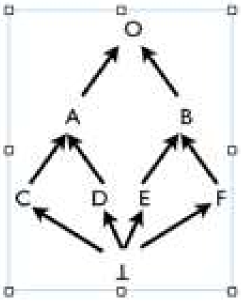
Compilation Methods SS 2013 - Assignment 4

Kastens, Pfahler, 28. Mai 2013

Exercise 1 (Data Flow Analysis Using a Type Lattice)

Consider the semi-lattice L for types of Slide C-2.24b:

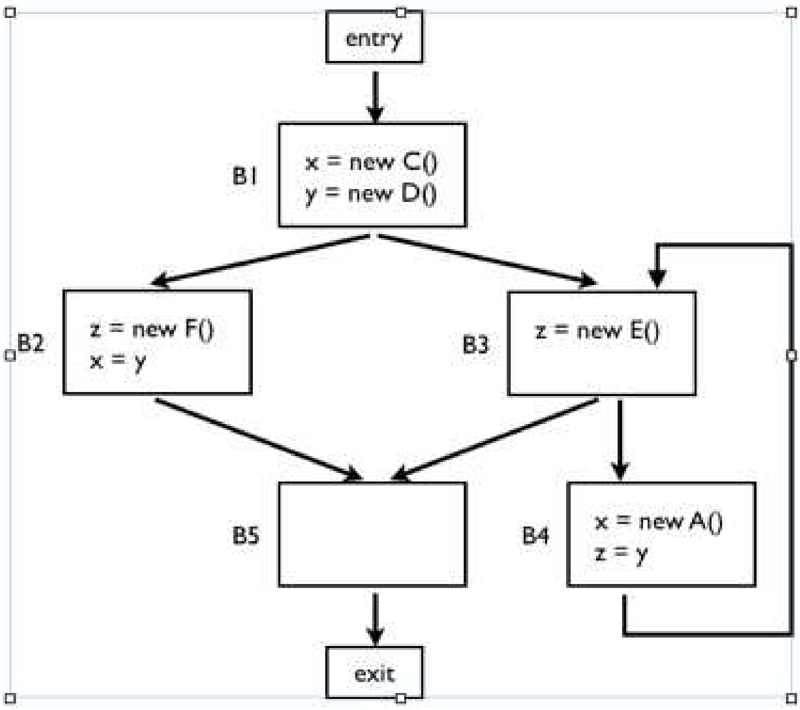


We extend it to a lattice `L3 = L x L x L`. The meet operation of L is defined by:
`x meet y` is the smallest common super-type of x and y. The meet operation of L3 is obtained by applying that of L componentwise.

The partial order in L3 is defined by
`(x1, x2, x3) <= (y1, y2, y3) iff x1<=y1 and x2<=y2 and x3<=y3`.

a) Fill in the missing items:
   1. (C, A, O) meet (D, C, F) = (_, _, _)
   2. (E, B, D) meet (D, C, D) = (_, _, _)
   3. (_, _, _) <= (C, B, O)
   4. (A, _, _) is not <= (_, B, _)
   5. (D, _, _) is not <= (_, O, _)

b) DFA using L3:



On the above CFG DFA using lattice L3 is to be applied to find out the types of values stored in the variables x, y, z at begin and end of basic blocks. Use the following table to specify the transformation functions for the blocks of the CFG:

Block function	x	y	z
1	C	D	z
2			
3			
4			
5			

c) Use the following table to compute the In- and Out-values for each block iteratively:

In, Out	x	y	z	x	y	z	x	y	z	x	y	z
In 1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
Out 1												
In 2												
Out 2												
In 3												
Out 3												
In 4												
Out 4												
In 5												
Out 5												

Exercise 2 (Graphs in Compiler Analysis)

Consider the following kinds of directed graphs used for program analysis: control flow graphs, call graphs, and class hierarchy graphs. What does it mean for each of the kinds of graphs if ...

1. a graph contains a cycle,
2. the undirected graph which corresponds to a given graph is disconnected,
3. a graph is a tree?(distinguish the cases whether the edges point to the root or to the leafs.

