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

```
typedef struct loopstack_ *loopstack;
struct loopstack_ {
  Temp_label whiletest;
  Temp_label whileend;
  loopstack next;
};
```

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

        ```
            T_Cjump(T_ne, unEx(test_exp), 0, e, e)
        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)

        ```
            T_Cjump(T_ne, unEx(test_exp), 0, e, f)
        f:
            unNx(elsee)
        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

```
    T_Cjump(T_ne, unEx(test_exp), 0, t, e)
t:
    unNx(then)
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

```
    T_Cjump(T_ne, unEx(test_exp), 0, t, f)
t:
    unNx(then)
    T_Jump(e)
f:
    unNx(elsee)
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)：whiletest, 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whiletest)，为假跳出while

```
whiletest:
    T_Cjump(T_ne, unEx(test_exp), 0, whiletest, whileend)
whileend:
    ...
```

其中doPatch的工作是将cx->trues填充为whiletest，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whiletest继续判断，为假跳出while

```
whiletest:
    T_Cjump(T_ne, unEx(test_exp), 0, whileloop, whileend)
whileloop:
    unNx(loop)
    T_Jump(whiletest)
whileend:
    ...
```

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

    ```
        T_Cjump(T_ne, unEx(left), 0, rightlabel, f)
    rightlabel:
        T_Cjump(T_ne, unEx(right), 0, t, f)
    t:  ...
    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

    ```
        T_Cjump(T_ne, unEx(left), 0, t, rightlabel)
    rightlabel:
        T_Cjump(T_ne, unEx(right), 0, t, f)
    t:  ...
    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)

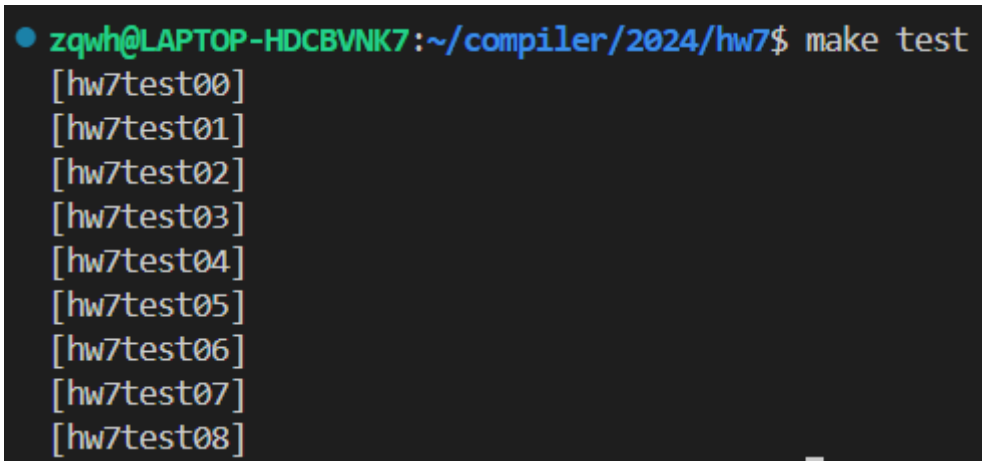
    3. !：!exp应该翻译成形如下面的语句

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



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: