Int J Software Informatics, Volume 5, Issue 1-2 (2011), Part I, pp. 21–28 E-mail: ijsi@iscas.ac.cn
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org
©2011 by ISCAS. All rights reserved. Tel: +86-10-62661040

Majority Vote Algorithm Revisited Again

Tobias Nipkow

(Technische Universität München, Germany)

Abstract In his article *Experience with Software Specification and Verification Using LP, the Larch Proof Assistant*, Manfred Broy verified (as one of his smaller case studies) the Majority Vote Algorithm by Boyer and Moore. LP requires that all user theories are expressed axiomatically. I reworked the example in Isabelle/HOL and turned it into a definitional development, thus proving its consistency. In the end, a short alternative proof is given.

Key words: majority vote algorithm; verification

Nipkow T. Majority vote algorithm revisited again. *Int J Software Informatics*, **Vol.5, No.1-2(2011), Part I: 21–28. http://www.ijsi.org/1673-7288/5/i74.htm**

1 Introduction

This paper is about Manfred Broy's verification[5] of the Majority Vote algorithm by Boyer and Moore[3] conducted with the help of LP, the *Larch Prover*[6]. LP was a theorem prover developed by Garland and Guttag at MIT. It grew out of Lescanne's *Reve* system[8], which was based on equational logic. Although LP later supported full first-order logic, the version that Broy was using was still restricted to equations.

This article is an attempt to evaluate the progress in theorem proving technology and tools over the past two decades by reworking Broy's proof in the state of the art theorem prover Isabelle/HOL[9]. Unsurprisingly, much less effort is involved today. But more interesting is the conceptual progress the field has made. Hence this article will not compare proof sizes but specification styles. LP was based on an axiomatic specification style: every function was introduced by axioms, and there was no mechanism to check the consistency of a specification. In contrast, the Boyer-Moore prover Nqthm[2], Mike Gordon's HOL[7], and Isabelle/HOL[9] all emphasise or even restrict to definitional extension, thus guaranteeing consistency. Broy accompanies his specification by informal proofs of consistency, sometimes indicating how a richer logic would have permitted a definitional specification. Having translated Broy's specifications into definitional form I can confirm that they are consistent, although in two places this is not trivial to show.

Of course one should not be too unkind to LP. It was not conceived as a program verification system in the way Nqthm was, and it allowed axiomatic reasoning in a way that Nqthm would not. Higher-order provers can combine the best of both worlds: Isabelle/HOL supports structured specifications called *locales*[1] which appear axiomatic but are in fact definitional.

Corresponding author: Tobias Nipkow, Email: nipkow@in.tum.de
Received 2010-09-16; Accepted 2010-12-20; Final revised version 2011-01-18.

2 MJRTY

Before we begin to formalise Broy's proof we explain *MJRTY*, the Boyer-Moore Majority Vote algorithm, by quoting their own vivid description:

> Imagine a convention center filled with delegates (i.e. voters) each carrying a placard proclaiming the name of his candidate. Suppose a floor fight ensues and delegates of different persuasions begin to knock one another down with their placards. Suppose that each delegate who knocks down a member of the opposition is simultaneously knocked down by his opponent. Clearly, should any candidate field more delegates than all the others combined, that candidate would win the floor fight and, when the chaos subsided, the only delegates left standing would be from the majority block. Should no candidate field a clear majority, the outcome is less clear; at the conclusion of the fight, delegates in favor of at most one candidate, say, the nominee, would remain standing—but the nominee might not represent a majority of all the delegates. Thus, in general, if someone remains standing at the end of such a fight, the convention chairman is obliged to count the nominee's placards (including those held by downed delegates) to determine whether a majority exists.

Like everybody else I concentrate on the fight phase of the algorithm, where a single candidate is determined—counting if he has a majority is trivial. Here is how Boyer and Moore describe a bloodless version of the fight phase:

> The chairman visits each delegate in turn, keeping in mind a current candidate *cand* and a count k, which is initialised to 0. Upon visiting each delegate, the chairman first determines whether k is 0; if it is, the chairman selects the delegate's candidate as the new value of *cand* and sets k to 1. Otherwise, the chairman asks the delegate whether his candidate is *cand*. If so, then k is incremented by 1. If not, then k is decremented by 1. The chairman then proceeds to the next delegate.

