An assertional proof for a construction of an atomic variable

Hesselink, Willem

Published in:
Formal Aspects of Computing

DOI:
10.1007/s00165-004-0038-5

DOI 10.1007/s00165-004-0038-5
BCS © 2004
Formal Aspects of Computing (2004) 16: 387–393

Formal Aspects of Computing

An assertional proof for a construction of an atomic variable

Wim H. Hesselink
University of Groningen, Groningen, The Netherlands

Abstract. The paper proves by assertional means the correctness of a construction of Haldar and Subramanian of an atomic shared variable for one writer and one reader. This construction uses four unsafe variables and four safe boolean variables. Assignment to a safe but nonatomic variable is modelled as a repetition of random assignments concluded by an actual assignment. The proof obligation consists of four invariants. These are proved using 25 auxiliary invariants. The proof has been constructed and verified with the theorem prover NQTHM.

Keywords: Safeness; Atomicity; Wait-free; Invariant

1. Introduction

In this note we discuss the construction of Haldar and Subramanian [HaS94] of an atomic shared variable for one writer and one reader by means of four unsafe variables together with four safe boolean variables. Haldar and Subramanian [HaS94] gave a correctness proof based on behavioural reasoning. It becomes more and more accepted that reliable reasoning about concurrent algorithms requires assertional reasong, i.e., reasoning on the level of individual states and single computation steps. The algorithm is a good case to show that assertional reasoning can be used for programs with variables that are not atomic but only safe or even unsafe.

In order to make sense of this, we have to introduce the concepts of safeness and atomicity of shared variables. We only consider variables that are written by a single process. Since processes themselves are sequential, it follows that different write operations to the same variable never overlap.

For every shared variable, we assume at least that every read operation that does not overlap with any write operation, returns the most recently written value. Following [Lam86], a shared variable is called *safe* iff every read operation that overlaps with a write operation returns some legitimate value of the domain of the variable. A variable is called *unsafe* iff it is not safe. For an unsafe variable, a read operation is not allowed to overlap with write operations, since otherwise chaos may result.

A shared variable is *atomic* iff read and write operations behave as if they never overlap but always occur in some total order that refines the precedence order (an operation precedes another one iff it terminates before the other operation starts).

The program of [HaS94] that we treat here is a critical case since it uses no atomic shared variables, four unsafe data variables, and four safe control bits. It can be compared with Tromp's algorithm in [Tro89], which needs eight safe bits, and which is a refinement of the version with four atomic bits. In [Hes98], we gave an assertional proof of the version with four atomic bits. The safe bits complicate matters since, during the assignment to a safe

Correspondence and offprint requests to: Wim H. Hesselink, Department of Mathematics and Computing Science, Rijksuniversiteit Groningen, P.O. Box 800, 9700 AV Groningen, The Netherlands. Email: wim@cs.rug.nl or W.H.Hesselink@cs.rug.nl

bit, the bit can repeatedly change its value. When there are only four safe bits, one cannot preclude that a bit is inspected during an assignment.

One may notice that the paper [HaS94] makes the assumption that the data variables are safe. This is because they do not consider unsafe variables. They call the algorithm conflict-free to express that the assumption of safeness is superfluous.

In section 2 we present the formal model and the algorithm. In section 3, we briefly sketch the method developed in [Hes02a] to prove atomicity of read-write objects and we apply this method to obtain the assertional proof obligation, which in this case consists of four invariants. In section 4, we prove preservation of these invariants. We summarize our conclusions in section 5.

2. The formal model and the algorithm

We first describe the formal model. Since assertional reasoning depends on the states between atomic steps, we have to model the actions as atomic steps of some kind. By convention, shared variables are always assumed to be unsafe unless otherwise specified. They are written in typewriter font. Actions on private variables are considered atomic. Private variables are written in italics.

