DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Ingeniería de software II

Trabajo Práctico N°2

Integrante	LU	Correo electrónico
Esteban Luciano Rey	657/10	estebanlucianorey@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

Índice

1. Consideraciones

El presente trabajo fue realizado con los resultados obtenidos de la herramienta SOOT 3.0.1 en un Windows 10 con JRE 8. TOdas las funciones de los ejercicios fueron incluidos en un mismo archivo A.java y luego de ejecutar la herramienta se separaron en archivos diferentes segun el ejercicio al cual correspondia la función analizada.

2. Respuestas

2.1. Implementación

merge

Dado que el analisis de division por cero es del tipo MAY, los conjuntos deben unirse. En el caso de que alguna variable este definida en los 2 caminos, se aplica la función supremo para decidir cual queda

add

Los únicos casos en los que podemos saber en la suma si el resultado es o no cero es cuando tenemos certezas de ambos operandos, cualquier otro caso nos da un MAYBE_ZERO

divideBy

La unica forma de que una division de cero es que el dividendo sea cero y el divisor distinto a cero. Asi como la unica forma de estar seguros de que no es cero es que ambos operandos no sean ceros.

multiplyBy

Si alguno de los operandos es cero entonces da cero, si ninguno lo es entonces el resultado tampoco. Todo otro caso no se puede determinar

substract

Solo puedo asegurar que da cero la resta entre ceros. Asi como solo puedo saber que no es cero cuando uno de los operandos lo es y el otro no

2.2. Ejercicio 1

```
public int ejercicio1(int, int)
{
    int i1, i2, $i3, i4;
    A r0;

    r0 := @this: A;

    i1 := @parameter0: int;

    i2 := @parameter1: int;

    $i3 = 0 * i2;

    i4 = i1 / $i3;
```

```
/*Possible division by zero here*/

        return i4;
    }
```

Luego de determinar que i3 (es decir $(x * n)$) es cero luego de la multiplicación, la regla detecta y marca como division por cero en la linea de asignación para i4 (variable j)

2.3. Ejercicio 2

```
    public int ejercicio2(int, int)
    {
        int i0, i1, i2, i4;
        A r0;

        r0 := @this: A;

        i2 := @parameter0: int;

        i0 := @parameter1: int;

        i1 = i0 - i0;

        i4 = i2 / i1;
/*Possible division by zero here*/

        return i4;
    }

Los parametros m y n se evaluan con el valor abstracto bottom, con lo cual en la primera
    resta se le adjudica a x un valor MAYBE, eso hace que la division en el calculo de
    la variable j (i4 en el formato jimple) presente una posible division por cero
```

2.4. Ejercicio 3

```
        public int ejercicio3(int, int)
    {
        int i0, i1, i2, i3;
        A r0;

        r0 := @this: A;

        i0 := @parameter0: int;

        i1 := @parameter1: int;

        if i0 == 0 goto label1;

        i3 = i0;

        goto label2;
```

```
    label1:
       i3 = 1;

    label2:
       i2 = i1 / i3;
/*Possible division by zero here*/

       return i2;
   }
```

En esta ocasión el analisis presenta un falso negativo, ya que de hacer un seguimiento de código se puede ver que nunca se puede producir una division por cero ya que la ejecución del if implica que x siempre es un valor numérico distinto a 0. El error surge de suponer el valor abstracto de m como un bottom y asi suponer que x, en el merge al final del if es un MAYBE, dando una posible division por cero.

2.5. Ejercicio 4

```
   public int ejercicio4(int, int)
   {
       int i0, i1, i2;
       A r0;

       r0 := @this: A;

       i0 := @parameter0: int;

       i1 := @parameter1: int;

       i2 = i0 / i1;
/*Possible division by zero here*/

       return i2;
   }
```

Se detecta una división por cero ya que los valores abstractos de los parametros de la función nunca dejan de ser bottom, y division entre bottom da MAYBE.