

Decaf PA4 实验报告

计 63 肖朝军 2016011302

一、 实验内容

本次 PA4 实验要求实现编译器的数据流分析功能,在本次实验中需要增加的部分为实现 DU 链的求解。提供的代码框架中已经实现了活跃变量的分析,也即实现了每一个代码块 B 的 $\text{Def}[B]$ 、 $\text{LiveUse}[B]$ 、 $\text{LiveIn}[B]$ 、 $\text{LiveOut}[B]$ 的求解。本次实验为了实现 DU 链的求解,我按照讲义 12 中的内容,对上述四个集合的定义进行了扩展,为集合中每个元素加上了位置信息,进而求解出 DU 链。

二、 实现方式

1、 拓展数据流方程中四个集合的定义

- a) $\text{Def}[B]$ 为 $\langle s, A \rangle$ 的集合,其中 s 是 B 中变量 A 的定义点,但 A 在 B 中被重新定值。(此处与 lecture 中定义有一些不同)
- b) $\text{LiveUse}[B]$ 为 $\langle s, A \rangle$ 的集合,其中 s 是 B 中某点, s 引用变量 A 的值,且 B 中在 s 前面没有 A 的定值点。
- c) $\text{LiveIn}[B]$ 为 $\langle s, A \rangle$ 的集合,其中 s 可以为 B 中某点或者 B 后某个代码块中的点, s 引用变量 A 的值。表示变量 A 在 B 入口处是活跃的,因为在 s 点引用了 A 的值。
- d) $\text{LiveOut}[B]$ 为 $\langle s, A \rangle$ 的集合, A 为在 B 出口处活跃的变量,相应的 s 为在之后引用变量 A 的点。

2、 数据流方程

数据流方程与原数据流方程相似

$$\text{LiveOut}[B] = \bigcup \text{LiveIn}(b), \quad b \in S[B]$$

$$\text{LiveOut}[B]' = \{ \langle s, A \rangle \mid \langle s, A \rangle \in \text{LiveOut}[B] \ \&\& \ A \text{ 在 } B \text{ 中没有定义} \}$$

$$\text{LiveIn}[B] = \text{LiveUse}[B] \cup \text{LiveOut}[B]'$$

3、数据流方程求解

数据流方程求解与之前框架中给的方程一样，需要注意的是，由于之前定义未拓展时，循环迭代的次数要比定义拓展后的次数少，因此在设置循环迭代停止条件时，应该当定义拓展后的集合不变时，循环停止。

4、DU 链的求解

定义拓展后， $\text{LiveOut}[B]$ 给出的信息是，在离开基本块 B 后，哪些变量的值还会被引用并给出引用的点的位置信息。有了这个信息之后可以直接求解 B 中任意一个变量 A 在定值点 u 的 DU 链。只要对 B 中 u 后面部分进行扫描：如果 B 中 u 后面没有 A 的其它定值点，则 B 中 u 后面 A 的所有引用点加上 $\text{LiveOut}[B]$ 中 A 的所有引用点，就是 A 在定值点 p 的 DU 链；如果 B 中 u 后面有 A 的其它定值点，则从 u 到与 u 距离最近的那个 A 的定值点之间的 A 的所有引用点，就是 A 在定值点 u 的 DU 链。

三、 样例分析

以测试样例中 `t0.decaf` 为例，编译器为这段代码生成了 8 个语句块，其中有 2 个语句块用作跳转语句块，1 个为返回语句块，其余与 lecture 中一一对应。

```

BASIC BLOCK 0 :
9  _T7 = 0 [ 10 ]
10 _T5 = _T7 [ ]
11 _T8 = 1 [ 12 ]
12 _T6 = _T8 [ ]
13 _T10 = 0 [ 14 ]
14 _T9 = _T10 [ 21 24 30 ]
15 _T11 = 2 [ 16 ]
16 _T3 = _T11 [ 18 ]
17 _T12 = 1 [ 18 ]
18 _T13 = (_T3 + _T12) [ 19 ]
19 _T4 = _T13 [ 28 ]
20 END BY BRANCH, goto 1

```

```

BASIC BLOCK 1 :
21 END BY BEQZ, if _T9 =
    0 : goto 7; 1 : goto 2

```

```

BASIC BLOCK 2 :
22 _T14 = 1 [ 23 ]
23 _T3 = _T14 [ 35 ]
24 END BY BEQZ, if _T9 =
    0 : goto 4; 1 : goto 3

```

```

BASIC BLOCK 4 :
27 _T15 = 1 [ 28 ]
28 _T16 = (_T4 + _T15) [ 29 ]
29 _T4 = _T16 [ 28 32 36 ]
30 END BY BEQZ, if _T9 =
    0 : goto 6; 1 : goto 5

```

```

BASIC BLOCK 5 :
31 _T17 = 4 [ 32 ]
32 _T18 = (_T4 - _T17) [ 33 ]
33 _T4 = _T18 [ 28 36 ]
34 END BY BRANCH, goto 6

```

```

BASIC BLOCK 6 :
35 _T5 = _T3 [ ]
36 _T6 = _T4 [ ]
37 END BY BRANCH, goto 1

```

```

BASIC BLOCK 7 :
38 END BY RETURN, void result

```

```

BASIC BLOCK 3 :
25 call _Main.f
26 END BY BRANCH, goto 4

```

如图，lecture 中 5 个代码块，从依次对应于 B1 - block0, B2 - block2, B3 - block4, B4 - block5, B5 - block6。

从 decaf 中可以发现，代码涉及两次判断跳转，分别是 if 跳转、while 跳转，一次函数返回跳转，得出的结论与 lecture 中图一致。

DU 链的求解：上图已经给出了编译器关于 DU 链的求解的结果，在每一个定值点后，都有相应的引用点的位置。五个变量 i, j, a, b, flag，分别对应程序中 _T3, _T4, _T5, _T6, _T9。i 的定值点有：16、23，j 的定值点有：19、29、33，a 的定值点有：10、35，b 的定值点有：12、36。

各个变量在定值点的 DU 链值与 lecture 中相同，程序正确。

四、 总结

本次实验补充了编译器对活跃变量数据流的计算，这是编译器进行程序优化的第一步，也是非常重要的一步。通过自己编程实现计算 DU 链，更好地掌握了其中的原理，以及体会了算法的正确性。

感谢助教、老师在课程中给予的帮助。