A read action of a shared variable $\mathtt{x}$ into a private variable v is denoted $v := \mathtt{x}$. A write action of a private expression E into an unsafe shared variable $\mathtt{x}$ is written

$$\text{(unsafe) } \mathtt{x} := E. \tag{0}$$

It is regarded as an atomic step, but the programmer has the proof obligation that, whenever a process is about to execute command (0), no other process is about to read or write $\mathtt{x}$ as well.

For a safe shared variable, we need to indicate that reading during a write action may return any value. We therefore denote a write action of a private expression E into a safe shared variable $\mathtt{x}$ by

$$\text{(flickering) } \mathtt{x} := E\ . \tag{1}$$

We model this in relational semantics, such as TLA [Lam94], by specifying that command (1) has the relational meaning

$$pc' = pc \quad \vee \quad (pc' = pc + 1 \ \wedge \ \mathtt{x}' = E)\ . \tag{2}$$

Here, the primes are used for the values of the variables after the step, pc stands for the location pointer, and by convention all shared or private variables apart from pc and $\mathtt{x}$ are unchanged. In other words, command (1) is modelled as a repetition of arbitrary assignments to $\mathtt{x}$ that ends with the actual assignment of E to $\mathtt{x}$. The value of $\mathtt{x}$ is indeterminate during the repetition. Liveness conditions are used to ensure that the repetition terminates.

The algorithm of [HaS94] uses an unsafe array `buf` of four items and four safe control variables to simulate an atomic variable of type *Item*. We use control variables `ww`, `rr`, and `c[0]` and `c[1]`, of the type $Bit = \{0, 1\}$. We thus declare

```
buf : array $Bit \times Bit$ of $Item$ ;
ww, rr : $SAFE\ Bit$ ;
c : array $Bit$ of $SAFE\ Bit$ .
```

The initial value v_0 of the abstract variable is stored in $\mathtt{buf}[0, 0] = v_0$. The control variables have the initial values $\mathtt{ww} = 0$, $\mathtt{rr} = 1$, and $\mathtt{c}[0] = \mathtt{c}[1] = 0$.

Recall that there is only one writer and one reader. The writer writes to `buf`, `ww` and `c`. The reader writes to `rr`. We represent the algorithm by means of two procedures *Write* and *Read*. The writing procedure writes to one of the four buffers and then inspects variable `rr` to get an estimate where the reader is reading. If the reader is reading a recent buffer, the writer writes again to a fresh one, see [HaS94]. For a bit b, we use $\neg b$ to denote $1 - b$.

```
proc Write ($vw$) :
    var $aw$ : $Bit$ := ww , $cw$ : $Bit$ ;
    20:   $cw := \neg$c[$aw$] ;
          (unsafe) buf[$aw, cw$] := $vw$ ;
    21:   (flickering) c[$aw$] := $cw$ ;
    22:   if $aw =$ rr then
    23:       $cw := \neg$c[$\neg aw$] ;
              (unsafe) buf[$\neg aw, cw$] := $vw$ ;
    24:        (flickering) $\texttt{c}[\neg aw] := cw$ ;
    25:        if  $aw = \texttt{rr}$  then
                   $aw := \neg aw$ ;
    26:            (flickering) $\texttt{ww} := aw$  fi ;
    27:    fi
end .
```

We have numbered the atomic steps for future reference. The numbers start at 20 to ease their use in our mechanical theorem prover. The grain of atomicity is such that every atomic step contains only one essential reference to a shared variable. The proviso "essential" is added, since the writer reads in steps 20 and 23 its own output variable c. This reading can easily be eliminated by introducing a private copy of c.

```
proc Read () :
    var $br : Bit := \texttt{rr}$ ,  $cr : Bit$ ;
    40:    if  $br \neq \texttt{ww}$  then
               $br := \neg br$ ;
    41:        (flickering) $\texttt{rr} := br$ ;
    42:        $cr := \texttt{c}[br]$ ;
    43:        $vr := \texttt{buf}[br, cr]$ ;
    44:    fi
end .
```

