Ingeniería del Software II

Parcial #2 – Análisis Estático de Programas

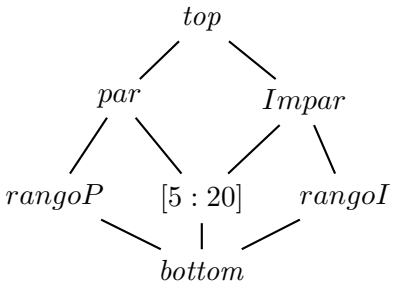
Ej1	Ej2	Ej3	Ej4	Nota

El exámen es a **libro abierto**. Cada ejercicio se evaluará como **Bien**, **Regular** o **Mal**. Para aprobar el examen es necesario tener al menos 2 ejercicios Bien y al menos 1 ejercicio Regular.

Bien = 2,5p, Regular = 1,25p, Mal = 0p.

Ejercicio 1

Sea R el siguiente reticulado que abstrae el valor de variables naturales en el contexto de un análisis dataflow. $[a : b]$ indica que el valor de la variable está en el rango entre a y b.



Se desea definir las operaciónes "+1" y "*2" de manera que modelen de la forma más precisa posible la funciones *sucesor*, y *duplicar* a los elementos del reticulado.

op1	**op1** $+1$
bottom	
$[5:20]$	
rangoP	
rangoI	
par	
Impar	
top	

op1	**op1** $*2$
bottom	
$[5:20]$	
rangoP	
rangoI	
par	
Impar	
top	

Ejercicio 2

Sea el reticulado del ejercicio 1 y las siguientes funciones de *transferencia* para un análisis *dataflow may*:

- "`z = k`" (con k constante):

$$OUT[n](z) = \begin{cases} par & \text{si } par(k) \wedge !enRango(k) \\ Impar & \text{si } Impar(k) \wedge !enRango(k) \\ rangoP & \text{si } par(k) \wedge enRango(k) \\ rangoI & \text{sino} \end{cases}$$

- "`z = t shift k`" (con $k \in \{1, 2\}$):

$$OUT[n](z) = \begin{cases} bottom & \text{si IN[n](t) = bottom} \\ [x:y] & \text{si } IN[n](t) = rangoP \\ rangoI & \text{si } (k = 1 \wedge IN[n](t) = [x:y]) \vee (k = 2 \wedge IN[n](t) = rangoP) \\ par & \text{si } (k = 1 \wedge IN[n](t) = rangoI) \vee (k = 2 \wedge IN[n](t) = [x:y]) \\ Impar & \text{si } (k = 1 \wedge IN[n](t) = par) \vee (k = 2 \wedge IN[n](t) = rangoI) \\ top & \text{sino} \end{cases}$$

Dado el siguiente programa:

```
foo(){
  z = 4
  z = z shift 2
  t = 1
  if (z >= 0) {
    t = t shift 1
  } else {
    z = z shift 1
  }
  return z
}
```

a. Construir el CFG del programa.

b. Utilizando el análisis propuesto, calcular el valor abstracto de las variables para los conjuntos IN y OUT de cada nodo en el CFG.

Ejercicio 3

Dado el siguiente programa P calcular el *points-to graph* usando un algoritmo no sensitivo a flujo:

```
int main(int choice) {
  var a,b,c,d;
  a = new Object();
  b = new Object();
  c = new Object();
  d = a;
  if (choice == 1) {
    d = b;
  } else {
    d = c;
  }
}
```

Ejercicio 4

Sea el sigiuente programa en `Java`:

```
//@ requires x > 0;
//@ ensures: \result == x * 2;
int duplicate(int x) {
  return x + 1;
}
```

Nota: La instrucción `return E`, asigna la expresión `E` a la variable de especificación `result`.

a. Calcular la *Weakest Precondition* de P

b. Demostrar que el programa es incorrecto usando el resultado del item a