FITR304: The Frama-C platform

Author : Christophe Garion <garion@isae.fr>
Audience : IN
Date :

In order to use Frama-C, you have to execute the following commands (or add to your `.bashrc`):

```
module load opam-softs
opam-init
```

The path to your personal repository for this lab session is `https://eduforge.isae.fr/repos/FITR304/deductive/your.login.name`. The `src` directory of your repository contains the source files you have to use.

1. **A first example**

   The aim of this exercise is to study a first example. Please follow your instructor's instructions.

   (a) use the `max.c` file (without assertions) and use the WP plugin of Frama-C on it. What do you observe?

   (b) use the `basic-annotated-max.c` file and use the WP plugin of Frama-C on it. What do you observe? Can you use the `\max` ACSL function in the specification?
   Change the implementation and verify that it always works.

   (c) use the `basic-annotated-max-2.c` file and use the WP plugin of Frama-C on it. What do you observe? Add RTE verification. Are there goals that are not proved? How can you solve the problem?

   (d) use the `annotated-max.c` file and use the WP plugin of Frama-C on it. What do you observe? How can you solve the problem?

2. **Floyd-Hoare basic constructs**

   The aim of this exercise is to write assertions in a simple program to understand how the Frama-C toolchain works. Use the `basics.c` file in your directory.

   (a) what do you think of the assertion `//@ assert a == 6;` in the file? Add some assertions after the assignment and the conditional (particularly on the value of `c`). Are they proved?

   (b) add an assertion after the while loop to verify that `a` is equal to `n` and that `c` is equal to $\frac{n \times (n-1)}{2}$.

3. **Factorial**

   Let us consider the following algorithm to compute factorial (beware, it is not the same than during the lecture!):

   ```
   {N ≥ 0}
   K := 1;
   F := 1;
   while (K ≤ N) do
         F := F * K;
         K := K + 1
   od
   {F = N!}
   ```

   (a) complete the `fact.c` program in your repository with annotations to prove that `fact.c` is correct. What should be the invariant if you want to prove the postcondition of the program? What happens if you forget the `assigns` statements?

   (b) enable RTE to check for runtime errors. Can you easily solve the problem?

4. **Hello, Euclid!**

   Remember the previous algorithm on greatest common divisor:

```
{A > 0 ∧ B > 0}
X := A
Y := B
while (X ≠ Y) do
      if (X > Y) then
            X := X - Y
      else
            Y := Y - X
      fi
od
{X = Y ∧ X > 0 ∧ X = gcd(A, B)}
```

Complete the `gcd.c` program in your repository with:

- an axiomatic characterization of GCD (see `fact.c` for instance)
- annotations to prove that `gcd.c` is correct.

5. **Divide ut imperes**

   Remember the previous algorithm on integer division:

```
{X ≥ 0 ∧ Y > 0}
Q := 0
R := X
while (Y ≤ R) do
      Q := Q + 1;
      R := R - Y
od
{X = Q × Y + R ∧ 0 ≤ Q ∧ 0 ≤ R < Y}
```

   (a) complete the `division_1.c` in your repository with annotations to prove that it correctly computes the euclidean division. What do you think of this program's coding style?

   (b) complete the `division_2.c` in your repository with annotations to prove that it correctly computes the euclidean division.

6. **Factorial for all**

   (a) consider now the `fact_tab` function defined in `fact_tab.c`. Simply complete the existing annotations. What happens when you try to prove it? Why?

   (b) add annotations to prove the `fact_tab` function. Beware of the `assigns` clause, try different solutions: assign the « right » element of the array, all the indexes previously assigned and all the array.

7. **Completing specifications…**

   Look at the `swap.c` file. It contains a `swap` function and its specification.

   (a) do you understand the specification?

   (b) there are **3** specification errors or omissions in `swap.c`. Find them and prove the program.