3. Proof obligations

Inspired by [Lyn96], we developed in [Hes02a] an assertional theory for atomicity of read-write objects, i.e., atomic shared variables. Theorem CRIT of [Hes02a] serves to prove atomicity. We shall apply it to the above protocol. We thus provide both processes with private integer ghost variables *start* and *sqn*. A shared integer ghost variable masq serves to denote the maximal *sqn* value of the completed operations. Initially $\texttt{masq} = t_0$ for some number t_0. Every process updates masq at the end of every operation by

$$\texttt{masq} := \max(sqn, \texttt{masq}) .$$

In every operation of a process, it updates its private variables *start* and *sqn* precisely once as described now.

Every operation of a process starts by copying the current value of masq to *start*. During every write operation, the writer increments *sqn* and attaches its new value of *sqn* as a kind of time stamp to the value to be written. When reading, the reader copies this attached number to *sqn*. The initial value v_0 of the abstract variable has the initial tag t_0.

According to theorem CRIT of [Hes02a], atomicity of the protocol is proved when we have the invariants that every write action has the postcondition $start < sqn$ for the writer and that every read action has the postcondition $start \leqslant sqn$ for the reader.

We therefore augment the procedures with the actions on the ghost variables. For convenience in the invariants, we replace the repeated procedure calls by an infinite loop for the writer and an infinite loop for the reader and we extend the lifetimes of the local variables of the procedures by replacing them with private variables of the processes such that initially $aw = 0$ and $br = 1$. The tags attached to the values are represented by a separate array tag, which is a ghost variable where initially $\texttt{tag}[0, 0] = t_0$. In this way, the code for the writer becomes:

```
Writer :   loop
    20:    $start := \texttt{masq}$ ;     $sqn := sqn + 1$ ;
           choose $vw$ ;     $cw := \neg\texttt{c}[aw]$ ;
           (unsafe) $\texttt{buf}[aw, cw] := vw$ ;
           $\texttt{tag}[aw, cw] := sqn$ ;
    21:    (flickering) $\texttt{c}[aw] := cw$ ;
    22:    if  $aw = \texttt{rr}$  then
    23:        $cw := \neg\texttt{c}[\neg aw]$ ;
               (unsafe) $\texttt{buf}[\neg aw, cw] := vw$ ;
               $\texttt{tag}[\neg aw, cw] := sqn$ ;
    24:        (flickering) $\texttt{c}[\neg aw] := cw$ ;
```

```
25:         if $aw = \texttt{rr}$ then
                $aw := \neg aw$ ;
26:             (flickering) $\texttt{ww} := aw$ fi fi ;
27:     $\texttt{masq} := \max(sqn, \texttt{masq})$ ;
    end loop .
```

Note that we now let the writer choose the value *vw* nondeterministically in 20.

```
Reader : loop
40:     $start := \texttt{masq}$ ;
        if $br \neq \texttt{ww}$ then
            $br := \neg br$ ;
41:         (flickering) $\texttt{rr} := br$ ;
42:         $cr := \texttt{c}[br]$ ;
43:         $vr := \texttt{buf}[br, cr]$ ;
            $sqn := \texttt{tag}[br, cr]$ fi ;
44:     $\texttt{masq} := \max(sqn, \texttt{masq})$ ;
    end loop .
```

We let q and r stand for the *pc* values of the writer and the reader, respectively.

Safeness of the assignments to the unsafe variable `buf` requires that whenever the writer is about to write $\texttt{buf}[i, j]$ and the reader is about to read $\texttt{buf}[m, n]$, then $(i, j) \neq (m, n)$. This is expressed in the two invariants

(Iq0) $q = 20 \quad \wedge \quad r = 43 \quad \wedge \quad aw = br \quad \Rightarrow \quad cr = \texttt{c}[br]$,
(Iq1) $q = 23 \quad \wedge \quad r = 43 \quad \wedge \quad \neg aw = br \quad \Rightarrow \quad cr = \texttt{c}[br]$.