They verified a Fortran version of this algorithm. I follow Broy in verifying a functional specification instead.

3 Consistency

Above I explained that working with LP raises the issue of consistency of an axiomatic specification. Broy discusses it as well and uses two mechanisms to ensure consistency: non-recursive definitions and structurally recursive definitions over a set of free constructors. We can prove the consistency of some axiomatic specification if we can give a consistent definition of the functions involved (e.g. using the two definition principles just mentioned) and we can prove the axioms from those definitions. Unfortunately Broy chooses the non-free data type of multisets for his work, which raises a thorny consistency issue. He is aware of this: at the end of section 3.2 of his article he acknowledges that for structural recursion equations over non-free data types one still needs to show that the result of the function is unique. He writes

> This proof, however, can be quite difficult for recursive functions.

and does not attempt such proofs in the paper. Most likely, LP would not have supported them.

Below we will show the consistency of Broy's specifications by expressing all functions purely definitionally in Isabelle/HOL. That is, in the end, every function is defined by a non-recursive equation $f(x) = t$, where f does not occur in t. Luckily, Isabelle/HOL helps: for example, terminating recursion equations over free data types can be given by the user and the system will turn them into non-recursive definitions from which it derives the original recursion equations as theorems. However, for multisets, no such automation is available and I had to get my hands dirty.

In places where the axiomatic specification follows easily from the definitional one, I will not comment on the issue.

4 Broy's Development

Broy decides to model the delegates on the floor by a multiset. He needed to specify multisets himself, in Isabelle/HOL we have a well-developed library theory of finite multisets of type $'a$ multiset—multisets of elements of type $'a$, where $'a$ is a type variable. It provides the empty and singleton multisets $\{\#\}$ and $\{\#x\#\}$, multiset union and difference $M + N$ and $M - N$, multiset membership $x \in \#M$, the number of times an element occurs in a multiset *count* M x, and the size of a multiset *size* M. Having such well-developed libraries of fundamental mathematical concepts is theoretically trivial but practically invaluable progress.

Broy also introduces a choice function *any* of type $'a\ multiset \Rightarrow\ 'a$ that he characterises axiomatically:

$$any(\{\#x\#\} + M) = x \vee M \neq \{\#\} \wedge any(\{\#x\#\} + M) = any\ M$$

In Isabelle/HOL we define *any* via the indefinite choice operator *SOME*:

$$any\ M = (SOME\, x.x \in \#M)$$

and abbreviate $M - \{\#any\ M\#\}$ by *drop M*.

Note that *any* does not satisfy Broy's axiom, something I only realized when I failed to prove the axiom. Nitpick[4] found the following counterexample for me. I present it in terms of sets instead of multisets, the issue remains the same. Let type $'a$ be some 3-element type, e.g. $\{1, 2, 3\}$.Let *any* behave like this: $\{1, 2, 3\} \longmapsto 3$, $\{1, 2\} \longmapsto 1$,$\{1,3\} \longmapsto 1$,$\{2, 3\} \longmapsto 2$,$\{1\} \longmapsto 1$,$\{2\} \longmapsto 2$,$\{3\} \longmapsto 3$,$\{\ \} \longmapsto 3$. This is consistent with our definition of *any*. But if we set $M = \{2, 3\}$ and $x = 1$, Broy's axiom for *any* is falsified because *any* $\{1,2,3\}$ is neither 1 nor *any* $\{2,3\}$. This reveals that Broy's axiom is stronger than necessary. The implication

$$M \neq \{\#\} \Longrightarrow any\, M \in \#M$$

would have sufficed and would have been equivalent to my definition of *any*. His axiom is consistent because in HOL it is provable that every type can be well-ordered. Hence *any* could pick out the least element w.r.t. that well-order, thus satisfying Broy's axiom.

Broy subdivides the rest of his development into the classical triple of *requirements*, *design* and *implementation*.

