编译原理 Lab3 实验报告

171860526，褚有刚ajeo0526@gmail.com　　171180554, 夏宇

2019 年 11 月 29 日

1　实验目的

在词法分析和语法分析的基础上编写一个程序，对 C–源代码进行语义分析和类型检查，并打印分析结果。

2　实验流程

2.1　翻译

2.1.1　翻译模式

和语义子程序类似，翻译模式采用遍历语法分析树的方式进行，由于语法分析树中的很多节点都是为了描述语言的结构而引入的，如果为每个文法符号都设计一个子过程执行翻译的动作代码量会很多，因此在遍历语法树的过程中，我们同样只需要关注一些**关键节点**[1]，为这些节点编写相应的翻译子过程即可；

2.1.2　跳转语句的翻译

由于我们的编译器是在语法分析树已经生成的前提下进行中间代码生成并且跳转目标采用语句标号的形式，因此我们只需要将语句标号作为**继承属性**，在翻译会生成跳转的语法结构时传入并且根据代码想要表达的语义在合适的位置上放上语句标号即可。

2.1.3　数组和结构体的翻译

对于数组和结构体，在定义这类变量的时候，使用用户定义的符号记录该变量对应的内存的首地址，而不是数据本身；在使用这类变量的时候，需要区别赋值语句和引用语句，在对该类变量进行赋值的时候，需要首先计算要写入的地址，然后使用左解引用[2]进行赋值；在引用这类变量的时候，同样需要首先计算要读取的地址，然后使用右解引用[3]获取具体要访问的数据；

ps: 我们的编译器支持任何维度的数组以及结构体，但是为了满足试验的要求，对于数组需要在编译选项中增加`__ARR__`宏的定义。

[1] 需要生成中间代码的节点

[2] *a = b 的形式

[3] a = *b 的形式

2.1.4 布尔表达式的翻译

布尔表达式在代码中有两种作用，作为改变控制流的翻译模式在课本中已经有了比较详细的介绍，对于作为计算逻辑值的情况，通过一点变通也可以采用改变控制流的翻译模式完成翻译，例如，对于`a = B;` 的代码[4]，采用如下的翻译方法：

```
t = 0;
if (B) t = 1;
a = t;
```

就可以完成布尔表达式的翻译；

2.2 优化

我们采取的优化策略都是基本块内部的优化，优化的方法是在重复遍历中间代码的过程中通过替换一些四元式中的符号发现一些无用的赋值表达式将其删除来达到优化的效果。

2.2.1 获取基本块

由于基本块内部代码要么全部执行，要么全不执行，因此在基本块内部进行优化能够保证优化后的代码不出现逻辑上的错误，在获取基本块的时候，需要关注一些会改变控制流以及进入一个新的过程的四元式，比如 **FUNCTION** 表示进入一个新的过程，此时之前的信息都变得无效；**LABEL** 表示其他地方的代码可能会跳转到这儿，因此之前的信息同样变得无效，etc；

2.2.2 赋值替换

对于赋值表达式, 获取其右部最近被修改的赋值表达式，如果该表达式右部没有被修改过，则可以直接使用其替换当前赋值表达式的右部，对于左部，其之前的值被“杀死”，因此需要更新信息（记录符号及其赋值信息的数据结构采用的是哈希表，通过再次插入就可以覆盖之前的值，效率十分高效），除此之外，对于一些简单的算术表达式也可以进行合并和预计算；

2.2.3 常量求值

对于加减乘除表达式右部的运算分量全部都是常数的情况，可以直接计算出表达式右部的值，将该算术表达式改为赋值表达式，除此之外，对于例如 **a = b * 1**；可以直接优化为 **a = b**; 对于 **a = b * 0**; 可以直接优化为 **a = 0**, etc；

2.2.4 标签删除

在翻译过程中可能会生成很多冗余的标签，比如多级嵌套的 **if-else** 结构，此时可以将连续出现的多个标签替换为一个标签，并且修改跳转到这些标签的 goto 语句的跳转目标为修改之后的标签；

[4]B 为布尔表达式

2.2.5 条件表达式

为了方便代码的阅读，在条件表达式中我们通常会加入很多的括号来显式表达条件表达式的求值次序，但是如果直接将这些括号当作一个表达式来翻译就会产生很多的冗余代码，对于这种情况可以直接将括号中的内容当作一个条件表达式进行翻译就会大大减少翻译之后的代码；此外，使用`LABEL_FALL`特殊的标签表示该条件表达式的下一句代码的就是跳转的目标，来减少冗余的 goto 语句。

2.2.6 赋值表达式

在执行了上述优化动作之后，会出现很多符号自从赋值之后再也没有被使用过的冗余四元式，对于这些四元式直接删除即可。

3 遇到的问题和解决方案

数组和结构体的翻译过程中，对于 **a = *b** 形式，如果将之后 **c = a** 的表达式直接优化为 **c = *b** 的形式就可能会出现问题，因为在这中间虽然 b 没有被修改过，但是 b 指向的内存中的数据可能被修改过，因此会出现问题；

和真实的机器不同，实验中所用的虚拟机在计算整数除法的时候采用的向下取整的策略，因此在进行优化的过程中可能出现一点问题，例如 **a = -3 / 2**，常量优化的时候如果直接求值将会得到 **a = -1**，但不进行常量优化直接在虚拟机上运行结果则是 **a = -2**，因此需要在进行常量优化的时候手动实现向下取整的代码。