decaf PA5 实验报告

电 53 魏宇轩 2015011942

一、基本块中两个结点连边的条件

对于基本块 bb 的每一条 TAC 语句 ti，如果该 TAC 语句对变量 A 定值，则将结点 A 与所有这样的结点 B 连边，$B \in \{bb.liveUse \cap ti.liveOut \backslash A\}$.

举例来说，如果有如下的 TAC 语句（TAC 语句的 liveOut 集合标注在后面中括号中），它在这样的基本块中：

```
Basic Block 0:
Def = …
liveUse = {_T0, _T1}
liveIn = …
liveOut = …


……
_T4 = …… [_T0, _T5]
……
```

则在_T4 与_T0 之间连边。

二、具体代码修改细节

1. 修改 InferenceGraph.java 的 InferenceGraph.addEdge 方法，增加对于两结点之间的边是否存在的判断，只有当两结点连边之前不存在时，才修改结点的度。

2. 修改InferenceGraph.java的InferenceGraph.makeEdges方法，对于定值的TAC类型（***ADD***, ***SUB***, ***MUL***, ***DIV***, ***MOD***, ***LAND***, ***LOR***, ***GTR***, ***GEQ***, ***EQU***, ***NEQ***, ***LEQ***, ***LES***, ***NEG***, ***LNOT***, ***ASSIGN***, ***LOAD_VTBL***, ***LOAD_IMM4***, ***LOAD_STR_CONST***, ***INDIRECT_CALL***, ***DIRECT_CALL***, ***LOAD***）按照“一”中所述的条件连边。

3. 修改GraphColorRegisterAllocator.java的GraphColorRegisterAllocator.alloc方法，首先增加TAC语句，将基本块的liveUse集合中的变量载入寄存器（从栈load到临时变量），并把基本块的liveIn集合作为这些TAC语句的liveOut集合。

然后实例化InferenceGraph对象，调用InferenceGraph.alloc方法实现基于干涉图染色的寄存器分配算法。

三、完整的干涉图染色寄存器分配算法

1. 结点连边条件修改为：

对于函数的每一条 TAC 语句 ti，如果该 TAC 语句对变量 A 定值，则将结点 A 与所有这样的结点 B 连边，$B \in \{ti.liveOut \backslash A\}$.

2. 关键伪代码

(1) 修改修改 InferenceGraph.java 的 InferenceGraph.makeEdges 方法的连边条件。

```
case ADD: case SUB: case MUL: case DIV: case MOD:
case LAND: case LOR: case GTR: case GEQ: case EQU:
```

```
case NEQ: case LEQ: case LES:
case NEG: case LNOT: case ASSIGN:
case LOAD_VTBL: case LOAD_IMM4: case LOAD_STR_CONST:
case INDIRECT_CALL:
case DIRECT_CALL:
case LOAD:
    for (Temp temp : tac.liveOut)
    {
        if (temp != null && !temp.equals(tac.op0)) {
            addEdge(tac.op0, temp);
        }
    }
    break;
```

(2) 修改 InferenceGraph.java 中的 InferenceGraph 类的 BasicBlock bb 属性为 BasicBlock[] bbs 属性，表示一个函数包含的基本块数组，同时修改 TAC 语句遍历部分。

```
for (BasicBlock bb : bbs)
{
    for (Tac tac = bb.tacList; tac != null; tac = tac.next) {……}
}
```

(3) GraphColorRegisterAllocator.java 中首先对于整个函数按照干涉图染色算法分配寄存器，再对每一个基本块进行处理。

```
InferenceGraph ig = new InferenceGraph();
ig.alloc();

for (BasicBlock bb : bbs)
{
    alloc(bb);
}
```