Simple Multireader Registers using Time-Stamp Schemes
(Extended Abstract)

Amos Israeli

Computer Science Department, Technion, Haifa, Israel

Ming Li

Aiken Computation Lab, Harvard, Cambridge, Mass.

Paul M.B. Vitanyi

Centrum voor Wiskunde en Informatica and Universiteit van Amsterdam, Amsterdam

ABSTRACT

We use the theory of time-stamp schemes to implement an atomic 1-writer n-reader variable (register) from n^2 atomic 1-writer 1-reader variables, using bounded time-stamps. The number of time-stamps needed is $(2n+2)^2$, so this scheme uses $O(n^2 \log n)$ control bits altogether. The construction is simple, transparant and optimal in worst-case number of control bits per subvariable. No other previous method did better than $\Omega(n)$ control bits per atomic subregister in the worst case, so in that measure we present a multiplicative $O(n/\log n)$ improvement. A similar scheme is given that uses only 2-bit variables written by readers, and $2n$-bit variables written by the writer. This uses altogether $O(n^2)$ control bits altogether. This scheme is optimal in worst-case overal number of control bits. Apart from being optimal in several ways, our constructions add an intuitive dimension which lacks in previous algorithms for this problem. We want to stress that it is really much simpler than previous such algorithms.

1980 Mathematics Subject Classification: 68C05, 68C25, 68A05, 68B20.
CR Categories: B.3.2, B.4.3, D.4.1., D.4.4.
Keywords and Phrases: Shared variable (register), concurrent reading and writing, atomicity, multiwriter variable.
Note: This paper is submitted for publication elsewhere.

The work of thee second author was supported in part by the National Science Foundation under Grants DCR-8301766, DCR-8606366, and Office of Naval Research Grant N00014-85-K-0445.

1. Introduction

A shared variable (or register) is *atomic*, if each read and write of it actually happens, or can be thought to happen, in an indivisible instant of time, irrespective of its actual duration. The time-instant in which the action is seemingly executed must be in between the beginning and the end of the actual duration. Usually, atomicity of operation executions is simply assumed or enforced by synchronization primitives like semaphores. However, active serialization of asynchronous concurrent actions always implies waiting by one action for another. In contrast, our aim is to realise the maximal amount of parallellism in concurrent actions by avoiding waiting altogether in our algorithms. In such a setting, serializability is *not* actively enforced, rather it is the result of a pre-established harmony in the way the execution of the algorithm by the various processors interact.

Communication plays a vital role in any type of concurrent computation. The means of communication have been the subject of growing interest. The fundamental work of [La] has started a new way of looking at processor communication. In particular, it was shown that atomic single reader single writer registers can be constructed from lower level existing hardware rather then just assuminng its existance. Naturally these ideas have aroused interest in construction of constructing multi-user atomic registers. Any of the references, say [La] or [VA], describes the problem area in some detail. Here a simple example in the area has to suffice. (This is a simpler problem than the one we attack below.) A flip-flop is a Boolean variable that can be read (tested) by one processor and written (set and reset) by one other processor. Suppose, we are given atomic flip-flops as building blocks, and are asked to implement an atomic variable with range 1 to n, that can be written by one processor and read by another one. Of course, $\log n$ flip-flops suffice to hold such a value. However, the two processors are asynchronous. Suppose the writer gets stuck after it has set half the bits of the new value. If the reader executes a read after this, it obtains a value that consists of half the new value and half the old one. Obviously, this violates atomicity. This is only one of the many problems that can occur in such a construction, see e.g. [Pe], [La] and [KKV].

Related work in the area of concurrent reading and writing is [Pe], [La], [B], [VA], [SAG], [Ly], [KKV], [PB], [BP], [AKKV], [IL], [NW]. It is extremely hard to understand the proposed algorithms, or the correctness proofs. In fact, with exception of [Pe], [La], the 2-writer Bloom algorithm [B] and the unbounded tag Vitanyi-Awerbuch algorithm [VA], the algorithms seem to defy comprehension by everybody but the designers, let alone the proofs of correctness. Here we present a simple algorithm, and a comprehensible proof of correctness. The algorithm is considerably more simple than any other algorithm for the same purpose [SAG], [KKV], [NW], [BP], [IL]. It is optimal in both time (number of accesses of subvariables) and space (number of control bits). Looking at the worst-case number of control bits per variable, our algorithm achieves $O(\log n)$, while all other algorithms known achieve only $\Omega(n)$, so we present a multiplicative $O(n/\log n)$ improvement.

