编译原理实验报告

PA4

姓名：谢兴宇

学号：2017011326

2020 年 1 月

1 工作简述

在此前的整体框架基础上，我继续实现了死代码消除。死代码消除即删去对不会对后续产生任何影响的语句，即删去对不活跃变量的定值语句，再递归删去由此前的删除而死亡的代码。

2 死代码消除的实现

借助框架中原本已实现好的 `dataflow` 模块计算每一行的活跃变量，对于每一个 TAC 指令，如果其（产生了赋值）且（被赋值的变量在此条指令后不再活跃）且（没有副作用，即没有发生函数调用），便把这一条指令删去。多次迭代上述过程，直到不再有指令被删掉为止。

代码十分简洁，仅有不足 50 行：

```
/**
  * Transformer entry.
  *
  * @param func a TAC program
  * @return also a TAC program, but optimized
  */
override def transform(tacProg: TacProg): TacProg = {
  // 调用 CFG
```

```
val analyzer = new LivenessAnalyzer[TacInstr]

var funcs = tacProg.funcs.asScala
var changed = false
do {
  changed = false

  funcs.foreach(func => {
    val instrSeq = func.getInstrSeq().asScala
    val cfg = CFG.buildFrom(instrSeq.toList)
    analyzer(cfg)

    // 进入 CFG 分析
    val block_it = cfg.iterator
    while (block_it.hasNext) {
      val loc_it = block_it.next().seqIterator
      while (loc_it.hasNext) {
        val loc = loc_it.next()
        val instr = loc.instr

        // 是否产生了赋值
        if (!instr.dsts.isEmpty) {
          // 被赋值的 Temp 是否是活跃变量
          if (!loc.liveOut.contains(instr.dsts(0))) {
            // printf(s"${instr.dsts(0)} is not live!\n")

            // 是否是一个 call 赋值给 Temp
            if (!instr.isInstanceOf[TacInstr.IndirectCall] && !instr
                  .isInstanceOf[TacInstr.DirectCall]) {
              instrSeq -= instr

              changed = true
            }
```

```
            }
          }
        }
      }
    })
  } while (changed)

  tacProg
}
```

3 实验中遇到的困难

- 不知道如何修改TacProg中的指令，或是根据传入的TacProg创建一个拥有被优化的TacProg。

  解决：与同学交流后发现，可以通过更改TacFunc.instrSeq来实现。之前误以为不能，是因为误解了 Java 和 Scala 的传递机制，其在传递对象时并不会传递对象的复值而是传递对象本身。

- Scala 框架的后端使用 Java 实现，对于在 Scala 中调用 Java 的对象产生了一定的困惑。

  有的 Java 对象可以直接隐式转换，比如 Java 的Array与 Scala 的Array，因为二者拥有完全相同 JVM 字节码（https://stackoverflow.com/questions/3940699/passing-java-array-to-scala）。比较复杂一些 Java 对象（比如 Java 的List和 Scala 的ListBuffer）可以通过collection.JavaConverters包中的toJava和toScala来转换。

4 性能测试结果

测例	basic-basic	basic-fibonacci	basic-math	basic-queue	basic-stack	mandelbrot	rbtree	sort	mycase
执行指令数（优化前）	41	7014	209	5827	867	5358318	2684498	593424	6
执行指令数（优化后）	41	7014	209	5827	867	5358318	2684498	593424	1

表 1: 优化前后指令执行条数的对比

其中，mycase 为我自己编写的测例，以测试我正确实现了死代码删除并检验其有效性，其内容为：

```
class Main {
    static void main() {
        int a = 1;
        int b = a;
        int c = b;
        int d = c;
    }
}
```