In order to distinguish the private ghost variables *start* and *sqn* of writer and reader, we write *startw* and *sqnw* for the values at the writer and *startr* and *sqnr* for the values at the reader. As indicated above, the proof obligations for atomicity are the invariants

(Jq0) $startw < sqnw$,
(Jq1) $r = 44 \quad \Rightarrow \quad startr \leqslant sqnr$.

Stricly speaking, the inequality of (Jq0) is required only when $q = 27$, but it is easier to prove the stronger invariant (Jq0). For simplicity, we use $t_0 = 1$ and assume that $\texttt{tag}[i, j] = 1$ for all i, j, and $sqnw = 1$ and $startw = sqnr = 0$ initially.

4. Verification

We thus have to verify that our concurrent program with 13 atomic statements satisfies the invariants (Iq0), (Iq1), (Jq0), (Jq1). We use the method described in [Hes02a] Section 3.3. The program is verified using two events files [Hes02b] for the theorem prover NQTHM of [BoM88]. The file `concprelude` defines the semantics for shared variable concurrency. The file `regvarHS` contains the program, the development of the invariants, and their initialization. We now proceed with a fairly detailed account of the proof.

Since the shared variables `ww` and `rr` are only safe variables, we avoid them as much as possible in the invariants and use their local copies *aw* and *br* instead, based on the following two invariants:

(Kq0) $r = 41 \quad \vee \quad \texttt{rr} = br$,
(Kq1) $q = 26 \quad \vee \quad \texttt{ww} = aw$.

Preservation of (Kq0) and (Kq1) is easy. They hold initially because of the initialization chosen. Since the assignments at 41 and 26 are flickering, nothing can be said about the value of `rr` at 41 or the value of `ww` at 26.

In order to prove (Iq0), we generalize it to

(Iq0) $q \in \{20, 26, 27\} \quad \wedge \quad r = 43 \quad \wedge \quad aw = br \quad \Rightarrow \quad cr = \texttt{c}[br]$.

This invariant is threatened only when $q = 22$ or $q = 25$, in which cases its preservation follows from (Kq0). Preservation of (Iq1) is threatened only when $q = 22$, in which case its preservation follows also from (Kq0).

In order to prove (Jq0), we postulate that *sqnw* is always the largest number around, as expressed in the invariants

(Kq2) $\texttt{masq} \leqslant sqnw$,
(Kq3) $sqnr \leqslant sqnw$,
(Kq4) $\texttt{tag}[i, j] \leqslant sqnw$.

In (Kq4), the variables *i* and *j* are free and range over *Bit*. The invariant (Jq0) is threatened only when $q = 20$. It is preserved at 20 because of (Kq2). We need (Kq3) to prove preservation of (Kq2) when $r = 44$, and (Kq4) to prove preservation of (Kq3) when $r = 43$. The remaining verifications are easy. For example, all three predicates hold initially. This concludes the treatment of the first three proof obligations.

4.1. The last proof obligation

The proof of invariance of (Jq1) is complicated. We provide it for the sake of completeness and illustration. Since we only want to consider invariants of a relatively simple form, the proof of (Jq1) turns out to need twenty auxiliary invariants.

We first settle a number of auxiliary invariants. Since masq never decreases and *startr* is only set equal to masq, we obviously have the invariant:

(Lq0) $startr \leqslant \texttt{masq}$.

Since tag, c, *sqnw*, are modified by the writer almost without interaction with the reader, there are some relatively easy invariants which express *sqnw* in terms of tag.

(Lq1) $q \neq 21 \;\Rightarrow\; \texttt{tag}[aw, \texttt{c}[aw]] = sqnw$,
(Lq2) $q = 21 \;\Rightarrow\; \texttt{tag}[aw, cw] = sqnw$,
(Lq3) $q = 21 \;\Rightarrow\; \texttt{tag}[aw, \neg cw] + 1 = sqnw$,
(Lq4) $q \in \{25, 26\} \;\Rightarrow\; \texttt{tag}[\neg aw, \texttt{c}[\neg aw]] = sqnw$,
(Lq5) $q = 24 \;\Rightarrow\; \texttt{tag}[\neg aw, cw] = sqnw$.

