Formal Verification of the Correctness of a Bubble Sort Implementation Using Coq

Jamshed Khan
University of Maryland
jamshed@cs.umd.edu

1 Proposal

In this project, I proposed to implement the bubble sort algorithm with the *Gallina* language embedded in `Coq`'s logic; and then provide a formal verification of the correctness of the implementation using `Coq`.

2 Final Deliverable

The bubble sort algorithm has been implemented in a functional manner. Then the correctness of the implementation has been formally verified using `Coq`. The complete code-base is available at this GitHub repository. Particularly, the files Perm.v and BubbleSort.v is pertinent to the project proposal, and the other `.v` files are preparatory materials used along the way for learning purposes.

3 Overview of the Algorithm Implementation

Provided a list, the classic bubble sort algorithm [Cormen *et al.*, 2009] repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted. Informally, at each step the smallest or the largest element "bubble" to the top (head or tail) of the current list; i.e. each pass places one element at its correct position in the sorted order. For a list of length n, typical implementations require $(n-1)$ passes; and the i'th (0-based) pass operates on a list of length $n-i$.

At the project implementation, each "bubble pass" repeatedly swaps two adjacent elements if the pair of the elements is in a decreasing order, starting from the tail of the provided list. Hence, a pass will make one of the smallest elements in the list "bubble" its way to the head of the list. The bubble sort algorithm repeatedly invokes this "bubble pass" on lists of progressively shorter lengths, bottoming out at empty lists.

Specifically, three functions are defined:

- $(bubble\_pass\ l)$ that makes a pass over the provided list l to "bubble" one of the smallest elements to the head of the output list;
- $(bubble\_sort'\ l\ n)$ that provided a list l and an integer n, invokes the $bubble\_pass$ function n times on l (each on a progressively shorter list by throwing out the head);
- and the wrapper $(bubble\_sort\ l)$ that just invokes $bubble\_sort'$ on the list l with $n = length(l)$.

4 Overview of the Formal Verification

The definition of the proposition defining sortedness is used as is in *Verified Functional Algorithms* [Appel, 2018]. Two properties of the implementation has to be formally verified: given a list l, (i) the result of the sort algorithm is a permutation of l; and (ii) the resultant list is sorted.

Note that, many existing lemmas and theorems, mainly from the `Permutation` library has been used along the way, and is not specifically mentioned here.

4.1 Verification of the Permutation Property

The most important lemma for the proof of this property is:

Lemma 1. *For a list l, $(bubble\_pass\ l)$ is a permutation of l.*

Then the following trivial helper lemma is proved:

Lemma 2. *For a list l and a natural number n, if $(bubble\_sort'\ l\ n)$ is empty, then l is empty.*

Using the lemmas 1 and 2, the following lemma is then proved:

Lemma 3. *For a list l and a natural number n, if the length of l is n, then $(bubble\_sort'\ l\ n)$ is a permutation of l.*

Wrapping up the lemma 3, the permutation property is proved in the following theorem:

Theorem 1. *For a list l, $(bubble\_sort\ l)$ is a permutation of l.*

4.2 Verification of the Sortedness Property

The following trivial helper lemma is required:

Lemma 4. *For a list l, if $(bubble\_pass\ l)$ is empty, then l is empty.*

Using lemma 4, the following important lemma is proved:

Lemma 5. *For some lists l and l' and natural numbers x and y, if $(bubble\_pass\ l = x :: l')$ and $y \in l$, then $x \leq y$.*

Then the following helper lemma is proved:

Lemma 6. *For a list l and a natural number x, $x \in l$ if and only if $x \in (bubble\_pass\ l)$.*

Using lemmas 4 and 6, the following crucial lemma is proved.

Lemma 7. *For a list l and a natural number x, if $(bubble\_sort'\ l\ (length\ l))$ is sorted and $\forall z \in l, x \leq z$, then $(x :: bubble\_sort'\ l\ (length\ l))$ is sorted.*

Then using the lemmas 4, 5, 6, and 7, the following is proved:

Lemma 8. *For a list l and a natural number n, if the length of l is n, then $(bubble\_sort'\ l\ n)$ is sorted.*

Wrapping up the lemma 8, the sortedness property is proved in the following theorem.

Theorem 2. *For a list l, $(bubble\_sort\ l)$ is sorted.*

The theorems 1 and 2 together proves the correctness of the implemented bubble sort algorithm.

5 Discussion

An ideal design to build formally verified correct software using `Coq` consists of writing the program using the embedded *Gallina* language, and then prove the program correct using `Coq`'s proof theory. At this course, we have extensively learned modeling and analysis techniques for programs and programming languages using the `Coq` proof assistant, and how to leverage those tools to mechanically reason about programs. `Coq` provides the necessary frameworks to program formal systems and assisted correctness proving for those. Having quite extensively exposed to the logical verification frameworks of `Coq` early in the course, I intended to deeply explore the formal verification component of the course.

One great takeaway from completing this project has been that, throughout the course we learned a lot about analysis and proof techniques for program properties using `Coq`, and the exercises mostly consisted of filling out some empty pieces of a larger construct. Specifically, most of the exercises required proving some **provided** properties or statements about program constructs. But the for the project, the case has not been like this. The property statements that would help proving the final two major goals had to be **designed** on my own, and this spiraled out by requiring to prove a good number of lemmas, that were not thought out initially. I think this has enriched my knowledge on functional algorithms verification to a great extent, by putting together most of the pieces learnt at the part of the course pertaining to logical foundations into proving self-designed lemmas and theorems. This aspect of designing required lemma and theorem statements gradually to build up to the final goal has been challenging, nevertheless enjoyable.

6 Resources

The primary learning materials and other relevant resources are present at *Software Foundations Volume I* [Pierce *et al.*, 2018] and *Verified Functional Algorithms* [Appel, 2018].

References

[Appel, 2018] Andrew W Appel. *Verified Functional Algorithms*. Software Foundations series, volume 3. august 2018.

[Cormen *et al.*, 2009] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[Pierce *et al.*, 2018] Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, and Brent Yorgey. *Logical Foundations*. Software Foundations series, volume 1. Electronic textbook, May 2018. Version 5.5. http://www.cis.upenn.edu/ bcpierce/sf.