Introduction to Software Verification - HW No. 5

Winter 2022-2023

TA in charge of the HW: Roy Deutch

Pay attention: an answer without explanation will not be checked.

In questions 1 and 2, assume you have the function $SatSolve(\alpha)$ which gets boolean formula α (not necessarily a CNF formula) and returns SAT if α if satisfiable and UNSAT if α is not satisfiable.
The algorithms should be efficient as possible,
Pay attention, efficient means with a minimal number of iterations.

Question 1

Let $M = (S, I, R, L)$ be a Kripke structure over atomic formulas AP and
ill transitions $R_{ill} \subseteq R$.
We say that the path $\pi = s_0, s_1, \ldots$ is ***respecting*** R_{ill} if for every ill transition is used at most once, i.e, for any $(s, t) \in R_{ill}$ there are no $i \neq j$ such that:
$s_i = s, s_{i+1} = t$ and $s_j = s, s_{j+1} = t$.
M is represented by CNF formulas over the variables vector $\bar{v} = (v_1, \ldots, v_n)$:

- $S(\bar{v})$ - CNF formula which represents the states of M.
- $I(\bar{v})$ - CNF formula which represents the initial states of M.
- $R(\bar{v}, \bar{u})$ - CNF formula which represents the transition relation of M.
- $R_{ill}(\bar{v}, \bar{u})$ - CNF formula which represents the ill transition relation of M.
    - Such that: $R_{ill}(\bar{v}, \bar{u}) \rightarrow R(\bar{v}, \bar{u})$
- For all $p \in AP$, $p(\bar{v})$ is a CNF formula which represents the states that satisfy the atomic formula p.

A. We say that $(s, t) \in R$ is a healthy transition if it is not ill.

   Write a propositional formula $R_{healthy}(\bar{v}, \bar{u})$ which represents the set of healthy transitions. Explain the formula's correctness.

B. Assume that the formula you wrote in the previous section is correct and it is given to you.

   Let $p \in AP$. Write a BMC iterative algorithm, efficient as possible, which returns $true$ if there is a path π in M which respects R_{ill} and satisfies $M, \pi \models Gp$, and returns $False$ otherwise.

Question 2

Let M be a Kripke structure represented by CNF formulas over the variables vector $\bar{v} = (v_1, \ldots, v_n)$:

- $S(\bar{v})$ - CNF formula which represents the states of M.
- $I(\bar{v})$ - CNF formula which represents the initial states of M.
- $R(\bar{v}, \bar{v}')$ - CNF formula which represents the transition relation of M.
- $p(\bar{v})$ - CNF formula which represents the states that satisfy the atomic formula p.
- $q(\bar{v})$ - CNF formula which represents the states that satisfy the atomic formula q.

Suggest a BMC iterative algorithm, efficient as possible, which returns $true$ if exists initial state $s_0 \in I$ such that $M, s_0 \vDash EG(EXp \wedge EXq)$, and $false$ otherwise.

Damp part (wet, but only a little)

As you (probably) already know, lately the “Atudaim’s national forum” was founded.
Its goal is to help all the Atudaim in Israel (for more information: https://forumatuda.wordpress.com). the forum’s Special Operations Department accepted to take one of the most significant missions of the organization - Decreasing the career service’s length for the Atudaim. The project attracted interest among the forum, and in a huge effort, the access control’s source code of the Atudaim department (hack_me.c) which attached to the assignment was attained. In this part, we will try to hack the given access control’s system, in order to change the necessary data in the Atudaim department’s database.

For the system’s research, Computer Science students were recruited. While students from Network Security course planning attacks the code, students from RE course decompiling the source code, and students from Parallel and Distributed Programming course debugging hacking attempts with a lot of servers, you were asked, as a student from Software Verification course, to use the CBMC tool to hack the system.

1. Now we will get to know the system. First, look at the code. As you see, students from the System Programming course (which one of the main lessons from it, is exaggerated knowledge about defines) changed it and made it unbearable. In this section there is no need to decode or understand what the function verify is doing. You need to refer to it as a black box that computes some function. **Solutions that rely on understanding the algorithm that is executed in the function verify, beyond what was described in this section, will not be accepted.**

   Look at the general structure of the code, and try to compile and run it.
   Pay attention, in this section, you shouldn’t change the code at all.
   Understand and write shortly how to use the system, how it gets inputs,
   and what is a correct input for it.

2. When a user signs up to the system, they pick a new username, and as result, they get a log-in password. In this section, we want to find log-in passwords for different usernames. Specifically, for every submission, we will refer to your id as a username and we want to find the password for your id (for example, if the students who submit this HW are Daniel Kimel, id 123456789, and Liam Zolotnitzki, id 987654321, we want to find the password for 123456789 and the password for 987654321). You need to use the CBMC tool (the instructions are at the end of this assignment) in order to find the passwords.

   To solve this section, you need to change the hack_me.c file, when you can change only the code in the main function. In particular, you mustn’t change the code in the function verify. We’ll say again, you need to use only the CBMC tool in this section, and in particular use the SAT SOLVER abilities it has, and every method including decoding the code can’t be used.

You need to present the changes which you did to the code (e.i. - present in the submission, the code parts you have changed. You don't need to present the unchanged code parts). You need to explain what you have changed, and how it helped you to solve this section.

3. The project's team found that beyond the personal passwords, there is a "master password" - a password that is suitable for all the users. You need to find it using the CBMC tool. The restrictions from section B are applied in this section as well.

   Pay attention: this section has two parts. You need to **find** another password, different from the one you found in section B, which is suitable for all the users. Then, you need to prove by using the CBMC tool, that it is indeed the master password, e.i. - it is suitable for all users.
   You need to explain what are the changes you have done to the code, and how they help you find the master password.

4. You need to submit the passwords you obtained in the previous sections, in csv format in a file named passwords.csv, in the next format:

   Id1, password1
   Id2, password2
   Master, master_password

5. The Atudaim department realized that the system was hacked (there has been a noticeable increase in the rate of smiling Atudaim), you was been recruited to help the department upgrade the system. In one of the modules of the system, the department executes a binary search in an array, by the given function in binsearch.c file. You need to use the functionality of the CBMC tool in order to check if the array accesses in the function are valid.
   a. The flag for this check is --bounds-check. Try to run the CBMC tool with this flag only on the function in binsearch.c (use the flag -- function). What is the result you got? Compare this run to the run on predictable.c with the same flags.
      Bonus: suggest a reasonable explanation to the differences between the two runs
   b. Try to run this check with an unwinding of 1 (--unwind 1). What is the result? Use the flag --no-unwinding-assertions to check if the verification with the current bound on the wind, succeeded or not. What is the flag's meaning? What you can conclude from this run about the program correctness?
   c. Now try increasing the bound on the unwinding. Wath is the result? What you can conclude from the new run about the program correctness?

CBMC - instructions

The CBMC tool is installed on the csl3 server, the tool can be found in:
usr/local/cbmc/cbmc
We recommend you to run alias in order to make the use in CBMC tool easier:
Alias cbmc=usr/local/cbmc/cbmc
To run the tool on the c file named example.c, you need to run the next line:
Cbmc example.c
For more information, you can get help by:

- Run cbmc --help.
- The tutorial slides.
- Google.

Pay attention that the tool can be used in other platforms as well, as noted in the tutorial.

Good luck!