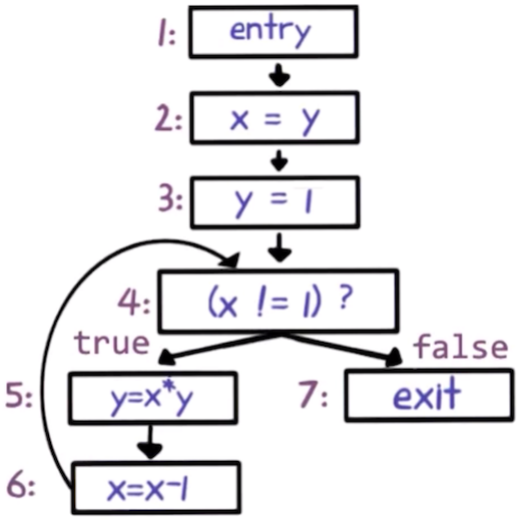
Ingeniería del Software II

Práctica #1 – Análisis Estático

Ejercicio 1

Sea el siguiente control-flow graph para una función:



Ejecutar el algoritmo caótico iterativo para el análisis de Reaching Definitions hasta alcanzar la estabilidad de los conjuntos IN y OUT. Completar la siguiente cuadro con el valor final de los conjuntos IN y OUT:

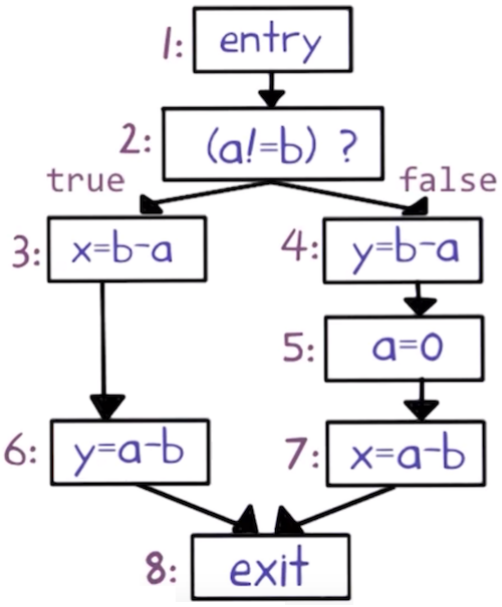
Nodo n	IN[n]	OUT[n]
1	-	$\emptyset$
2	$\emptyset$	$\{\langle x, 2\rangle\}$
3	$\{\langle x, 2\rangle\}$	$\{\langle x, 2\rangle, \langle y, 3\rangle\}$
4		
5		
6		
7		

Ejercicio 2

Sean los conjuntos KILL[n] y GEN[n] para el análisis de Reaching Definitions que matan y generan respectivamente la información de dataflow, completar las ecuaciones de dataflow que caracterizan los conjuntos IN[n] y OUT[n] para dicho análisis.

Ejercicio 3

Sea el siguiente control-flow graph para una función:



Ejecutar el algoritmo caótico iterativo para el análisis de Very Busy Expressions hasta alcanzar la estabilidad de los conjuntos IN y OUT. Completar la siguiente cuadro con el valor final de los conjuntos IN y OUT:

n	IN[n]	OUT[n]
1	-	
2		
3		
4		
5	$\emptyset$	$\{a-b\}$
6	$\{a-b\}$	$\emptyset$
7	$\{a-b\}$	$\emptyset$
8	$\emptyset$	-

Ejercicio 4

Sean los conjuntos KILL[n] y GEN[n] para el análisis de Very Busy Expressions que matan y generan respectivamente la información de dataflow, completar las ecuaciones de dataflow que caracterizan los conjuntos IN[n] y OUT[n] para dicho análisis.

Ejercicio 5

Sea el siguiente programa, donde `MASK`, `IA`, `IQ`, `IR`, `IM` y `AM` son constantes.

```
float foo(int pid) {
1:  int i, j, h;
2:  i = pid ^ MASK;
3:  int k = i / IQ;
4:  h = IA * (i - k * IQ) - IR * k;
5:  h = j ^ MASK;
6:  if (h < 0)
7:    h = h + IM;
8:  float answer = AM * h;
9:  return answer * pid / k;
}
```

a) Construir su control-flow graph.
b) Computar el análisis Live Variables.

Ejercicio 6

Computar los conjuntos IN y OUT para el análisis de Available Expressions.

```
void foo(int[] m) {
1: int a = 3;
2: int i = 0;
3: while (i <= a) {
4:    int t = m[i];
5:    m[i] = t;
6:    i = i + 1;
   }
8: bar(M, a);
}
```

Ejercicio 7

Categorizar los análisis dataflow Live Variables, Reaching Definitions, Very Busy Expressions y Available Expressions.

	Forward	***Backward***
May		
Must		

Ejercicio 8

Sea el siguiente programa:

```
class BinaryTree {
 static class Node {
   int data;
   Node parent, left, right;
 }
 Node root;
 void leftmostInsertNode(int data) {
   if (root == null) {
      Node n1 = new Node(); // allocation site #1
      n1.data = data;
      root = n1;
    } else {
      Node h = root;
      while (h.left != null) {
        h = h.left;
      }
      Node n2 = new Node(); // allocation site #2
      n2.data = data;
      h.left = n2;
      n2.parent = h;
    }
  }
}
```

Construir el points-to graph resultante de aplicar un may-alias análisis insensible a flujo y con abstracción de heap basada en allocation sites.

Ejercicio 9

Completar el programa Datalog que está a continuación llenando las reglas de inferencia para el análisis de Live variables.

- Relaciones de entrada:

```
kill(n:N, v:V)
gen(n:N, v:V)
next(n:N, m:N)
```

- Relaciones de salida:

```
in(n:N, v:V)
out(n:N, v:V)
```