Manual Taller #1: SOOT

El objetivo de este documento es presentar el modo de utilización de la herramienta SOOT.
La herramienta SOOT contiene un conjunto de algoritmos ya implementados para ***dataflow analysis***.

Todos los pasos están pensados para una máquina Linux. Tener en cuenta que si se tiene Windows o IOS los comandos pueden cambiar, pero los pasos a seguir deberían ser los mismos.

A continuación, explicaremos cómo configurar SOOT para luego ver cómo utilizarlo.

Requisitos Java

● Instalar JAVA 8:
https://www.oracle.com/ar/java/technologies/javase/javase8-archive-downloads.html

● Setear ***JRE*** a la dirección donde está el rt.jar. En mi caso es con el comando:

```
export JRE=/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/rt.jar
```

● Setear ***JAVA_HOME***. En mi caso es con el comando:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
```

● Setear ***PATH*** . En mi caso es con el comando:

```
export PATH=/usr/lib/jvm/java-8-openjdk-amd64/bin/:$PATH
```

Nota: Realizar esto cada vez que se abra una terminal nueva ya que la variable JRE,JAVA_HOME y PATH son temporales para la sesión actual.

Configurando SOOT

● Bajar el .jar de SOOT desde este link:
https://drive.google.com/file/d/1bL-bU5g8O5Npfs1yMjH4lSa_j7uX-77n/view

● Colocar en una carpeta que se llame "soot" el archivo .jar bajado.

Compilando una clase Java

- Crear una carpeta “bin”.
- Crear una carpeta “src”.
- Copiar el siguiente archivo Foo.java en la carpeta “src”.

```
class Foo {
        public static void main(String[] args) {
                Foo f = new Foo();
                int rv = f.bar(0);
                System.out.println(rv);
        }
```

```
        public int bar(int x) {
                int c;
                if (x == 0) {
                        c = x;
                } else {
                        c = x + 1;
                }
                return x;
        }
}
```

- Ejecutar el siguiente comando para compilar Foo.java y colocar la clase compilada en la carpeta “bin”.

```
$javac -g -d bin src/Foo.java
```

- La ejecución correcta del comando no producirá ninguna salida por consola y se creará el archivo bin/Foo.class
- Para ejecutar el programa Java debemos invocar a la máquina virtual de Java utilizando el comando “java” e indicando donde está el programa compilado y cuál es su clase con el punto de entrada (i.e. la función main).

```
$java -classpath bin Foo
```

- La ejecución de la función main de la clase Foo debería imprimir por consola el siguiente resultado:

```
The value is 0.
```

Utilizando SOOT

Supongamos que queremos realizar un Reaching Definition Tagger sobre una clase.
Para esto, lo primero es compilar el archivo java.
Consideremos entonces que tenemos el archivo **Foo.java** en la carpeta soot creada.
Vamos hasta la ubicación de la carpeta “soot” y compilamos el archivo **Foo.java** utilizando javac

```
$javac -d bin -g src/Foo.java
```

Esto creará el archivo bin/**Foo.class**
Ahora que tenemos el código compilado debemos ejecutar el análisis con el siguiente comando

```
java -classpath .:soot-3.3.0-jar-with-dependencies.jar:. soot.Main -cp bin:$JRE -f J
 Foo -print-tags -p jap.rdtagger enabled:true -p jb use-original-names:true -p jb.cp
                            off -keep-line-number
```

Nota: en Windows es necesario reemplazar los dos puntos “:” que separan los .jar en el classpath por punto y coma “;”.

¿Qué hace este comando?

- ***-classpath*** setea el classpath que contiene la aplicación SOOT.

- ***-cp*** setea el classpath que contiene la aplicación que será analizada (en este caso, “bin” ya que allí se encuentra Foo.class). Además, debemos indicar la ruta del rt.jar para el correcto funcionamiento de SOOT.
- ***-f*** establece el formato del output de SOOT. En este caso le pasamos el parámetro J para que genere un archivo en formato “Jimple” (formato recomendado por la cátedra).
- **Foo** es el nombre de la clase que SOOT analizará.
- ***-print-tags*** le indica a SOOT que imprima en pantalla los tags luego de la línea
- ***-p*** indica que tipo de análisis se quiere realizar. Un manual con todos los tipos de análisis posibles puede verse aquí.
- En nuestro ejemplo utilizamos ***jap.rdtagger,*** este le pede a soot que realice un reaching definition tagger.
- El resto de los flags: ***use-original-names:true*** y ***jb.cp off***, deben mantenerse porque así es como mejor se comporta SOOT.

Entendiendo el output de SOOT

La herramienta debería crear un directorio *“sootOutput”* donde podrán encontrar los archivos en formato Jimple. Dependiendo del análisis que se corra se tendrá un output distinto, pero explicaré levemente el output de un reaching definition tagger.

return x;
```
/*16*/
/*x has reaching def: x := @parameter0: int*/
```

Luego de la instrucción “return x” de la línea ***16*** se tiene que la variable ***x*** usa la definición que fuera pasada por argumento en la función.

Cada análisis tendrá su nomenclatura.
Podrán darse cuenta solos que es lo que el análisis está diciendo.

Otros Análisis de Soot

SOOT provee los siguientes análisis predefinidos:

- Null Pointer Checker (jap.npc)
- Null Pointer Colourer (jap.npcolorer)
- Array Bound Checker (jap.abc)
- Profiling Generator (jap.profiling)
- Side Effect tagger (jap.sea)
- Field Read/Write Tagger (jap.fieldrw)
- Call Graph Tagger (jap.cgtagger)
- Parity Tagger (jap.parity)
- Parameter Alias Tagger (jap.pat)
- Live Variables Tagger (jap.lvtagger)
- Reaching Defs Tagger (jap.rdtagger)
- Cast Elimination Check Tagger (jap.che)
- Unreachable Method Transformer (jap.umt)
- Loop Invariant Tagger (jap.lit)
- Available Expressions Tagger (jap.aet)
- Dominators Tagger (jap.dmt)