2015/1/3 沈哲言 计 34 2013011371

DECAF 实验阶段四（PA4）

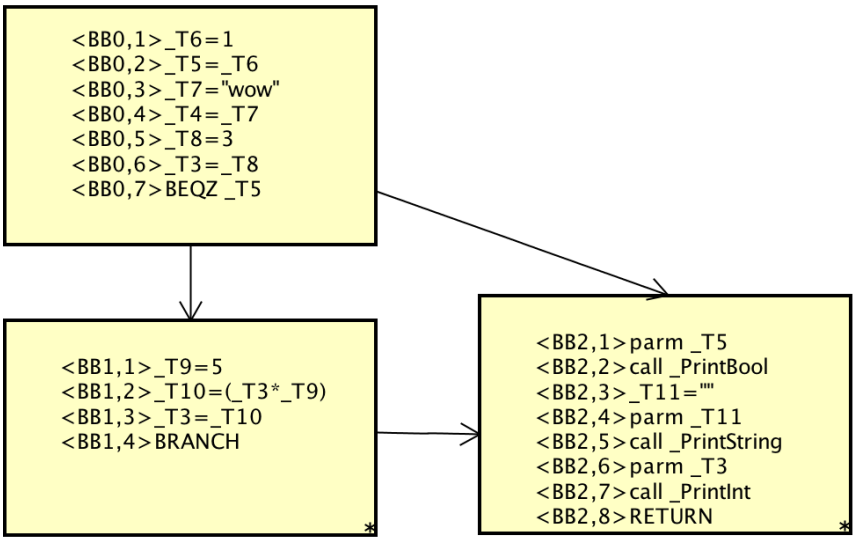
1. 实验内容：本实验基于框架中已有的活跃变量分析实现了定值点的 DU 链，并以 filename.du 的文件格式保存在 OUTPUT 文件夹下（与活跃变量分析以及运行结果在同一文件夹下）；输出形式参考了活跃变量分析，在定值点（TAC 语句）后附加了该定值点的引用信息。为了便于描述 DU 链中的元素，我用（基本块号，块内行号）的二元组来描述一个引用点，并以基本块为单位统计了行号，在 TAC 语句之间输出，便于观察和调试。（下图为一个示例，后续会用图示进一步说明）

```
FUNCTION main :
BASIC BLOCK 0 :
1:_T6 = 1 [ BB 0: line 2; ]
2:_T5 = _T6 [ BB 0: line 7; BB 2: line 1; ]
3:_T7 = "wow!" [ BB 0: line 4; ]
4:_T4 = _T7 [ ]
5:_T8 = 3 [ BB 0: line 6; ]
6:_T3 = _T8 [ BB 2: line 6; BB 1: line 2; ]
7:END BY BEQZ, if _T5 =
    0 : goto 2; 1 : goto 1
BASIC BLOCK 1 :
1:_T9 = 5 [ BB 1: line 2; ]
2:_T10 = (_T3 * _T9) [ BB 1: line 3; ]
3:_T3 = _T10 [ BB 2: line 6; ]
4:END BY BRANCH, goto 2
BASIC BLOCK 2 :
1:parm _T5 []
2:call _PrintBool []
3:_T11 = " " [ BB 2: line 4; ]
4:parm _T11 []
5:call _PrintString []
6:parm _T3 []
7:call _PrintInt []
8:END BY RETURN, void result
```

2. 实验完成思路：实验完成的思路基于***深度优先搜索策略***，框架中的控制流图已经为基本块之间建立了清晰的执行顺序，那么我只要从一个定值点出发，沿着执行流程一路走下去，图中记录所有的引用点信息，直至碰到***新的定值点***或者遇到递归边界退出即可。递归边界的判定有两种：一种是当前基本块已经被搜索过，即陷入了循环，故循环路径上所有的引用点肯定已经被记录过了，可以退出；第二种是函数返回了（RETURN），没有再进一步的条件跳转。故整个搜索的复杂度只是基本块个数的线性复杂度。

3. 主要修改：BASICBLOCK 中加入了统计行号，打印 DU 链的操作；FLOWGRAPH 中加入了搜索函数，遍历待分析函数中的每一条定值语句，从该定值语句的基本块出发，通过深度优先搜索模拟程序的执行流，从而记录引用点信息；OPTION 中加入了新的调试级别，对应 DU 链的输出；DRIVER 中增加了相应的打印语句（仿照活跃变量分析）；RUNALL.PY 中增加了 FILENAME.DU 的生成语句

4. 实验结果：还是通过上面那个输出实例来解释，这次采用更直观的基本块流图的方式



DU-链

定值点	DU 链
<BB0,1>_T6	<BB0,2>
<BB0,2>_T5	<BB0,7> <BB2,1>
<BB0,3>_T7	<BB0,4>
<BB0,5>_T8	<BB0,6>
<BB0,6>_T3	<BB2,6> <BB1,2>
<BB1,1>_T9	<BB1,2>
<BB1,2>_T10	<BB1,3>
<BB1,3>_T3	<BB2,6>
<BB2,3>_T11	<BB2,4>

5. 实验小结：通过本次的实验，我了解了最基本的现代编译器优化中间代码的流程，通过基本块的划分和流图的生成，编译器可以做死代码删除以及寄存器分配等级别不一的优化操作；同时我也明白了要让程序的运行效率更高，除了程序自身实现的巧妙之外，还离不开编译器的支持。