编译原理

MiniDecaf 编译器实验报告 -- STAGE 2

2021010706 岳章乔

一、 思考题

step 5:

1.

```
addi    sp, sp, -16
```

2.

需要对当前符号表进行检查：修改 `scope/scope.cpp` 的 `lookup` 函数。对于新的重名的变量，按变量的位置获取其最靠近的实际引用的变量。当发现重名的时候，建立函数原有名字和新的加了后缀的名字之间的映射。在之后读取的时候，按这样的映射在 `scope stack` 读取实际的变量。

step 6:

1. 框架以 `C++` 实现，词法分析、语法分析分别由 `Flex/Bison` 完成，而 `Bison` 语法分析器处理悬挂 `else` 时触发 `shift/reduce` 冲突，而 `bison` 默认选择 `shift` 展开，而不是 `reduce` 展开，因此 `else` 匹配最近的 `if` 。
2. 当前之所以时短路求值，是因为在中间代码生成阶段，三目判断语句

   ```
   T a = CONDITION ? TRUE_EXPR : FALSE_EXPR;
   ```

   被转化为其等价的 `if` 语句 实现：

   ```
   T ret = 0;
   if(CONDITION)
        ret = TRUE_EXPR ;
   else
       ret = FALSE_EXPR ;
   DUMP(ret);
   ```

   那么如果是不短路求值，就应该修改其转化的等价形式，具体如下：

   ```
   T ret = FALSE_EXPR;
   T true_val = TRUE_EXPR;
   if(CONDITION)
        ret = true_val;
   DUMP(ret);
   ```

二、实验内容

2.0. 约定

parser.y

非终结符 `VarDecl` 代表声明语句， `Lvalue` 代表左值，`LvalueExpr` 代表左值表达式。

2.1. 实验需求

实现声明、赋值语句；实现判断语句和三目判断表达式。

2.2. 需求分析

< 与 stage 1 对应部分相同 >

2.3. 具体实现

2.3.0. 同 stage 1.

2.3.1. 词法分析

编译函数第 11 行 `parseFile` 调用语法分析器，语法分析器调用词法分析器。

（**更改1**）由于在 `frontend/parser.y` 已经定义了终结符，因此只要按定义补全 `frontend/scanner.l` 的单词表即可，本次更新针对声明、赋值运算，以及三目判断语句，分别增加**等号**（赋值运算符）；**问号**和**冒号**。

2.3.2. 语法分析

修改 `frontend/parser.y` 产生式的定义。

step 5:

要求实现变量处理，包括**变量声明**、**变量取值**、**变量赋值**三种情况；

变量声明：

按 C 语言的语法规范，声明语句不是语句，因此不能存在语句到声明语句 的推导。

（**更改2**）如果对声明语句(`VarDecl`)进行展开，必须在语句块部分进行，具体是修改 `StmtList` 的推导规则：

```
StmtList    : /* skip */
+           | StmtList VarDecl
+               { $1->append($2);
+                   $$ = $1; }
            ;
```

（**更改3**）然后在下方对 `VarDecl` 分别定义由赋值和没有赋值的推导。

（**更改4**）变量取值： 在非终结符 `Expr` 的部分新增推导左值表达式的关于推导规则的定义。

（**更改5**）变量赋值：在非终结符 `Expr` 的部分新增推导赋值表达式的关于推导规则的定义： `Lvalue ASSIGN Expr`

step 6 要求实现**三目判断表达式**；

（**更改6**）在非终结符 `Expr` 的部分新增推导三目判断表达式的关于推导规则的定义： `Expr QUESTION Expr COLON Expr`

后面调用的构造函数对应 `ast/ast.hpp` 上对各个运算符的定义。

2.3.3. 语义分析

（**更改7**）编译器第一次遍历语法树，构造符号表，需要补全对变量声明的处理，具体参考函数声明（`FuncDefn`）的实现。

编译器第二次遍历语法树，进行类型检查，在 `translation/type_check.cpp` 新增遍历三目判断求值的遍历函数。由于另外的关于变量读取的遍历函数已经实现，因此不用处理。 三目判断求值遍历函数的实现参考 `if` 判断语句以及其他运算符的实现。

2.3.4. 中间代码生成

编译器第三次遍历 AST，生成中间三地址代码：

需要修改 `translation/translation.hpp` 和 `translation/translation.cpp` ，把节点翻译为三地址码。

（**更改8-10**）需要实现声明语句、赋值语句、左值语句和三目判断表达式的翻译。

一个变量通常以 `ATTR(sym)` 的形式出现。

这次值得注意的是获取临时变量的方式：

对于新的变量，可以调用 `getNewTempI4()` 获得新的临时变量，接着调用 `attachTemp()` 把刚生成的临时变量接入到当前作用域的所属符号上。

对于原有的变量，例如在赋值表达式被修改的变量和左值表达式的变量，可以通过 `getTemp()` 读取临时变量。

（**更改11**）三目判断表达式的翻译，等价于判断语句和赋值语句的综合，具体参考思考题 6.2 。

2.3.5. 中间代码优化

略。

2.3.6. 目标代码生成

由于在中间代码生成阶段，已经把 minidecaf 语言的声明、左值、三目判断等抽象的运算以更基本的形式实现，于是现阶段只需处理赋值语句。

赋值语句在汇编语言层面，就是寄存器读写的 `mov` 操作（8086），于是只要在翻译三地址码阶段把赋值语句转化为 `mov` 语句即可。

由于 `mov` 是上一阶段（stage-1）能独立存在的条件之一，因此不需要修改最终的汇编代码生成部分。

综上所述，这个阶段需要修改 `asm/riscv_md.cpp` ，在 `RiscvDesc::emitTac` 函数新增赋值的枚举，具体如下：

（**更改12**）

```
case Tac::ASSIGN:
    emitUnaryTac(RiscvInstr::MOVE, t);
    break;
```