

PROVING ARRAY REVERSAL

DANIEL SMITH, RAMYAA

ABSTRACT. We develop an array reversal algorithm and verify its correctness with PVS. We give an informal analysis of the algorithm which provides reason to trust its validity. We describe our testing strategy and discuss the feasibility of complete testing. We present the results of actual testing which are sufficient to demonstrate the validity of the algorithm. We then present some of the subtle points of the proving process.

CONTENTS

1. Introduction	1
2. Thoughts on Development	2
3. Semantic Verification - The Functional Model	2
4. Correctness Checking Infeasibility and an Alternative	3
5. The Automated Correctness Checker	4
6. Test Results	5
7. The PVS Theory	6
8. The Verification Process	7
9. Properties of the Array Reversal Algorithm	7
10. Philosophy of Verification	8
11. Conclusion	9

1. INTRODUCTION

This article is an account of the development of a formally verifiable array reversal algorithm. The organization of the paper is as follows. First, thoughts on the development of the array reversal algorithm are presented. Second, a discussion of correctness is offered which presents a semantic argument for the validity of the algorithm based on a functional model. Following this section, the feasibility of exhaustive verification is examined and a philosophy of testing is suggested. In the

Date: April 24, 2008.

subsequent section, the design model of the testing algorithm is presented along with a correctness discussion. We then present the results of correctness testing. Next, we describe the formulation of the PVS theory followed by some of the more subtle and significant points in the formal verification process. Finally, we present a collection of properties we've proven of the algorithm and conclude that the algorithm has been successfully verified.

2. THOUGHTS ON DEVELOPMENT

In designing an array reversal algorithm we want the termination conditions to be natural, but easy for PVS to typecheck automatically. This is a property that is difficult to achieve without experience or experimentation. In the beginning, we reasoned that keeping a single index the value of which is between 0 and half the size of the array is sufficient to achieve this goal. It is certainly natural. Unfortunately, designing the algorithm in this way means that when the algorithm terminates the index has moved beyond the range of indices that receive elements from a higher index during the reversal process. After some experimentation we found that the best solution is to weaken our termination condition and perform a sequential post-calculation to finish the algorithm. This strategy proved to be very beneficial. Our algorithm functions as follows. After retrieving the input, a variable, s , which marks the position in the array for input purposes, will contain the number of elements in the array. Setting a new variable, k , to the value 0, so as to have the effect of pointing to the beginning of the array, we switch the entries in k and $s - 1 - k$ and increment k . When $s - 2 * k < 3$ we exit the loop portion of the algorithm and for the last time switch the k and $s - 1 - k$ entries, after which every element which was required to move during the array reversal has been moved.

3. SEMANTIC VERIFICATION - THE FUNCTIONAL MODEL

Below is presented the pseudo-code for the array reversal algorithm. Let A be an array. Let B be the reverse of A :

Precondition $\{s = \text{Size}(A) \wedge k = 0\}$
while $(s - 2k \geq 3)$
Invariant $\{(\forall j)((j < k \vee j > s - 1 - k) \Rightarrow A(j) = B(s - 1 - j))\}$
 $A(k) \leftrightarrow A(s - 1 - k)$
 $k \leftarrow k + 1$
end while
 $A(k) \leftrightarrow A(s - 1 - k)$
Postcondition $\left\{ \begin{array}{l} s - 2k < 3 \\ \wedge (\forall j) ((j \leq k \vee j \geq s - 1 - j) \\ \Rightarrow A(j) = B(s - 1 - j)) \end{array} \right\}$

In particular, it is important in the design process to keep in mind two things: the loop invariant, and a measure function on the loop. In this case, the loop invariant is a little bit difficult to specify. Let us call the original array A and the array after reversal B. Then the loop invariant can be expressed $(\forall j)((j < k \vee j > s - 1 - k) \Rightarrow A(j) = B(s - 1 - j))$ where s is the size of the array. The measure function being used is $s - 2k$.

As long as the loop hasn't terminated, the measure function has a positive integer value as indicated by the loop condition. It can be shown that after an iteration of the loop, the measure function decreases, and this proves termination of the algorithm. Furthermore, termination corresponds to the value of the measure function becoming less than 3. It can also be shown that for each loop iteration the measure decreases by two; therefore, upon completion of the loop $s - 2k = 2$ or $s - 2k = 1$ which means that for the sequential code after the loop either two adjacent entries are being switched if the size of the array is even or an entry is switched with itself in the case that the size of the array is odd. In both cases there are no elements which were left unmoved and require moving to satisfy array reversal. Furthermore, the invariant condition along with the loop termination condition and the sequential code guarantee us that all elements are in the proper position.