Conventions The architecture is the one used in the basic unbounded tag algorithm in [VA], and the algorithms we propose below essentially imitate the unbounded tag algorithm used in [VA]. The system comprises processors $0,1,..,n$. Processor 0 is the *writer*, and processors $1,..,n$ are *readers*. Each processor i owns $n+1$ 1-reader 1-writer shared atomic variables, named $R_{i,0} \cdots R_{i,n}$. Only processor i can write these variables, one at a time. Additionally, processor i can read all variables $R_{0,i},...,R_{n,i}$, one at a time. The reads and writes of the subvariables $R_{i,j}$ are assumed to be atomic. We implement an atomic compound variable that can be read by processors $1,..,n$, and can be written only by processor 0, under the following conditions. The readers and writers of the shared variable are assumed to be totally asynchronous. Each read and write must consist of an a priori bounded number of elementary actions, i.e., the program that the reader or writer executes is loop-free (e.g., no busy waiting). The main results of this paper:

Theorem. *An atomic 1-writer n-reader compound variable with range* $\{1,..,N\}$ *can be constructed:*

(i) *using* n^2 *1-reader 1-writer variables, using* $O(\log n)$ *control bits per subregister (total* $O(n^2 \log n)$*), and about* $3n$ *accesses of variables per operation execution; or*

(ii) *using* n^2 *1-reader 1-writer variables, using* $2n$ *control bits in each one of n subregisters, and 2 control bits in each one of* n^2 *subregisters (total* $O(n^2)$*), and about* $3n$ *accesses of variables per operation execution.*

(iii) *Both in algorithm (i) and (ii), actually only n subregisters need to contain two copies of the stored value each, to a total of* $2n \log N$ *value bits.*

(For conciseness we abbreviate "i-writer j-reader" by "(i,j)" in the sequel.) The fundamental problem in implementing multi-user atomic register using atomic $(1,1)$-registers is to keep the temporal precedence relations between members of the set of the actions of the implemented register. The task of keeping this order is complicated because some of these actions might be overlapping. The work of [VA] presents a simple and elegant (n,n) atomic register using time-stamps. Time-stamps are very nice because they naturally induce a total order on the set actions. There is only one draw-back of time-stamps: they are unbounded. The reason why the constructions using bounded tags are so hard to understand and to verify is that they need to use many interacting ad hoc methods to keep track of the precedence relation. Thus, the intuition of the time-stamp method is gone. We would like to closely follow the algorithm in [VA] and cycle through the time-stamps, re-using old ones. To do so, we first look at *time-stamp* schemes [IL].

In [IL] a new construction for multi-writer multi-reader atomic register is presented. This protocol uses a novel paradigm called *Bounded Time-Stamps*. This simple idea enables a set of processors working concurrently to keep track of the temporal ordering of their actions. In the present work we exploit the bounded Time-Stamp paradigm to present a very simple and yet efficient protocol for a single writer multi-reader atomic register. Our new protocol restores the crisp intuition of the time-stamp method.

2. The Unbounded Protocol

We start by presenting a simple $(1,n)$-protocol. This protocol is actually the protocol of [VA], reduced to one writer. The label size of this protocol is unbounded and depends on the number of register operations performed. This is not satisfactory, at least from a theoretical point of view. The architecture is the matrix architecture described in **conventions.** Each pair i,j of processors communicate through the pair $R_{i,j}$, $R_{j,i}$ of atomic $(1,1)$-registers. The writer keeps a local variable called the time-stamp counter which is initialized to 0.

The writer's algorithm:

1. Increase your time-stamp counter by 1 to get the new time-stamp.
2. Append the new time-stamp to the value to be written and write the combination to all readers.

A reader's algorithm:

1. Read values from all readers;
2. Read the value from the writer;

3. Choose value with maximal time-stamp.

4. Write the chosen value, with its time-stamp, to all readers.

Some may believe the correctness of this protocol on sight. For the wary investigator: it also follows from the proofs in [VA], [AKKV] or [Ly]. Close observation will also reveal the following fact:

Fact 1: Among the time-stamps scanned by a reader, the writers time-stamp is always either the most recent or one smaller then the most recent.