4.1 The requirements specification

Let *majority M x* (“*x* has the majority in *M*”) be an abbreviation for *size M* $<$ *2* $*$ *count M x*. Broy states the requirements on the majority vote algorithm by introducing a function *major* of type $'a\ multiset \Longrightarrow\ 'a$ and asserting that if there is a majority in *M*, then *major M* must return that element. Due to the lack of existential quantifiers in LP, Broy needs to specify two functions simultaneously, *major* and *anarchic*. The latter, he notes, should really be defined like this, which I did:

$$anarchic\ M\ = (\nexists\, x.\ majority\ M\ x)$$

Introducing the requirements specification as a function *major* is unsatisfactory as it is not clear yet if such a function exists. In Isabelle, there is an alternative: *locales*[1]. Here is the requirements specification as a locale:

locale *Mjrty-req* =
fixes *major* :: $'a\ multiset \Rightarrow\ 'a$
assumes $\neg\ anarchic\ M \Longrightarrow majority\ M\ (major\ M)$

This looks like an axiomatic specification of a function *major* but defines a predicate *Mjrty-req* where *major* is just a parameter:

$Mjrty\text{-}req\ major = (\forall M.\ \neg\ anarchic\ M \longrightarrow majority\ M\ (major\ M))$

That is, *Mjrty-req* is a higher-order function that defines the requirement to be a majority finder. LP being first-order, such a specification was not open to Broy. He comments on this very issue in his quicksort proof in the same article.

4.2 The design specification

Broy calls a multiset *homogeneous* iff all its elements are the same:

$homo\ M = (\forall x\ y.\ x \in\#\ M \longrightarrow y \in\#\ M \longrightarrow x = y)$

His approach is to compute the majority by splitting the multiset into a homogeneous and an anarchic part. Simplifying and translating Broy’s specification into a locale yields

locale *Mjrty-design* =
fixes *hom* :: $'a\ multiset \Rightarrow\ 'a\ multiset$
assumes $hom\ M \leqslant M$ **and** $homo(hom\ M)$ **and** $anarchic(M - hom\ M)$

Locale *Mjrty-design* specifies the above requirements for splitting off a homogeneous part. Note that $\leqslant$ is the submultiset relation. In the context of the assumptions of *Mjrty-design* it is easy to show that $any \circ hom$ satisfies the requirements specification:

$Mjrty\text{-}req\ (any \circ hom)$

4.3 Implementation

Now we come to the “bloodless” algorithm where the chairman visits each delegate in turn (see Section 2). Broy’s version looks quite different from Boyer and Moore’s but is in fact very similar: he operates on homogeneous multisets, and a homogeneous multiset is isomorphic to an element and a multiplicity—the *cand* and k by Boyer and Moore. Broy specifies a function $scan :: 'a\ multiset \Rightarrow 'a\ multiset$ that iterates over a multiset and is supposed to satisfy *Mjrty-design*. Here is his specification:

$scan\ \{\#\} = \{\#\}$

$scan\ (\{\#any\ M\#\} + drop\ M) =$
$(\textsf{if}\ homo\ (\{\#any\ M\#\} + scan\ (drop\ M))\ \textsf{then}\ \{\#any\ M\#\} + scan\ (drop\ M)$
$\ \textsf{else}\ drop\ (scan\ (drop\ M)))$

This is a subtle axiomatisation. Defining a recursive function on multisets (or sets) is nontrivial because they are non-free data structures. Broy does not write the second axiom as $scan\ (\{\#x\#\} + M) = \ldots$, which would have been inconsistent: if the argument of *scan* is anarchic, it matters in which order the elements of the multiset are visited. Broy avoids the problem by examining the elements in the canonical order determined by *any*. Although this leaves open in what sense this can be considered executable or an implementation, it seems to avoid the inconsistency. But are his axioms really consistent? In the definitional approach we have to face this issue, and this is where HOL comes into its own. Isabelle/HOL's multiset library offers a recursion combinator for iterating over a multiset. But the function to be iterated needs to be insensitive to the order in which multiset elements are examined, but *scan* is not. Hence we have to do some real work.

I follow the standard road for defining beastly functions: define an inductive predicate/relation and show that it is in fact a function. Here is the inductive definition of a binary predicate *Scan*:

$Scan\ \{\#\}\ \{\#\}$

$Scan\ (drop\ M)\ N \Longrightarrow$
$Scan\ (drop\ M + \{\#any\ M\#\})$
$\ (\textsf{if}\ homo\ (\{\#any\ M\#\} + N)\ \textsf{then}\ \{\#any\ M\#\} + N\ \textsf{else}\ drop\ N)$

The point is that (monotone) inductive definitions are always consistent. Isabelle knows about this. That is, if you tell it to interpret a set of implications of a particular form as an inductive definition, it will internally create a proper definition from which it derives the given implications as theorems. No axioms are involved.