4. CORRECTNESS CHECKING INFEASIBILITY AND AN ALTERNATIVE

A complete test of the correctness of this program over all inputs is impossible. To even verify that the reversal algorithm succeeds on all inputs of size smaller than 100000 with entries smaller than 2^{16} would require the construction of $\frac{2^{1600016}-1}{2^{16}-1} - 1$ arrays, the subsequent reversal

of all arrays, and finally the same number of comparisons of the arrays. This is not feasible.

To test the program, some assumptions about the operation of the program must be granted. Perhaps the most useful assumption is that the value in an array location has no effect on the outcome of the algorithm. If the arrangement of a few arbitrary numbers in the array affects the outcome of the algorithm, there is no feasible way of exposing the error from testing alone. Perhaps it is possible on suitably built machines to test for a comparison involving side channel methods.

With the above assumption, there is little left that can affect the validity of the algorithm other than the size of the input array. As a result it seems sufficient to test the program on a random input of each size from 1 to 100000. If there is any error in the computation caused by the size of the array, the problem will be revealed.

5. THE AUTOMATED CORRECTNESS CHECKER

The pseudo-code of the correctness checker follows. We assume that the checker has been given two array inputs A and B with $B = R(A)$, that is B is the result of the array reversal algorithm:

Precondition $\{s = \text{Size}(A) = \text{Size}(B) \wedge k = 0\}$
while $(s - 2k \geq 3) \wedge (A(k) = B(s - 1 - k)) \wedge (A(s - 1 - k) = B(k))$
Invariant $\{(\forall j)((j < k \vee j > s - 1 - k) \Rightarrow A(j) = B(s - 1 - j))\}$
 $k \leftarrow k + 1$
end while
 $s - 2k < 3 \wedge (A(k) = B(s - 1 - k)) \wedge (A(s - 1 - k) = B(k))$
Postcondition $\left\{ \begin{array}{l} B \text{ is } A \text{ reversed} \Leftrightarrow (s - 2k < 3 \\ \wedge (\forall j)((j \leq k \vee j \geq s - 1 - k) \Rightarrow A(j) = B(s - 1 - j)) \end{array} \right\}$

The loop invariant is for all j $j < k$ or $j > s - 1 - k$ implies $A(j) = B(s - 1 - j)$. There is assurance at each step that each of the elements which have already been checked have satisfied the array reversal condition. We use the same measure function, $s - 2k$, for simplicity.

While the loop hasn't terminated, the measure function has a positive integer value. After each iteration of the loop, the measure function decreases by two. Termination is caused by one of two conditions, either an error in array reversal, or the measure function becoming smaller than 3. In the latter case this means $s - 2k = 1$ or $s - 2k = 2$ and by the loop invariant we are guaranteed that if $A(k) = B(s - 1 - k)$ and $A(s - 1 - k) = B(k)$ then B is in fact the reversal of A . Thus termination occurs and the check fails if a counterexample is found or succeeds if $s - 2k < 3$, $A(k) = B(s - 1 - k)$, and $A(s - 1 - k) = B(k)$. Thus the correctness checker is valid.

6. TEST RESULTS

The results of a simple testing algorithm, using the above correctness checker, which creates a random array of integers from 0 to $2^{16} - 1$ of size $n = 1, 2, \dots, 100000$, runs array reversal and verifies that the array has indeed been reversed are as follows. *Range* indicates the size of the final array in the subsequence of checks. Therefore each entry corresponds to 10000 tests.

Range	Total Failures
10000	0
20000	0
30000	0
40000	0
50000	0
60000	0
70000	0
80000	0
90000	0
100000	0

It took 3 minutes and 5 seconds to perform this test.

The results of the testing show that there are no occurrences of failure for any test size in the range from 0 to 100000. There are, therefore, no pathological instances of the size of an input array causing an error. In conjunction with the discussion in the fourth section, this provides a strong argument for the validity of the array reversal algorithm.

