Formal Verification of a Selection Sort Algorithm in Dafny

Zitian Wang

April 2, 2025

1 Introduction

This report presents the formal verification of a selection sort algorithm implemented in Dafny. The aim is to prove that the algorithm sorts the array in non-decreasing order and preserves the multiset of elements. This assignment demonstrates the application of deductive verification techniques and the use of Dafny's specifications, invariants, and assertions.

2 Algorithm Selection

The algorithm chosen for this verification is selection sort. In each iteration, the algorithm finds the minimum element from the unsorted portion of the array and swaps it with the first unsorted element. Selection sort was chosen because:

- Its structure is simple, allowing for clear and modular verification.
- The invariants are relatively straightforward compared to those needed for algorithms like insertion sort or quicksort.
- The modular design allows us to verify helper methods (such as the function to find the minimum index) independently.

3 Specification and Implementation

The implementation is structured as follows:

- **Predicates:** Two predicates are defined:
  - isSorted asserts that the entire array is sorted.
  - isSortedPrefix asserts that the first n elements of the array are sorted.
- MinIndex: The `MinIndex` method locates the minimum element in the subarray starting from a given index. Its correctness is ensured by loop invariants that establish the minimum element condition.
- SelectionSort: The main sorting method uses a loop invariant that ensures:
  1. The prefix of the array (of length `pos`) is sorted.

2. Every element in the sorted prefix is less than or equal to every element in the unsorted portion.

The algorithm repeatedly swaps the minimum element from the unsorted portion into the first unsorted position.

4 Verification Process

Dafny automatically generates verification conditions (VCs) based on the specifications, loop invariants, and assertions. These VCs are then discharged by the SMT solver, ensuring that:

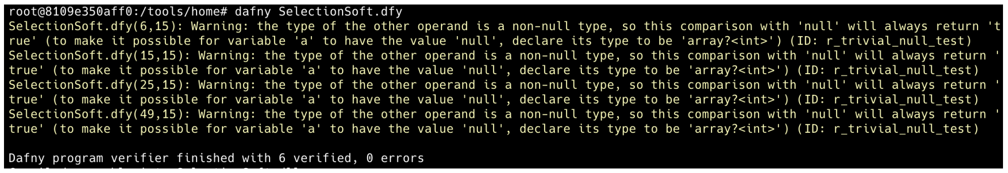
- The loop invariants hold at the beginning and end of every iteration.
- The postconditions are satisfied once the algorithm terminates.
- The `MinIndex` method correctly identifies the index of the minimum element in the specified subarray.

5 Discussion and Challenges

One challenge in formal verification is designing invariants that are strong enough to prove the desired properties but not so complex that the SMT solver struggles to discharge them. In this project, selection sort was chosen over insertion sort because its invariants are more natural and easier to manage. Additionally, the modular design of helper functions like `MinIndex` simplifies the overall verification process.

6 Conclusion

The verified Dafny implementation of selection sort demonstrates that the algorithm correctly sorts an array while preserving its multiset of elements. This project illustrates the effectiveness of deductive verification techniques in ensuring program correctness and provides a basis for verifying more complex algorithms in future work.



```
root@8109e350aff0:/tools/home# dafny SelectionSoft.dfy
SelectionSoft.dfy(6,15): Warning: the type of the other operand is a non-null type, so this comparison with 'null' will always return 't
rue' (to make it possible for variable 'a' to have the value 'null', declare its type to be 'array?<int>') (ID: r_trivial_null_test)
SelectionSoft.dfy(15,15): Warning: the type of the other operand is a non-null type, so this comparison with 'null' will always return '
true' (to make it possible for variable 'a' to have the value 'null', declare its type to be 'array?<int>') (ID: r_trivial_null_test)
SelectionSoft.dfy(25,15): Warning: the type of the other operand is a non-null type, so this comparison with 'null' will always return '
true' (to make it possible for variable 'a' to have the value 'null', declare its type to be 'array?<int>') (ID: r_trivial_null_test)
SelectionSoft.dfy(49,15): Warning: the type of the other operand is a non-null type, so this comparison with 'null' will always return '
true' (to make it possible for variable 'a' to have the value 'null', declare its type to be 'array?<int>') (ID: r_trivial_null_test)

Dafny program verifier finished with 6 verified, 0 errors
```

Figure 1: Screenshot

A insertionSort code without Comment

Listing 1: MySelectionSort.dfy

```
module MySortModule {

  predicate isSorted(a: array<int>)
    requires a != null
    reads a
  {
    isSortedPrefix(a, a.Length)
  }

  predicate isSortedPrefix(a: array<int>, n: int)
    requires a != null
    requires 0 <= n <= a.Length
    reads a
  {
    forall i, j :: 0 <= i < j < n ==> a[i] <= a[j]
  }

  method MinIndex(a: array<int>, start: int) returns (minIdx: int)
    requires a != null
    requires 0 <= start < a.Length
    ensures start <= minIdx < a.Length
    ensures forall i :: start <= i < a.Length ==> a[minIdx] <= a[i]
  {
    minIdx := start;
    var j := start;
    while j < a.Length
      invariant start <= j <= a.Length
      invariant start <= minIdx < a.Length
      invariant forall k :: start <= k < j ==> a[minIdx] <= a[k]
      decreases a.Length - j
    {
      if a[j] < a[minIdx] {
        minIdx := j;
      }
      j := j + 1;
    }
  }

  method SelectionSort(a: array<int>)
    requires a != null
    modifies a
    ensures isSorted(a)
  {
    var n := a.Length;
    var pos := 0;
    while pos < n
      invariant 0 <= pos <= n
      invariant forall i, j :: 0 <= i < pos <= j < n ==> a[i] <= a[j]
      invariant isSortedPrefix(a, pos)
      decreases n - pos
    {
      var mi := MinIndex(a, pos);
      var temp := a[pos];
      a[pos] := a[mi];
      a[mi] := temp;
      pos := pos + 1;
    }
  }
}
```