Resorting to Isabelle/HOL's definite description operator *THE* we can trivially define *scan*:

$scan\ M = (THE\ N.\ Scan\ M\ N)$

Of course, now the real work starts: we have to derive the recursion equations from this definition. First, injectivity of *Scan*:

$Scan\ M\ N \Longrightarrow Scan\ M\ N' \Longrightarrow N = N'$

The proof is a simple induction on the derivation of the first premise. All the work is in this lemma:

$$\begin{aligned}&\mathit{Scan}\ (\mathit{drop}\ M + \{\#\mathit{any}\ M\#\})\ N \Longrightarrow\\ &\exists K.\ \mathit{Scan}\ (\mathit{drop}\ M)\ K\ \wedge\\ &\qquad N = (\textsf{if}\ \mathit{homo}\ (\{\#\mathit{any}\ M\#\} + K)\ \textsf{then}\ \{\#\mathit{any}\ M\#\} + K\ \textsf{else}\ \mathit{drop}\ K)\end{aligned}$$

It holds because nonempty multisets can be decomposed uniquely into an element *any M* and a rest *drop M*. The proof is a bit technical. Now we can prove that *Scan* is not just injective but also total:

$$\exists! N.\ \mathit{Scan}\ M\ N$$

The quantifier $\exists!$ means "there exists a unique". This proof is by induction on the size of *M*. If *M* is nonempty, $\mathit{Scan}\ (\mathit{drop}\ M)\ L$ follows by induction hypothesis, from which we can derive $\mathit{Scan}\ M\ \ldots$ (for suitable ...) by the second rule for *Scan* because $M = \mathit{drop}\ M + \{\#\mathit{any}\ M\#\}$. By injectivity of *Scan*, the result ... is unique. Now the two equations for *scan* are easily derived using this basic lemma:

$$\exists! x.\ P\ x \Longrightarrow P\ a \Longrightarrow (\mathit{THE}\ x.\ P\ x) = a$$

Until now, except for the derivation of the *scan* equations, we hardly needed to prove anything. But now we need to show that *scan* satisfies the design specification:

Mjrty-design scan

We prove the three properties that *Mjrty-design* stands for by a simultaneous induction on *M*:

$$\mathit{scan}\ M \leqslant M \qquad \mathit{homo}\ (\mathit{scan}\ M) \qquad \mathit{anarchic}\ (M - \mathit{scan}\ M)$$

Because of the nonstandard pattern on the left-hand side of the second *scan* equation, the induction is on the size of *M*. The proof is about as long as the derivation of the *scan* equation.

We have now come to the end of our little case study. Broy still has to prove the consistency of his specifications of *homo* and *anarchic*. He does so by giving recursion equations over multisets. Again this is not a complete proof because multisets are non-free data types (see Section 3). But now a proper consistency proof is simpler. We already have definitions for those two functions, we merely need to show that Broy's equations follow from them, which turns out to be almost trivial due to Isabelle's automation. We show the equations without further comment:

$$\mathit{homo}\ \{\#\} \qquad \mathit{homo}\ \{\#x\#\}$$

$$\mathit{homo}\ (A + \{\#b\#\} + \{\#a\#\}) = (a = b \wedge \mathit{homo}\ (A + \{\#b\#\}))$$

$$\mathit{anarchic}\ M = \mathit{comp}\ M\ (\mathit{size}\ M)$$

$$\mathit{comp}\ \{\#\}\ n \qquad \mathit{comp}\ \{\#a\#\}\ n = (1 < n)$$

$$\mathit{comp}\ (\{\#a\#\} + M)\ n = (2 * \mathit{count}\ (\{\#a\#\} + M)\ a \leqslant n \wedge \mathit{comp}\ M\ n)$$

Here *comp* is a suitably defined function, which the reader is invited to figure out herself. Note that the definition does not matter as long as we obtain the desired properties.

