

# hw7实验过程记录

---

## 参考资料

---

hw5\_6的代码。

## 实验过程

---

### 策略

总的思路：一边做类型检查，一边翻译成IR。

- 类型检查写在 `semant.h/c: transA_xxx`
- 翻译部分写在 `translate.h/c: Tr_xxx`
- 在每个节点的类型检查完成后，`transA_xxx`会调用`Tr_xxx`函数，返回 (return语句) 得到的`Tr_exp`

### 数据结构

1. `T_xxx` (translate模块): IR所用的数据类型
  - 所有的函数和类方法都翻译成`T_funcDecl`
  - 所有的`Stm`翻译成`T_stm`，所有的`Exp`翻译成`T_exp`
2. `Tr_exp`: 将所有的`T_stm`和`T_exp`都抽象成`Tr_exp`
  - 3种类型: `Ex` (需要返回值的`exp`)，`Nx` (`stm`)，`Cx` (特别处理`Cjump`，包含真假`label`对应的`patchList`)
  - 对应的包装函数: `Tr_Ex`，`Tr_Nx`，`Tr_Cx`
  - 对应的解包函数: `unEx`，`unNx`，`unCx`
  - 对于条件语句以及比较语句的特殊处理：
    - `unEx`: 若是`Tr_cx`，填充`patchList`，并包装成`escExp`，通过`Cjump`实现根据真假`label`返回1或0
    - `unCx`: 若是`Tr_ex`，返回一个含有待填充真假`patchList`的`Cjump`语句，`condition`为`unEx(exp) != 0`
3. `expty` (semant模块): 表达式返回类型，包含以下3个成员
  - `exp`: `Tr_exp`类型
  - `value`: `Ty_ty`类型
  - `location`: `Ty_ty`类型
4. `loopstack` (semant模块): `while` loop的标签栈，存储当前`while`语句的`whiletest`、`whileend`两个标签

```

1  typedef struct loopstack_ *loopstack;
2  struct loopstack_ {
3      Temp_label whiletest;
4      Temp_label whileend;
5      loopstack next;
6  };

```

## 翻译过程

### 整体

依次处理主方法中的VarDeclList部分和StmList部分。

### VarDecl

只需要在语义检查的基础上增加：

1. 申请一个Temp\_temp，并把它记录在venv的对应表项中
2. 在vd->elist不为空时，返回相应的赋值语句

int/float类型处理：由于在VarDecl中，右边的exp只可能是Const，所以直接根据变量类型返回intConst/floatConst，不需要cast语句

### Stm

值得注意的是条件语句和循环体的判断。

#### 1. if

- 流程：将条件表达式通过unCx解包成Cx，然后利用doPatch对Cx的真假patchList进行填充，最后利用Tr\_Nx重新包装成Tr\_exp
- 共有3个label (一般)：t, f, e，分别标识if body, else body和if-else之后的语句
- 可以分为以下4种情况：
  1. if body和else body都为空：相当于先计算exp，无论真假都跳到end label

```

1      T_Cjump(T_ne, unEx(test_exp), 0, e, e)
2  e:  ...

```

其中doPatch的工作是将cx->trues和cx->falses都填充为end label

2. if body为空，但else body不为空：计算exp后，为真直接跳到end label，为假继续执行else body (即跳到false label)

```

1      T_Cjump(T_ne, unEx(test_exp), 0, e, f)
2  f:
3      unNx(elsee)
4  e:  ...

```

其中doPatch的工作是将cx->trues填充为end label，将cx->falses填充为false label

3. if body不为空, else body为空: 计算exp后, 为真继续执行if body (即跳到true label), 为假直接跳到end label

```
1      T_Cjump(T_ne, unEx(test_exp), 0, t, e)
2  t:
3      unNx(then)
4  e:  ...
```

其中doPatch的工作是将cx->trues填充为true label, 将cx->falses填充为end label

4. if body和else body都不为空: 计算exp后, 为真继续执行if body且完成后直接跳出, 为假继续执行else body

```
1      T_Cjump(T_ne, unEx(test_exp), 0, t, f)
2  t:
3      unNx(then)
4      T_Jump(e)
5  f:
6      unNx(elsee)
7  e:  ...
```

其中doPatch的工作是将cx->trues填充为true label, 将cx->falses填充为false label

## 2. while

- 流程: 和if处理过程完全类似
- 共有3个label (一般): whilettest, whileloop, whileend, 分别标识condition exp, while body和while之后的语句
- 可以分为以下2种情况:

1. while body为空: 计算exp后, 为真继续执行exp (跳到whilettest), 为假跳出while

```
1  whilettest:
2      T_Cjump(T_ne, unEx(test_exp), 0, whilettest, whileend)
3  whileend:
4      ...
```