7. THE PVS THEORY

To translate this algorithm into something we can prove in PVS, we choose a functional representation. We formulate the array reversal algorithm as follows.

$$\begin{aligned}
&R(a:ARRAY[_{nat} \rightarrow_{int}], s: \{j | j > 0\}, d: \{i | 2i < s\}): \\
&\quad RECURSIVE ARRAY[_{nat} \rightarrow_{int}] = \\
&\quad IF \ s - 2d < 3 \\
&\quad THEN \ a \ WITH \ [(d) := a(s - 1 - d), (s - 1 - d) := a(d)] \\
&\quad ELSE \ R(a \ WITH \ [(d) := a(s - 1 - d), (s - 1 - d) := a(d)], s, d + 1) \\
&\quad ENDIF \\
&MEASURE \ s - 2d
\end{aligned}$$

Except for the boundary conditions given in the IF statement, this is a very natural formulation. The purpose for using this choice of boundary condition is that it assures that the value of the measure function is always a natural number.

For ease of verification, and to match our checking algorithm, we would also like to choose a functional representation for the specification. The following function definition is a desirable model for our specification for two arrays of the same size having the property that they are reverses of each other.

$$\begin{aligned}
&R_{SPEC}(a:ARRAY[_{nat} \rightarrow_{int}], b:ARRAY[_{nat} \rightarrow_{int}], s: \{j | j > 0\}, d: \{i | 2i < s\}): \\
&\quad RECURSIVE \ bool = \\
&\quad (s - 2d < 3 \ AND \ a(d) = b(s - 1 - d) \ AND \ a(s - 1 - d) = b(d)) \\
&\quad OR \\
&\quad (a(d) = b(s - 1 - d) \ AND \ a(s - 1 - d) = b(d) \ AND \ R_{SPEC}(a, b, s, d + 1)) \\
&MEASURE \ s - 2d
\end{aligned}$$

We use the same measure function and exit condition in this model as in the reversal definition so that recursion can work smoothly when the same variable is shared by each function in a theorem statement.

This formulation of the specification is not natural and is designed to accept R as defined above. To use this model as our specification, we need to show that this specification implies a natural specification to which everyone can agree. Our natural specification is as follows.

$$\begin{aligned}
&SPEC(a:ARRAY[_{nat} \rightarrow_{int}], b:ARRAY[_{nat} \rightarrow_{int}], s: \{j | j > 0\}, d: \{k | 2k < s\}): \\
&\quad bool = \\
&\quad FORALL \ (k: \{i | i \geq d \ AND \ 2i < s\}): \ a(k) = b(s - 1 - k) \ AND \ a(s - 1 - k) = \\
&\quad b(k)
\end{aligned}$$

8. THE VERIFICATION PROCESS

A fundamental conflict in the proof process that continually arises when trying to prove propositions is that although it is typically easier to prove a more general statement than a specific statement, it is necessary to prove very specific properties for the theorems to be proven. This is a complicated process of taking a broad viewpoint and narrowing the possibilities until one notices that another property is needed, and then generalizing this property. For example, to verify the array reversal algorithm in PVS we need to prove our big theorem.

Theorem 1. *For all arrays a with size $s > 0$, $SPEC(a, R(a, s, 0), s, 0)$.*

In fact, in light of the following lemma, this theorem reduces to showing that R satisfies an analogous statement for R_{SPEC} instead of $SPEC$.

Lemma 1. *For all $s > 0$, all arrays a and b with size s , and all k such that $2k < s$, if $R_{SPEC}(a, b, s, k)$ is true then $SPEC(a, b, s, k)$ is true.*

It suffices, however, to prove a more general lemma.

Lemma 2. *For all $s > 0$, all arrays a with size s , and all k such that $2k < s$, $R_{SPEC}(a, R(a, s, k), s, k)$.*

Proving the above lemma requires two properties, one dependent on the other.

Lemma 3. *For all arrays a and b of size $s > 2$, k such that $2k < s$ and $d \geq 0$, if $d < k$ then $R_{SPEC}(a, b, s, k) = R_{SPEC}(a_{WITH}[(d) = a(n), (n) = a(d)], b, s, k)$ where $n = s - 1 - d$.*

The latter two of the above lemmas as well as the main result depend on the following lemma which we prove as two separate results.

