CS5218 Assignment3 Report

Guo Shijia A0191309E

April 30, 2019

1 Part 1 - Difference Analysis

1.1 Task 1: Designing Difference Analysis

For Difference Analysis, First we should introduced our concrete and abstract domain.
Our concrete domain is $C = 2^z$ - set of sets of integers.
Our abstract domain is A - set of intervals

$$A = \{\bot\} \cup \{[p,q]\} |\ p \in \{-\infty\} \cup Z, q \in Z \cup \{\infty\}, p \leq q$$

The interval lattice is:

$$\beta_\sharp = \{[a,b]\ | a \in I \cup \{-\infty\}, b \in I \cup \{+\infty\}, a \leq b\} \cup \{\bot_b^\sharp\}$$

The Galois connection for Intervals is:

$$\gamma([p,q]) = \{x \in Z \mid p \leq x \leq q\}\ where\ \ \gamma(\bot) = \emptyset$$

$$\alpha(c) = [inf(c), sup(c)],\ if\ c \neq \emptyset \quad where\ \ \alpha(\emptyset) = \bot$$

where $\alpha,\ \gamma$ is monotonic function.
We also need introduce the interval abstract arithmetic operators:

$$[c, c^{'}]_b^\sharp = [c, c^{'}]$$

$$-_b^\sharp[a,b] = [-b, -a]$$

$$[a,b] +_b^\sharp [c,d] = [a+c, b+d]$$

$$[a,b] -_b^\sharp [c,d] = [a-d, b-c]$$

$$[a,b] \times_b^\sharp [c,d] = [min(ac,ad,bc,bd), max(ac,ad,bc,bd)]$$

$$[a,b]/_b^\sharp[c,d] = \begin{cases} \bot_b^\sharp, & if \quad c = d = 0 \\ [min(a/c, a/d, b/c, b/d), max(a/c, a/d, b/c, b/d)] & else\ if \quad 0 \leq c \\ [-b,-a]/_b^\sharp[-d,-c] & else\ d \leq 0 \\ ([a,b]/_b^\sharp[c,0]) \cup_b^\sharp ([a,b]/_b^\sharp[0,d]) & otherwise \end{cases}$$

$where \pm\infty \times 0 = 0, 0/0 = 0, \forall x : x/\pm\infty = 0, \forall x > 0 : x/0 = +\infty, \forall x < 0 : x/0 = -\infty$
How we widen the interval(on single elements form interval):

$$w([a,b]) = [max\{i \in B \mid i \leq a\}, min\{i \in B\ b \leq i|\}]$$
$$w(\perp) = \perp$$

The function w is defined pointwise on L^n.
Our abstraction is: we set the threshold for the analysis, we treat the value over 1000 as $+\infty$ and value lower than -1000 as $-\infty$.
So the abstract semantics is defined as:

$$\chi^\sharp : L \to D \textbf{ any solution of} \begin{cases} \chi_e^\sharp \textbf{ such that } \chi_e \subseteq \gamma(\chi_e^\sharp) \\ \chi_{\zeta \neq e}^\sharp \supseteq^\sharp \cup\ C^\sharp[c]\chi_{\zeta'}^\sharp \end{cases}$$

$$\forall \zeta \in L : \gamma(\chi_e^\sharp) \supseteq \chi_\zeta$$

1.2 Task 2 & 3: Difference Analysis Implementation

For Implementation, we use the abstract analysis combine with Galois connection to solve this. First, we transfer the separation to determine the each variable's interval, then use the interval to calculate their separation. We follow the template of demo, but made some modifications. We maintain an activeBlocks list, which contain the blocks that updated in the previous iteration(if one block is changed, then we add this block and the successor of this block into the activeBlocks list). Then, when update the each block, first we collect all the information from the predecessor blocks, and union each variable's interval by using the widen method. Then we use the intervals collect from the predecessor blocks to handle the different instructions.
Besides the arithmetic instructions , we also need to handle the load,store and allocate instruction. For each allocate variable, we assign the interval as $[-\infty, +\infty]$. After each iteration, we check the update information to seek for the fixed point. If the interval value is above our pre-defined threshold, we treat as $-\infty$ *or* $+\infty$. After the arrive the fixed point, we use the variable's interval to calculate the separation. For example, the interval [a,b] and [c,d], their separation should be :$max(abs(d-a), abs(b-c))$.

