PA3 实验报告

计73 王焱 2017050024

本阶段任务，将带属性修饰的AST翻译为更底层的一种中间表示 TAC 。

除零错

仿照Index的处理，在visitBinary中，对于div 和 mod 两种情况，进行右操作数是否为零的检查。

```
default void visitBinary(Tree.Binary expr, FuncVisitor mv) {
    if ((expr.op.equals(Tree.BinaryOp.EQ) || expr.op.equals(Tree.BinaryOp.NE)) &&
            expr.lhs.type.eq(BuiltInType.STRING)) {...}

    expr.lhs.accept( v: this, mv);
    expr.rhs.accept( v: this, mv);

    if(expr.op.equals(Tree.BinaryOp.DIV)||expr.op.equals(Tree.BinaryOp.MOD)){
        var zero = mv.visitLoad( value: 0);
        var error = mv.visitBinary(TacInstr.Binary.Op.EQU, expr.rhs.val, zero);

        var handler = new Consumer<FuncVisitor>() {...};
        emitIfThen(error, handler, mv);
    }

    var op = switch (expr.op) {...};

    expr.val = mv.visitBinary(op, expr.lhs.val, expr.rhs.val);
}
```

ABSTRACT

在TacGen中访问函数体之前，添加对于abstract属性的判断，避免出现空指针。

VAR

不用额外处理

扩展call

由于新特性的添加，对原来的call进行调整，统一为一种形式。约定 expr.func.val 存储的格式为偏移量0处为函数指针，偏移量为4处为外部参数个数，对于membermethod 是object，对于lambda 是捕获变量。在call中需要做的是将参数一次压栈，然后调用新添加的函数 visitcall，其中使用IndirectCall的方式对函数进行调用。

```
    default void visitCall(Tree.Call expr, FuncVisitor mv) {
        if (expr.isArrayLength) {...}
        expr.func.accept( v: this, mv);
        expr.args.forEach(arg -> arg.accept( v: this, mv));
        var temps = new ArrayList<Temp>();
        expr.args.forEach(arg -> temps.add(arg.val));
        var entry = mv.visitLoadFrom(expr.func.val,  offset: 0);
        var parmnum = mv.visitLoadFrom(expr.func.val,  offset: 4);
        var zero = mv.visitLoad( value: 0);
        var one = mv.visitLoad( value: 1);
        var four = mv.visitLoad( value: 4);
        var eight = mv.visitLoad( value: 8);
        var parmaddr = mv.visitBinary(TacInstr.Binary.Op.ADD, expr.func.val, eight);
        Function<FuncVisitor, Temp> test = v -> v.visitBinary(TacInstr.Binary.Op.NEQ, parmnum, zero);
        var body = new Consumer<FuncVisitor>() {
            @Override
            public void accept(FuncVisitor v) {
                var parameter = v.visitLoadFrom(parmaddr);
                v.visitBinarySelf(TacInstr.Binary.Op.ADD, parmaddr, four);//paramaddr++
                v.visitBinarySelf(TacInstr.Binary.Op.SUB, parmnum, one);//num--
                v.visitParm(parameter); // push stack
            }
        };
        emitWhile(test, body, mv.freshLabel(), mv);
        for (var arg : temps)
            mv.func.add(new TacInstr.Parm(arg));
        expr.val = mv.visitCall(entry,  needReturn: true);
TacEmitter › visitCall()
```

函数变量

在visitvarsel中，按照约定格式将函数的地址，调用对象等信息存储进一块内存，返回这个函数对象。在这里需要取得函数的地址，对于static的情况，使用了一个全局虚表 Static_orz 存储了所有静态函数。对于非static情况，则从相应的vtable中取出即可。