Thus it is enough for a reader to check whether the time-stamp of any other reader is larger then the writer's time-stamp. If this is the case the value of that reader is the most recent. In the other case the value of the writer is most recent. Our construction keeps the structure of this protocol. The time-stamp mechanism is replaced by a Bounded Time-stamp Scheme. Fact 1 is used to reduce the complexity of the time-stamp scheme to a minimum.

3. Time-stamp Schemes

Time-Stamp schemes stems from the observation that the time-stamp method is actually a representation of precedence relation in a graph form. The nodes of the graph are the time-stamps and the edges are directed in a way that a node corresponding to a later time-stamp *dominates* all nodes corresponding to earlier time-stamps. The simple idea of bounded time-stamps is to replace the infinite graph by a finite graph while preserving the conceptual frame-work of the protocol. The induced relation is now much more complicated but a correct protocol will make sure that the data gathered by any participant of the protocol will always be inerpretable in only one consistent way.

In order to solve a certain problem we reduce it to a pebble game on graphs. the players of each such game are pebblers (which could be of many kinds) and an adversary. Each pebbler has some pebbles on the graph and a pebble moving protocol. This protocol consists of several operations. The game proceeds by the adversary directing the moves of the player; but these moves must follow the protocol, i.e., be *legal*. The adversary has only control over timing and speed of the submoves. The adversary wins if some pebbler cannot execute its next move legally. A *time-stamp scheme* is a pair (G,P) where G is a graph and P is a protocol that the adversary cannot win. The exact rules of the game depend on the nature of the problem we try to solve. After defining the particular game we have to prove that:

1. The reduction is correct (i.e., a correct implementation of the game solves the problem).

2. That the protocol presented for the game is correct.

4. The Pebble Game

In this section we define the pebble game for the single writer multiple reader atomic register and prove the correctness of the reduction. The rules of the game should reflect the fact that the corresponding problem is implementing a $(1,n)$ register using $(1,1)$ registers. Consider a directed graph G where $a \leftarrow b$ means that b *dominates* a. G has no cycles of length 1 or 2 (not both $a \leftarrow b$ and $b \leftarrow a$). The *pebble game* on the graph is played by 1 writer (black pebble) and n readers (white pebbles) and by the adversary.

Imagine the game as being played on a physical board. Pebblers correspond to asynchronous serial processors, and as such have the following properties: All actions of players are completely asynchronous and can overlap each other in arbitrary ways (the global time model is used). Actions by the same player are totally ordered, and do not overlap. The particular problem that we solve (i.e., using (1,1)-registers) is reflected by the following properties of observation of pebbles and moves of pebbles. A player can determine the instant position of another player's pebble only when he looks for this specific pebble. He can not look for two pebbles at the same time. While pebbles are moved from one node to another, each player looking for a particular pebble may *see* it at the originating node or at the target node, but not both.

The rules of the game are the following: A move of the writer consists of selecting a node that *dominates* its old node, and is not pebbled by a reader or dominated by a node pebbled by a reader (in case the reader has selected its destination, both the source node and the target node count as being pebbled). After a target node has been selected, the writer moves its pebble to the new node. A move of the reader consists in selecting either a node where it sees the writer's pebble, or a node with a reader's pebble that dominates the observed writer's node. (I.e., in the latter case, another reader has already seen the result of the next move of the writer and has irrevocably decided to move, moves, or has moved, its pebble to the writer's target node.) After having selected its target node, the reader moves its pebble to the selected node. Both the readers and the writer follow a given protocol. The protocol allows each player to attach messages for each other player to its own pebble. (Such a message will announce a proposed target node.) It can only attach one such message to each addressee at a time. No player can read a message intended for another player. We associate with each node a unique identifier called a *time-stamp*. The pebble game is a *legal* pebble game if the protocol for the readers and the writer uses a strategy that ensures the writer can always move. The players play the pebble game by reading and writing to a shared register, and in this way observe the other's pebbles, and move their own pebble.

Lemma 1. (atomicity) *A shared register such that the readers and the writer play a legal pebble game is atomic.*

Proof. To aid intuition, first assume that G is unbounded. Let the semantics be that there is a unique *primary* value in the register, which at any time corresponds with the last node actually pebbled by the writer.

Map each reader's move to the time instant it last saw the pebble it finally selects to move to. Map each move by the writer to just before *, where * is a time instant which is the earliest among the following time instants. One time instant is the end of the duration of the writer's move, and each other time instant consists of the last time instant that a reader, that selects the writer's target node to move to, actually saw the writer's pebble at the target node.

