Assignment 3 : Point-to Analysis via Dataflow

Description:

You are supposed to implement a flow-sensitive, field- and context-insensitive algorithm to compute the points-to set for each variable at each distinct program point.

One approach is to extend the provided dataflow analysis framework as follows:

1) Implement your representation of points-to sets
2) Implement the transfer function for each instruction
3) Implement the MEET operator
4) Extend the control flow graph to be inter-procedural 程序间CFG

Note that we do not require the analysis to be context-sensitive or field-sensitive.

Output:

Print the callee functions at every call instructions. The printed format should be ${line} : ${func_name1}, ${func_name2}, here ${line} is unique in output. And NULL is optional.

Note that you should ignore llvm intrinsic functions in output like llvm.memset.p0i8.i64 .

Examples:

1. Simple example.

```
1   #include <stdlib.h>
2   struct fpstruct {
3      int (*t_fptr)(int,int);
4   } ;
5   int clever(int x) {
6      int (*a_fptr)(int, int) = plus;
7      int (*s_fptr)(int, int) = minus;
8      int op1=1, op2=2;
9      struct fpstruct * t1 =
            malloc(sizeof (struct fpstruct));
10     if (x == 3) {
11         t1->t_fptr = a_fptr;
12     } else {
13         t1->t_fptr = s_fptr;
14     }
15     unsigned result = t1->t_fptr(op1, op2);
16     return 0;
17     }
```

Output:

```
9 : malloc
15 : plus, minus
```

2.We don’ t require your analysis to be path-sensitive, so you can ignore all branch conditions(like line : 6).

```
1   #include <stdlib.h>
2   struct fpstruct {
3      int (*t_fptr)(int,int);
4   } ;
5   void foo(struct fpstruct *t, int flag) {
6     if (flag == 3) {
7         t1->t_fptr = plus;
8     } else {
9         t1->t_fptr = minus;
10    }
11  }
12  int clever(int x) {
13     int (*a_fptr)(int, int) = plus;
14     int (*s_fptr)(int, int) = minus;
15     int op1=1;
16     struct fpstruct * t1 =
             malloc(sizeof (struct fpstruct));
17     foo(t1, op1);
18     unsigned result = t1->t_fptr(op1, 2);
19     return 0;
```

```
Output:
16 : malloc
17 : foo
18 : plus, minus
```

3.We don’ t require your analysis to be context-sensitive, so in every call site, the function call will give the same result.

```
1   #include <stdlib.h>
2   struct fpstruct {
3      int (*t_fptr)(int,int);
4   } ;
5   void foo(struct fpstruct *t,
            int (*a_fptr)(int,int)) {
6      t1->t_fptr = a_fptr;
7  }
8  int clever(int x) {
9     int (*a_fptr)(int, int) = plus;
10     int (*s_fptr)(int, int) = minus;
11     struct fpstruct * t1 =
                malloc(sizeof (struct fpstruct));
12     foo(t1, a_fptr);
13     unsigned result = t1->t_fptr(1, 2);
14     foo(t1, s_fptr);
15     result = t1->t_fptr(1, 2);
14     return 0;
```

```
Output:
11 : malloc
12 : foo
13 : plus, minus
14 : foo
15 : plus, minus
```