Sādhanā, Vol. 21, Part 2, April 1996, pp. 245–259. © Printed in India.

Weak atomicity: A helpful notion in the construction of atomic shared variables

K VIDYASANKAR

Department of Computer Science, Memorial University of Newfoundland, St. John's, Newfoundland, Canada A1C 5S7

Abstract. A new class of 1-writer shared variables, called *weakly atomic* variables, is defined, and an elegant general method of constructing atomic variables from weakly atomic ones is presented in this paper. Four examples of atomic variable constructions that use this method are described. Two of these constructions are new.

Weak atomicity provides an intermediate step between regularity and atomicity. In addition to enabling new constructions, this concept helps to derive simple correctness proofs of the constructions.

Keywords. Weakly atomic variables; atomic variable constructions; atomic shared variables.

1. Introduction

A *shared variable* is an abstraction of asynchronous interprocess (persistent) communication, where the senders and the receivers are called the *writers* and the *readers*, respectively, and the states of the communication medium are the *values* of the shared variable. A writer *writes*, that is, puts a value in the variable, and a reader *reads*, that is, reports a value from the domain of the variable. Writing and reading are the only operations in a *Read/Write variable* (*variable*, for short, in this paper). In this paper, 'Write' and 'Read' are used as nouns, referring, respectively, to a write operation execution and a read operation execution, and 'write' and 'read' as verbs.

Recent interest is on constructions of shared variables which have the following properties: (i) the operation executions are not assumed to be *atomic*, that is, they are not instantaneous; and (ii) they are *wait-free*, that is, no operation execution waits for any other operation to finish its execution. The first property allows treating operation executions uniformly, even when some operations are high level ones implemented using low level operations. The second property implies that each operation execution will take at most a fixed amount of time, irrespective of the presence of other operation executions and their relative speeds, and the (harmless) failure of one operation execution does not affect other operation executions.

The above two properties give rise to a classification of shared variables, depending on their output characteristics. Lamport (1986) defines three categories for 1-writer variables, using a precedence relation on operation executions defined as follows: for operation executions a and b, *a precedes b*, denoted $a \longrightarrow b$, if a finishes before b starts; a and b *overlap* if neither a precedes b nor b precedes a. We note that a 1-writer variable is written by one and only one process, and not by many processes mutually exclusively. In 1-writer variables, all the Writes are totally ordered by "$\longrightarrow$". (In this paper, we are concerned only with 1-writer variables.)

DEFINITION 1
(Awerbuch *et al* 1988) An *execution* of a shared variable construction (called a *run* in Awerbuch *et al* 1988) is a tuple $\langle A, \longrightarrow, \pi\rangle$, where A is a set of operation executions (Reads and Writes) on the shared variable, $\longrightarrow$ is a precedence relation on A and π is a partial *reading mapping* from the set of Reads to the set of Writes in A, such that for Read r if $\pi(r) = w$ then r returns the value written by w, and if $\pi(r)$ is undefined then r may return any value from the domain of the variable. □

DEFINITION 2
A Read r in an execution $\langle A, \longrightarrow, \pi\rangle$ is *regular* if

(i) $\pi(r)$ is defined,
(ii) $r \not\longrightarrow \pi(r)$ and
(iii) for no Write w in A, $\pi(r) \longrightarrow w \longrightarrow r$. □

DEFINITION 3
(Awerbuch *et al* 1988) An execution $\langle A, \longrightarrow, \pi\rangle$ on a shared variable is

(a) *safe* if each Read in that execution that does not overlap with any Write is regular,
(b) *regular* if each Read, whether it overlaps with a Write or not, is regular, and
(c) *atomic* if it is regular and in addition there is a total order $\Longrightarrow$ on the set of operation executions as follows:
    (i) for a, b in A, if $a \longrightarrow b$ then $a \Longrightarrow b$;
    (ii) for each Read r in A, $\pi(r) \Longrightarrow r$; and
    (iii) for each read r in A there is no Write w in A such that $\pi(r) \Longrightarrow w \Longrightarrow r$. □

DEFINITION 4
(Awerbuch *et al* 1988; Lamport 1986) A shared variable is *safe, regular* or *atomic* if each execution on the variable is safe, regular or atomic, respectively. □

A shared variable is *boolean* or *multivalued* depending upon whether it can hold only boolean or any number of desired values.

With these classifications we can define a hierarchy on shared variables, with 1-writer 1-reader boolean safe variable in the lowest level and multiwriter multireader multivalued atomic variable in the highest level. Higher-level shared variables can be constructed from lower-level ones. Several such constructions have been proposed in the literature.

In this paper we consider wait-free construction of 1-writer atomic variables from regular variables. We first state a proposition that describes the distinguishing property which makes a regular variable atomic. In 1-writer variables, the precedence relation $\longrightarrow$ induces a total order on the Writes. We use the notation $\prec$ for the total order. That is, for Writes a and b, $a \longrightarrow b$ if and only if $a \prec b$. We also use $a \preceq b$ to denote a equals b or $a \prec b$.

