Static Analysis and Verification
HT2023

A.Rezine, Ulf Kargén

Linköping University

There are two parts in this lab. Both parts are mandatory. In order to pass this lab, you will need to:

1. go through each question and write down (using a pen or electornically) your answers (in pairs)
2. demonstrate your answers in a lab session to one of the teachers. You can demonstrate one part at a time. The demonstrations are individual.
3. after you have demonstrated all parts, send a report by email in a pdf form where your name is clearly stated (in pairs). The mapping between questions and their corresponding answers should be made clear.

1 Abstract Interpretation

Abstract interpretation [1] techniques are extensively used in compiler optimization, software analysis and program verification frameworks. We will use the ambitious open source frama-c software analysis framework for c programs (`http://frama-c.com/`). Frama-c already incorporates several static analysis techniques. One important technique, the so called value analysis, makes use of numerical abstract (interval) domains based abstract interpretation. In this assignment, you will focus on the value analysis feature of frama-c (eva, for evolved value analysis). You can check both the value analysis tutorial (`https://frama-c.com/download/frama-c-eva-manual.pdf`) and the tutorial of the frama-c tool itself (`https://frama-c.com/download/frama-c-user-manual.pdf`).

Value analysis deduces an over-approximation of the values each variable may assume. If the over-approximation still implies that no variable overflows, that no out-of-bound array index is used, or that some code is not executed, then such properties hold on all runs of the program. Remember this is a research framework targeting very difficult problems. For instance, the analysis builds on specific assumptions when it comes to allocating and manipulating memory. Such assumptions might hinder the value analysis from capturing certain bugs. Still the analysis is much deeper then just lexically matching keywords in the code.

Login to your account on `"thinlinc.edu.liu.se"`. Download and unzip the file `"sana.zip"`. Change to the obtained directory, create a symbolic link `"ln -s /courses/TDDC90/static/frama-c-gui.sh"`. and execute `"./frama-c-gui.sh -eva -main foo simple.c"`. This should launch the gui of frama-c and run the value analysis with the procedure foo as an entry point of the program (the

default being main). To the left, you have a list of the procedures in the file. To the right, you have the original file (cannot be edited from the gui, each time you want to make a change, you need to make the change in your favorite editor and then click "reparse" under the file menu). In the middle you have a reorganized version of your file, the correspondence should be clear. In the lower part of the gui, you can access different tablets. We will focus on one of the tablets, namely the **Values** tablet. By clicking on a variable in the middle window (say `i` after the assignment in `"simple.c"`) you will get the set of values computed by frama-c. This set is always an over-approximation. Choose the foo procedure from the left window and click on the value returned by foo. This value is included in the set $\{0, 10\}$. This means that this value **cannot be** something else than 0 or 10. Also, choose procedure `bar`. Parts of the code are colored in red. By clicking on it, the information tablet informs you this code is dead. It is never going to be executed no matter what the `read()` method returns as value for `i` in `foo()`.

The value analysis in frama-c achieves this by computing, as precisely as possible, over-approximations of the possible values. Sometimes, this over-approximations are strict. If you change the `(i>0)` condition in foo to `(i!=0)`, you see that the value assumed by `i` can be any integer, and hence frama-c value analysis alone cannot exclude the possibility of a division by zero. possibility of an overflow in `abs`. It still manages to establish that there is deadcode in `bar`. The lost precision can be recovered with other more expensive abstract numerical domains, but these are not discussed here. Instead frama-c can leverage on other approaches such as user or automatically generated and proved *Hoare triples* in order to formally establish such run time errors do not occur. There are three parts in this value analysis assignment.

Loops with a fixed number of iterations. This is a "warm-up" part. You do not have to answer this part, but it is recommended you go through it. Execute `"frama-c-gui.sh -eva -main fixed_repeat fixed_repeat.c"` from the command line.

1. Check the possible values of `x` and `y` inside the while-loop according to the value analysis. Do the values associated to `y` seem reasonable to you?
2. In the lower left corner of the Gui, under the "Eva" part, there is a parameter "slevel". Change it to 20. Re-run the analysis by clicking on reparse under the file menu. Observe that the new values associated to `y` are more precise.
3. Intuitively, "slevel" influences how many times a loop is unfolded (hence avoiding the precision loss entailed by the join at each iteration). This yields a more precise and more expensive value analysis.