The invariant (Lq1) is preserved at 21 and 25 because of (Lq2) and (Lq4). Predicate (Lq3) is preserved at 20 because of (Lq1). Predicate (Lq4) is preserved at 24 because of (Lq5). The rather surprising preservation of (Lq4) at 25 is due to (Lq1). Note that the mutual dependence of (Lq1) and (Lq4) does not introduce circular reasoning: all invariants are postulated in the preconditions of all steps, and we then prove that they hold in the postconditions.

Predicate (Jq1) only contains private variables of the reader. The writer therefore preserves (Jq1). Since the reader can reach $r = 44$ from 40 and 43, predicate (Jq1) is threatened at 40 and 43. For preservation of (Jq1) at 43, it is necessary and sufficient to postulate the invariant

(Sq0) $r = 43 \;\Rightarrow\; startr \leqslant \texttt{tag}[br, cr]$.

In the remainder of this text, we only give the remaining invariants and why we need them, but we omit the arguments to show that they are sufficient. The reader interested in the details and willing to read lisp-like code is referred to [Hes02b].

We postpone the treatment of (Jq1) and first settle (Sq0). For this purpose, we introduce the following invariants concerning *startr*.

(Sq1) $r \in \{41, 42\} \;\Rightarrow\; startr \leqslant \texttt{tag}[br, \texttt{c}[br]]$,
(Sq2) $q = 24 \;\wedge\; aw \neq br \;\Rightarrow\; startr \leqslant \texttt{tag}[br, \neg cw]$,
(Sq3) $q = 23 \;\wedge\; aw \neq br \;\Rightarrow\; startr \leqslant \texttt{tag}[br, \texttt{c}[br]]$.

Predicate (Sq1) is introduced to preserve (Sq0) at 42. Similarly, (Sq2) is introduced for (Sq1) at 24 and (Sq3) for (Sq2) at 23. Preservation of (Sq3) at 22 is worth mentioning. Because of the test in 22 and invariant (Kq0), the writer only arrives at 23 with $aw \neq br$ when the reader is at 41. Therefore (Sq3) is preserved because of (Sq1).

In addition to (Lq2) and (Lq3), the next invariants express that *sqnw* is large when $q = 21$:

(Tq0) $q = 21 \;\wedge\; r \notin \{40, 44\} \;\Rightarrow\; \mathtt{masq} < sqnw$,
(Tq1) $q = 21 \;\wedge\; aw \neq br \;\Rightarrow\; \mathtt{masq} < sqnw$,
(Tq2) $q = 21 \;\wedge\; aw \neq br \;\Rightarrow\; sqnr < sqnw$,
(Tq3) $q = 21 \;\Rightarrow\; \mathtt{tag}[\neg aw, j] < sqnw$.

The variable j in (Tq3) ranges over *Bit*. The predicates (Tq0) and (Tq1) are introduced to preserve (Sq1) at 21 and 40, respectively; (Tq2) is introduced for (Tq1) at 44; (Tq3) is introduced for (Tq2) at 43. These predicates together are strong enough to prove preservation of (Sq0).

Remark. The invariants (Tq0) and (Tq1) imply that when $q = 21$ and $\mathtt{masq} = sqnw$, then $aw = br$ and $r \in \{40, 44\}$. Indeed, the antecedent can occur during an execution, namely when the writer is repeatedly executing 21 (see formula (2) in section 2), while the reader completes reading with $br = aw$ and $cr = cw$. It then sets `masq` *equal to sqnw and can freely move between 40 and 44.* □

Preservation of (Jq1) at 40 is also subtle, since `ww` may be flickering at 26. For treating this case, we introduce the following invariants to express that *sqnr* is large enough.