PROPOSITION 1
(Awerbuch *et al* 1988; Lamport 1986) *A 1-writer multireader multivalued regular shared variable is atomic if it has the additional property that, in each execution on that variable, for any two Reads r and r′ such that* $r \longrightarrow r'$, $\pi(r) \preceq \pi(r')$. □

In a regular variable, it is possible that for r, r' such that $r \longrightarrow r'$, $\pi(r') \prec \pi(r)$. This situation is called *new-old inversion*. Proposition 1 identifies "no new-old inversion" as the single additional property that is required of a regular variable to become atomic. Different techniques have been employed to overcome new-old inversion in the literature, for example, in the constructions of Lamport (1986, Construction 5), Vidyasankar (1991), and Burns & Peterson (1988). There is no uniformity in the techniques, and each construction warrants a different, and sometimes involved, proof of correctness.

In this paper, we first identify a general method of converting regular variable constructions to atomic ones. This method is conceptually simple, but is difficult to implement. We then show that for a new class of regular variables, called *weakly atomic* variables, the implementation of this method is easier. Now the construction of weakly atomic variables from regular variables appears to be a much simpler problem. Some regular variable constructions that have appeared in the literature are also weakly atomic. Some others are convertible to weakly atomic ones in simple manner. Thus, with the identification of weak atomicity as a possible intermediate step between regularity and atomicity, the problem of constructing atomic variables from regular ones is simplified.

This paper is organized as follows. The above mentioned general method of converting regular variable constructions to atomic ones, the weak atomicity concept, and a general method of converting weakly atomic variable constructions to atomic variable ones are described in § 2. We then take, in § 3, four weakly atomic variable constructions and derive atomic variable constructions from them using the general method. Two of these weakly atomic variable constructions have been obtained by slightly modifying the regular variable constructions of Lamport (1986, Construction 4), Chaudhuri & Welch (1990). The resulting atomic variable constructions are new. The other two weakly atomic variable constructions have been obtained the other way, from atomic variable constructions in the literature. Here our intention is to illustrate that the weak atomicity concept can provide better insight into some existing atomic variable constructions, and also simplify the correctness proofs. Section 4 concludes the paper.

2. Weak atomicity

In an execution, we define a Read r to be *dependable* if for all Reads r' such that $r \longrightarrow r'$, $\pi(r) \preceq \pi(r')$. In an execution on a regular variable, some Reads are dependable, some are not. If all Reads, in all executions, are dependable then, by proposition 1, the variable is atomic.

Let us consider the following method of converting regular variable constructions to atomic ones. Let V be a regular variable and **read**(V) and **write**(V) denote the Read and Write operations on V. We define a Read operation **read***(V) on V as follows:

> It consists of a sequence of one or more **read**(V) executions and returns the value read in one of them such that **read***(V) is dependable; that is, if R is a **read***(V) execution, then for any other **read***(V) execution R' such that $R \longrightarrow R', \pi(R) \preceq \pi(R')$.

This means that if R consists of **read**(V) executions $r_1, r_2, \cdots, r_n$ and returns the value read by r_i for some i, and R' consists of $r'_1, r'_2, \cdots, r'_m$ and returns the value read by r'_j for some j, then $\pi(r_i) \preceq \pi(r'_j)$.

There are two problems with this approach: (i) the number of times **read**(V) is executed by a **read***(V) execution must be bounded by a constant for the wait-freedom property, and (ii) the check for dependability must be feasible. Both these problems are difficult in general. In the following, we define a special class of regular variables, called weakly atomic ones, for which these problems appear to be easier to solve.

First we introduce some additional terminology and state a basic property. For operation executions a and b on a shared variable, $a \dashrightarrow b$ will mean that a starts before b finishes. That is, if $a \dashrightarrow b$, then either a precedes b or it overlaps b; in other words, $b \not\longrightarrow a$. We also assume that if $b \not\longrightarrow a$, then $a \dashrightarrow b$. That is, we assume global time model (Lamport 1986).

PROPOSITION 2

(Lamport 1986) *For operation executions b and c on a shared variable, and any operation executions a and d, if* $a \longrightarrow b \dashrightarrow c \longrightarrow d$, *then* $a \longrightarrow d$.

Proof. The implication follows by the transitivity of (i) a finishes before b starts, (ii) b starts before c finishes and (iii) c finishes before d starts. □

PROPOSITION 3

Consider an execution on a 1-writer multireader multivalued regular variable. Suppose r and r′ are Reads such that $r \longrightarrow r'$ *and* $\pi(r) \prec \pi(r')$. *Then for any Read r″ such that* $r' \longrightarrow r''$, $\pi(r) \preceq \pi(r'')$.

Proof. Suppose that $\pi(r) \not\preceq \pi(r'')$. Then, since all the Writes are totally ordered in the 1-writer case, $\pi(r'') \prec \pi(r)$. Now, since the variable is regular, $\pi(r') \dashrightarrow r'$, by definition 2(ii). Therefore we have $\pi(r'') \longrightarrow \pi(r) \longrightarrow \pi(r') \dashrightarrow r' \longrightarrow r''$. This implies, by proposition 2, that $\pi(r'') \longrightarrow \pi(r) \longrightarrow r''$. This violates part (iii) of definition 2, contradicting the regularity of r''. □

We note that, if for the Reads r, r' such that $r \longrightarrow r'$ in the above proposition, we only have $\pi(r) = \pi(r')$, then it is not necessarily true that for any r'' that succeeds r', $\pi(r) \preceq \pi(r'')$. (A counterexample is where r, r' and r'' all overlap $\pi(r)$, and $\pi(r'')$

immediately precedes $\pi(r)$, that is, there is no other Write in between $\pi(r'')$ and $\pi(r)$. Here $\pi(r'')$ is the most recently completed Write for r'', and hence r'' can return the $\pi(r'')$ value without violating the regularity property.) We call those regular variables for which this property holds weakly atomic.