Lemma 4. *For all arrays a of size $s > 0$, k such that $2k < s$, and $d \geq 0$, if $d < k$ then $a(d) = R(a, s, k)(d)$ and $a(n) = R(a, s, k)(n)$ where $n = s - 1 - d$.*

We stress that the proofs of the above claims are not difficult, rather they are straightforward, but the knowledge of what claims were needed is difficult to achieve. There are other possible claims that are correct and can yield the main result but are too difficult to prove easily.

9. PROPERTIES OF THE ARRAY REVERSAL ALGORITHM

In this section we present some of the properties that we expect the array reversal algorithm and the specification function to have which

we have verified using PVS. The first collection are properties of the specification function, R_{SPEC} , which assure us that it acts consistently with the behavior we expect from the specification. The second is a pair of results which show that, as expected, R has a self-inverse property.

Lemma 5. *For all arrays a and b of size $s > 2$, if the loop invariant of the specification algorithm holds true for all indices k such that $2k < s$ then a and b are reverses of each other.*

This result supports our previous informal analysis.

Lemma 6. *R_{SPEC} is symmetric with respect to a and b .*

In other words, if a is the reverse of b , then b is the reverse of a .

Most important among the properties of R_{SPEC} for proving the self-inverse property of R is the following induction lemma which allows us to check for reversal in specific symmetric regions within an array.

Lemma 7. *For all arrays a and b of size $s > 0$, k such that $2k < s$, and d such that $k \leq d$ and $2d < s$, if $R_{SPEC}(a, b, s, k)$ is true then $R_{SPEC}(a, b, s, d)$ is also true.*

For the self-inverse property of R we see once again that generalization is necessary. We prove the following lemma using the above induction lemma.

Lemma 8. *For all arrays a and b of size $s > 0$ and k such that $2k < s$ if $R_{SPEC}(a, b, s, ac)$ is true then for all d such that either both $k \leq d$ and $2d < s$ or both $s - 1 - d \geq k$ and $s - 1 - 2d < 0$ we have $R(a, s, ac)(d) = b(d)$.*

As a consequence of this lemma we derive the self-inverse property of R .

Theorem 2. *For all arrays a of size $s > 0$ and all $0 \leq d < s$ $R(R(a, s, 0), s, 0)(d) = a(d)$.*

10. PHILOSOPHY OF VERIFICATION

It is interesting to note that the useful methods for proving the properties we desire to prove about R rely on proving properties of R_{SPEC} . The intuitive justification for this is that R_{SPEC} is our tool for recognizing the properties of R , and in proving a property of R we must in some sense factor information about the contents of an array through both R and R_{SPEC} . Thus it seems our philosophy should be to generalize the properties we want our algorithm to realize into properties of arbitrary instances of the underlying objects. In this way, we can formulate relationships by factoring the information in the objects through only the

specification function, in our case R_{SPEC} , instead of composed functions.

In addition, it seems also that most of the generalization seems to be removing instances of our algorithm to prove properties of arbitrary objects. The strategy of proving properties of more generic objects under certain conditions and then proving that our algorithm gives us those conditions has proven to be very useful. In this way we can see that we are generalizing away from our algorithm to prove properties and then specialize to the case which applies to our algorithm instead of complicating our proofs by getting buried in the details of the algorithm.

Most of the times in which it became necessary to specialize a claim seem to arise when we are proving properties of the specification function. This suggests that there is a duality between the function of which we are trying to prove properties, and the specification function. Statements about the functional representation of the algorithm require generalization while statements about the specification function require specialization.

11. CONCLUSION

We have tested all possible array sizes under 100000 to determine whether the size of the array affects the outcome of the array reversal algorithm. The results of this testing offer convincing evidence that the algorithm is safe for use. In the subsequent sections we prove using PVS that the array reversal algorithm meets the expected specifications and has the properties we should expect of such an algorithm. There is further justification for labeling the array reversal algorithm correct in the semantic argument using a functional model of the algorithm. Consequently, we conclude that the array reversal algorithm has been shown to terminate with the correct output.

We have also speculated about the meaning of our successes and failures in the proof process. We have concluded that there is some sort of duality between a functional representation of an algorithm and its specification. The proof process has shown us that proving properties of the functional representation require generalization while proving properties of the specification to some extent requires specialization of the claim. This property we suspect generalizes to all such verification tasks, and we conclude that this should become part of our verification philosophy.