1.3 Result

How to compile:
clang++-3.5 -o part1 Assignment3_part1.cpp ‘llvm-config-3.5 –cxxflags‘ ‘llvm-config-3.5 –ldflags‘ ‘llvm-config-3.5 –libs‘ -lpthread -lncurses -ldl
the execute command: ./part1 test1.ll
The screenshots of example program output:

```c
int main(){
 int a,b,c,d = 0;
 if(a>0)
   c = 5;
 else
   c = 10;
 if(b>0)
   d = -11;
}
```

(a) source code

```
Block name is: %10
a and b sep is: Infinity
a and c sep is: Infinity
a and d sep is: Infinity
b and c sep is: Infinity
b and d sep is: Infinity
c and d sep is: 21
```

(b) result

Figure 1: Task2 Example testcase

```c
int main(){
  int a,b,x,y,z = 0;
  int i = 0,N=100;
  while(i < N){
    x = -((x+2*y*3*z)%3);
    y = (3*x+2*y+z)%11;
    z++;
  }

}
```

(a) source code

```
Block name is: %6
N and a sep is: Infinity
N and b sep is: Infinity
N and i sep is: 100
N and x sep is: 102
N and y sep is: 100
N and z sep is: Infinity
a and b sep is: Infinity
a and i sep is: Infinity
a and x sep is: Infinity
a and y sep is: Infinity
a and z sep is: Infinity
b and i sep is: Infinity
b and x sep is: Infinity
b and y sep is: Infinity
b and z sep is: Infinity
i and x sep is: 2
i and y sep is: 10
i and z sep is: Infinity
x and y sep is: 12
x and z sep is: Infinity
y and z sep is: Infinity
```

(b) result

Figure 2: Task3 Example testcase

2 Part 2 - Interval Analysis

2.1 Task 1: Designing Interval Analysis

This part analysis is similar like part1, but we do not collect all the information of predecessor blocks, we need to consider the path sensitive. Below is how we narrow down our intervals: if X = [a,b] and Y = [c,d], we can define:

$$C^\sharp(X <= Y) = \begin{cases} \bot_b^\sharp, & if \quad a > d \\ \chi^\sharp[X \to [a, min(b,d)], Y \to [max(c,a), d]\ ] & otherwise \end{cases}$$

$$C^\sharp(X < Y) = \begin{cases} \bot_b^\sharp, & if \quad a >= d \\ \chi^\sharp[X \to [a, min(b,c-1)], Y \to [max(a+1,c), d]\ ] & otherwise \end{cases}$$

$$C^\sharp(X > Y) = \begin{cases} \bot_b^\sharp, & if \quad b <= c \\ \chi^\sharp[X \to [min(a,c+1), b], Y \to [c, min(b-1,d)]\ ] & otherwise \end{cases}$$

$$C^\sharp(X >= Y) = \begin{cases} \bot_b^\sharp, & if \quad b < c \\ \chi^\sharp[X \to [max(a,c), b], Y \to [c, min(b,d)]\ ] & otherwise \end{cases}$$

$$C^\sharp(X == Y) = \begin{cases} \bot_b^\sharp, & if \quad b < c \ or \ a > d \\ \chi^\sharp[X \to [max(a,c), min(b,d)], Y \to [max(a,c), min(b,d)]\ ] & otherwise \end{cases}$$

$$C^\sharp(X! = Y) = \begin{cases} \bot_b^\sharp, \ if max(a,c) <= min(b,d) \\ \chi^\sharp[X \to [max(a,c), min(b,d)], Y \to [max(a,c), min(b,d)]\ ] & otherwise \end{cases}$$

we can treat the X or Y as interval or constant, combine this and above analysis, we can have path sensitive analysis.

