0.1. Live variable analysis

Es backward. En cada paso agrega los predecesores a un stack.

$$transfer[n](X) = gen(n) \cup (X - kill(n)) \quad (1)$$
$$out[n] := \cup\{in[m] | m \in succ(n)\} \quad (2)$$
$$in[n] := transfer[n](out[n]) \quad (3)$$
$$gen(n) = \text{Variables leídas en } n \quad (4)$$
$$kill(n) = \text{Variables escritas en } n \quad (5)$$

0.2. Inductive analysis

$$gen(n) = \text{Variables de GOTO } \cup \text{ sizes de arrays/multiarrays} \quad (6)$$
$$kill(n) = \emptyset \quad (7)$$
$$transfer[n](X) = ? \quad (8)$$

Transfiere el *out* al *in*.
transfer[n](X) = ?

Describo la función de transferencia por partes:
Primero se llena la lista *addSet*

- CASO x = f(b) Si el *out* contiene a la variable x, entonces meto en *addSet* a...x y las variables de $f(b)$ que estén vivas en ese momento.
- CASO conditional $if\ x \neq\ 0\ GOTO\ s$

  Si el in contiene a una variable involucrada en la condición del GOTO, decimos que tiene una inductiva. En ese caso, volvemos a recorrer las variables de la condición, y metemos en *addSet* las que están vivas (ignorando s)

  Para mí no deberíamos fijarnos si alguna variable de la condición está en el in. Si es un GOTO agregamos todas las variables y listo no?

- Está comentado el caso invokeExpression, lo cual me parece un poco turbio.

Luego se genera *in*:
$in = A_1 \cup A_2 \cup addset$,

donde $A_1 = (genSet \cap vivas)$ y $A_2 = (out \cap vivas)$

Observación: el killset no juega ningún rol.

- En el código están swappeados el in y el out, no sé si porque cuando uno extiende BackwardFlowAnalysis de soot hay que llamarlos así y no al revés como en la teoría. Chequearlo
- NO estamos usando el análisis de inductivas; está apagado un switch. Estamos usando como inductivas a las relevantes, que son las variables vivas.

A ver, para abreviar, asumamos que todas las variables están vivas.

Entonces $in = variablesde\ GOTO \cup sizesdearrays \cup out \cup addSet$,

donde $addSet$ tiene a las variables que modifican una variable del out y también tiene todas las variables del stmt si el statement es un GOTO.

el $addSet$ no hace falta que tenga lo segundo, porque ya forma parte del genSet :/

Lo malo de todo esto es que se "propagan demasiado"...pudiendo salirse del loop.

No sé está haciendo esto porque el switch está apagado, pero se deberían diferenciar las relevantes de las inductivas, con relevantes = live variables, inductivas = las del análisis.

Tal vez una pequeña optimización podría ser hacer un killset. Podrían ser, por ejemplo, las variables que son usadas solamente fuera de los loops. Será muy difícil de hacer?

Eso, o que la propagación sea siempre "dentro del loop", pero me suena a que esto es más difícil, por alguna razón.....aunque supongo que es igual de dificíl.

Propuesta mía: yo quiero a las variables que se usan para modificar a las variables inductivas (propagar para ese lado), parecido a lo que hace el slicing de alexis.

Bah, en realidad esto no funciona muy bien, porque seguramente se va un poco al carajo. Tengo que ver las variables que modifican la condición del loop pero "dentro del loop". Pero entender cuál es el cuerpo del loop me parece un poco complicado.