compiler-lab5 中间代码优化

1. 实验目标

通过数据流分析对中间代码优化。

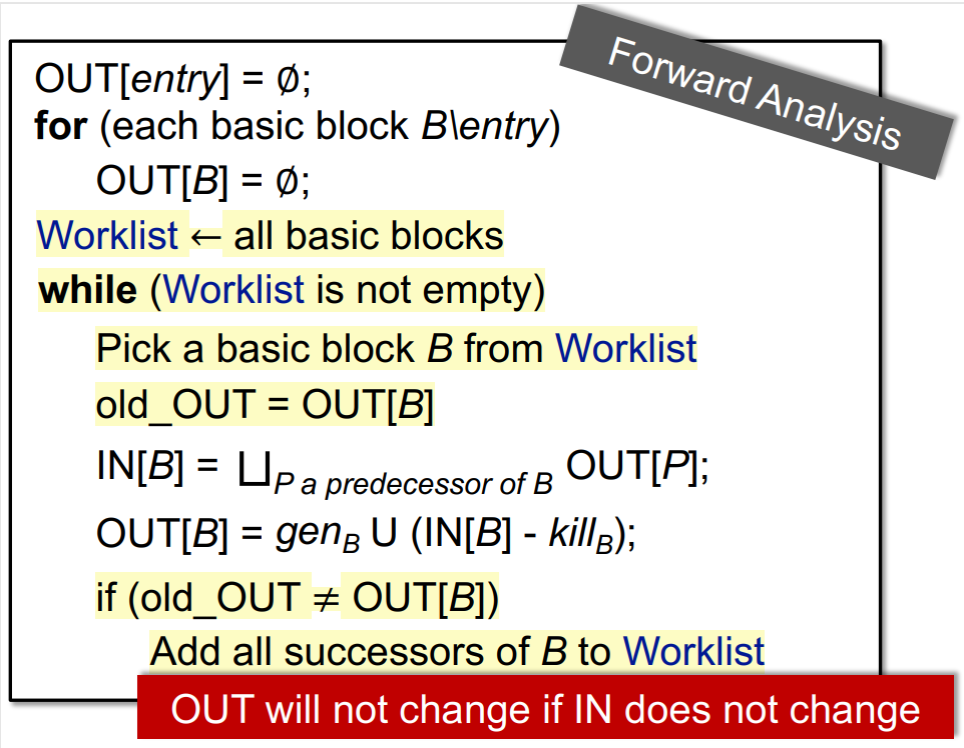
2. 完成的任务

在实验框架的基础上，完成了活跃变量分析，复制传播，常量传播以及可用表达式分析，并用分析结果完成中间代码优化。

3. 总体思路

3.1. 求解器

工作集算法是迭代算法的有效改进，他能够有效减少重复计算。前向分析的伪代码如下（参考软件分析第4讲）。



大致思路是经过迁移函数后，如果OUT改变那么就将所有后继加入到工作集中。

3.2. 数据流分析框架

工作集算法有几个重要的部分，分别是边界`Fact`的设置，非边界`Fact`的初始化，交汇运算以及传递函数。因此整个数据流分析算法可以抽象成一下几个函数：

- `newBoundaryFact`：返回新的边界`Fact`
- `newInitialFact`：返回新的初始化`Fact`
- `transferNode`：传递函数
- `meetInto`：交汇函数

3.3. 数据流分析汇总

不同数据流分析的上述四个函数都大同小异，无非需要注意的是通常传递函数先从`Fact`中剔除`kill`，再将`gen`加入`Fact`。以及在基本块内部对语句的分析顺序需要与整体的分析顺序（前向或后向）保持一致。另外，**复制传播**（copy propagation）作为全局表达式合并的关键，在课本中并未提及，这里简单总结一下该分析的关键点。非常类似于可用表达式分析。

- 分析方向：前向分析
- $boundaryFact = \top$
- $initialFact = \bot$
- $kill = \{x = y | x或y在基本块内部被重定义\}$
- $gen = \{x = y | 从entry到当前复制语句x或y未被重定义的语句\}$

框架代码使用`def-use`和`use-def`的`map`记录最近一次复制的def和使用情况。因此在kill和gen时需要同时更新两个`map`，否则可能导致

重要的赋值语句被删除。

其他数据流分析汇总在下表中（参考软件分析第四讲），参照下表即可较轻松地完成四个核心函数。

	Reaching Definitions	Live Variables	Available Expressions
Domain	Set of definitions	Set of variables	Set of expressions
Direction	Forwards	Backwards	Forwards
May/Must	May	May	Must
Boundary	OUT[entry] = ∅	IN[exit] = ∅	OUT[entry] = ∅
Initialization	OUT[B] = ∅	IN[B] = ∅	OUT[B] = U
Transfer function	OUT/IN = gen U (IN/OUT - kill)		
Meet	U	U	∩

3.4. 代码优化相关

使用活跃变量分析和常量传播分析进行死代码消除。具体而言，如果变量在定义点不活跃（不在活跃变量结果中），那么该条代码就是无用代码，删除即可；常量传播分析不仅可用于常量折叠，也能用于分支死代码消除。即假设一条条件判断语句为`if a > 1 then...`如果`a`为常量就可以直接用常量信息判断数据流去往哪条分支（假如a的常量分析结果为2那么为假的分支就是死代码）。

复制传播算法和可用表达式分析用于冗余表达式的消除。首先将所有相同右值（表达式）统一定义赋值给一个变量exp_i。并将所有接受该表达式的变量右值替换成exp_i，使用复制传播将**定义**换成**使用**。于是此时代码中就会出现无效的定义语句，此时再执行活跃变量分析，将所有不活跃的变量删除即可，效果也较好。

4. 总结

本次实验完成了四种数据流分析，并结合框架进行了中间代码优化。总的来说在实验框架的帮助下（再次感谢hby大佬拯救期末月~）以及软件分析课程的基础，实验的完成较为顺利，也从优美的框架组织中学到了不少巧妙的写法（宏的替换、c的面向对象）大大简化了冗余代码。

本次实验的完成也标志着c--编译器的初具雏形。回顾词法分析，语法分析，语义分析和中间代码生成，以及最后的中间代码优化和目标代码生成，完成了编译器的前中后端。至此，2022的编译原理实验告一段落，完结撒花~