

Ingeniería del Software II

Taller #2 – Implementando Análisis de Dataflow

Deadline: Jueves 29 de abril 23:59 hs

Introducción

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.

Por lo tanto, para un conjunto $IN[n]$ de un nodo n del control-flow graph, podemos interpretar el valor abstracto de una variable x del siguiente modo:

- $IN[n](x) = \perp$ (indefinido) si el $IN[n]$ no contiene tuplas con x .
- $IN[n](x) = Z$ (zero) si $IN[n]$ únicamente contiene una sola tupla de x y es la tupla $\langle x, \text{ZERO} \rangle$.
- $IN[n](x) = NZ$ (not_zero) si $IN[n]$ únicamente contiene sola tupla de x y es la tupla $\langle x, \text{NOT_ZERO} \rangle$.
- $IN[n](x) = MZ$ (maybe_zero) si $IN[n]$ contiene simultáneamente las tuplas $\langle x, \text{ZERO} \rangle$ y $\langle x, \text{NOT_ZERO} \rangle$.

Del mismo modo podemos interpretar los valores posibles de x para un conjunto $OUT[n]$.

Las ecuaciones de dataflow que caracterizan este análisis son las siguientes

$$IN[n] = \bigcup_{n' \in \text{pred}(n)} OUT[n']$$

$$OUT[n] = \text{Transfer}[n, IN[n]]$$

Ejercicios

Parte 1: Definiendo el Zero Analysis

Ejercicio 1

Completar la siguiente tabla con los valores esperados del conjunto $OUT[n]$ para la variable x cuando n es la asignación de una constante, con $K \in \mathbb{Z} - \{0\}$:

n	$OUT[n](x)$
$x = 0$	
$x = K$ // con K distinto de 0	

Ejercicio 2

Completar la siguiente tabla para caracterizar la función de transferencia (*Transfer*) para la sentencia de copia ($x = y$):

IN[n](y)	OUT[n](x)
\perp	
<i>Z</i>	
<i>NZ</i>	
<i>MZ</i>	

Ejercicio 3

Completar la siguiente tabla para caracterizar la función de transferencia (*Transfer*) cuando se calcula la información de dataflow para una sentencia de suma ($x = y + z$):

IN[n](y)	IN[n](z)	OUT[n](x)
\perp	\perp	
<i>Z</i>	\perp	
<i>NZ</i>	\perp	
<i>MZ</i>	\perp	
\perp	<i>Z</i>	
<i>Z</i>	<i>Z</i>	
<i>NZ</i>	<i>Z</i>	
<i>MZ</i>	<i>Z</i>	
\perp	<i>NZ</i>	
<i>Z</i>	<i>NZ</i>	
<i>NZ</i>	<i>NZ</i>	
<i>MZ</i>	<i>NZ</i>	
\perp	<i>MZ</i>	
<i>Z</i>	<i>MZ</i>	
<i>NZ</i>	<i>MZ</i>	
<i>MZ</i>	<i>MZ</i>	

Ejercicio 4

Completar la siguiente tabla para caracterizar la función de transferencia (*Transfer*) cuando se calcula la información de dataflow para una sentencia de resta ($x = y - z$):

IN[n](y)	IN[n](z)	OUT[n](x)
\perp	\perp	
Z	\perp	
NZ	\perp	
MZ	\perp	
\perp	Z	
Z	Z	
NZ	Z	
MZ	Z	
\perp	NZ	
Z	NZ	
NZ	NZ	
MZ	NZ	
\perp	MZ	
Z	MZ	
NZ	MZ	
MZ	MZ	

Ejercicio 5

Completar la siguiente tabla para caracterizar la función de transferencia (*Transfer*) cuando se calcula la información de dataflow para una sentencia de multiplicación ($x = y * z$):

IN[n](y)	IN[n](z)	OUT[n](x)
\perp	\perp	
Z	\perp	
NZ	\perp	
MZ	\perp	
\perp	Z	
Z	Z	
NZ	Z	
MZ	Z	
\perp	NZ	
Z	NZ	
NZ	NZ	
MZ	NZ	
\perp	MZ	
Z	MZ	
NZ	MZ	
MZ	MZ	

Ejercicio 6

Completar la siguiente tabla para caracterizar la función de transferencia (*Transfer*) cuando se calcula la información de dataflow para una sentencia de división **entera** ($x = y / z$):

IN[n](y)	IN[n](z)	OUT[n](x)
\perp	\perp	
Z	\perp	
NZ	\perp	
MZ	\perp	
\perp	Z	
Z	Z	
NZ	Z	
MZ	Z	
\perp	NZ	
Z	NZ	
NZ	NZ	
MZ	NZ	
\perp	MZ	
Z	MZ	
NZ	MZ	
MZ	MZ	