DEFINITION 5
A 1-writer regular shared variable is *weakly atomic* if the following property holds in each execution on that variable: if r and r' are Reads such that $r \longrightarrow r'$ and $\pi(r) = \pi(r')$, then for any r'' such that $r' \longrightarrow r''$, $\pi(r) \preceq \pi(r'')$. □

Thus weakly atomic variables have the nice property that for a Read r in an execution, if for some Read r' that succeeds r, $\pi(r) \preceq \pi(r')$, then for all Reads r'' that succeed r', $\pi(r) \preceq \pi(r'')$. Therefore a **read***(V) execution R could consist of a sequence of one or more **read**(V) executions $r_1, r_2, \cdots, r_n$ such that, for some i, $\pi(r_i) \preceq \pi(r_n)$, returning the value read by r_i. The following proposition implies that n needs to be at most 3.

PROPOSITION 4
Consider an execution on a 1-writer multireader multivalued regular variable. For any three Reads r, r' and r'' such that $r \longrightarrow r' \longrightarrow r''$, if $\pi(r') \not\preceq \pi(r'')$ then $\pi(r) \preceq \pi(r')$.

Proof. Assume the contrary. Then $\pi(r'') \prec \pi(r') \prec \pi(r)$, that is, $\pi(r'') \longrightarrow \pi(r') \longrightarrow \pi(r)$. Since $\pi(r) \dashrightarrow r$, we have $\pi(r'') \longrightarrow \pi(r') \longrightarrow \pi(r) \dashrightarrow r \longrightarrow r''$. It follows by proposition 2 that $\pi(r'') \longrightarrow \pi(r') \longrightarrow r''$, contradicting the regularity of r''. □

Therefore, in a weakly atomic variable, for any three Reads r, r' and r'' such that $r \longrightarrow r' \longrightarrow r''$, r' is dependable when $\pi(r') \preceq \pi(r'')$, and r is dependable when $\pi(r') \not\preceq \pi(r'')$. Hence the operation **read***(V) can be described as follows.

```
Function read*(V):
  $r_1$ : $val_1$ := read($V$) ;
  $r_2$ : $val_2$ := read($V$) ;
  $r_3$ : $val_3$ := read($V$) ;
    if $\pi(r_2) \preceq \pi(r_3)$ then return $val_2$ else return $val_1$.
```

We note that the relation $\preceq$ in the above function can be replaced by $=$ also; this is justified by the following proposition.

PROPOSITION 5
Consider an execution on a 1-writer multireader multivalued regular variable. For any three Reads r, r' and r'' such that $r \longrightarrow r' \longrightarrow r''$, if $\pi(r') \neq \pi(r'')$, then either $\pi(r) \preceq \pi(r')$ or $\pi(r) \preceq \pi(r'')$.

Proof. Suppose both $\pi(r')$ and $\pi(r'')$ precede $\pi(r)$. Now either $\pi(r')$ precedes $\pi(r'')$ or vice versa. In the former case, we have $\pi(r') \longrightarrow \pi(r'') \longrightarrow \pi(r) \dashrightarrow r \longrightarrow r'$. This implies $\pi(r') \longrightarrow \pi(r'') \longrightarrow r'$, contradicting the regularity of r' (part (iii) of definition 2). In the latter case, we have $\pi(r'') \longrightarrow \pi(r') \longrightarrow \pi(r) \dashrightarrow r \longrightarrow r''$. This implies $\pi(r'') \longrightarrow \pi(r') \longrightarrow r''$, contradicting the regularity of r''. □

The above description of **read***(V) is a schematic one – proper code must be substituted for checking $\pi(r_2) \preceq \pi(r_3)$ (or $\pi(r_2) = \pi(r_3)$) to get a complete function definition. The method of checking the condition depends on the variable V. Also, it may be possible to do the checking without actually performing r_3. Then, the third Read can be eliminated in the **read***(V) execution. Further optimization is also possible in some cases. These are illustrated in the constructions in the next section.

DEFINITION 6
For a **read***(V) operation execution R returning $\pi(r_i)$ value, $\pi(R)$ is defined to be $\pi(r_i)$. □

3. Atomic variable constructions

In this section, we present four atomic variable constructions derived from weakly atomic variable constructions using the general procedure described in § 2. The **read***(V) description involves specifying how the condition $\pi(r_i) \preceq \pi(r_j)$ or $\pi(r_i) = \pi(r_j)$ is checked in the **if** statement. The **write** procedure and the **read** and **read*** functions are written in a Pascal-type language. The blocks are shown by indentation, rather than with 'begin's and 'end's.

In the proofs we use the following notation. For an operation execution a on the variable V, $a[x]$ denotes the execution of the suboperation of a on the (sub)variable x; the argument is expanded as $[x = value]$ to indicate the value that is read or written. For an atomic variable y, the total ordering imposed on the operation executions on y will be denoted $\Longrightarrow_y$; the subscript y will be omitted if it is clear from the context. For two operation executions a and b on y, we recall from definition 3 that if $a \longrightarrow b$ then $a \Longrightarrow b$. The property that if $a \Longrightarrow b$ then $a \dashrightarrow b$ is easy to verify.

