Logical verification

Kjell Zijlemaker, student number: 2603017

January 2019

1 Introduction

For the formalization project I have chosen to formalize the mergesort. This algorithm has a good performance of O(n log n) [1] and has been invented by John von Neumann in 1945 [1].

The reason that his sorting algorithm has been chosen is because of this. It's fairly old but delivers good performance, is stable, and is still used in a lot of projects nowadays.

2 Reflection

Of course there were some problems while formalizing the mergesort. First of all, I was not able to fully formalize the mergesort, but was able to partially formalize the merge definition, though it is not the same as the mergesort itself.

Also, some of the basic proves were deemed to be difficult enough. This is why at some of the proves: "sorry" has been written. The greatest problem with this was the concatenation of the merge. For example proving: merge (x :: xs) xs = merge xs (x :: xs).

Although I have not finished the project in its total, I am grateful to have learned so much in just 2 months. I did not had any knowledge about functional programming and a little bit about recursion, so I had to learn these things quickly before I could follow up to program in Lean. Thankfully, I was able to complete all assignments within time and even before the exam. This was already an accomplishment for me. To sum it all up, it was hard and maybe I did not passed the exam, but it was great to have come so far.

In the next sections I will explain my attempt on formalizing the mergesort.

3 Definitions

Some of the main definitions for this formalization project are the following:

- Creating two halves to be merged together when sorted
- The merge definition which will merge two lists together

- The mergesort itself which will merge two sorted lists together recursively

Some extra definitions have also been made, which are then used to prove some definitions:

- The reverse definition which will reverse a list
- The concat definition which will concatenate two lists togther

4 Properties

For proving some of the more advanced properties, first basic properties have to be defined. These were the following ones:

- Appending two reverse lists together, which is required when proving reverse(xs ++ ys) = reverse xs ++ reverse ys
- Concatinating two lists is the same as appending one list with a singleton list, which is required when proving the reverse(reverse(n))

4.1 Reverse properties

Before I wanted to prove the merge with reverse properties, I first proved some basic reverse properties itself. These were the following:

- reverse (xs ++ ys) = reverse xs ++ reverse ys, which will help to prove the property below
- reverse (x::xs) = reverse xs ++ (x :: list.nil)
- reverse (reverse n) = n, which has been a test before I wanted to prove the reverse with the mergesort

4.2 Merge properties

For the merge properties I first proved some of the most basic ones, namely:

- merge ([] : list nat) n = n
- merge n ([] : list nat) = n

Both are required in the double reverse of the merge: reverse(reverse(merge n m)) = merge n m

Other properties simple properties also have been proven:

- merge xs (x::xss) = merge (x::xss) xs, which is required for the property below
- merge m n = merge n m, which proves that the two merges are commutative, i.e. it does not matter if we merge [2,3,4] [4,3,2] or [4,3,2] [2,3,4]

5 Unfinished

Some parts of the sorting algorithm were not finished. This subsection will attempt to give more information about them.

5.1 Mergesort

The merge algorithm itself of course does not merge the two sorted lists, this is the job of the mergesort algorithm. The definition of the algorithm was as follows:

```
def mergesort : list nat -> list nat
| [] := []
| [a] := [a]
| (x :: xs) :=
  have list.sizeof (fhalf xs) < x + (1 + list.sizeof xs), from sorry,
  have list.sizeof (sndhalf xs) < x + (1 + list.sizeof xs), from sorry,
  merge (mergesort (fhalf xs)) (mergesort (sndhalf xs))
```

As can be seen, some properties have to be proven because it cannot prove if the sorting algorithm is decreasing by itself. Therefore, these properties have to be proven and are now annotated with 'sorry'.
Unfortunately I was not able to prove these two properties and therefore the mergesort could not be used within the project.

References

[1] T. T. J. Katajainen, Jyrki; Pasanen, "Practical in-place mergesort," *Nordic Journal of Computing*, vol. 3, pp. 27–40, 7 1996.