5 A Functional Program

Although I could prove the consistency of Broy's specifications, the result is not satisfactory from a programmer's point of view: a lot of logic, and in the end we do not even obtain an executable functional program but a set of recursion equations over multisets that still contain a choice operator. Surely we can do better! For a start, let us work on lists, multisets only complicate matters: they appear to be more abstract but *scan* has to operate on canonically ordered multisets, i.e. lists in disguise. Let us also drop all ideas of stepwise refinement: the algorithm is simple enough for a direct approach. The "bloodless algorithm" is trivially expressed as a function on lists:

$cand\ c\ k\ [] = c$
$cand\ c\ k\ (x{\cdot}xs) =$
$(\textsf{if}\ x = c\ \textsf{then}\ cand\ c\ (k + 1)\ xs$
$\ \textsf{else case}\ k\ \textsf{of}\ 0 \Rightarrow cand\ x\ 1\ xs \mid Suc\ n \Rightarrow cand\ c\ n\ xs)$

where [] is the empty list and $x{\cdot}xs$ is the list with head x and tail xs. Without further ado, here is the correctness theorem

$majority\ xs\ m \Longrightarrow cand\ c\ 0\ xs = m$

where $majority\ xs\ m$ abbreviates $length\ xs\ div\ 2 < length(filter\ (\lambda x.\ x{=}m)\ xs)$. The theorem follows trivially from this lemma

$majority\ (replicate\ k\ c\ @\ xs)\ m \Longrightarrow cand\ c\ k\ xs = m$

which is proved by induction on xs followed by simplification. The key is the no-frills formulation on lists with its rich background library that already provides the functions *length*, @ (append), *filter* and *replicate* (where $replicate\ k\ c$ is the list of k c's) for a compact description of the specification and its generalisation.

6 Conclusion

This paper has compared an axiomatic theorem prover from the early 1990s (LP) with a state-of-the-art definitional theorem prover of today (Isabelle/HOL). The progress that we have seen is largely conceptual. Most importantly, all of the axiomatic specifications could be replaced by definitions from which the axioms are derived. This is partly the result of the richer logic HOL that supports polymorphism, higher-order functions and choice operators. It is also the result of many years of system development: Isabelle/HOL offers inductive and recursive definition facilities that appear axiomatic on the surface but are definitional internally. The same is true of structuring facilities like locales that help to express axiomatic requirements specifications in a definitional manner. Last but not least, the continuous development of libraries is absolutely essential: The pleasingly succinct verification of the functional program in Section 5 would not have been possible without Isabelle/HOL's well-developed list library. The latter proof is also an excellent example of the *keep it simple* approach to verification.

Acknowledgement

I would like to thank Manfred Broy for his long-term support of the Isabelle project and many years of extremely pleasant cooperation.

References

[1] Ballarin C. Locales and locale expressions in Isabelle/Isar. In: Berardi S, Coppo M, Damiani F, eds. Types for Proofs and Programs (TYPES 2003). Lecture Notes in Computer Science, Springer-Verlag, 2004, 3085: 34–50.

[2] Boyer RS, Moore JS. A Computational Logic. Academic Press, 1979.

[3] Boyer RS, Moore JS. MJRTY: A fast majority vote algorithm. Automated Reasoning: Essays in Honor of Woody Bledsoe. Kluwer, 1991. 105–118.

[4] Blanchette JC, Nipkow T. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In: Kaufmann M, Paulson L, eds. Interactive Theorem Proving (ITP 2010). Lecture Notes in Computer Science, Springer-Verlag, 2010, 6172: 131–146.

[5] Broy M. Experiences with software specification and verification using LP, the Larch proof assistant. Formal Methods in System Design, 1996, 8(3): 221–272.

[6] Garland S, Guttag J. An overview of LP, the Larch prover. In: Dershowitz N, ed. Rewriting Techniques and Applications. Lecture Notes in Computer Science, Springer-Verlag, 1989, 355: 137–151.

[7] Gordon MJC, Melham TF, eds. Introduction to HOL: a Theorem-proving Environment for Higher Order Logic. Cambridge University Press, 1993.

[8] Lescanne P. Computer experiments with the Reve term rewriting system generator. Proc. 10th ACM Symp. Principles of Programming Languages. ACM Press, 1983. 99–108.

[9] Nipkow T, Paulson L, Wenzel M. Isabelle/HOL — A Proof Assistant for Higher-Order Logic. Lecture Notes in Computer Science, Springer-Verlag, 2002, 2283.