EC4219: Software Engineering (Spring 2024)
Homework 3: Interval Analysis

100 points
Due: 6/19, 23:59 (submit via GIST LMS)

Instructor: Sunbeom So

Important Notes

- **Evaluation criteria**
  The correctness of your implementation will be evaluated using testcases:

  $$\frac{\#\text{Passed}}{\#\text{Total}} \times \text{point per problem}$$

  - "Total" indicates a set of testcases prepared by the instructor (undisclosed before the evaluation).
  - "Passed" indicates testcases whose expected outputs match with the outputs produced by your implementations.

- **Executable**
  Before you submit your code, please make sure that your code can be successfully compiled. That is, the command `./build` should not report any errors. Otherwise, you will get 0 points for that HW.

- **No Plagiarism and No Discussion**
  Cheating (i.e., copying assignments by any means) will get you an F. See the slides for Lecture 0. Code-clone checking will be conducted irregularly. Furthermore, discussions at all levels are strictly disallowed.

- **No Changes on Template/File Name/File Extension Changes**
  Your job is to complete `(* TODO *)` parts in provided templates; you should not modify the other existing code templates. Do not change the file names. The submitted files should have `.ml` extensions, not the others (e.g., `.pdf`, `.zip`, `.tar`).

- **No Posting on the Web**
  You should not post your implementations on public websites (e.g., public GitHub repositories). Violating this rule gets you an F, even after the end of the semester.

1 Goal

Your goal is to implement an interval analyzer for detecting assertion violations.

2 Structure of the Project

You can find the following files in the `hw3` directory. **Your job is to complete and submit the two files**:

`absDom.ml`, `itvAnalysis.ml`

As in Hw1 and Hw2, feel free to implement any additional helper functions that you think are necessary, even if they are not specified in the templates.

- `absDom.ml`: This file implements the abstract domains and abstract operators.
  - `module AbsMem`: aims to implement an abstract memory state $\widehat{m} \in \widehat{\mathbb{M}}$, a mapping from each variable to an abstract value.
  - `module AbsVal`: aims to implement an abstract value domain. An abstract value is a pair of an interval $i \in \mathbb{I}$ and an abstract boolean $\widehat{b} \in \widehat{\mathbb{B}}$.
  - `module AbsBool`: aims ot implement an abstract domain for boolean values.
  - `module Itv`: implements an interval domain.
- `itvAnalysis.ml`: aims to implement the abstract transformers (abstract semantic functions).
  - For Hw3, we assume all elements of an integer array are abstracted into a single interval value. For example, given an array a containing three elements $\{1, 5, 10\}$, the interval values of $a[0]$, $a[1]$, and $a[2]$ are all $[1, 10]$.
  - You should consider implementing widening and narrowing operators to ensure termination.
  - Moreover, you may consider applying widening operators only when you do not reach a fixed point even after x iterations, where x is a predetermined threshold (e.g., $x = 20$).
- The rest files are the same as Hw1 and Hw2.
  - An exception for Hw3 is that, we assume (1) pre- and postconditions are ignored in the analysis, and (2) all loop invariants are unknown (hence marked using "?") in each test code.

3 How to Build

After completing the files (`absDom.ml`, `itvAnalysis.ml`), you can build the project as follows.

```
$ ./build
```

Then, the executable `./main.native` will be generated. You can run it as follows.

```
$ ./main.native -input TESTCODE
```

4 Running Example

Consider the test code from `test/loop3`:

```
@pre ... (* ignore pre- and postconditions for Hw3 *)
@post ...
simple_loop3 (int n) returns (bool rv) {
  int x;
  int y;

  x = 0;
  y = 0;

  while @L{?} (x<10) {
    x = x+1;
    (* y = y+1; *)
  }

  assert (x==10 && y>=0); (* must be proven *)
  assert (y>=10); (* cannot be proven using our interval analysis *)

  return true;
}
```

If you run the command

```
$ ./main.native -input test/loop3
```

you should obtain the following result:

```
...
========== Processing ==========
[INFO] initial memory state
n |-> ([-oo, +oo],bot)
[INFO] memory state @ Return
n |-> ([-oo, +oo],bot)
x |-> ([10, 10],bot)
y |-> ([0, 0],bot)
========== Verificaiton Result ==========
1
Time : ...
```

where `1` is the number of proven assertions.