3.1 *Lamport's Construction (Lamport 1986) (with atomic bits)*

The construction of Lamport [(Lamport 1986): Construction 4] implements 1-writer multireader multivalued regular variable from 1-writer multireader boolean regular variables. Lamport has shown in the same paper that even if the boolean variables (bits) are atomic, that construction does not implement an atomic variable. We show that when the bits are atomic, the same construction implements a weakly atomic variable. Then we define **read***(V) to get an atomic variable construction.

The bits are $v_1, v_2, \cdots, v_n$. The construction uses unary encoding in which a value k is denoted by zeroes in bits 1 through $k-1$ and a one in bit k. To write the value k, the writer first sets bit k to one and then sets bits $k-1$ through 1 to zero, writing from right to left. A reader reads the bits from left to right (1 to n) until it finds a one. The write and read operations for V and the read operation **read***(V) are as follows. We assume that the initial values of all the variables are zeroes, and that the first operation execution on V is an initializing Write that does not overlap with any Read.

Construction 1.

```
Procedure write(V) writing value k:
  write 1 in $v_k$;
  for $i := k - 1$ step $-1$ until 1 do write 0
  in $v_i$.

Function read(V):
  for $k := 1$ until $n$ do
    $val :=$ read($v_k$);
    if $val = 1$ then return $k$ and exit.
  (* $\pi(r)$ is $w$, where $\pi(r[v_k]) = w[v_k]$ *)

Function read*(V):
  $r_1 : k_1 :=$ read(V) ;
  $r_2 : k_2 :=$ read(V) ;
  $r_3 : k_3 :=$ read(V) ;
    if $k_2 \le k_3$ then return $k_2$ else return $k_1$.
```

The regularity of V, even when the bits are regular, has been shown in Lamport (1986). In the following, we show the regularity of V (with atomic bits) to make this subsection self-contained, and then show the weak atomicity of V and the dependability of **read***(V) operations.

Lemma 1. Suppose r is a Read of V returning value k and $\pi(r)$ is w. Then there is no Write w' such that for some l, $l \le k$, $w[v_l] \Longrightarrow w'[v_l] \Longrightarrow r[v_l]$.

Proof. Suppose on the contrary that the statement of the lemma does not hold. Let w' be the latest Write succeeding w, and l the largest index such that $w[v_l] \Longrightarrow w'[v_l] \Longrightarrow r[v_l]$. For the case $l = k$, we get a contradiction to $\pi(r[v_k]) = w[v_k]$. For $l < k$ and w' writing 1 in v_l, our choice of w' implies that r should read 1 from v_l, contradicting the assumption that it reads 0. In the remaining case, for $l < k$ and w' writing 0 in v_l, we have $w'[v_{l+1}] \longrightarrow w'[v_l] \Longrightarrow r[v_l] \longrightarrow r[v_{l+1}]$, implying $w[v_{l+1}] \Longrightarrow w'[v_{l+1}] \Longrightarrow r[v_{l+1}]$, which contradicts the choice of l. □

Lemma 2. The variable V is regular.

Proof. Let r be a Read in an execution of V. We show that all the three properties of definition 2 are satisfied for r. Suppose r reads 0 from $v_1, \ldots, v_{k-1}$, and 1 from v_k. It is a simple exercise to show that there is such a k. Then $\pi(r)$ is defined as the Write w such that $\pi(r[v_k]) = w[v_k]$. From the atomicity, and hence the regularity, of v_k, we have $r[v_k] \not\longrightarrow w[v_k]$. Hence it follows that $r \not\longrightarrow w$. That there is no Write w' such that $w \longrightarrow w' \longrightarrow r$ follows from lemma 1. □

Lemma 3. For **read**(V) *executions r_a and r_b such that $r_a \longrightarrow r_b$, reading values k and k' respectively, if $k \le k'$ then $\pi(r_a) \preceq \pi(r_b)$.*

Proof. Suppose that $k \leq k'$, but $\pi(r_b) \prec \pi(r_a)$. Denote $\pi(r_a)$ as w_a, and $\pi(r_b)$ as w_b. Then we have $w_b[v_k] \Longrightarrow w_a[v_k] \Longrightarrow r_a[v_k] \Longrightarrow r_b[v_k]$, which contradicts lemma 1 (with r_b as r). □

Lemma 4. The variable V is weakly atomic.

Proof. Suppose for **read**(V) operation executions r and r' such that $r \longrightarrow r'$, $\pi(r) = \pi(r')$. We show that for any r'' such that $r' \longrightarrow r''$, $\pi(r) \preceq \pi(r'')$. Denote $\pi(r)$ as w and $\pi(r'')$ as w''. Let k and k'' be the values written by w and w'', respectively. If $k \leq k''$, then by lemma 3, $\pi(r) \preceq \pi(r'')$. So assume $k'' < k$. Suppose $\pi(r'') \prec \pi(r)$. Then $w'' \longrightarrow w[v_k] \Longrightarrow r[v_k] \longrightarrow r'$ implies $w'' \longrightarrow r'$ (by the property that if $a \Longrightarrow b$ then $a \dashrightarrow b$, and by proposition 2). Then $w''[v_{k''} = 1] \Longrightarrow r'[v_{k''} = 0]$ implies that there exists some w''' such that $w''[v_{k''}] \Longrightarrow w'''[v_{k''} = 0] \Longrightarrow r'[v_{k''}]$. This implies $w''[v_{k''}] \Longrightarrow w'''[v_{k''}] \Longrightarrow r''[v_{k''}]$, contradicting lemma 1 (with r'' as r). □

Theorem 1. *Construction 1, with* **write**(V) *and* **read***(V) *, implements an atomic variable.*

Proof. The dependability of **read***(V) follows from the weak atomicity of V, by lemma 4, and (i) from lemma 3, when k_2 is returned, and (ii) from proposition 5 and the fact that if $k_2 \neq k_3$ then $\pi(r_2) \neq \pi(r_3)$, when k_1 is returned. □

3.2 *Tree Construction (Chaudhuri & Welch 1990) (with atomic bits)*

The construction of Chaudhuri & Welch (1990) implements 1-writer multireader multivalued regular variable from 1-writer multireader boolean regular variables, in a way different from that of Lamport's in the above subsection. We show that, here also, when the boolean variables are atomic, the same construction implements a weakly atomic variable. Then we define **read***(V) to get an atomic variable construction.

