

hw8实验过程记录

参考资料

参考hw6的代码和hw7/docs/lab7_8.md文档。

实验过程

Array

关于Array的翻译过程一共有以下几个操作：

1. 定义/初始化： `int[] a = {ConstList};` 以及 `a[] = {ExpList};`

- 处理 `ConstList` / `ExpList` 的流程类似，最终返回 `Tr_expList`

对应 `semant` 模块中的 `transA_ExpList_NumConst` 和 `transA_ExpList_Num` 函数。

- 只在int/float转换时有区别：`ConstList` 可以直接在翻译时转成对应类型，无需加 `Cast`

- 翻译流程

对应 `translate` 模块中的 `Tr_ArrayInit` 函数。

1. 遍历 `initExpList`，获取数组长度 `len`

2. emit IRP:

- 获取新的Temp_temp `newArr`
- 调用malloc函数分配 `(len+1) * SEM_ARCH_SIZE` bytes的空间并Move到 `newArr`
- 在-1的位置存入长度 `len`

3. 再次遍历 `initExpList`，emit IRP:

- 计算每个元素的偏移量，然后取Mem，并将初始值Move进去
- 此处的小优化是若偏移量为0则无需将其翻译成 `T_Binop` 相加，而是直接取Mem

2. 分配空间： `a = new int[exp];`

- 翻译：类似上面，唯一不同的是需要计算malloc空间的大小

```
1 T_Binop(T_mul, T_Binop(T_plus, T_Temp(sizeTemp), T_IntConst(1)),  
  T_IntConst(SEM_ARCH_SIZE))
```

- 优化：当 `size` 表达式为常数时，可以直接计算出malloc大小，而不用翻译成 `T_Binop` 的形式

3. 存取元素： `exp[exp]`

- 翻译：通过 `pos` 表达式计算偏移量，再取Mem

- 优化：

1. 若 `pos` 为常量0，无需计算偏移量，直接返回取Mem的结果

2. 若 `pos` 为非0常量, 可以直接计算偏移量 (IntConst), 不用翻译成T_Binop的形式

4. 获取长度: `length(exp)`

- 直接取-1位置即可

Class

预处理

1. 类环境表cenv: 和hw6一样

2. 偏移表 (Unified Object Record): 将所有类的变量和方法合并起来

- 初始化: 初始化两张表 (S_table), 分别记录类变量的偏移varoff和类方法的偏移methoff, 全局变量globaloff=0
- 第一遍预处理 (transA_xxx_basic函数) 建立偏移表
 - 每遇到一个类变量, 若还没有varid的记录, 在表varoff中记录varid->globaloff的映射, 然后globaloff递增
 - 每遇到一个类方法, 若还没有methid的记录, 在表methoff中记录methid->globaloff的映射, 然后globaloff递增
- 完成: 最终globaloff记录了所有类的变量和方法数量 (即Unified Object的大小), 表varoff和methoff记录相应类变量和方法的偏移量
- 代码: 由于off为int, 与void *不符, 将S_table再包装一层

```
1 static void offtable_enter(S_table t, S_symbol key, int off) {
2     int *p = checked_malloc(sizeof(int));
3     *p = off;
4     S_enter(t, key, p);
5 }
6
7 static int offtable_look(S_table t, S_symbol key) {
8     int *p = S_look(t, key);
9     if (!p) {
10         return -1;
11     }
12     return *p;
13 }
```

翻译过程

1. 整体的T_funcDeclList

- 在method列表中, main method在第一个
- 方法的表示: T_funcDecl

1. 命名

- main method的名字为main
- 其他类方法的名字为c\$m (调用S_link函数), 其中c为类名, m为方法名

2. 参数列表

对应 `semant` 模块的 `transA_FormaList` 函数。

- `main method`的参数列表为`NULL`
- 其他类方法的参数列表的第一个参数都是`this` (在`temp.c`中定义为`t99`), 剩余的是声明参数 (可能为`NULL`).

