Assignment 3

Task 1

According to the abstract interpretation of program analysis, we shall approximate our interval analysis with the following Galois connection $(\mathbb{P}(\mathbb{Z}), \alpha, \gamma, \mathfrak{I})$ that describes the relation between sets of integers to intervals as shown below:

$$\alpha: \mathbb{P}(\mathbb{Z}) \to \mathfrak{I}$$
$$\gamma: \mathfrak{I} \to \mathbb{P}(\mathbb{Z})$$
$$\mathfrak{I}: (\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{\infty\}) \; where \; infimum(\mathfrak{I}) \leq supremum(\mathfrak{I})$$
$$\sigma^{\#}: \mathbb{V} \to \mathfrak{I}$$
$$f_{\ell}^{\#}: \sigma_{\ell'}^{\#} \to \sigma_{\ell}^{\#} \; \forall (\ell', \ell) \in Forwards$$

where α is the abstraction function, γ the concretization function, $\mathfrak{I}$ the interval of infimum and supremum, $\sigma^{\#}: \mathbb{V} \to \mathfrak{I}$, and the transfer function $f_{\ell}^{\#}$ at each program point. We can also define abstract interval arithmetic as follows:

$$-^{\#}[a,b] \stackrel{\text{def}}{=} [-b,-a]$$
$$[a,b] +^{\#} [c,d] \stackrel{\text{def}}{=} [a+c, b+d]$$
$$[a,b] -^{\#} [c,d] \stackrel{\text{def}}{=} [a-d, b-c]$$
$$[a,b] \times^{\#} [c,d] \stackrel{\text{def}}{=} [min(ac,ad,bc,bd), max(ac,ad,bc,bd)]$$
$$[a,b] \cup^{\#} [c,d] \stackrel{\text{def}}{=} [min(a,c), max(b,d)]$$
$$[a,b] \cap^{\#} [c,d] \stackrel{\text{def}}{=} [min(ac,ad,bc,bd), max(ac,ad,bc,bd)]$$

and the abstract semantics

$$\mathcal{A}^{\#}[\![v]\!]\sigma^{\#} \stackrel{\text{def}}{=} \sigma^{\#}(v)$$
$$\mathcal{A}^{\#}[\![-e]\!]\sigma^{\#} \stackrel{\text{def}}{=} -^{\#}\mathcal{A}^{\#}[\![e]\!]\sigma^{\#}$$
$$\mathcal{A}^{\#}[\![e_1 + e_2]\!]\sigma^{\#} \stackrel{\text{def}}{=} \mathcal{A}^{\#}[\![e_1]\!]\sigma^{\#} +^{\#} \mathcal{A}^{\#}[\![e_2]\!]\sigma^{\#}$$
$$\mathcal{A}^{\#}[\![e_1 - e_2]\!]\sigma^{\#} \stackrel{\text{def}}{=} \mathcal{A}^{\#}[\![e_1]\!]\sigma^{\#} -^{\#} \mathcal{A}^{\#}[\![e_2]\!]\sigma^{\#}$$
$$\mathcal{B}^{\#}[\![e_1 \; op \; e_2]\!]\sigma^{\#} \stackrel{\text{def}}{=} \mathcal{B}^{\#}[\![e_1]\!]\sigma^{\#} op^{\#} \mathcal{B}^{\#}[\![e_2]\!]\sigma^{\#}$$
$$\mathcal{B}^{\#}[\![\neg \, e]\!]\sigma^{\#} \stackrel{\text{def}}{=} \neg^{\#}\mathcal{B}^{\#}[\![e]\!]\sigma^{\#}$$

where $v \in \mathbb{V}$, e an expression, and $op^{\#}$any Boolean operator. Note that $\mathbb{V}$ is a set of variables in a program to be analyzed.

Task 2

Based on the framework from Task 1, the implementation is shown as the following algorithm for loop-free program analyses.

Algorithm

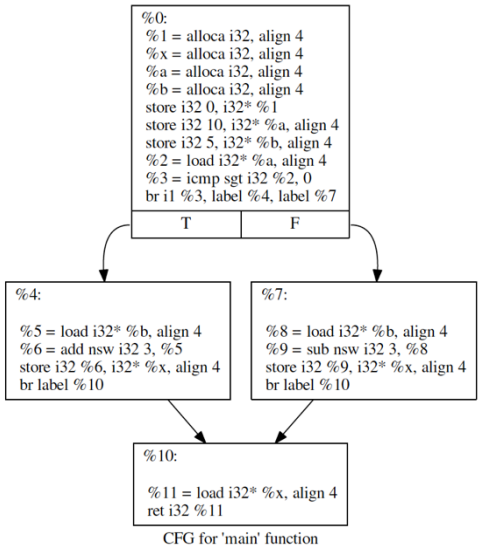
Interval: an object that represents an interval of infimum and supremum, implemented with its corresponding operations such as overloaded operators like addition, subtraction, and multiplication.
intervalMap: a mapping of a variable to its corresponding interval

boolMap: a mapping of a comparison instruction to its corresponding path traversal Boolean values, the basics of the path-sensitive analysis
transfer: a function that calculates the values of variables according to the current instructions in a basic block
backwardUpdate: a function that updates the values that pass down a specific path based on a comparison instruction
In each basic block, first compute the interval for each variable in an instruction, and with the latest **boolMap** that is modified by the **transfer** function, decide whether or not to traverse over the next basic block for different paths. Continuously do it until no more blocks from the control flow graph are found. Finally, the algorithm takes a union of the maps from different paths and print out the **intervalMap**.

