Ingeniería del Software II

Taller #2 – Implementando Análisis de Dataflow

Deadline: 1 de octubre a las 23:59 hs

Un `zero-analysis` es un *may forward dataflow analysis* cuyo objetivo es detectar si una variable puede ser (o no) cero durante la ejecución de un programa. El análisis utiliza un conjunto de tuplas para determinar el valor abstracto de cada variable. Si el conjunto no tiene tuplas definidas para una variable, significa que se desconoce su valor abstracto. Las tuplas tienen la siguiente forma:

$\{\langle x, \text{ZERO}\rangle\}$ representa que la variable x vale cero.

$\{\langle x, \text{NOT\_ZERO}\rangle\}$ representa que la variable x no vale cero.

$\{\langle x, \text{ZERO}\rangle, \langle x, \text{NOT\_ZERO}\rangle\}$ representa que la variable x puede (o no) valer cero. En la implementación lo representamos con una única tupla $\langle x, \text{MAYBE\_ZERO}\rangle$.

La función de transferencia debe soportar las siguientes operaciones:

- `x = constant;`
- `x = y;`
- `x = y + z;`
- `x = y - z;`
- `x = y * z;`
- `x = y / z;`

Para este taller se pide implementar un dataflow analysis para detectar si hay una división por cero. Para este objetivo tendrán que completar una implementación ya existente que deberán bajar de la página de la materia. Deberán completar los métodos del `enum ZeroAbstractValue`; los métodos `visitDivExpression` y `visitIntegerConstant` de la clase `ZeroValueVisitor`; el método `merge` de la clase `DivisionByZeroAnalysis` y el método `union` de la clase `ZeroAbstractSet`.

Realizar el zero-analysis para los siguientes programas Java:

```
public int ejercicio1(int m, int n) {
#1: int x = 0;
#2: int j = m / (x * n);
#3: return j;
}
```

```
public int ejercicio2(int m, int n) {
#1: int x = n - n;
#2: int i = x + m;
#3: int j = m / x;
#4: return j;
}
```

```
public int ejercicio3(int m, int n) {
#1: int x = 0;
#2: if (m != 0){
#3:     x = m;
#4: }else{
#5:     x = 1;
#6: }
#7: int j = n / x;
#8: return j;
}

public int ejercicio4(int m, int n) {
#1: int x = 0;
#2: int j = m / n;
#3: return j;
}
```

Manual de Usuario

En la página de la materia, puede bajarse un proyecto Maven que posee las siguientes carpetas: **examples, sootOutput, target, utils, zero-analysis**.

- **examples** contiene las clases a ser analizadas.
- **sootOutput** es la carpeta donde se colocaran los outputs de soot.
- **target** es la carpeta donde maven colocará los jars provenientes del comando `mvn install`.
- **utils** contiene la implementación del patrón de diseño visitor para visitar los distintos **statement** del programa.
- **zero-analysis** es la carpeta donde está el código a modificar.
    - **Launcher:** Es el entry point del analizador.
    - **DivisionByZeroAnalysis:** es la clase que implementa el zero-analysis (extends `ForwardFlowAnalysis` de Soot).
    - **ZeroValueVisitor:** contiene la implementación del patrón de diseño visitor para visitar las distintas **expresiones** del programa (extends `AbstractValueVisitor` de utils).
    - **ZeroAbstractSet:** es un conjunto de tuplas cuya primer componente es el nombres de variable y la segunda componente es un `ZeroAbstractValue`.
    - **ZeroAbstractValue:** implementa los valores abstractos mencionados anteriormente.

Para compilar y generar el jar que utilizaremos para el análisis deberán correr el comando `mvn install` desde la carpeta `soot-dataflow-analysis`. Es necesario tener instalado Maven (confío en que podrán instalar Maven por su cuenta).

Una vez adquirido el jar, utilizar el siguiente comando que utiliza el jar generado por Maven (zero-analysis-1.0-SNAPSHOT-jar-with-dependecies.jar) para correr el análisis sobre la clase A (que contiene los ejercicios que están arriba).

```
java -jar
zero-analysis/target/zero-analysis-1.0-SNAPSHOT-jar-with-dependencies.jar
-cp .:./examples/:$JRE  -f J A -v -print-tags  -p jtp.DivisionByZeroAnalysis on
```

Recuerden que para que este comando funcionen deben tener bien seteada la $JRE. Ver taller 1.

Formato de Entrega

El taller debe ser subido al campus. Debe ser un archivo zip con el siguiente contenido.

1. Un archivo `src.zip` con el código implementado correctamente comentado.
2. Los outputs por cada método (ejercicio1-4).
3. Un archivo `report.pdf` con la descripción de la resolución, incluyendo una breve explicación de los resultados obtenidos por el analizador. Si la forma de ejecutarse es distinta a la presentada en la práctica, explicar los pasos a seguir para ejecutar el código.