A Proof of the Boyer-Moore Majority Protocol

Jayadev Misra

06/11/2013

1 Introduction

This note is inspired by discussions with Greg Plaxton who has observed that the Boyer-Moore Majority Protocol [1] provides an excellent example for teaching program verification in an undergraduate course. I write out the proof in detail here. The first program uses certain abstract data structures. Its proof is relatively easy to construct. The next program is a refinement, replacing the abstract data structures by concrete representations.

The Boyer-Moore Majority Protocol Given is a finite bag B of items. An item is a *majority* if it occurs more often than all other items combined. Equivalently, a majority item of B occurs more than $|B|/2$ times on B. It is required to determine if B has a majority item and return it if there is one. The Boyer-Moore algorithm consists of two passes, each pass consuming linear time. The first pass either determines that there is no majority item or finds a *candidate* item that is the only possible majority item. The second pass is run only if there is a candidate in order to determine if the candidate is indeed a majority item. The second pass is straightforward and is omitted from further discussion. The rest of this note is about the first pass.

Terminology An *item* may occur multiple times in a bag. Each of its occurrences is called an *element*. Henceforth, we overload the usual set notation to apply to bags. Symbol ϕ denotes the empty bag. □

The protocol description below is from Misra and Gries [3] where a generalized version of this problem treated. A *reduction* step removes any two distinct elements from B. The algorithm applies the reduction step to B repeatedly as long as there are distinct elements in B; the resulting bag, R, is a *reduced bag*. Termination is guaranteed because each step strictly decreases the bag size. Bag R contains copies of at most one item, because otherwise a reduction step can be applied to R.

Observe that the reduced bag is not unique. For example, starting with the bag $\{2, 2, 3, 4\}$ a reduction might eliminate $\{3, 4\}$ leaving $\{2, 2\}$ as the reduced bag whereas a different sequence of reduction steps that eliminate $\{2, 3\}$ and $\{2, 4\}$ would leave ϕ as the reduced bag.

Informal Proof of the Protocol We assert that a majority item of B is in R. Equivalently, any item z not in R is a non-majority in B. Consider the bag $B - R$, the bag of elements removed using the reduction steps. Each reduction step removes two distinct elements from B. So, every element in $B - R$ is paired with an element of different value; so, $B - R$ has no majority, and z is not a

majority item in $B-R$. Further, since z does not occur in R it is not a majority in R. Therefore, z is not a majority in $(B-R)\cup R$, or B.

2 Formal Algorithm Description and Proof

The original description of the algorithm by Boyer and Moore [1] was given in Fortran, which makes its analysis very difficult. It is a tribute to the authors' abilities in automated verification that they could even prove the result. But such a description is wholly unsuitable for teaching program verification to novices.

2.1 Properties of a reduced bag

We use the notation $B \hookrightarrow R$ to stand for "B can be reduced to R". First, we note some simple properties of $\hookrightarrow$. These properties can be established directly from the intuitive description of $\hookrightarrow$, or they can simply be taken as the definition of $\hookrightarrow$.

$$\phi \hookrightarrow \phi \tag{R1}$$

$$B \hookrightarrow R \text{ and } (R = \phi \vee x \in R) \text{ imply } B \cup \{x\} \hookrightarrow R \cup \{x\} \tag{R2}$$

$$B \hookrightarrow R \text{ and } (R \neq \phi \wedge x \notin R) \text{ imply } B \cup \{x\} \hookrightarrow R - \{s\}, \text{ where } s \text{ is any item in } R \tag{R3}$$

2.2 Properties of Reduction

The goal of this section is to prove formally that any majority item of bag B belongs to the reduced bag R. We do so through a number of smaller proof steps.

2.2.1 Proposition (P1): R is a subbag of B

The proof is by induction on the number of times rules (R1) through (R3) are applied in reducing B to R.

- Given $\phi \hookrightarrow \phi$, we have $\phi \subseteq \phi$.
- Given that $R \subseteq B$, it is easy to see that $R \cup \{x\} \subseteq B \cup \{x\}$ and $R - \{s\} \subseteq B \cup \{x\}$, for any s in R.

2.2.2 Proposition (P2): There is no majority item in $B - R$

For bag b and item z let $nomaj(z, b)$ denote that z is not a majority item in b. Slightly abusing notation, let $nomaj(b)$ denote $nomaj(z, b)$ for all z; that is, b has no majority item. (This predicate is a simpler version of one proposed in Plaxton [4].)

We show that $nomaj(B-R)$ by induction on the number of times rules (R1) through (R3) are applied in reducing B to R. We need the following results whose proofs are straightforward using any formal definition of $nomaj(z, b)$.

$$z \notin b \;\Rightarrow\; nomaj(z, b) \quad \text{(Q1)}$$
$$(nomaj(z, b) \wedge nomaj(z, b')) \;\Rightarrow\; nomaj(z, b \cup b') \quad \text{(Q2)}$$
$$(nomaj(b) \wedge nomaj(b')) \;\Rightarrow\; nomaj(b \cup b') \quad \text{(Q3)}$$

Proof of $nomaj(B - R)$

- Given $\phi \hookrightarrow \phi$, we have to show $nomaj(\phi - \phi)$, or $nomaj(\phi)$, which follows trivially.
- In applying (R2), $B - R$ does not change; so, $nomaj(B - R)$ is preserved.
- In applying (R3), the precondition is $B \hookrightarrow R$, $(R \neq \phi \wedge x \notin R)$ and $nomaj(B - R)$. Note that from $(R \neq \phi \wedge x \notin R)$ and $s \in R$, we have $s \neq x$.

