Writing Correct Programs, 32 Years Later: Binary Search Revisited

Alex Groce, Oregon State University

March 4, 2015

1 Introduction

In 1983 John Bentley wrote an article [?] for his beloved "Programming Pearls" column in CACM called "Writing Correct Programs." That column described an attempt to prove binary search correct, in the context of the ongoing effort to verify computer programs. Bentley wrote:

> In the late 1960's people were talking about the promise of programs that verify the correctness of other programs. Unfortunately, in the intervening decades, with precious few exceptions, there is still little more than talk about automated verification systems. In spite of unrealized expectations, though, research on program verification has given us something far more valuable than a black box that gobbles programs and flashes "good" or "bad"—we now have a fundamental understanding of computer programming.

2 Scaling Up

3 Testing the Tester

Andrews' tool generates 63 mutants of `binsearch.c`. Of these, using signed characters as the index and value types, CBMC finds counterexamples for 56, and 3 fail to compile (because they delete a variable declaration). The remaining 4 mutants, shown in Figure 2 are of interest.

One of these is easily dismissed. Because the `rep_op` mutant is to the `else if` condition *after* we have already checked `midval < key`, `midval != key` is semantically equivalent to `midVal > key`. The `rep_const` and `del_stmt` mutants are not detected with signed character indices because this code only matters with an unsigned index type. Finally, the first mutant shows a (slightly) surprising property of binary search: it doesn't really matter if we set `low` initially to -1 or to 0, the behavior will end up being binary search-like. Changing the

```
index_t binsearch(value_t key, value_t a[], index_t size) {
  index_t low = 0;
  index_t high = size - 1;
  while (low <= high) {
    index_t mid = (low + high) / 2;
    value_t midVal = a[mid];
    if (midVal < key)
      low = mid + 1;
    else if (midVal > key)
      high = mid - 1;
    else
      return mid; // key found
  }
  return -1; // key not found
}
```

Figure 1: Binary search, version 1

index type to `unsigned char`, 59 of the mutants are killed, 3 fail to compile, and exactly one verifies successfully: the obviously semantically equivalent change to the `midVal` comparison.

While not a proof that our test harness captures all conceivable bugs in binary search (for example, it doesn't make sure binary search leaves the array un-modified).

4 Scaling Up, Again

```
#include "sortarray.h"

int acc = -1;

index_t ind[MAX_ITEMS];
value_t val[MAX_ITEMS];

value_t a(index_t n) {
  int i;
  for (i = 0; i <= acc; i++) {
    if (ind[i] == n) {
      /* printf ("LOG: found %u with val %d\n", n, val[i]); */
      return val[i];
    }
  }
  value_t v = nondet_value();
  for (i = 0; i <= acc; i++) {
    if (ind[i] < n) {
      __CPROVER_assume(v >= val[i]);
    } else {
      __CPROVER_assume(v <= val[i]);
    }
  }
  acc = acc + 1;
  assert(acc < MAX_ITEMS);
  val[acc] = v;
  ind[acc] = n;
  return v;
}
```

```
 /* MUTANT (rep_const) */  index_t low = -1;
 /* MUTANT (rep_op) */    else if (midVal != key)
 /* MUTANT (rep_const) */      if (mid == -1) {
 /* MUTANT (del_stmt) */ /*     return 0; */
```

Figure 2: Mutants not killed