Ingenieria del Software II

Segundo Cuatrimestre de 2020

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Taller 1

SOOT

Integrantes	LU	Correo electrónico
Petri, Javier	306/15	javierpetri2012@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Ejercicio 1

Para corroborar el resultado del análisis, Reaching Definition Analysis, del analizador **SOOT**, realizo el **Grafo de Control de Flujo** para el programa del Ejercicio 1.

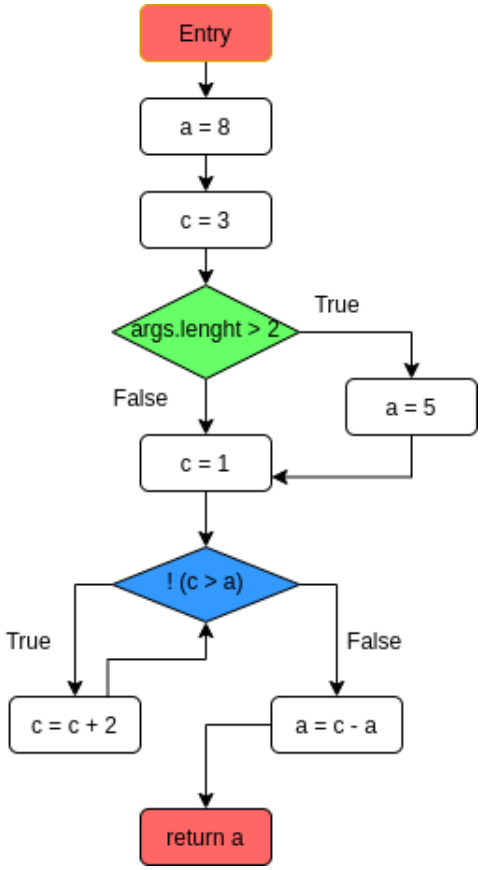

Figura 1: Grafo de Control de Flujo del Ejercicio 1

1.1. ¿Qué definiciones de las variables a y c alcanzan la línea 5?

Como vimos en la teórica, en la definición del análisis **Reaching Definition**, el conjunto IN y OUT para cada nodo del Grafo de Control de Flujo se define de la siguiente manera:

$$IN[n] = \bigcup OUT[n'] \tag{1}$$

Donde n' son los predecesores de n,

$$OUT[n] = (IN[n] - KILL[n]) \cup GEN[n] \tag{2}$$

Por lo tanto, las definiciones de las variables **a** y **c** en la linea 5 del código (**c = 1**) tienen el siguiente estado:

IN	OUT
<a,1>,<c,1>,<a,3>	<a,1>,<c,5>,<a,3>

Se puede ver que en el conjunto OUT la definición <**c,1**> se reemplaza por <**c,5**>, ya que es en esta linea, la 5, que se redefine la variable **c**, reemplazando la definición de **c** en la linea 1. Es por este "*reemplazo*" y por el hecho de que la información se propaga desde el "*entry node*" hacia adelante, que se llama a este tipo de análisis, **"forward"**.
También es importante destacar que se mantienen dos posibles definiciones para la variable **a**. Esta situación refleja el hecho de que el código del ejercicio tiene un **if** en la linea 2, tal que si se cumple la condición del mismo, se redefine **a** en la linea 3. Es por este comportamiento, de mantener las posibles definiciones, que también se caracteriza a este análisis como **"may"**.

1.2. ¿Qué definiciones de las variables a y c alcanzan la línea 9?

Las definiciones de las variables **a** y **c** en la linea 9 del código (**a = c - a**) tienen el siguiente estado:

IN	OUT
<**a,1**>,<**c,5**>,<**c,7**>,<**a,3**>	<**c,7**>,<**c,5**>,<**a,9**>

Este resultado se corresponde con lo obtenido con la herramienta **SOOT**.

Notar que en la linea 9 se redefine la variable **a** por lo que se reemplazan sus definiciones anteriores por esta.

En la linea 7, dentro del **while** se redefine la variable **c** pero no se reemplaza a la definición de la linea 5 ya que puede pasar (**"may"**) que no se ingrese al while.

1.3. ¿Qué definiciones de la variable a alcanza la línea 10?

Las definiciones de las variables **a** y **c** en la linea 10 del código (**return a**) tienen el siguiente estado:

IN	OUT
<**c,5**>,<**c,7**>,<**a,9**>	–

Podemos obtener sencillamente el conjunto IN de este nodo, tomando la unión del conjunto OUT del nodo anterior que mostramos en el ejercicio previo. Siendo éste el ultimo nodo del Grafo de Control de Flujo, el conjunto **OUT** será vacío.

