Assignment 3

1 Student Information

Name	: Hady Januar Willi
Student ID	: A0174396R
School Email	: E0210429@u.nus.edu
Email	: hadywilli@outlook.com

2 Design

Both task 1 and task 2 analysis are implemented on the basis of interval design, even though task 1 does not require completion of interval analysis. The ability of getting the maximum difference between two variables is a side effect of the result of interval analysis, which is a more general analysis than maximum difference analysis.

The analysis is divided into two steps, widening first and then narrowing. In the implementation and analysis of result, any value that is -1000 represents negative infinity and any value that is 1000 represents positive infinity. All widening and narrowing handle of variable interval (Range struct in code) will check against this sanity, that is any value above 1000 will be set to 1000 and any value below -1000 will be set to -1000.

The transfer function of instructions are simplified for generalization of interval analysis, therefore may not be fully accurate. Each instruction type has its implementation for narrowing and widening. Widening implementation is a simple widening of narrowing instruction transfer functions.

In loops during narrowing after widening, the widenened interval of the while end condition can spill over to while condition and while body, making the analysis less accurate than it should be. As a result, analysis is not as accurate as what the assigment expects. The implementation to cater for such case is neither general nor trivial, requiring hard coding of block names and first iteration check, therefore is not included in the implementation.

2.1 Branch Comparison

This analysis covers the following comparison instruction, with the assumption that either one side of the comparison is a constant, or the right hand side is a user input (i.e. not part of the analysis):

1. Greater than (>)
2. Greater than or equal to(>=)
3. Less than (<)
4. Less than or equal to(<=)
5. Equals(==)

Not equals comparison is not covered in the analysis as the condition for interval analysis is complex. The complexity can be thought of being analoguous to having two comparison of greater than and less than.

2.2 Instructions

The following instructions are covered according to the requirement in the assignment handout:

1. Alloca
2. Store
3. Load
4. SAdd
5. SSub
6. SMul
7. SDiv
8. SRem

2.3 Structure

The structure of the algorithm follows the general guideline in the demo and absint example code, with more emphasis on intervals rather than LLVM instructions. The transfer functions of branch comparisons, instructions, and branch merging are defined in terms of intervals rather than instructions.

2.4 Task 1 Printing Flags

In task 1 source file main function, there are two flags, print infinity and print steps. Print infinity when set to true will print out all difference analysis that does not have a bounded values. Print steps when set to true will print out interval analysis narrowing and widening steps.

2.5 Task 2 Printing Flags

In task 2 source file main function, there is only one flag which is print steps. Print steps when set to true will print out interval analysis narrowing and widening steps.

3 Submission Folder Structure

> zip	–	Folder containing all files
> bin	–	Folder containing compiled assignment binary
> graphs	–	Folder containing dot and pdf files generated from LLVM intermediate representation
> installation	–	Folder containing LLVM 7 installation script
> llvms	–	Folder containing LLVM intermediate representation
> source	–	Folder containing assignment source code
> target	–	Folder containing target C code

4 Running the Implementation

After extracting the file, the following commands can be run in sequence:

```
./compile
./llvm
./run
```

Run command will run against all four examples in the assignment sheet. After compilation, to run against any other LLVM intermediate representation file, the following command can be used:

For task 1:

```
./bin/assignment3-1 other-file.ll
```

For task 2:

```
./bin/assignment3-2 other-file.ll
```

To generate the graphs, the following command can be run:

```
./graph
```

If all else fails, the compiled binary is also included in the assignment, and can be used with the following command which hopefully runs with or without LLVM libraries:

For task 1:

```
./bin/assignment3-1 other-file.ll
```

For task 2:

```
./bin/assignment3-2 other-file.ll
```

5 Misc

The implementation can be found at `https://github.com/hjw95/program-analysis` which will be available after the end of the course. I believe this module is good and important, however I find that the examples in the module is lacking concrete implementation that students can run and see what exactly is happening.

I am keen to help with developing example code. If you are open to discussion on creating example analysis code, do send email to my personal email address or comment in github.