Static analysis of programs with Frama-C and **the WP plugin**
Lab Session

Olivier Hermant [*]

september 2020

Résumé

Those exercises aim at getting more familiar with Frama-C and **the** WP **plugin**. You will specify, implement and verify some small algorithms adapted from standard libraries of |C| and |C++|. Do not hesitate to ask questions.

1 Introduction

In the following exercises, you will use Frama-C together with its WP plugin in order to verify some small C programs.

1.1 Specifying, Coding and Verifying in a Formal World

In the following exercises, you shall make a formal specification of your algorithm, implement it and check that your implementation really matches your specification. The first two phases are part of the well-known *V* development cycle (and others), even if the specifications we will give are far less informal. The verification phase replaces or, usually, supplements the tests phase in the case of critical software.

You will have a first overview of the additional amount of work required by a formal verification of an implementation. The confidence we gain through the use of formal methods demands a certain involvement from the user, even if this work is more and more automated.

You will see however that it will become step by step more natural to specify and verify your programs.

[*] olivier.hermant@mines-paristech.fr

1.2 Using Frama-C and the WP plugin

What has to be reminded for these tools (as well as for other tools) is the address of the documentation. For Frama-C, you need to type in a terminal :

```
frama-c -help
```

and for Jessie (the WP plug-in), to type :

```
frama-c -wp-help
```

Moreover, you will find a quick tutorial (a bit outdated) at `http://frama-c.com/jessie_tutorial_index.html` and the complete manual and resources at `https://frama-c.com/support.html`. This can be an excellent complement of what you have already seen.

Lastly, if you have a C file, say `max.c` correctly written and specified in ACSL, you can launch the analysis in the terminal by typing :

```
frama-c -wp example.c
```

You can also enjoy the graphical user interface, that you can get this way :

```
frama-c-gui -wp example.c
```

for an automatic analysis (orange dots : not able to prove the invariant, green dots : OK) and

```
frama-c-gui example.c
```

for an analysis by hand (need to right-click the annotation and to ask the WP plugin to prove it). Finally, if you have a version of alt-ergo newer, than `0.99`, it might be necessary to add the option `"-backward-compat"`, for backward compatibility o the output of alt-ergo with Frama-C, like this (and similarly for other commands)

```
   frama-c --wp -wp-alt-ergo-opt='-backward-compat' example.c
```

2 Finding the minimum

We are going to look at the minimum in a sequence of integers. This function has the following signature :

```
int min_element(const int *a, int n);
```

The function finds the lowest element in the interval `a[0,n[`. More precisely, it returns the smallest valid index `i` such that :

1. for all index `k` such that $0 \leq k < n$, we have $a[k] \geq a[i]$;
2. for all index `k` such that $0 \leq k < i$, we have $a[k] > a[i]$.

The value returned by `min_element` is `n` if and only if there is no minimum (that is to say, when `n == 0`).

1. propose a formal specification. You can use behaviors. (`behavior`).
2. propose and implementation.
3. Verify that your algorithm is correct. It might well be that you need to state explicitely some loop variants and invariants, in order to succeed.

3 Research of an element

Here we want to implement an algorithm performing *sequential search* on a sequence of integers. The signature of the function is :

```
int find(const int *a, int n, int v);
```

This function gives the smallest valid index `i` of `a` such that :

```
a[i] == v
```

is true. Otherwise, it returns `n`, the length of the array.

1. propose a formal specification in ACSL.
2. propose an implementation.
3. check that your implementation corresponds to its specification.

4 Equality

Let us continue with an equality test on two arrays of integers. It means that you need to specify, implement and verify a function that corresponds to the following signature :

```
int equal (const int *a, int n, const int *b);
```

This C function returns the value `1` if both arrays `a` and `b` are pointwise identical (at each index, elements are equal). Otherwise (at least one element of `a` and `b` is different), it returns `0`.

1. propose an ACSL specification of the function `equal`. You can ask yourself the following questions in order to define your specification :
   (a) What are the prerequisites of the function ?
   (b) What does it assign ?
   (c) Can you make distinction between different expected `behaviors` ?
   Those are standard questions that you can ask yourself systematically at specification time, in particular for the next exercises.
2. Propose an implementation of this function. You probably shall need to specify the loop variants and invariants that correspond to your implementation, in order to succeed in the next step.
3. Verify that your implementation and your specification correspond to each other.

5 Fill an array

This is our first example of a function that modifies the values of an array, admittedly in a very simple way : it fills an array of size `n` with a default value `val`. This function has the following signature :

```
void fill(int *a, int n, int val);
```

1. propose a specification ACSL of its behavior.
2. implement this function.
3. verify that your implementation matches your specification. You probably will need to specify loop variants and invariants.

6 Copy

The copy function allows, as indicated by its name, to copy the content of a sequence in another one. We restrict the application of this function to duplication of integer arrays. So, the signature of the function is :

```
void copy(const int *a, int n, int *b);
```

This function copies the elements from `a` to `b`. In an informal way, each element of `a` is copied at the same place in `b`.

1. Specify this function with ACSL.
2. Implement `copy`.
3. Check your implementation. Beware the loop invariants and variants!
4.

(a) Is your implementation correct with respect to the specification if `a` and `b` are partially superposable ? (*i.e.* they share a memory zone) ?

(b) How to tell the WP plugin, that both arrays must be in two separate memory regions ?

(c) were you able to prove the soundness of your implementation, at the beginning ?

7 Iota

`iota` is a function ot the STL C++ library. It assigns strictly increasing values to a sequence. The starting value is given by the user. The signature we will be using for `iota` is :

```
void iota(int *a, int n, int val);
```

Starting from the initial value `val`, this function assigns increasing values to the array `a`.

1. Specify formally `iota` in ACSL. How to specify the behavior of this function in the case when the assigned integers go beyond the limits of your computer ?
2. Implement the function.
3. Check your implementation against your specification.