CS5218: Assignment 3 – Abstract Interpretation

This assignment is not a group assignment.

1 Part 1 - Difference Analysis

The purpose of this part of the assignment is to perform an abstract interpretation analysis without path sensitivity. Your task would be mainly defining an abstract domain and performing an analysis which requires widening.

1.1 Introduction

Consider a special kind of "difference" analysis. We want to know, at each program point, what is the difference in value between any *pair* of variables, eg. for variables x and y, what is their *separation*? We now define the (symmetric) function:

$$sep(x,y) = \begin{cases} x-y & if \quad x \geq y \\ y-x & otherwise \end{cases}$$

The analysis serves to determine, for each program point, and for each pair of variables, the *largest* separation.

1.2 Task 1: Designing Difference Analysis

Define your analysis using the framework of *abstract interpretation* (Galois connection, abstract semantics, etc).

1.3 Task 2: Implementing the Difference Analysis in LLVM

You have seen a sample abstract interpretation framework in the demo session. In this task, implement an LLVM pass to perform the analysis on loop-free programs. The LLVM pass should find, for each program point, and for each pair of variables, the *largest* separation.

Example 1: For example, consider the program below:

```
int main() {
    int a,b,c,d = 0;
    if (a > 0)
```

```
        c = 5;
    else
        c = 10;
    if (b > 0)
        d = -11;
}
```

One difference analysis, for the pair of variables c and d, could return for $sep(c,d)$, at the terminal program point, the value 21.

Note: Your analysis can be limited to the following instructions: Alloca, Add, Sub, Div, Mul, Rem and Load/Store.

1.4 Task 3: Adding Support for Loops to the Analysis

Extend your implementation to accomodate cycles in the CFG. This time you need to be concern about the termination of your analysis. In order to guarantee termination, the analysis may now not be able to produce a finite bound at all times. You therefore need to perform some kind of *abstraction*.

Example 2: Consider the program below:

```
int main() {
    int a,b,x,y,z = 0 ;
    // assume N is an input value
    int i = 0;
    while (i < N) {
        x = -((x + 2*y * 3*z) % 3);
        y = (3*x + 2*y + z) % 11;
        z++;
    }
}
```

One difference analysis, for the pair of variables x and y, could return for $sep(x,y)$ the value 12 at the end of the loop body. For the pair of variables x and z, on the other hand, it may not be possible to obtain a finite bound.

Note: One sample abstract domain can be $[-\infty,-20,...,-1,0,1,...,20,\infty]$ which can represent all the values that z can have in an abstract way. Explain the abstraction that you are use in the report.

2 Part 2 - Interval Analysis

The purpose of this part of the assignment is to perform an abstract interpretation analysis with some path sensitivity.

2.1 Introduction

Interval analysis is a famous analysis developed in 1950s and 1960s. In this approach, an interval for a variable x is represented by $x : [a, b]$ where a is an upper bound and a b is a lower bound on the possible values that x can contain. For example, in the program below, the interval for x before the assertion would be: $[-2, 8]$. Since, the value of x is at most 8, it can be inferred that the assertion will never be violated.

```
int main() {
  int x, a;
  int b = 5;
  if (a > 0)
    x = 3 + b;
  else
    x = 3 - b;
  assert (x < 10);
  return x;
}
```

Note that the intervals for the other variables in this program would be: $a : [-\infty, \infty]$ and $b : [5, 5]$. Interval analysis is a light-weight analysis that is widely used to infer the possible values for variables at different program points.

In this part, you will be implementing an Interval analysis with some path sensitivity for all the variables in the input program. All the algorithms you need are available from the lecture material.

For example, consider the program below. A non-path sensitive interval analysis would infer that the interval of values of x before the assertions will be: $[-2, 8]$. As a result, it would infer that the assertion will be violated. However, considering the interval for $a : [10, 10]$, we can see that the false branch of the if-statement is infeasible (meaning the values of a are always greater than 0 and a cannot have any non-positive values).

Considering this point, the update on value of x in the false branch should be ignored. As a result, the interval for x would change to $x : [8, 8]$ which does not violate the assertion in the end of the program.

```
int main() {
  int x, a;
  a = 10;
  int b = 5;
  if (a > 0)
    x = 3 + b;
  else
    x = 3 - b;
  assert (x >= 0);
  return x;
}
```

Finally, the analysis will return the following intervals for the other variables in this program: $a : [10,10]$ and $b : [5,5]$.

2.2 Task 1: Designing the Interval Analysis

Define your analysis using the framework of *abstract interpretation* (Galois connection, abstract semantics, etc).

2.3 Task 2: Implementing the Interval Analysis in LLVM

You have seen a sample abstract interpretation framework with some path sensitivity in the demo session. In this task, implement an LLVM pass to perform the analysis on loop-free programs.

Note: The abstract interpretation framework in the demo session considered the effect of equality and non-equality on the abstract domain. In this task, your analysis should consider the effect of dis-equalities on the abstract domain too.

2.4 Task 3: Adding Support for Loops to the Analysis

Extend your implementation to accommodate cycles in the CFG. This time you need to be concerned about the termination of your analysis. In order to guarantee termination, the analysis may now not be able to produce a finite bound at all times. You therefore need to perform some kind of *widening*.

Example 2: Consider the program below:

```
int main() {
  int a = -2,b = 5,x = 0, y;
  // assume N is an input value
  int i = 0;
  while (i++ < N) {
     if (a > 0){
        x = x + 7;
        y = 5;
     }else{
        x = x - 2;
        y = 1;
     }
     if (b > 0){
        a = 6;
     }else{
        a = -5;
     }
  }
}
```

A non path-sensitive interval analysis, would return $[-\infty, \infty]$ for x. A path-sensitive interval analysis, however, would return the following intervals for the variables (assuming loop iterates at least once):

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

3 Submission - Due Date: Sunday 28th April, 23:59

Please submit the following in a single archive file (`zip` or `tgz`):

1. A report in PDF (`asg3.pdf`). It should describe, for each part, your design for task 1 and the implementation of tasks 2 and 3, and the algorithm used. How to build and run. The output of examples programs tested.
2. Your source code.
3. Your test C files with corresponding `ll` files.

For technical support on LLVM, you can contact either Rasool (`rasool@comp.nus.edu.sg`) or Sanghu (`sanghara@comp.nus.edu.sg`). Make sure your subject line begins with “CS5218”.

Make sure your reports and source files contain information about your name, matric number and email. Your zip files should have the format *Surname-Matric*-asg4.zip (or `tgz`). Submit all files above to the appropriate IVLE workbin.