数函数调用都比较方便，但是对于不定长参数函数 (在这里只有 printf 函数) 需要在静态区存下每次调用该函数的参数个数。

对与之前的求最大公约数的函数，下面是其对应的 MIPS 代码。

```
1  _gcd:
2      sw $ra, ($sp)
3      lw $t2, 4($sp)
4      lw $t1, 8($sp)
5      rem $t0, $t1, $t2
6      sw $t0, -4($sp)
7      li $t2, 0
8      lw $t1, -4($sp)
9      seq $t0, $t1, $t2
10     sw $t0, -8($sp)
11     lw $t0, -8($sp)
12     bne $t0, $0, L2
13     b L3
14 L2:
```

```
15      lw $v0, 4($sp)
16      sw $v0, 4($sp)
17      jr $ra
18      b L1
19  L3:
20      lw $t2, 4($sp)
21      lw $t1, 8($sp)
22      rem $t0, $t1, $t2
23      sw $t0, -12($sp)
24      lw $t0, 4($sp)
25      sw $t0, -20($sp)
26      lw $t0, -12($sp)
27      sw $t0, -24($sp)
28      subu $sp, $sp, 28
29      jal _gcd
30      addi $sp, $sp, 28
31      sw $v0, -16($sp)
32      lw $ra, ($sp)
33      lw $v0, -16($sp)
34      sw $v0, -16($sp)
35      jr $ra
36  L1:
37      jr $ra
```

Chapter 6

结语

这个编译器是笔者目前做过的最大的 project，笔者花了大量时间在这上面，在做这个 project 的过程中笔者也在努力不断调整，从一开始的茫然低效到后来的渐通门路。这个过程中代码能力得到提高，对如何完成一个 project 也更加有经验了。

但是很遗憾的是最终有个别几个数据无法通过，原因在于之前在中间表示时开始对结构体的处理方式不是很好（新建一个指针指向原结构体），最后评测前也没有时间进行大改。以后再做 project 的时候还是应适当了解接下来的阶段做的事情，不要在早期留下后面难以处理的东西。

最后，感谢各位助教的耐心指导，感谢倪昊斌同学在整个过程中为笔者解答大量问题。

Bibliography

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman Compilers: Principles, Techniques and Tools (Second Edition). Pearson Education, 2006.

[2] Andrew W. Appel, Jens Palsberg. Modern Compiler Implementation in Java (Second Edition). Cambridge University Press, 2002

[3] Scott Hudson. JCup User’s Manual. 2014

[4] Gerwin Klein, Steve Rowe, and Regis Decamps. JFlex User’s Manual. 2015