(Uq0) $r \in \{40, 44\} \;\wedge\; q = 26 \;\Rightarrow\; \mathtt{masq} \leqslant sqnr$,
(Uq1) $r \in \{40, 44\} \;\wedge\; aw = br \;\Rightarrow\; \mathtt{masq} \leqslant sqnr$,
(Uq2) $r \in \{40, 44\} \;\wedge\; aw = br \;\wedge\; q \in \{26, 27\} \;\Rightarrow\; sqnw = sqnr$.

Indeed, (Uq0) and (Uq1) are introduced for (Jq1) at 40. Predicate (Uq2) is needed for (Uq1) at 27. In order to prove preservation of (Uq0), (Uq1), and (Uq2) at 43, we finally postulate the invariants

(Uq3) $r = 43 \;\wedge\; q = 26 \;\Rightarrow\; \mathtt{masq} \leqslant \mathtt{tag}[br, cr]$,
(Uq4) $r = 43 \;\wedge\; aw = br \;\Rightarrow\; \mathtt{masq} \leqslant \mathtt{tag}[br, cr]$,
(Uq5) $r = 43 \;\wedge\; aw = br \;\wedge\; q \in \{26, 27\} \;\Rightarrow\; sqnw = \mathtt{tag}[br, cr]$.

It turns out that this completes the construction of a sufficient set of constituent invariants: their conjunction is stable, holds initially, and implies the four proof obligations of section 3.

5. Conclusion

The proof that the unsafe variables $\mathtt{buf}[i, j]$ are treated correctly in the lines 20 and 23 is straightforward and only requires the invariants (Iq0), (Iq1), and (Kq0). The atomicity proof is complicated because of the flickering assignments in lines 21, 24, 26, and 41. Line 21 is the most critical location.

The proof of [HaS94] is based on Lamport's method [Lam86] that uses the precedence relation between operation executions. We find this behavioural proof extremely difficult to follow and therefore unconvincing. Our proof may be tedious, but it is conceptually straightforward. In any case, it gives independent confirmation of the correctness of the algorithm.

We only consider our proof trustworthy, however, because of our use of the theorem prover NQTHM. This prover also enabled us to modify the set of constituent invariants. Indeed, in earlier versions of our proof, we did not yet have (Lq3), and the invariants (Tq) had inequalities with "$\leqslant \mathtt{tag}[aw, \neg cw]$" instead of "$< sqnw$". When writing this paper, we changed this since we find the present form somewhat easier to understand. Mechanical verification enabled us to make the change in a reliable way and with little effort.

References

[BoM88] Boyer RS, Moore JS (1988) A computational logic handbook. Academic, Boston
[HaS94] Haldar S, Subramanian K (1994) Space-optimum conflict-free construction of 1-writer 1-reader multivalued atomic variable. In: Proceedings of the eighth international workshop on distributed algorithms, (LNCS 857), Springer, Berlin Heidelberg New York, pp 116–129
[Hes98] Hesselink WH (1998) Invariants for the construction of a handshake register. Inf Process Lett 68:173–177
[Hes02a] Hesselink WH (2002) An assertional criterion for atomicity. Acta Informatica 38:343–366
[Hes02b] Hesselink WH `www.cs.rug.nl/~wim/mechver/imperative`: NQTHM proof of a construction of an atomic variable, the events files of `concprelude` and `regvarHS`

[Lam86] Lamport L (1986) On interprocess communication, Parts I and II. Distrib Comput 1:77–101
[Lam94] Lamport L (1994) The temporal logic of actions. ACM Trans Program Lang Syst 16:872–923
[Lyn96] Lynch NA (1996) Distributed algorithms. Morgan Kaufman, San Francisco
[Tro89] Tromp J (1989) How to construct an atomic variable. In: Bermond J-C, Raynal M (eds) Distributed algorithms, proceedings nice. (LNCS 392), Springer, Berlin Heidelberg New York, pp 292–302

Received May 2002
Accepted in revised form March 2004 by D.J.Cooke
Published online 5 May 2004