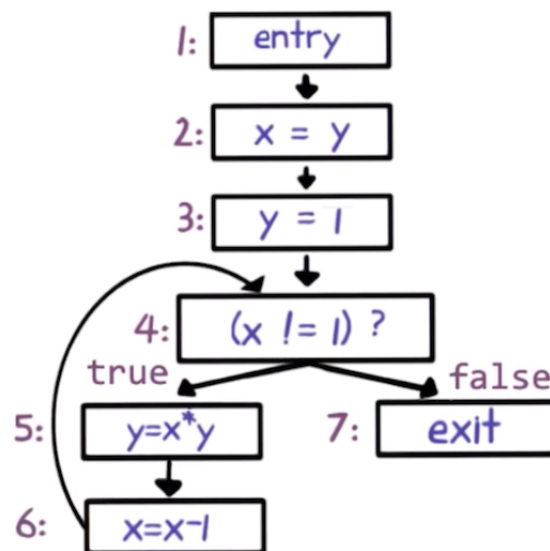
Ejercicio 7

Sea el siguiente programa y su correspondiente control-flow graph:

```

1 public int productoria(int y) {
2   int x = y;
3   y = 1;
4   while (x != 1) {
5     y = x * y;
6     x = x - 1;
7   }
8   return y;
9 }

```



Ejecutar el algoritmo caótico iterativo hasta obtener valores estables de *IN* y *OUT* para toda variable del programa. **Nótese que el valor inicial de los parámetros es siempre MZ ya que no sabemos que valor pueden tomar al ejecutarse el programa.**

n	IN[n](x)	IN[n](y)	OUT[n](x)	OUT[n](y)
1	\perp	MZ		
2				
3				
4				
5				
6				
7				

Parte 2: Implementando el Zero Analysis en SOOT

Para este taller se pide implementar un dataflow analysis que infiere información sobre el valor de una variable entera con el fin de 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.

La implementación debe soportar (al menos) las siguientes operaciones:

- `x = 0;`
- `x = K;` // donde K es una constante distinta de cero
- `x = y;`
- `x = y + z;`
- `x = y - z;`
- `x = y * z;`
- `x = y / z;` // división entera

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`.

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-dependencies.jar`) para correr el análisis sobre la clase `ZeroAnalysisTest1` (repetir para `ZeroAnalysisTest2` hasta `ZeroAnalysisTest7`).

```
java -jar
zero-analysis/target/zero-analysis-1.0-SNAPSHOT-jar-with-dependencies.jar
-cp ../examples/:$JRE -f J ZeroAnalysisTest1
-v -print-tags -p dataflow.DivisionByZeroAnalysis on
```

Recordar que para que este comando funcione se debe tener bien seteada la `$JRE`. Ver Taller #1.

El análisis implementado debe calcular correctamente la información de dataflow como se muestra a continuación, indicando los posibles errores de división por cero si los hubiere:

```
1  public static int test1(int m, int n) {
2      int x = 0;
3      int k = x * n;
4      int j = m / k;
5      return j;          // IN(x)=IN(k)=Z, IN(m)=IN(n)=MZ, IN(j)=Undef
6  }

1  public static int test2(int m, int n) {
2      int x = n - n;
3      int i = x + m;
4      int j = m / x;
5      return j;          // IN(m)=IN(n)=IN(x)=IN(i)=MZ, IN(j)=MZ
6  }

1  public static int test3(int m, int n) {
2      int x = 0;
3      int j = m / n;
4      return j;          // IN(m)=IN(n)=MZ, IN(x)=Z, IN(j)=MZ
5  }

1  public static int test4(int m, int n) {
2      int x = 0;
3      if (m != 0) {
4          x = m;
5      } else {
6          x = 1;
7      }
8      int j = n / x;
9      return j;          // IN(m)=IN(n)=IN(x)=MZ, IN(j)=MZ
10 }

1  public static int test5(int y) {
2      int x = y;
3      y = 1;
```

```
4      while (x != 1) {
5          y = x * y;
6          x = x - 1;
7      }
8      return y;          // IN(x)=IN(y)=MZ
9  }
```



```
1  public static int test6(int x) {
2      int y;
3      if (x == 0) {
4          y = 1;
5      } else {
6          y = 2;
7      }
8      int r = x / y;
9      return r;          // IN(x)=IN(r)=MZ, IN(y)=NZ
10 }
```

```
1  public static int test7() {
2      int i = 0, j = 1;
3      int d = j / i;
4      if (d > 0) {
5          d = 1;
6      }
7      return d;          // IN(i)=Z, IN(j)=NZ, IN(d)=NZ
8  }
```

Formato de Entrega

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

1. Un archivo `answers.pdf` con las respuestas a los ejercicios.
2. Un archivo `src.zip` con la implementación del zero analysis correctamente comentado.
3. Un archivo con el output producido por el análisis por cada test.