实验五——中间代码优化

姓名：肖丹妮　　学号：201220199　　院系：计算机科学与技术系

一、实验环境与编译方法

实验环境与手册一致；

使用Mikefile文件进行编译，执行make就可以编译成功。

二、框架代码重点梳理

程序完整的工作

1. 解析输入文件：

```
IR_parse(argc >= 2 ? argv[1] : NULL);
```

2. 进行中间代码优化：

```
IR_optimize();
```

3. 将优化后的中间代码打印到指定的文件：

```
IR_output(argc >= 3 ? argv[2] : NULL);
```

中间代码优化IR_optimize();

流程为：
常量传播、可用表达式分析、复制传播、第二次常量传播、活跃变量分析。

涉及到的分析为常量传播、可用表达式分析、复制传播、活跃变量分析。

进行数据流分析

求解器

solver.c已经给出了迭代求解器和worklist求解器的前向求解器的完整实现，后向求解器的内容待补充完整。

数据流分析接口

```
struct DataflowAnalysis {
    struct {
        void (*teardown) (DataflowAnalysis *t);
        bool (*isForward) (DataflowAnalysis *t);
        Fact *(*newBoundaryFact) (DataflowAnalysis *t, IR_function *func);
        Fact *(*newInitialFact) (DataflowAnalysis *t);
        void (*setInFact) (DataflowAnalysis *t, IR_block *blk, Fact *fact);
        void (*setOutFact) (DataflowAnalysis *t, IR_block *blk, Fact *fact);
        Fact *(*getInFact) (DataflowAnalysis *t, IR_block *blk);
```

```
        Fact *(*getOutFact) (DataflowAnalysis *t, IR_block *blk);
        bool (*meetInto) (DataflowAnalysis *t, Fact *fact, Fact *target);
        bool (*transferBlock) (DataflowAnalysis *t, IR_block *block, Fact
*in_fact, Fact *out_fact);
        void (*printResult) (DataflowAnalysis *t, IR_function *func);
    } const *vTable;
};
```

每个分析都有这样一个虚函数表，初始化时每一个API都会指定成自己特定的分析方法，在每个分析的c文件中单独具体实现。

数据流分析

available_expressions_analysis.c、constant_propagation.c、copy_propagation.c、live_variable_analysis.c：
Dataflow Analysis部分实现虚函数表vTable的每个API。

求解器调用这些API，IR_optimize()中建立分析，调用求解器，得到分析结果。

根据分析结果优化中间代码

available_expressions_analysis.c、constant_propagation.c、copy_propagation.c、live_variable_analysis.c：
Optimize部分实现优化操作。

IR_optimize()中得到分析结果后调用对应的优化操作，完成中间代码优化。

三、补全框架代码

后向求解器

和前向求解器不一样的处理是：

1. 初始化时，func的OutFact需要初始化，exit需要单独处理，每个basicblock都需要初始化InFact。
2. 分析时，OUT[blk] = meetAll(IN[succ] for succ in AllSucc[blk])；若In[blk]发生update, 则进行继续迭代。

可用表达式分析

1. 边界条件是空集，OutFact[Entry] = Bottom: empty set，因此return NEW(Fact_set_var, false);
2. 是Must Analysis，InitFact =Top: universal set，因此return NEW(Fact_set_var,true);
3. 是Must Analysis，IN[blk] = intersect_with (all OUT[pred_blk])，因此return VCALL(target->set, intersect_with, &fact->set);

常量传播

1. NAC ⊓ v = NAC，UNDEF ⊓ v = v，c ⊓ c = c， c1 ⊓ c2 = NAC
2. v1.kind == CONST && v2.kind == CONST的情况已经给出；若v1是NAC或v2是NAC，若v2是常量0且运算类型是除或者取余，则得到UNDEF，否则得到NAC；若不是上述两种情况，则得到UNDEF
3. 常量传播是前向分析
4. 在Boundary(Entry)中, 函数参数初始化为NAC
5. 分析IR_ASSIGN_STMT：def=……，更新def的值为use的值
6. 分析IR_OP_STMT：def=……，更新def的值为由calculateValue()计算得到的值
7. 分析其他有新的变量定义的stmt：def=……，更新def的值为NAC

复制传播

1. 复制传播是前向分析
2. 边界条件是空集，故return NEW(Fact_def_use, false);
3. 是Must Analysis，InitFact =Top: universal set，因此return NEW(Fact_set_var,true);
4. 删除以new_def为def的关系：new_def=use，即删除def_to_use[new_def]中的use以及use_to_def[use]中的def
5. 删除以new_def为use的关系：def=new_def，即删除use_to_def[new_def]中的def以及def_to_use[def]中的use
6. 建立关系：def=use，即设置def_to_use[def]=use以及use_to_def[use]=def

活跃变量分析

1. 活跃变量分析是后向分析
2. 交汇运算是并，因此return VCALL(*target, union_with/intersect_with?, fact);
3. 因为是后向分析，因此应该先执行kill再执行gen，kill对应的是delete def，gen对应的是insert use
4. def不是活跃变量时可以被标记为死代码，即VCALL(*new_out_fact, exist, def) == false

四、实验心得

首先理解数据流分析的通用框架是很重要的，其次应该能够根据各个分析不同的需求和背景确定其方向、边界条件、交汇运算、传递函数等等的具体情况。中间代码优化没有一定的标准，但是经过数据流分析得到的结果进行针对性的冗余代码消除后，总能够达到使代码更加高效和简洁的目的。

另：

实验使用了贺柄毓同学提供的框架代码，非常感谢贺柄毓同学的分享和帮助！！！