The shared variables (bits) are the internal nodes of a binary tree, whose leaves correspond to the values being written. The tree represents a sort of binary search conducted by the Reads to find the value written. The Reads take a path from the root to a leaf, whereas the Writes follow the path starting from a leaf to the root. The path in the tree taken by a Read, along with the values read from the internal nodes, uniquely defines the value read from the tree. In the following, we describe these operations in more detail.

A Write, writing a value v, writes into the set of variables which form the path between the root and the leaf labelled v, as follows:

- The first internal node written is the parent of the leaf labelled v. If the leaf node is the left/right child, the value written is 0/1.
- The ith internal node written is the parent of the $(i-1)$st node. If the $(i-1)$st node is the left/right child, the value written is 0/1.
- The last node written is the root.

A Read reads the set of variables which form the path from the root to a leaf labelled v, for some v. It subsequently returns v.

- The root node is the first node read.
- Suppose the ith node read has value 0/1. Then, if its left/right child is a leaf, then the value v, where v is the label of the leaf, is returned. Otherwise, the left/right child of the ith node is the $(i+1)$st node read.

For the case where the variables and values form a *complete* binary tree, we describe the algorithm formally. Let $v_m v_{m-1} \cdots v_1$ be the binary representation of the n-ary value v, where $m = \log n$ and n is a power of 2. The root variable is labelled ϵ. For each variable labelled with the binary string l, the strings $l0$ and $l1$ are the labels of its left and right children, respectively. We assume that the initial values of all the variables are zeroes, and that the first operation execution on V is an initializing Write that does not overlap with any Read. The operations are as follows. (The notation $v_m \cdots v_m$ refers to the variable v_m and $v_m \cdots v_{m+1}$ refers to the root variable ϵ.)

Construction 2.
Procedure **write**(V) writing value $v = v_m v_{m-1} \cdots v_1$:

```
for $p := 1$ to $m$ do
    write $v_p$ in variable $v_m \cdots v_{p+1}$.
```

Function **read**(V):

```
for $p := m$ step $-1$ until 1 do
    $v_p :=$ read variable $v_m \cdots v_{p+1}$
return $v_m \cdots v_1$.
(* $\pi(r)$ is $w$, where $\pi(r[v_m \cdots v_2]) = w[v_m \cdots v_2]$*)
```

Function **read***(V):

```
$r_1 : k_1 :=$ read$(V)$ ;
$r_2 : k_2 :=$ read$(V)$ ;
$r_3 : k_3 :=$ read$(V)$ ;
    if $k_2 = k_3$ then return $k_2$ else return $k_1$.
```

The regularity of V has been shown in Chaudhuri & Welch (1990) with regular bits. In the following, we show the regularity with atomic bits.

Lemma 5. Suppose r is a Read of V returning value k and $\pi(r)$ is w. Then there is no Write w' such that for some node x, in the path from the root to the leaf representing k, $w[x] \Longrightarrow w'[x] \Longrightarrow r[x]$.

Proof. Suppose on the contrary that the statement of the lemma does not hold. Let w' be the latest Write succeeding w, and x be the farthest node from the root such that $w[x] \Longrightarrow w'[x] \Longrightarrow r[x]$. If x is the parent of the leaf representing k, we get a contradiction to $\pi(r[x]) = w[x]$. In the remaining case, assume without loss of generality that w writes 0 in x. If w' writes 1 in x, our choice of w' implies r should read 1 from x and go to a leaf different from the one representing k, a contradiction. If w' writes 0 in x, we have

$w'[x0] \longrightarrow w'[x] \Longrightarrow r[x] \longrightarrow r[x0]$, implying $w[x0] \Longrightarrow w'[x0] \Longrightarrow r[x0]$, which contradicts the choice of x. □

Lemma 6. The variable V is regular.

Proof. Let r be a Read in an execution of V. It is straightforward to verify the first two properties of definition 2 for r. The third property follows from lemma 5. □

Lemma 7. For **read**(V) *executions r_a and r_b such that $r_a \longrightarrow r_b$, reading values k and k' respectively, if $k = k'$ then $\pi(r_a) \preceq \pi(r_b)$.*

Proof. Suppose $k = k'$, but $\pi(r_b) \prec \pi(r_a)$. Denote $\pi(r_a)$ as w_a, and $\pi(r_b)$ as w_b. Then, if the parent of the leaf representing value k is the node x, we have $w_b[x] \Longrightarrow w_a[x] \Longrightarrow r_a[x] \Longrightarrow r_b[x]$, which contradicts lemma 5 (with r_b as r). □

Lemma 8. The variable V is weakly atomic.

