SWEN90010: HIGH INTEGRITY SYSTEMS ENGINEERING
DEPARTMENT OF COMPUTING AND INFORMATION SYSTEMS
THE UNIVERSITY OF MELBOURNE

Workshop SPARK #2

Introduction

In the lectures, we discussed proving properties of small programs using the SPARK Prover. In this workshop you will apply those same ideas to a simple arithmetic program.

Exercises

The focus of this workshop is a SPARK program for computing integer division and remainder via repeated subtraction. A small package `DivMod` is provided to perform this calculation, as well as a main driver program `main.adb`.

1. Open the project, using the provided `default.gpr` project file. Compile the source code the GPS environment, run it a few times, providing different inputs until you understand what the program is computing.

2. Now run the SPARK prover: *SPARK → Prove All.* You will see that the provided `main.adb` contains a number of `pragma Assert` statements. These are assertions that the SPARK prover tries to prove always hold.

   You will see that the assertion `X = K * N + R and R < N` in `main.adb` cannot be proved. (You may also see potential problems reported in the `DivMod` package, but we will come to those later.)

   This is because the `DivMod` procedure has no contract (pre/postcondition annotations), so the SPARK prover cannot tell anything about `K` and `R` after it is called.

3. Add a postcondition annotation to the `DivMod` procedure in `divmod.ads` to allow the failing assert to be proved. *Hint: this postcondition should state what is true about `K` and `R`, in terms of `X` and `N`, after `DivMod` returns.*

4. Now run the SPARK Prover again. Now the assertions in `main.adb` should be able to be proved, using the contract on `DivMod`. However, the SPARK prover cannot actually prove that the contract holds.

   It also cannot prove that the loop in `DivMod` won't cause integer overflow.

5. To help it prove these, we need to add a suitable loop invariant annotation for the while-loop in `DivMod`.

   To work out what the invariant should say, you can add print statements to this loop to get it to print out the values of `Y` and `K` each time through the loop. Then look for a relationship that always holds between `Y`, `K`, `N` and `X`.

   Once you have figured out the invariant, add an appropriate annotation to the while-loop:
   `pragma Loop_Invariant (`*... your invariant goes here ...*`);`

6. Now re-run the SPARK prover. If your invariant is correct, you should find that the SPARK prover does not report any problems. You have proved the correctness of your first program. Congratulations!

7. *If you have time:* Look at the assert statements in `main.adb` more closely. The final one asserts that `X / N = K`, i.e. that `K` does in fact hold the result of performing integer division on `X` by `N`.

   Try commenting out each of the assert statements above and re-running the SPARK prover for each. You should find that when one of these assertions is commented out, one of the following assertions cannot be proved.

This means that, to prove that following assertion, the SPARK prover first needs to know that the preceding one holds, i.e. it cannot derive the following assertion in one go but it needs some help: we first have to tell it to derive the intermediate assertion and, only then, can it derive the subsequent one.

This can sometimes happen with automated provers like the SPARK prover. Using intermediate assertions like this can be a useful way, therefore, helping to derive extra facts that cannot be inferred automatically.