2. 函数调用 (`callStm / callExp`): `callExp`和`callStm`基本一样, 下面只陈述`callExp`

对应 `translate` 模块的 `Tr_CallExp` 函数。

- 首先, 检查实参列表并返回结果`Tr_expList`

对应 `semant` 模块的 `transA_ExpList_Call` 函数。

- 在`methoff`中找到方法对应的偏移, 通过偏移获取方法的地址
 - 封装在 `translate` 模块的 `Tr_ClassMethExp` 函数中

```
1 Tr_exp Tr_ClassMethExp(Tr_exp thiz, int offset) {
2     if (offset == 0) {
3         return Tr_Ex(T_Mem(unEx(thiz), T_int));
4     }
5     return Tr_Ex(T_Mem(T_Binop(T_plus, unEx(thiz), T_IntConst(offset *
6         SEM_ARCH_SIZE)), T_int));
7 }
```

- 注意在参数列表的最前面加上对象地址 `thiz`
- `emit`的`IRP`语句要套一层`T_escExp`: 避免再次用到的时候重复执行`call`

3. 获取类变量 (`classVarExp`)

- 通过查`varoff`表找到对应的偏移即可

4. `new`类对象 (`newObjExp`)

- 代码组织

```
1 // "semant.h"
2 expty transA_NewObjExp(FILE *out, A_exp e);
3 Tr_exp transA_NewObjClassVar(FILE *out, S_table vtbl, Tr_exp tmpobj,
4 Tr_exp newObjStm);
5
6 // "translate.h"
7 Tr_exp Tr_NewObjAlloc(Tr_exp tmpobj, int size);
8 Tr_exp Tr_NewObjTemp(Temp_temp tmp);
9 Tr_exp Tr_ClassMethLabel(Temp_label label);
10 Tr_exp Tr_ClassVarExp(Tr_exp thiz, int offset, T_type type);
11 Tr_exp Tr_ClassMethExp(Tr_exp thiz, int offset);
```

- 首先新建一个`Temp_temp`, 调用`malloc`分配`globaloff * SEM_ARCH_SIZE bytes`的空间 (即 `Tr_newObjAlloc` 函数)

- 初始化类变量：遍历对应的vtbl，根据表项中记录的A_varDecl判断是否需要初始化，若需要则emit初始化语句（即 `transA_NewObjClassVar` 函数）
 - 类似 `transA_VarDeclList` 的处理方式
- 初始化所有类方法：遍历对应的mtbl，根据表项中记录的来源类from，将对应的namedLabel from\$m（根据 `Tr_ClassMethLabel` 函数生成）Move到对应偏移量的内存中
- 最后在以上所有初始化语句的外面套一层T_escExp，返回新建的temp即可

遇到的问题

1. `new A().m()` 的this问题

- 首先会处理 `new A()`，返回一个obj，形式是T_escExp（其中的T_stm即为初始化语句）
- 在处理函数调用 `obj.m()` 时，也会将obj作为参数列表的第一个传入
 - 此时应该用对应的Temp_temp，而非escExp本身，因为不能多次执行初始化语句
- 故在 `Tr_CallExp` 和 `Tr_CallStm` 中，对obj为T_escExp的情况不断解包，直到找到对应的Temp_temp

```

1 Tr_exp Tr_CallExp(string meth, Tr_exp thiz, Tr_exp methAddr, Tr_expList
  e1, T_type type) {
2   T_exp obj = unEx(thiz);
3   while (obj && obj->kind == T_ESEQ) {
4     obj = obj->u.ESEQ.exp;
5   }
6
7   return Tr_Ex(T_Call(meth, unEx(methAddr), T_ExpList(obj,
  unTrExpList(e1)), type));
8 }

```

测试结果

hw8/test下只有一个测试文件：hw8test00.fmj，将make test和make ast2irp的结果比对后发现无误。

开发过程

git提交记录如下：

 	hw8: fix callexp side effect bug	28 Apr 2024 16:35	Jopqior
	hw8: remove vendor/tools	25 Apr 2024 21:34	Jopqior
	hw8: merge master	25 Apr 2024 21:32	Jopqior
	hw8: remove duplicate codes	25 Apr 2024 21:30	Jopqior
	hw8: refactor call codes	25 Apr 2024 09:16	Jopqior
	hw8: merge master	25 Apr 2024 08:14	Jopqior
	hw8: fix eseqexp for this in callexp	25 Apr 2024 08:09	Jopqior
	hw8: add external test	23 Apr 2024 21:29	Jopqior
	hw8: add call escexp debug and refactor	23 Apr 2024 21:28	Jopqior
	hw8: optimize arrayexp pos calculation	23 Apr 2024 18:05	Jopqior
	hw8: fix mainmethod set curClassId bug	23 Apr 2024 17:51	Jopqior
	hw8: add class	23 Apr 2024 17:33	Jopqior
	hw8: add arr	23 Apr 2024 00:06	Jopqior
	hw8: initial commit	22 Apr 2024 21:48	Jopqior