Proof. Suppose for **read**(V) executions r and r' such that $r \longrightarrow r'$, $\pi(r) = \pi(r')$. We show that for any r'' such that $r' \longrightarrow r''$, $\pi(r) \preceq \pi(r'')$. Denote $\pi(r)$ as w and $\pi(r'')$ as w''. Let k and k'' be the values written by w and w'', respectively. If $k = k''$, then by lemma 7, $\pi(r) \preceq \pi(r'')$. Assume that k'' is different from k. Suppose $\pi(r'') \prec \pi(r)$.

Let y be the parent of the leaf representing k. Then, from $w'' \longrightarrow w[y] \Longrightarrow r[y] \longrightarrow r'$, we have $w'' \longrightarrow r'$. Now let x be the boolean variable farthest from the root common to the paths from the root to the leaves representing k and k''. Without loss of generality, assume that k is in the left subtree rooted at x and k'' is in the right subtree. Then $w''[x = 1] \Longrightarrow r'[x = 0]$ implies there exists some w''' such that $w''[x] \Longrightarrow w'''[x = 0] \Longrightarrow r'[x]$. This implies $w''[x] \Longrightarrow w'''[x] \Longrightarrow r''[x]$, contradicting lemma 5 (with r'' as r). □

Theorem 2. *Construction 2, with* **write**(V) *and* **read***(V) *, implements an atomic variable.*

Proof. The dependability of **read***(V) follows from the weak atomicity of V, by lemma 8, and (i) from lemma 7, when k_2 is returned, and (ii) from proposition 5 and the fact that if $k_2 \neq k_3$ then $\pi(r_2) \neq \pi(r_3)$, when k_1 is returned. □

3.3 *Alternating Write construction (Vidyasankar 1989)*

With **write**(V) and **read***(V), this is an atomic variable construction from Vidyasankar (1989). The underlying weakly atomic variable construction is brought out to provide more insight into the atomic variable construction. Here **read***(V) is an optimized version.

Here all the variables are 1-writer multireader ones. The multivalued variable V is composed of a boolean atomic variable c and two multivalued regular variables B_0 and B_1, called *buffers*, which are used alternately for writing successive new values. Immediately after B_i is written, c is set to i. (We denote the boolean values as 0 and 1.) Thus c contains the index of the buffer B_i which was written by the most recent Write.

The writer uses a local (not shared) boolean variable cl. It is assumed that the initial value of cl is either 0 or 1, and the first operation execution on V is an initializing Write that does not overlap with any Read.

Construction 3.
Procedure **write**(V) writing value *newval*:

```
  $cl := \neg cl$;
  **write** $newval$ in $B_{cl}$;
  **write** $cl$ in $c$.
```

Function **read**(V):

```
  **read** $k$ from $c$;
  **read** and **return** $val$ from $B_k$.
```

Function **read***(V):

```
  **read** $k_1$ from $c$;
  **read** $val1$ from $B_{k_1}$;
  **read** $k_2$ from $c$;
  **if** $k_2 = k_1$ **then return** $val1$
  **else**
    **read** $val2$ from $B_{k_2}$;
    **read** $k_3$ from $c$;
    **if** $k_3 = k_2$ **then return** $val2$
    **else** (* $k_3 = k_1$ *) **return** $val1$.
```

A direct proof of construction 3, with **write**(V) and **read***(V), is given in Vidyasankar (1989). Here we give a proof using the weak atomicity concept.

We note that in this construction, for a Read r that reads and returns the value from B_i, $\pi(r) = w$, where $\pi(r[B_i]) = w[B_i]$.

Lemma 9. For a Read r in an execution on V, $\pi(r[c]) \preceq \pi(r)$.

Proof. Assume without loss of generality that r reads 0 from c. Denoting $\pi(r[c])$ as w_c, we have $w_c[B_0] \longrightarrow w_c[c] \Longrightarrow r[c] \longrightarrow r[B_0]$, implying $w_c[B_0] \longrightarrow r[B_0]$. From the regularity of B_0, it follows that $w_c \preceq \pi(r)$. □

Lemma 10. The variable V is regular.

Proof. Let r be a Read in an execution on V. The first two properties of definition 2 follow from the regularity of the buffers. For the third property, suppose there is a Write w' such that $\pi(r) \longrightarrow w' \longrightarrow r$. Denoting $\pi(r)$ as w, we have $w[c] \longrightarrow w'[c] \longrightarrow r[c]$, implying $w \prec \pi(r[c])$, that is, $\pi(r) \prec \pi(r[c])$, contradicting lemma 9. The assertion follows. □

Lemma 11. For **read**(V) *executions r and r' such that $r \longrightarrow r'$ reading values k and k', respectively, from c, if $k = k'$ then $\pi(r) \preceq \pi(r')$.*

Proof. Suppose on the contrary that $\pi(r') \prec \pi(r)$. Denote $\pi(r)$ as w and $\pi(r')$ as w'. Assume without loss of generality that $k = k' = 0$, that is, both w and w' write in buffer B_0 and write 0 in c. Since no two consecutive Writes write in the same buffer, there is a Write w'', such that $w' \longrightarrow w'' \longrightarrow w$, writing in B_1 and writing 1 in c. Then we have $w'[c] \longrightarrow w''[c] \longrightarrow w[B_0] \dashrightarrow r[B_0] \longrightarrow r'[c]$, implying that $w' \prec \pi(r'[c])$, that is, $\pi(r') \prec \pi(r'[c])$, contradicting lemma 9 . □

Lemma 12. The variable V is weakly atomic.