Consider the following example program

$$[a := 10]^1; [b := 5]^2; if\ [a > 0]^3\ then\ [x := 3 + b]^4\ else\ [x := 3 - b]^5$$

and its control flow graph on the right. We shall use our algorithm as described above to traverse over the basic blocks in the graph and implement a path-sensitive program analysis as follows:



CFG for 'main' function

Block 1: // %0
x: [-∞, ∞], a: [10, 10], b: [5, 5]

Block 2: // %4
x: [8, 8], a: [10, 10], b: [5, 5]

Block 3: // %10
x: [8, 8], a: [10, 10], b: [5, 5]

Since **a** is always greater than 0, it is unnecessary to traverse down the other path from **%7**.

Task 3

Using the algorithm elaborated on in Task 2, we can show how to support a loop program analysis and describe a modified algorithm that reaches a fixed point in the following. Note that the same information that has been aforementioned will not appear here again.

Algorithm

wideMap: a function that widens an old **intervalMap** with a new **intervalMap**.
reachFixedPoint: a function that checks if an old **intervalMap** is exactly the same as a new **intervalMap**
masterTraversedBlocks: the set of traversed basic blocks on the master branch
masterBlockQueue: the queue that includes the next basic block to traverse from the master branch, not from any sub-branches
The algorithm is similar to the previous one except that the traversal mechanism is different in that at some point the analysis will encounter the same basic block that has the loop condition. To decide whether or not to traverse over the loop again or choose the

false path that is out of loop, we first observe whether or not **reachFixedPoint** shows that the old map from the start of the loop, and the new map after the end of the loop are the same. If yes, remove the loop condition block from **masterBlockQueue** to prevent the traversal of it again and jump out of loop. Otherwise, if the current basic block is in both **masterTraversedBlocks** and **masterBlockQueue**, return to the master branch and execute **wideMap** such that it speeds up the abstract interval analysis. Following that, do the analysis again until it reaches a safe fixed point.

Consider the following example program

$[a := -2]^1; [b := 5]^2; [x := 0]^3; [i := 0]^4 while\ [i < N]^5\ ;$
$do(if\ [a > 0]^6\ then\ [x := x + 7]^7; [y := 5]^8\ else\ [x := x - 2]^9;\ [y := 1]^{10}; if\ [b > 0]^{11}; then\ [a := 6]^{12} else\ [a := -5]^{13};)$

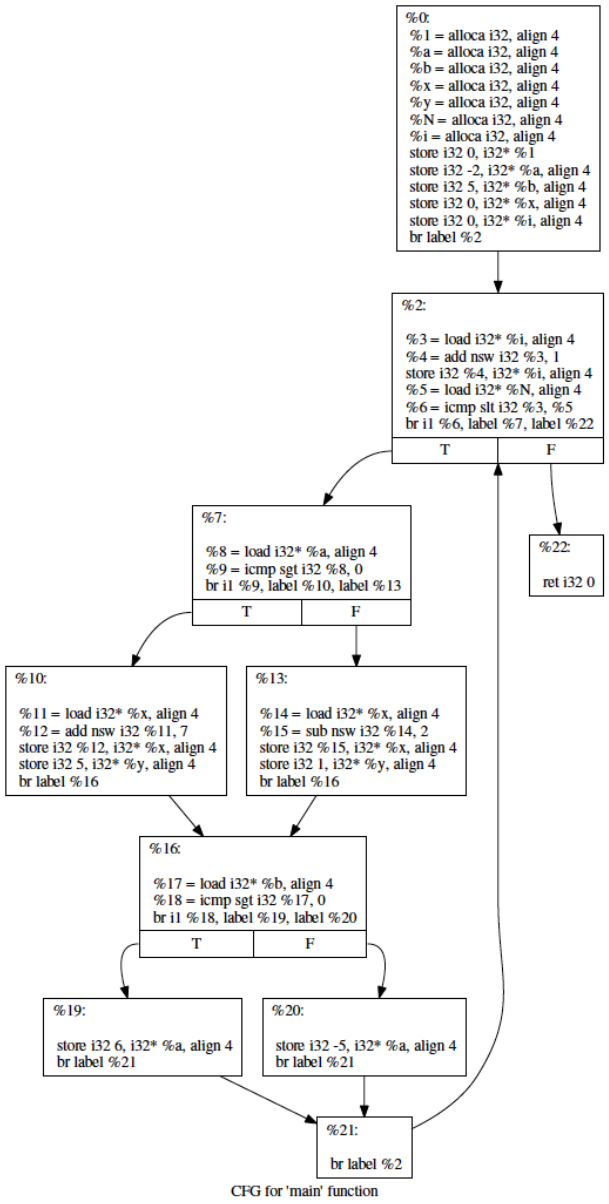
and its control flow graph below.

For simplicity, here is only presented the final **intervalMap** after the program analysis using the algorithm.

Block 5128:

x: [-∞, ∞]
y: [1, 5]
a: [-2, 6]
b: [5, 5]
i: [0, ∞]
N: [-∞, ∞]

In conclusion, although the analysis may not be as safe as we expect it to be, it is indeed on possible analysis result in this case.



CFG for 'main' function