Ingeniería del Software II

Taller #6 – Análisis de Dataflow

LEER EL ENUNCIADO COMPLETO ANTES DE ARRANCAR.

Fecha de entrega: 30 de Mayo de 2024
Fecha de re-entrega: 13 de Junio de 2024 (no hay extensiones)

Soot

La herramienta Soort contiene un conjunto de algoritmos ya implementados para *Dataflow Analysis*. Puede encontrar más información acerca de su funcionamiento, opciones y documentación en los siguientes links:

- Wiki oficial: `https://github.com/soot-oss/soot/wiki`
- Documentación de la API Java:
  `https://soot-oss.github.io/soot/docs/4.4.1/jdoc/index.html`
- Documentación con las opciones para usar la herramienta por línea de comando:
  `https://soot-oss.github.io/soot/docs/4.4.1/options/soot_options.html`

Requisitos

Para poder correr esta herramienta es necesario tener instalado Java 8. En Ubuntu, puden correr el siguiente comando para instalarlo: `sudo apt-get install openjdk-8-jdk`. Para Windows, descargar el instalador de Java SE Runtime Environment 8 desde `https://www.oracle.com/ar/java/technologies/javase/javase8-archive-downloads.html` y seguir las instrucciones de instalación.

Utilizando Soot

Para este taller, hemos preparado una tarea de Gradle que permite ejecutar Soot. Para correr esta tarea, debe ejecutar el siguiente comando desde la carpeta raíz del taller:

```
./gradlew soot -PtargetClass=<targetClass> -Panalysis=<sootAnalysis>
```

Donde "`<targetClass>`" es la clase objetivo sobre la cual se desea correr Soot, y "`<sootAnalysis>`" es el análisis de Soot que se desea correr.

Es importante mencionar que Soot puede ejecutarse también por línea de comando de manera similar a como lo hace la tarea Gradle internamente. Algunos de los argumentos de Soot que se usan en la implementación del taller son:

- El primer argumento es el nombre completo (i.e., incluyendo el paquete) de la clase Java que se desea analizar. Por ejemplo, `com.example.Bar`.
- `-cp` setea el classpath que contiene la aplicación que será analizada. Por ejemplo, para el taller será el JAR con las clases a analizar. Además, debemos indicar la ruta del `rt.jar` (provisto por nuestra instalación de Java JRE 8) 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*.
- `-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.

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. Por ejemplo, las siguientes líneas corresponden a un análisis de "*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.

Otros Análisis de Soot

Soot provee los siguientes análisis predefinidos. Un manual con todos los tipos de análisis posibles puede verse aquí: `https://www.sable.mcgill.ca/soot/tutorial/phase/phase.html`.

- 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)

Ejercicio 1

Dado el siguiente programa Java:

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

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

Ejecutar un análisis de Soot usando el *Reaching Defs Tagger* (`jap.rdtagger`) y responder qué definiciones son usadas en las siguientes sentencias:

a) `return c;`
b) `System.out.println(rv);`

Explique y justifique el output de la herramienta para las sentencias pedidas.

Ejercicio 2

Dado el siguiente programa Java:

```
public class ReachingDefinitionsExample {
  ReachingDefinitionsExample() {}

  int doSomething(String[] args) {
      int a = 8; int c = 3;
      int len=args.length;
      if(len > 2) {
          a = 5;
      }
      c = 1;
      while (!(c > a)) {
          c = c + 2;
      }
      a = c - a;
      return a;
  }
}
```

Ejecutar un análisis de Soot usando el *Reaching Defs Tagger* (`jap.rdtagger`) y responder qué definiciones son usadas en las siguientes sentencias:

a) `a = c - a;`
b) `return a;`

Explique y justifique el output de la herramienta para las sentencias pedidas.

Ejercicio 3

Dado el siguiente programa Java:

```
public class LiveVariablesExample {
  LiveVariablesExample() {}

  int doSomething(int a, int b) {
      int c = a + b;
      int d = a - b;
      int r;
      if (a < b) {
          r = c;
      } else {
          r = d;
      }
      return r;
  }
}
```

Ejecutar el análisis Soot usando el *Live Variables Tagger* (`jap.lvtagger`) y completar qué variables reporta Soot que siguen vivas luego de la ejecución de las siguientes sentencias. Completar con **SI** si la variable está viva y con **NO** si la variable no está via.

Sentencia	a	b	c	d	r
`d = a - b;`					
`r = c;`					
`r = d;`					
`return r;`					

Formato de Entrega

El taller debe ser entregado en el campus de la materia durante la fecha de entrega indicada en el mismo campus. La entrega debe incluir el siguiente material:

1. Un archivo `readme.txt` con las respuestas para cada ejercicio que se pide en el enunciado. Incluya instrucciones sobre como obtener los resultados del taller (i.e., qué comandos se ejecutaron y qué archivos se analizaron).