Proof. Suppose for **read**(V) executions r and r' such that $r \longrightarrow r'$, $\pi(r) = \pi(r')$. We show that for any r'' such that $r' \longrightarrow r''$, $\pi(r) \preceq \pi(r'')$. Denote $\pi(r)$ as w and $\pi(r'')$ as w''. Let k and k'' be the values that r and r'' read, respectively, from c. If $k = k''$, then, by lemma 11, $\pi(r) \preceq \pi(r'')$. Suppose k and k'' are different. Assume without loss of generality that k is 0 and k'' is 1.

First we claim that $w[c] \Longrightarrow r'[c]$. For, suppose on the contrary that $r'[c] \Longrightarrow w[c]$. Since r' reads 0 from c, w writes 0 in c, and two consecutive Writes write different values in c, it follows that $r'[c] \Longrightarrow w^p[c]$, where w^p is the Write immediately preceding w. Then we have $r[B_0] \longrightarrow r'[c] \Longrightarrow w^p[c] \longrightarrow w[B_0]$, implying $r[B_0] \longrightarrow w[B_0]$. This contradicts the regularity of B_0.

Since $r' \longrightarrow r''$, we have $w[c] \Longrightarrow r'[c] \longrightarrow r''[c]$, that is, $w[c] \Longrightarrow r''[c]$. That is, $w[c = 0] \Longrightarrow r''[c = 1]$, implying that, if w^s is the Write that immediately follows w, $w[c = 0] \longrightarrow w^s[c = 1] \Longrightarrow r''[c = 1]$. That is, $w \prec \pi(r''[c])$. Therefore, if $w'' \prec w$, then $w'' \prec \pi(r''[c])$, contradicting lemma 9.

The assertion follows. □

Theorem 3. *Construction 3, with* **write**(V) *and* **read***(V) , *implements an atomic variable.*

Proof. The dependability of **read***(V) follows from lemmas 11 and 12. □

3.4 *An Optimal 1-Reader atomic variable construction*

Here, the resulting atomic variable construction, consisting of **write**(V) and **read***(V) operations, is a slight variation of that of Burns & Peterson (1988). However, the approach of deriving the atomic variable, by identifying the weakly atomic variable V and using a general method to get **read***(V) operation, is new. This approach facilitates a simple correctness proof.

This is a 1-writer 1-reader atomic variable construction from regular variables. A multivalued regular buffer b and two boolean regular variables RC and WC are used. The operations are given in the following. The variables lr, lw, *savebuff* and *savebuff1* are local variables. All the variables could have any initial values. We assume an initializing Write that precedes all other operation executions on V. Here also, the **read***(V) is an optimized version.

Construction 4.
Procedure **write**(V) writing value *newval*:

```
  **write** *newval* in $b$;
  **read** $lr$ from $RC$;
  **write** $\neg lr$ in $WC$.
```

Function **read**(V):

```
  **read** $lw$ from $WC$;
  **if** $lw = RC$ **then return** *savebuff* value
  **else**
    **write** $lw$ in $RC$;
    **read** *savebuff* from $b$;
    **return** *savebuff* value.
```

Function **read***(V):

```
  **read** $lw$ from $WC$;
  **if** $lw = RC$ **then return** *savebuff* value
  **else**
    **write** $lw$ in $RC$;
    **read** *savebuff* from $b$;
    **read** $lw$ from $WC$;
    **if** $lw = RC$ **then return** *savebuff* value
    **else**
      **write** $lw$ in $RC$;
      **read** *savebuff1* from $b$;
      **read** $lw$ from $WC$;
      **if** $lw = RC$ **then** *savebuff* := *savebuff1*;
      **return** *savebuff* value.
```

Lemma 13. The variable V is regular.

Proof. Consider a Read r in an execution of V. It may return a value either from buffer b, or from *savebuff* without reading b. Let r^o be the most recent Read equal to or preceding r, which reads from b. Then $\pi(r)$ is $\pi(r^o)$ which equals $\pi(r^o[b])$, and this is defined by the regularity of b. If $r \longrightarrow \pi(r)$, then $r^o \longrightarrow \pi(r^o)$, and hence $r^o[b] \longrightarrow \pi(r^o[b])$, a contradiction to the regularity of b. Therefore property (ii) of definition 2 is satisfied for r. For property (iii), suppose there is a Write w' such that $\pi(r) \longrightarrow w' \longrightarrow r$. Now $\pi(r^o) \longrightarrow w' \longrightarrow r^o$ would imply $\pi(r^o[b]) \longrightarrow w'[b] \longrightarrow r^o[b]$, contradicting the regularity of b. Thus property (iii) is satisfied for r^o. Suppose $r \neq r^o$. Then, from the above discussion, we have $w'[b] \not\longrightarrow r^o[b]$. That is, $r^o[b] \dashrightarrow w'[b]$. Then $r^o[RC] \longrightarrow r^o[b] \dashrightarrow w'[b] \longrightarrow w'[RC] \longrightarrow w'[WC] \longrightarrow r[WC]$ implies r will find $WC \neq RC$. (Note that by our choice of r^o, no Read in between r^o and r writes RC since it does not read b.) Hence r will read b, contradicting the assumption that it returns *savebuff* value without reading b. □

Lemma 14. Consider three Reads r, r' and r'' such that $r \longrightarrow r' \longrightarrow r''$. If r'' reads from b, then $\pi(r) \preceq \pi(r'')$.