2. Ejercicio 2

Para corroborar el resultado del análisis, Reaching Definition Analysis, del analizador **SOOT**, realizo el **Grafo de Control de Flujo** para el programa del Ejercicio 2.

Coloquialmente, una variable estará "*viva*" cuando exista un camino a un uso de esa variable que no contenga a una redefinición de la misma en el medio del camino.

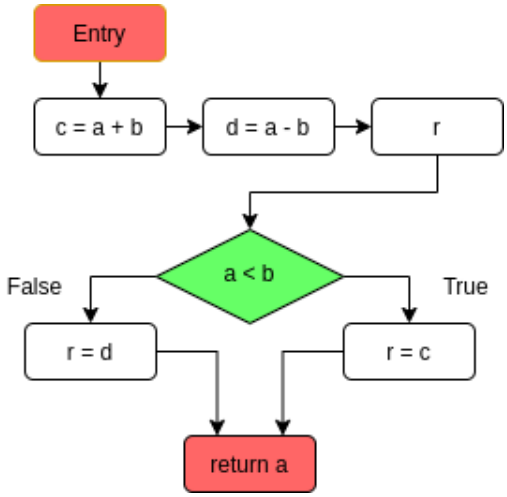

Figura 2: Grafo de Control de Flujo del Ejercicio 2

2.1. ¿Cuál es el conjunto de variables vivas en la línea 5?

Como vimos en la teórica, en la definición del análisis **Live Variables**, el conjunto IN y OUT para cada nodo del Grafo de Control de Flujo se define de la siguiente manera:

$$OUT[n] = \bigcup IN[n'] \tag{3}$$

Donde n' son los sucesores de n

$$IN[n] = (OUT[n] - KILL[n]) \cup GEN[n] \tag{4}$$

Por lo tanto, el conjunto IN y OUT de las variables vivas en la línea 5 ($\mathbf{r} = \mathbf{c}$) son:

IN	**OUT**
c	**r**

Este resultado concuerda con lo obtenido al realizar el Live Variable Analysis.

Este análisis, a diferencia del Reaching Definition, es "*backward*", por lo que se comienza el mismo a partir del nodo final. Donde el conjunto OUT es la unión de los conjuntos IN de los nodos sucesores. En este caso, el nodo que contiene a la definición $\mathbf{r} = \mathbf{c}$ toma como OUT a la variable **r** que pertenece al conjunto IN del unico nodo sucesor, el nodo final, **return r**.

2.2. ¿Cuál es el conjunto de variables vivas en la línea 7?

El conjunto IN y OUT de las variables vivas en la línea 7 (**r** = **d**) son:

IN	OUT
d	**r**

Por razones similares a las explicadas en el punto anterior, concluimos que el conjunto IN de este nodo es la variable **d** ya que es la variable que se "*usa*" en la expresión **r** = **d**. Notar que el nodo sucesor de este es el mismo que el nodo sucesor del ejercicio anterior, por lo tanto tienen el mismo conjunto OUT.

2.3. ¿Cuál es el conjunto de variables vivas en la línea 9?

El conjunto IN y OUT de las variables vivas en la línea 9 (**return r**) son:

IN	OUT
r	$\emptyset$

El conjunto OUT de este nodo sera **vació** ya que es el ultimo nodo que tiene alguna expresión en el cual se usan variables. Y para el conjunto IN, tomamos a la variable **r** ya que se está usando en la expresión, y no se esta redefiniendo.

3. Ejercicio 3

El análisis **Null Pointer Checker** encuentra las instrucciones que tienen el potencial de arrojar un **NullPointerException** y además agrega anotaciones indicando, si el puntero siendo dereferenciado puede, o no, ser determinado estáticamente a un valor nulo.

3.1. ¿Qué valores abstractos de las variables c1 y c2 pueden alcanzar la línea 3?

Lo mas importante que detecta el análisis es que no es posible que la expresión **return c1.value** (la línea 3) arroje un **NullPointerException**. Luego los valores abstractos que pueden alcanzar segun **SOOT**, en la línea 3, las variables c1 y c2 serán: **not null**.

Si bien cuando analizamos el output del análisis realizado por SOOT, vemos que a las variables c1 y c2 les asigna un valor **unknow** en la línea 1 y 2, respectivamente, al final, en la línea 3 (**return c1.value**), nos devuelve **not null**, para la variable c1. Como c1 está definida de la misma manera que c2, asumo que ésta variable también alcanzaría un valor **not null**, como c1 en la línea 3, si fuera dereferenciada.