Loops without a fixed number of iterations. You need to demonstrate and to answer this part. Execute `"frama-c-gui.sh -eva -main repeat repeat.c"` from the command line.

1. The analysis fails to show an assertion about the values of `y` and therefore prints it. This is how frama-c reports a possible error. What is this error?

This is a false positive. Explain what is a false positive and why this reporting is one.
2. Change "slevel" to 100. Re-run the analysis by chooising reparse under file. Did you get rid of the false-positive? Do you think changing "slevel" will help? Explain.

Array indices out of bounds. Execute `"frama-c-gui.sh -eva -main bounds bounds.c"` from the command line.

1. What are the possible values of `i` and `j` (in the respective loops) according to the value analysis? Are these strictly over-approximated or they possible?
2. Change `seq[100]` to `seq[10]`. Frama-c prints assertions because it fails to establish them. This is how it reports possible errors. What is this error? Is is a false positive or a true positive?

2 Symbolic Execution

The original idea is not new [3]. Put simply, constraints along specific paths are collected in order to check the possibility of the considered path, and to explore new paths by negating some of the constraints. The entailed checks are left for a satisfiability modulo theory solver (i.e., smt solver). Such a solver may establish unsatisfiability of the constraints representing a specific path, or may return values for the involved variables (input variables in particular). Many smt solvers adopt the SMTLIB common input format [4]. Z3 [2] is a well established and efficient solver. It is installed on `"thinlinc.edu.liu.se"` where you can invoke it from the command line.

Example 1 (Simple). The **assertion** in this simple program is **violated exactly when** the `then` branch of the `if` statement is taken with `result==b`.

```
#define CAP 5


int compute(int a, int b){


  int res=0;


  if((a % CAP) != (b % (CAP + 2))){
    res = a + 1;


    assert(res != b);
}
```

```
(declare-const CAP Int)
(assert (= CAP 5))

(declare-const a0 Int)
(declare-const b0 Int)

(declare-const res0 Int)
(assert (= res0 0))

(assert (not (= (mod a0 CAP)
                (mod b0 (+ CAP 2)))))
(declare-const res1 Int)
(assert (= res1 (+ a0 1)))
(assert (= res1 b0))

(check-sat)
(get-model)
```

We generate the constraint at the right of the table. The constraint encodes the execution of a path that violates the assertion. You call Z3 with `"z3 -smt2 if.smt2"`. The smt solver checks the satisfiability of the constraint and returns a satisfying valuation to the variables (hence a test that is guaranteed to violate the assertion).

In this assignment, you will manually encode paths of a simple method in a lottery program. Do not use the "if-then-else" construct (i.e. `"ite"`) in Z3. When encoding a path, use SSA form and do not forget to negate the assertion that is to be checked since we will use Z3 to check satisfiability. The `main` method in `"lottery.c"` chooses three numbers `a`, `b` and `c` in $\{0, 1\}$. The `check` method checks whether the triplet `(a,b,c)` is a winning combination and prints a corresponding message. One can encode each of the paths of the `check` method and establish, using Z3, which are the winning triplets and which are the losing ones. Your assignment is to:

1. Find out how many paths there are. Explain. (Just a number an an explanation)
2. Encode (as an input to Z3) the two following paths:
   - first, the path corresponding to the "then" branch followed by the "else" branch followed by the "then" branch, and
   - the path corresponding to the "else" branch followed by the "then" branch followed by the "else" branch.
3. By querying Z3, check which one, if any, of the two paths has a satisfiable constraints[1]? to which `(a,b,c)` triplets do they correspond to?

Note: Remember the (check-sat) and (get-model) commands at the end of your query, otherwise Z3 will not output anything.

References

1. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977.
2. L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.
3. J. C. King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.
4. S. Ranise and C. Tinelli. The smt-lib standard: Version 1.2. Technical report, Technical report, Department of Computer Science, The University of Iowa, 2006. Available at www. SMT-LIB. org, 2006.

[1] Observe that the (get-model) Z3 command returns an error in case the constraints are unsat. This is ok. If you comment the command out, Z3 will simply report whether the constraints are sat or not.