Proof. Suppose on the contrary that $\pi(r'') \prec \pi(r)$. Denote $\pi(r)$ as w and $\pi(r'')$ as w''. Then $w'' \longrightarrow w$. We claim that there cannot be any other Write in between w and w''. For, if there is one, say w''', then $w'' \longrightarrow w''' \longrightarrow w \dashrightarrow r \longrightarrow r''$ implies $w'' \longrightarrow w''' \longrightarrow r''$; then the assumption that $\pi(r'')$ is w'' contradicts the regularity of V. Also, again from the regularity of r'', $w \not\longrightarrow r''$, that is, $r'' \dashrightarrow w$. In fact, we have $r''[b] \dashrightarrow w[b]$, since r'' reads b. From $w'' \longrightarrow w \dashrightarrow r \longrightarrow r'$, we have $w'' \longrightarrow r'$. Therefore, $w''[b] \longrightarrow w''[WC] \longrightarrow r'[WC] \longrightarrow r''[WC] \longrightarrow r''[b] \dashrightarrow w[b] \longrightarrow w[WC]$. That is, $w''[WC] \longrightarrow r'[WC] \longrightarrow r''[WC] \longrightarrow w[WC]$. Thus both r' and r'' read the same value of WC. Since r' would have made sure that $RC = WC$, r'' would certainly find that $RC = WC$, and hence would not read the buffer b, contrary to the assumption. □

Lemma 15. The variable V is weakly atomic.

Proof. Consider three Reads r, r' and r'' such that $r \longrightarrow r' \longrightarrow r''$ and $\pi(r) = \pi(r')$. Let r^o be a Read such that (i) it is the most recent Read equal to or preceding r'', (ii) it succeeds r' and (iii) it reads from b. From lemma 14, $\pi(r) \preceq \pi(r^o)$, and of course $\pi(r^o) = \pi(r'')$. If no such r^o exists, then r'' returns the same *savebuff* value as was returned by r'. That is, $\pi(r) = \pi(r'')$. □

Theorem 4. *Construction 4, with* **write**(V) *and* **read***(V), *implements an atomic variable.*

Proof. The assertion follows from lemma 15 and the observation that **read***(V) specification is according to the general procedure, described in § 2, with some simplification. Nevertheless, we give a detailed proof showing that new-old inversion does not occur with **read***(V) executions.

Consider a **read***(V) execution R. It starts with the first **read**(V) execution, say r. If it returns *savebuff* value (line 2 of the procedure), which is the value returned by the predecessor **read***(V) execution, then clearly there is no new-old inversion. Suppose r is performed to completion and the next **read**(V) execution, say r', is started. If *savebuff* value is returned, we have $\pi(r) = \pi(r')$ and the weak atomicity (lemma 15) justifies the return of this value. In the remaining case, r' is performed to completion, saving the value read from b in *savebuff1*, and a third **read**(V) execution, say r'', is started. If r'' were to return *savebuff1* value, then again due to weak atomicity, R can return *savebuff1* value. If r'' were to read from b, then R returning the value read by r is justified by lemmas 14 and 15. (Hence r'' need not read b at all, since that value is not going to be used in any way.) □

The optimality of the atomic variable construction, with respect to the shared space requirement, has been shown in Burns & Peterson (1988).

4. Discussion

The weak atomicity concept provides a general intermediate step in the construction of atomic variables from regular ones. It facilitates new constructions, and simplifies the correctness proofs.

Lamport (1986) has shown that there is no construction of atomic variable from regular ones in which only the writer writes. Since we are able to construct atomic variables from weakly atomic ones without the necessity of the readers writing, it follows that there is no construction of weakly atomic variable from regular ones in which only the writer writes.

Sibsankar Haldar's comments on the earlier versions of this paper and the reports of two anonymous referees helped to improve the presentation considerably.

This research is partially supported by the Natural Sciences and Engineering Research Council of Canada Individual Operating Grant A-3182. Part of this manuscript was prepared when the author was visiting Utrecht University, The Netherlands, with support from the Netherlands Organization for Scientific Research (NWO) under contract NF 62-376 (NFI project ALADDIN: *Algorithmic Aspects of Parallel and Distributed Systems*).

References

Awerbuch B, Kirousis L M, Kranakis E, Vitányi P M B 1988 A proof technique for register atomicity. In *Proc. 8th Conf. on Foundations of Software Technology & Theoretical Computer Science*, Pune, LNCS (Berlin: Springer Verlag) 338: 286–303

Burns J E, Peterson G L 1988 Sharp bounds for concurrent reading while writing. Tech Rep GIT-ICS-87/31, Georgia Institute of Technology, Atlanta, Georgia, June (Revised)

Chaudhuri S, Welch J L 1990 Bounds on the costs of register implementations. In *4th International Workshop on Distributed Algorithms*, Italy

Kirousis L M, Kranakis E 1989 A brief survey of concurrent readers and writers. Centre for Mathematics and Computer Science, Amsterdam. *CWI Quarterly* 2: 307–330

Lamport L 1986 On interprocess communication, Part I: Basic formalism, Part II: Algorithms. *Distrib. Comput.* 1: 77–101

Vidyasankar K 1989 An elegant 1-writer multireader multivalued atomic register. *Inf. Process. Lett.* 30: 221–223

Vidyasankar K 1991 A very simple construction of 1-writer multireader multivalued atomic variable. *Inf. Process. Lett.* 37: 323–326