这里static_orz 存在的一个问题是，它的offsets的key按照原来的实现方法，在全局中会有重复，所以我为它写了相应putOffsetsForStaticVtlb和 getOffsetsForStaticVtlb 。

```
    default void visitVarSel(Tree.VarSel expr, FuncVisitor mv) {
//        System.out.println("visitVarsel");
        if(expr.symbol.isVarSymbol()){...}
        if (expr.symbol instanceof MethodSymbol) {
            var symbol = (MethodSymbol)expr.symbol;
            if (symbol.isMemberMethod()) {
                if (symbol.isStatic()) {
//                    System.out.println("Static");
                    var vtbl = mv.visitLoadVTable( clazz: "Static_orz");
//                    System.out.println("aaaaaaaaaa offset :"+mv.ctx.getOffsetForStaticVtlb("Static_orz",symbol.owner.name , expr.name));
                    var entry = mv.visitLoadFrom(vtbl, mv.ctx.getOffsetForStaticVtlb( vtblName: "Static_orz",symbol.owner.name ,expr.name));
                    var memory = mv.visitIntrinsicCall(Intrinsic.ALLOCATE,  needReturn: true, mv.visitLoad( value: 8));
                    mv.visitStoreTo(memory,  offset: 0, entry);
                    mv.visitStoreTo(memory,  offset: 4, mv.visitLoad( value: 0));
                    expr.val = memory;
                } else {
//                    System.out.println("notStatic");
                    expr.receiver.get().accept( v: this, mv);
                    var object = expr.receiver.get().val;
                    var vtbl = mv.visitLoadFrom(object);
                    var entry = mv.visitLoadFrom(vtbl, mv.ctx.getOffset(((MethodSymbol) expr.symbol).owner.name, expr.name));
                    var memory = mv.visitIntrinsicCall(Intrinsic.ALLOCATE,  needReturn: true, mv.visitLoad( value: 12));
                    mv.visitStoreTo(memory,  offset: 0, entry);
                    mv.visitStoreTo(memory,  offset: 4, mv.visitLoad( value: 1));
                    mv.visitStoreTo(memory,  offset: 8, object);
                    expr.val = memory;
                }
TacEmitter › visitVarSel()
```

Lambda表达式

- 捕获变量

  在Typer中，用一个lambdaStack记录当前访问过的lambda表达式，在visitvarsel中遍历lambdaStack，遇到lambda在变量之后的，进行捕获。

```
for(var lambda : lambdaStack){//capture var
    if((varsymbol.pos).compareTo(lambda.pos) < 0){//before
        lambda.capture.add(varsymbol);
    }
}
```

- 新建函数

  在Typer阶段，记录整个程序的所有lambda表达式，记录在Tree.toplevel中，然后在TacGen中为每个lambda表达式新建函数，并为他们新建一个全局虚表 Lambda_orz 。

```
    // emit tac instructions for every lambda.
    for (var lambda : tree.lambdas) {
        FuncVisitor mv;
        // Remember calling convention: pass `this` (if non-static) as an extra argument, via reversed temps.
        var numArgs = lambda.params.size()+lambda.capture.size()+1;
        mv = pw.visitFunc( className: "Lambda_orz",  funcName: "lambda"+lambda.pos, numArgs);
        var i = 1;
        for(var capturevar : lambda.capture){
            capturevar.temp = mv.getArgTemp(i);
            i++;
        }
        for (var param : lambda.params) {
            param.symbol.temp = mv.getArgTemp(i);
            i++;
        }

        if (lambda.expr != null) {
            lambda.expr.accept( v: this, mv);
            mv.visitReturn(lambda.expr.val);
        }
        if (lambda.body != null) {
            lambda.body.accept( v: this, mv);
        }
        mv.visitEnd();
    }
```

- 新建函数对象

  与varsel不同的是，lambda需要存入的参数多了若干捕获变量，将他们也依次存进内存中即可。

```
@Override
default void visitLambda(Tree.Lambda expr, FuncVisitor mv){
    var vtbl = mv.visitLoadVTable( clazz: "Lambda_orz");
    var entry = mv.visitLoadFrom(vtbl, mv.ctx.getOffset( clazz: "Lambda_orz",  member: "lambda" + expr.pos));
    var size = 12 + expr.capture.size() * 4;

    var memory = mv.visitIntrinsicCall(Intrinsic.ALLOCATE,  needReturn: true, mv.visitLoad(size));
    mv.visitStoreTo(memory,  offset: 0, entry);
    mv.visitStoreTo(memory,  offset: 4, mv.visitLoad( value: 1 + expr.capture.size()));
    mv.visitStoreTo(memory,  offset: 8, expr.inStatic ? mv.visitLoad( value: 0) : mv.getArgTemp( index: 0));
    for (int i = 0;i < expr.capture.size();i ++) {
        mv.visitStoreTo(memory,  offset: 12 + 4 * i, expr.capture.get(i).temp);
    }
    expr.val = memory;
}
```

小结

开始的挺早，明白的太晚。中途还换了一种方法，最开始用的是指导书推荐的方法，然后一直也没调出来，甚至一度不想写了，后来换了一种比较简单的方法，即建全局虚表，再加上同学的讲解，过了ddl两天才写完orz.