Parseo y Generación de Código – 2do semestre 2016
Licenciatura en Informática con Orientación en Desarrollo de Software
Universidad Nacional de Quilmes

Práctica 9
Análisis de flujo de datos

Ejercicio 1. Dado el siguiente programa:

```
x := 1
y := 0
while x < n {
  if y > x {
      y := x
      x := x + y
  } else {
      x := x + y
      y := x
  }
}
```

1. Compilarlo a código de tres direcciones y construir su grafo de flujo de control.
2. Aproximar las definiciones de alcance (*reaching definitions*) en cada punto del programa usando análisis de flujo de datos.

Ejercicio 2. Exhibir un programa en el que haya una asignación $a := 42$ y una asignación $b := a$ de tal modo que:

- El análisis reporte la instrucción $a := 42$ como una definición de alcance para la instrucción $b := a$.
- No haya ninguna ejecución del programa en la que a valga 42 cuando se ejecuta la asignación $b := a$.

Ejercicio 3. Dado el siguiente programa:

```
x := 1
y := 10
z := 2 * x
while x < y {
    x := x + 2
}
z := 1
y := 2 * z
while z < x {
    z := z + 3
}
```

1. Compilarlo a código de tres direcciones y construir su grafo de flujo de control.

2. Aproximar las variables vivas (*live variables*) en cada punto del programa usando análisis de flujo de datos.

Ejercicio 4. Considerar el siguiente programa:

```
x := 0
.start:
y := 2 * x
x := y + 1
•
jumpIf≤ y x .start
z := y
```

En el punto del programa marcado con •: ¿qué variables están vivas? ¿cuál es el conjunto de variables vivas que reporta el análisis?

Ejercicio 5. En el siguiente programa:

```
for i = 1 to n
    v[i] := 2 * v[i] + v[i + 1]
end
```

la variable `v` representa un arreglo de enteros de 64 bits, de modo que para acceder al i-ésimo elemento en el código objeto se debe multiplicar el subíndice i por un factor (por ejemplo, un factor de 8 bytes si suponemos direccionamiento a byte). El código de arriba se compiló al código de tres direcciones siguiente:

```
i := 1
start:
jumpIf> i n end
t1 := 8 * i
t2 := v[t1]
t3 := 2 * t2
t4 := i + 1
t5 := 8 * t4
t6 := v[t5]
t7 := t3 + t6
t8 := 8 * i
v[t8] := t7
i := i + 1
jump start
end:
```

Analizar las expresiones disponibles en cada punto del programa, y determinar si alguna asignación puede simplificarse para utilizar una variable que contenga la correspondiente expresión disponible, ahorrando algún cómputo.

Ejercicio 6. En un punto del programa decimos que una variable x es **nula** si en todas las ejecuciones del programa que llegan a ese punto podemos asegurar que el valor de x es 0. Se puede asegurar que el valor de x es 0 en cualquiera de los siguientes casos:

- La última escritura sobre x es de la forma $x := a$, donde a es la constante 0 o una variable nula.
- La última escritura sobre x es de la forma $x := a + b$, donde a es 0 o una variable nula, y también b es 0 o una variable nula.
- La última escritura sobre x es de la forma $x := a * b$, donde a es 0 o una variable nula, o alternativamente b es 0 o una variable nula.

El problema de determinar las variables nulas se puede aproximar haciendo análisis de flujo de datos.

1. ¿Corresponde aplicar un análisis **may** o **must**? ¿Corresponde aplicar un análisis **forward** o **backward**?
2. ¿Cuáles son los conjuntos *gen* y *kill* para una instrucción de asignación $x := E$?
   Nota: en este ejercicio los conjuntos *gen* y *kill* pueden depender de los valores de los conjunto *in* y *out*, de manera que puede ser necesario recalcular sus valores a lo largo del análisis.
3. Usando el análisis propuesto, calcular las variables nulas en cada punto del siguiente programa:

```
z := 0
x := 0
i := 0
.start:
a := i * z
x := x + a
i := i + 1
jumpIf≥ i n .start
```

Ejercicio 7. En el ejercicio anterior se diseñó un nuevo análisis para calcular las variables nulas. En realidad no es necesario diseñar un nuevo análisis: el conjunto de variables nulas se puede aproximar conociendo el conjunto de definiciones de alcance. Explicar cómo se podría utilizar el resultado del análisis de definiciones de alcance para calcular el conjunto de variables nulas.