Report on the project of

Formal Approaches

- Submitted by: Jagruti Kapgate (10978)

Introduction:

The project Formal Verification of a sorting function aims at finding a Frama-C proof for given sorting algorithm programmed in C language. Frama-C is a set of interoperable program analyzers for C programs and a tool to perform static analysis on them. Static analysis of source code is the technique of verifying the source code without executing it. The 3 types algorithms are provided i.e. Plain numbers, Three numbers and Four numbers. I have chosen *sort2.c* from Plain numbers.

Current state of verification:

In each sort algorithm, functions like *sort()* or *swap()* work on similar principle. The key difference is how loops are written. In the chosen file *sort2.c*, the *sort()* function has two loops: both the loops are for loops. First one is running from $i = 0$ to $i = l$ and the second one is a running from $j = 0$ to $j = (l - 1)$.

In my verification, *sort()* has a predicate sorted, which verifies the output being generated is actually a sorted array.

```
/*@ requires \valid(t + (0 .. l - 1));
    requires l > 0;

    behavior sorted:
      ensures
        ∀ ℤ a;
          0 ≤ a < \old(l) →
          (∃ ℤ b;
             0 ≤ b < \old(l) → \old(*(t + b)) ≡ \at(*(\old(t) + a),Here));
      ensures sorted(\old(t), \old(l) - 1);

    complete behaviors sorted;
    disjoint behaviors sorted;
 */
void sort(int *t, int l)
```

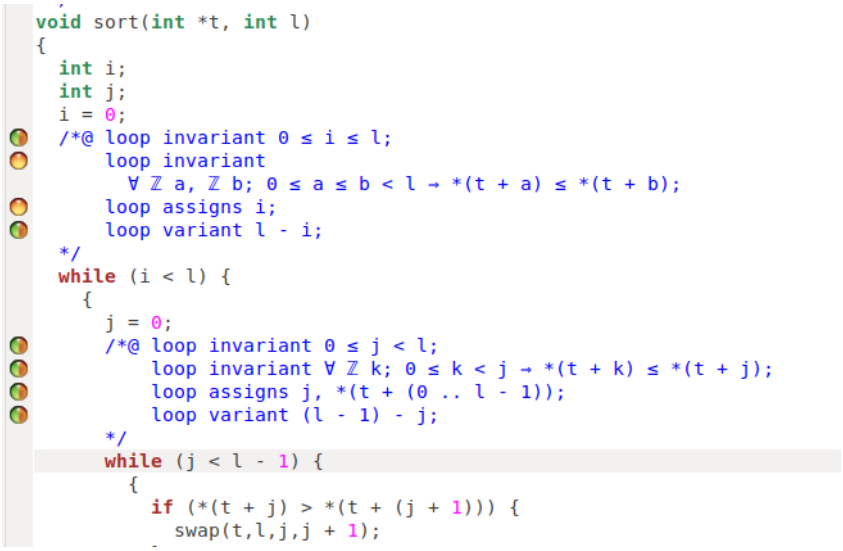
The *swap()* function is totally verified in my proof.

```
/*@ ensures \old(*(t + i)) ≡ *(\old(t) + \old(j));
    ensures \old(*(t + j)) ≡ *(\old(t) + \old(i));
    assigns *(t + i), *(t + j);
 */
void swap(int *t, int l, int i, int j)
{
  int tmp;
  tmp = *(t + i);
  *(t + i) = *(t + j);
  *(t + j) = tmp;
  return;
}
```

Future improvements:

The code in current state has two problems:

1. Predicate sorted is partially verified.
2. The outer loop is not verified because the loop invariant is not working and loop assigns is showing error.



```
void sort(int *t, int l)
{
  int i;
  int j;
  i = 0;
  /*@ loop invariant 0 ≤ i ≤ l;
      loop invariant
        ∀ ℤ a, ℤ b; 0 ≤ a ≤ b < l → *(t + a) ≤ *(t + b);
      loop assigns i;
      loop variant l - i;
  */
  while (i < l) {
    {
      j = 0;
      /*@ loop invariant 0 ≤ j < l;
          loop invariant ∀ ℤ k; 0 ≤ k < j → *(t + k) ≤ *(t + j);
          loop assigns j, *(t + (0 .. l - 1));
          loop variant (l - 1) - j;
      */
      while (j < l - 1) {
        {
          if (*(t + j) > *(t + (j + 1))) {
            swap(t,l,j,j + 1);
```

To carry on the future improvements, I will read the reference documents and try to program for the loops correctly.

To conclude, I will say this project introduced me to the new tool and the new way of code analysis. By working on this project, I learned how to use this tool to analyze the c program.

References:

1. Frama-C official website and documentation
2. Stack Overflow
3. Allan Blanchard
4. https://frama-c.com/download/publications/tutorial_tap2013_slides.pdf