  We have to show that $nomaj((B \cup \{x\}) - (R - \{s\}))$. From induction hypothesis,

  $nomaj(B - R)$
  $\Rightarrow$ {since $s \neq x$, $nomaj(\{x, s\})$}
  $nomaj(B - R)$ and $nomaj(\{x, s\})$
  $\Rightarrow$ {from (Q3)}
  $nomaj((B - R) \cup \{x, s\})$
  $\equiv$ {from (P1) $R \subseteq B$. So, $(B - R) \cup \{x, s\} = (B \cup \{x\}) - (R - \{s\})$}
  $nomaj((B \cup \{x\}) - (R - \{s\}))$

2.2.3 Proposition (P3): Any majority item of B is in R

We prove the contrapositive, that $z \notin R \;\Rightarrow\; nomaj(z, B)$.

$z \notin R$
$\Rightarrow$ {from (Q1)}
$nomaj(z, R)$
$\Rightarrow$ {from (P2), $nomaj(B - R)$}
$nomaj(z, B - R)$ and $nomaj(z, R)$
$\Rightarrow$ {from (Q2)}
$nomaj(z, (B - R) \cup R)$
$\Rightarrow$ {from (P1), $R \subseteq B$; so, $(B - R) \cup R = B$}
$nomaj(z, B)$

2.3 Computing a reduced bag

Properties (R1) through (R3) also prescribe a procedure for computing a reduced bag. To compute a reduced bag for $B \cup \{x\}$, we may first compute the reduced bag R for B. If R is empty or it contains x, then add x to R. Otherwise, remove an item from R as an application of the reduction step.

The obvious way to formally describe the algorithm is to have an array that stores the elements to be processed, and a program that processes the array

elements in sequence. We propose a different style of description that completely avoids dealing with array indices. This style is inspired by UNITY [2] where the emphasis is placed on the actions that manipulate data rather than on the control or data structures. A similar approach is advocated in Plaxton [4] for this problem.

Imagine that some external agent supplies a stream of x values that make up bag B. There are two actions, $increase(x)$ and $reduce(x)$, that either add x to R or remove an element from R, respectively. These actions are executed repeatedly as long as values are supplied.

initially $B, R := \phi, \phi$

$increase(x) :: R = \phi \ \vee x \in R \longrightarrow \ B, R := B \cup \{x\}, R \cup \{x\}$

$reduce(x) \ :: R \neq \phi \ \wedge x \notin R \longrightarrow \ B, R := B \cup \{x\}, R - \{s\}$, where $s \in R$

Correctness The computation procedure implements (R1) through (R3). Therefore, $B \hookrightarrow R$ is an invariant of this program. Consequently, propositions (P1) through (P3) hold. In particular, from (P3), any majority item of B is in R.

Invariant for the Structure of R We can establish that R contains (possibly multiple copies of) at most one item. That is, we have the invariant

$$y \in R \wedge z \in R \ \Rightarrow \ y = z, \text{ for all } y \text{ and } z. \tag{I}$$

The proof of the invariance of (I) is as follows.

- Initially, (I) holds vacuously.
- $increase(x)$ adds x to R provided $(R = \phi \vee x \in R)$. Then R contains no item other than x following the action execution if (I) holds prior to the execution.
- $reduce(x)$ removes an item from R; so, it preserves (I).

3 Optimizing the Data Structure

The description of the algorithm in Section 2 uses bags B and R. Bag B is an auxiliary variable introduced solely for stating the correctness condition and the invariant. So, we eliminate this variable in the final algorithm. Bag R can be represented more efficiently as follows.

Given (I) we may represent R by a pair of variables, c and s, where c is the size of R and s the unique item in R provided $c > 0$; if $c = 0$ the value of s is irrelevant. The representation invariant linking the concrete data structure (c, s) to the abstract data structure R is: the bag consisting of c copies of element s equals R.

The initial assignment to R, $R := \phi$, can be implemented by $c := 0$. Additionally, assign s a dummy value *NIL* so that s always has a defined value. Similar translations for all tests and assignments are shown in Table 1.

Abstract Operation	Concrete Implementation
$R := \phi$	$c, s := 0, NIL$
$R = \phi$	$c = 0$
$x \in R$	$x = s$
$R := R \cup \{x\}$	$c, s := c + 1, x$
$R := R - \{s\}$	$c := c - 1$

Table 1: Implementing the Reduced Bag

These transformations result in the following program.

initially $c, s := 0, NIL$

$increase(x) :: c = 0 \ \vee x = s \ \rightarrow \ c, s := c + 1, x$

$reduce(x) \ :: c \neq 0 \ \wedge x \neq s \ \rightarrow \ c := c - 1$

4 Concluding Remarks

The Boyer-Moore majority algorithm is an excellent example of an algorithm whose description is simple but whose proof is not. It appears to be a very good exercise in teaching program verification. Unfortunately, it does not include enough imperative features —such as multiple program states, nested loops, or termination arguments— that would illustrate other aspects of imperative program verification.

References

[1] R.S. Boyer and J Moore. MJRTY - a fast majority vote algorithm. In R.S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Automated Reasoning Series, pages 105–117. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1991.

[2] K. M. Chandy and J. Misra. *Parallel Program Design. A Foundation.* Addison-Wesley, 1988.

[3] Jayadev Misra and David Gries. Finding repeated elements. *Science of Computer Programming*, 2:143–152, 1982.

[4] Greg Plaxton. Personal communication.