其中doPatch的工作是将cx->trues填充为whilettest, 将cx->falses填充为whileend

2. while body不为空: 计算exp后, 为真继续执行while body且完成后跳回whilettest继续判断, 为假跳出while

```
1  whilettest:
2      T_Cjump(T_ne, unEx(test_exp), 0, whileloop, whileend)
3  whileloop:
4      unNx(loop)
5      T_Jump(whilettest)
6  whileend:
7      ...
```

其中doPatch的工作是将cx->trues填充为whileloop, 将cx->falses填充为whileend

### 3. continue / break

- 语义检查：利用标签栈是否为空判断是否在循环体内
- 翻译成IR：获取当前栈顶的标签
  - continue：跳转到whiletest
  - break：跳转到whileend

## Exp

值得注意的是运算表达式和numConst的处理。

1. 算术运算：直接翻译成T\_Binop即可，返回Tr\_Ex
  - 对于minusExp (-x)，处理成0-x即可
2. 比较运算：直接翻译成T\_Cjump语句，创建好真假label的patchList待填充，返回Tr\_Cx
3. 逻辑运算：返回Tr\_Cx
  1. and：left && right应该翻译成形如下面的语句

```
1      T_Cjump(T_ne, unEx(left), 0, rightlabel, f)
2  rightlabel:
3      T_Cjump(T_ne, unEx(right), 0, t, f)
4  t:    ...
5  f:    ...
```

其中应该将left的true label填充为rightlabel (调用doPatch)，将left和right的false patchList连接起来待填充 (调用joinPatch)

2. or：left || right应该翻译成形如下面的语句

```
1      T_Cjump(T_ne, unEx(left), 0, t, rightlabel)
2  rightlabel:
3      T_Cjump(T_ne, unEx(right), 0, t, f)
4  t:    ...
5  f:    ...
```

其中应该将left的false label填充为rightlabel (调用doPatch)，将left和right的true patchList连接起来待填充 (调用joinPatch)

3. !：!exp应该翻译成形如下面的语句

```
1      T_Cjump(T_ne, unEx(exp), 0, f, t)
2  t:    ...
3  f:    ...
```

即将exp利用unCx解包成Cx，然后其真假label的patchList，返回Tr\_Cx

### 4. numConst

- 可以通过传入类型进行优化：若传入类型非空，则直接翻译成对应类型；否则再通过 `num == (int)num` 判断num是否是整数

# 测试结果

运行 `make test`，结果如下 (无报错):

```
● zqwh@LAPTOP-HDCBVNK7:~/compiler/2024/hw7$ make test
[hw7test00]
[hw7test01]
[hw7test02]
[hw7test03]
[hw7test04]
[hw7test05]
[hw7test06]
[hw7test07]
[hw7test08]
```

人工确认输出的IR应该正确，输出文件太长不贴到报告里。

# 开发过程

git提交记录如下:

|                                     |                   |         |
|-------------------------------------|-------------------|---------|
| hw7: fix numconst cast              | 22 Apr 2024 21:04 | Jopqior |
| hw7: update tools                   | 21 Apr 2024 20:34 | Jopqior |
| hw7: fix Tr_Cast and A_ne           | 21 Apr 2024 20:33 | Jopqior |
| hw7: add Tr_Cast and fix putch cast | 20 Apr 2024 22:32 | Jopqior |
| hw7: merge master                   | 20 Apr 2024 14:04 | Jopqior |
| hw7: refactor codes                 | 18 Apr 2024 22:12 | Jopqior |
| hw7: merge master                   | 18 Apr 2024 09:07 | Jopqior |
| hw7: update ast2irp                 | 18 Apr 2024 08:35 | Jopqior |
| hw7: remove duplicate code          | 17 Apr 2024 16:51 | Jopqior |
| hw7: merge master                   | 15 Apr 2024 17:07 | Jopqior |
| hw7: modify Makefile and .gitignore | 15 Apr 2024 09:15 | Jopqior |
| hw7: complete exp                   | 15 Apr 2024 09:14 | Jopqior |
| hw7: complete statement             | 14 Apr 2024 14:10 | Jopqior |
| hw7: remove hw6                     | 13 Apr 2024 22:25 | Jopqior |
| hw7: add if, while and assignment   | 13 Apr 2024 00:17 | Jopqior |
| hw7: chmod for tools                | 13 Apr 2024 00:12 | Jopqior |
| hw7: merge hw6                      | 12 Apr 2024 21:53 | Jopqior |
| hw7: add general                    | 12 Apr 2024 20:53 | Jopqior |
| hw7: initial commit                 | 12 Apr 2024 17:25 | Jopqior |