2.2 Task 2 & 3: Interval Analysis Implementation

The difference between part1 and part2 is how we collect the predecessor block's information. If the predecessor branch is unconditional jump, we will collect all the information; if it is a conditional jump, we will based on the condition to narrow down our variable's interval, how we narrow down is explained in the part 2.1.

2.3 Result

How to compile:
clang++-3.5 -o part2 Assignment3_part2.cpp ‘llvm-config-3.5 –cxxflags‘ ‘llvm-config-3.5 –ldflags‘ ‘llvm-config-3.5 –libs‘ -lpthread -lncurses -ldl
the execute command: ./part2 test5.ll
The screenshots of example program output:

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

}
```

(a) source code

```
Block name is:%0
a       [ 10 , 10 ]
b       [ 5 , 5 ]
x       [ NEG_INF , POS_INF ]
Block name is:%10
a       [ 10 , 10 ]
b       [ 5 , 5 ]
x       [ 8 , 8 ]
Block name is:%4
a       [ 10 , 10 ]
b       [ 5 , 5 ]
x       [ 8 , 8 ]
Block name is:%7
```

(b) result

Figure 3: Task2 Example testcase

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


}
```

(a) source code

```
Block name is:%0
a       [ NEG_INF , POS_INF ]
b       [ 5 , 5 ]
x       [ NEG_INF , POS_INF ]
Block name is:%10
a       [ NEG_INF , POS_INF ]
b       [ 5 , 5 ]
x       [ -2 , 8 ]
Block name is:%4
a       [ 1 , POS_INF ]
b       [ 5 , 5 ]
x       [ 8 , 8 ]
Block name is:%7
a       [ NEG_INF , 0 ]
b       [ 5 , 5 ]
x       [ -2 , -2 ]
```

(b) result

Figure 4: Task2 additional testcase 1

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


}
```

```
Block name is:%0
a       [ -10 , -10 ]
b       [ 5 , 5 ]
x       [ NEG_INF , POS_INF ]
Block name is:%10
a       [ -10 , -10 ]
b       [ 5 , 5 ]
x       [ -2 , -2 ]
Block name is:%4
Block name is:%7
a       [ -10 , -10 ]
b       [ 5 , 5 ]
x       [ -2 , -2 ]
```

(a) source code (b) result

Figure 5: Task2 additional testcase 2

```c
int main(){
    int a = -2,b = 5,x = 0, y;
    int N = 100;
    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;
        }
    }
}
```

```
Block name is:%21
N       [ 100 , 100 ]
a       [ 6 , 6 ]
b       [ 5 , 5 ]
i       [ 1 , 99 ]
x       [ NEG_INF , POS_INF ]
y       [ 1 , 5 ]
Block name is:%22
N       [ 100 , 100 ]
a       [ -2 , 6 ]
b       [ 5 , 5 ]
i       [ 100 , 100 ]
x       [ NEG_INF , POS_INF ]
y       [ NEG_INF , POS_INF ]
```

(a) source code (b) result

Figure 6: Task3 example testcase

```c
int main(){
    int a = 2,b = 5,x = 0, y;
    int N = 100;
    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) source code

```
Block name is:%21
N       [ 100 , 100 ]
a       [ 6 , 6 ]
b       [ 5 , 5 ]
i       [ 1 , 99 ]
x       [ 7 , POS_INF ]
y       [ 5 , 5 ]
Block name is:%22
N       [ 100 , 100 ]
a       [ 2 , 6 ]
b       [ 5 , 5 ]
i       [ 100 , 100 ]
x       [ 0 , POS_INF ]
y       [ NEG_INF , POS_INF ]
```

(b) result

Figure 7: Task3 additional testcase 1