Fill your answers in the following table:

	control flow graph	call graph	class hierarchy graph
contains a cycle			
the corresponding undirected graph is disconnected			
is a tree, edges point to the (a) root., (b) leaves			

Exercise 3 (Call Graph)

Construct the context-insensitive call graph of the program:

```
int p1 = 0, p2 = 0, p3 = -1, p4 = 0;
void h() { p1++; }
void i() { g(); h(); }
void f() { if (p1<3) { h(); f(); } }
int g()  { h(); if (p1<4) i(); return p1; }
int main()
{ int i;
  for(i=0; i<3; i++)
  { if (p1)
    { do
      { f(); }
      while (p2);
    } else {
      if (p3)
      { g();
      } else {
        while (p4)
        { g(); }
        break;
      } /* if */
      f();
    } /* if */
  } /* for */
  return p1;
}
```

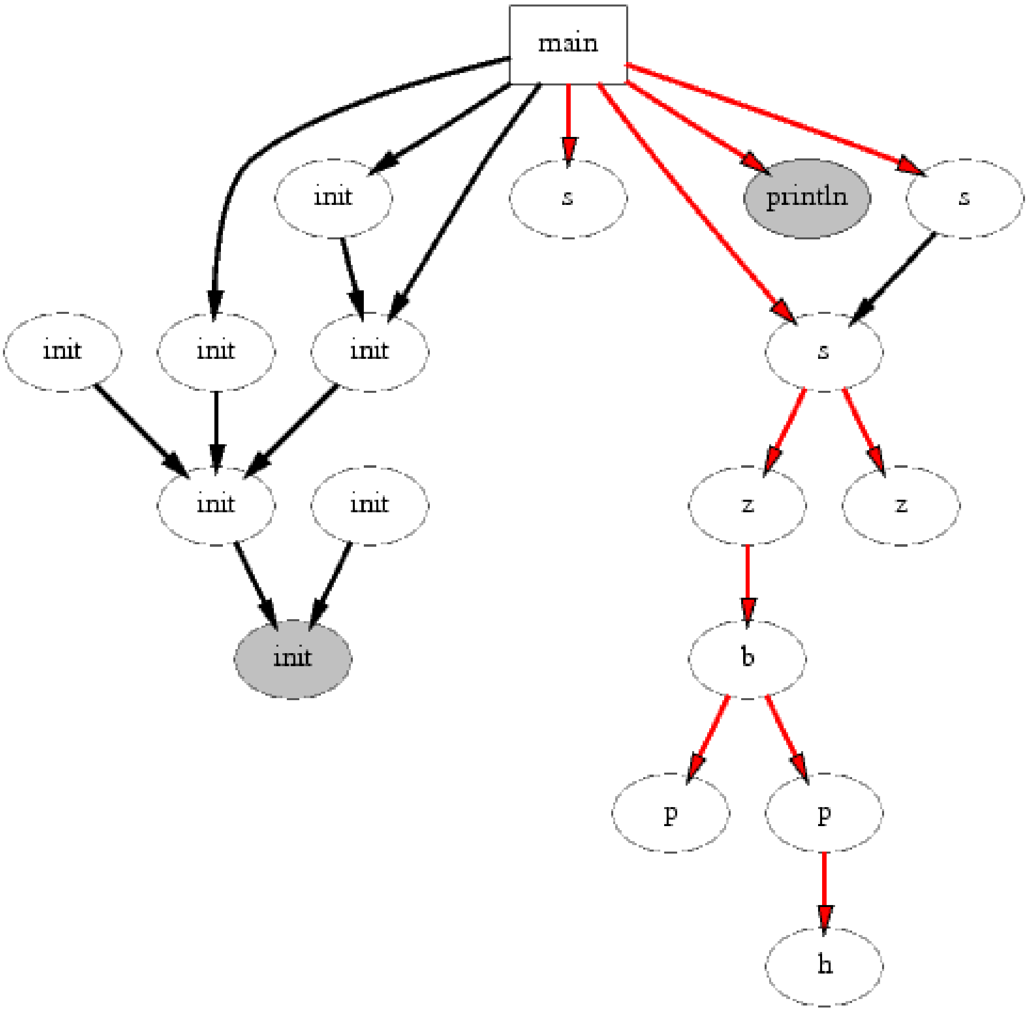
Exercise 4 (Object Oriented Call Graphs)

a) Draw the class hierarchy graph for the following program:

```
package packneu;
abstract class E   {
   int h() { return 9; }
   int z() { return 10; }
   int s() { return z(); }
}

class K extends E   {
   int s() { return 0; }
}

class A extends E   {
   int s() { return super.s() + 1; }
}

class B extends E   {
   int b() { return 2 * p(); }

   int p() { return 3; }
}

class L extends B   {
   int z() { return b(); }
   int p() { return h(); }
}

class Main   {
   public static void main(String[] args)
   {  E e;
      e = new A(); System.out.println(e.s());
      e = new B(); System.out.println(e.s());
      e = new L(); System.out.println(e.s());
   }
}
```

b) Write the call chains that are initiated by the three calls of println. What does the program print?

c) Our Java Bytecode analysis tool constructed the following call graph for this program:



Add the missing class names to the method and constructor nodes.

d) Find edges which could be eliminated by more powerful analysis methods (see Slide C-2.33).