Software Analysis and Verification - Fall 2025

Ruzica Piskac

September 9, 2025

The deadline for Project 01 is September 30, 11:59PM ET.

You are encouraged to work with a partner.

1 Verification Condition Generation

Build a verification condition generator (VCG) based on computation of weakest preconditions, as described in the lectures.

Your VCG should compute verification conditions, based on the code and provided annotations. To prove the resulting formulas, connect your VCG to a theorem prover to prove the verification conditions. You should use SMT-LIB format and an SMT solver of your choice (e.g., Z3 or cvc5).

We developed a code skeleton for Haskell, located in the files as proj1-haskell.zip. The code skeleton should contain the necessary information for building itself. While we are not aware of any bugs in the code skeleton, we do not guarantee it is bug free. If there are any differences between the language specification and the behavior of the code skeleton, the language specification is correct.

Check , on Canvas, to understand the syntax used to define the programs that you will verify.

2 Generic instructions

Start by modifying the parser to handle logical assertions, pre- and post-conditions and loop invariants (inspire your code from what is already in the skeleton). After that, write a recursive function that goes through the program AST and computes the weakest precondition. Only at the end, write a printing function that outputs the verification condition using the SMT-LIB format, and write code to interact with the SMT solver.

3 Input

Your tool should output “Verified” all postconditions and loop invariants have been verified. Otherwise, it should output “Not verified”. Your output should not contain any other characters, EXCEPT it may contain arbitrary whitespace. If you do not follow these instructions, you will be asked to resubmit.

4 Other information

It is possible that we issue some amendments to the language or that I add extra material. If so, I will signal it in a Canvas announcement.

Additionally, keep the benchmarks that you use to test your code. You should submit five (or more) original benchmarks by alongside your project, so that we can prepare them for the competition. Your benchmarks should include at least five programs in the original IMP language with loops and annotations. Please try to write programs that do something interesting, e.g., sort an array, as opposed to programs that are meaningless/artificially constructed.

Notes on the language:

1. Double assignment to arrays is not supported. (i.e., you cannot write a[0], a[1] = a[1], a[0])