Firstly, this mapping maps each move to an instant inside its duration, (and can be made one-one by infinitely small shifts of the images). Secondly, the image of the writer's move to a particular node precedes the images of all readers' moves to that node. Thirdly, no writer's move, say m, to another node is mapped in between a writer's move, say m' ($m' \neq m$) to a node and a reader's move to the same node. The first two statements are obvious, and the third one follows since otherwise the reader would have seen the writer's pebble on the target node of move m at a time it had already arrived at the target node of move m', which is impossible.

Suppose G is bounded and the pebble game is legal. Then the mapping of the moves gives the same order as it would give for the play on the unbounded graph G' defined as follows. G' has infinitely many versions (i,j) $(j=1,2,...)$ of each node i in G. The topology of G' can be easily choosen such that, whenever the writer moves in G for the jth time to a node i, he moves in G' to the j-th copy (i,j) of that node. ●

5. Protocol 1. (Minimum Control Bits per Subvariable)

The architecture is the matrix of [VA], as described in **conventions** above. The Read and Write actions consist of first reading the associated 'column' of subregisters, and then writing the associated 'row' of subregisters. An important feature is that the subregister in the writer's row $(R_{0,1},..,R_{0,n})$ is read last in the read phase of a Read. For easy mnemonics in the protocol below,

- *Read (Rj,rrj)* denotes ''reader i reads contents rrj from the subregister $R_{j,i}$ it shares with reader j'' $(1 \le i, j \le n)$;
- *Read (W,wk)* denotes ''reader j reads contents wk from the subregister $R_{0,j}$ it shares with the writer 0, that was written in write wk'' $(1 \le j \le n$ and $1 \le k \le 2)$;
- *Read (Rj,rwj)* denotes ''writer 0 reads contents rwj from the subregister $R_{j,0}$ it shares with reader j'' $(1 \le j \le n)$;
- *Write (Rj,x)* denotes ''processor i writes contents x to the subregister $R_{i,j}$ it shares with reader j'' $(0 \le i \le n,\ 1 \le j \le n)$;
- *Write (Wk,x)* denotes ''reader i writes contents x to field k of the subregister $R_{i,0}$ it shares with writer 0'' $(1 \le i \le n,\ 0 \le k \le 1)$. (So the subregisters in the first column of the matrix, that are read by the writer, have double the contents of the other subregisters.)

The algorithm uses 'tickets' $1,..,2n+2$. Each subregister will store one or two tags consisting of a ticket pair (old,new). This pair corresponds to the last two tickets selected by the writer, and is written by the writer when it selects the "new" ticket. The registers in the first (writer) column store two tags, and those in the other (readers) colums one tag. This scheme requires $2n+2$ tickets. Each register needs to store one ticket pair, except those in the first (writer) column, which need to store two ticket pairs.

Define a graph to play a pebble game as follows. Let $V = A \times A$ and A a set, $|A| = 2n+2$. Let $G = (V, E)$ be a directed graph with $a \leftarrow b$ if $a = (A, B)$ and $b = (B, D)$, $D \ne A$. The pebble game is played by 1 writer and n readers, using the matrix construction. The local variables old,new,ticket are elements of A, $rr1,...,rrn,w,w1,w2,w3,m$ are elements of $A \times A$, and $rw1,...,rwn$ are elements of $A \times A \times A \times A$. The value $val@i, val, val.new$ belong to the value domain $V = \{NIL, 1,...,N\}$, and $M \in A \times A \times V$. Initialize everything so that the protocol makes sense.

Writer algorithm: /* w = ((old,new), val), writer writes val.new */

1) Read $(R1, rw1),...,(Rn, rwn)$;
   old := new; new := ticket not in 1st coordinates tags in $rw1,...,rwn,w$;
      /* I.e., $2n+2$ tickets are sufficient */
   w := ((old,new), val.new); /* select node */
2) Write $(R1, w),...,(Rn, w)$

Reader algorithm for j:

1) Read $(W, w1)$;
2) Write $(W0, w1)$;
3) Read $(R1, (rr1, val@1)),...,(Rn, (rrn, val@n)), (W, (w2, val))$; /*Writer last*/

   Case 1a: if $w1 \ne w2$ and $first$=**true** then ($w1 := w2$; first:= bold false ; goto 2);
   Case 1b: if $w1 \ne w2$ and $first$=**false** then $M := (NIL, val@w1)$;

/* NIL dominates no node */

Case 2: if $w1 = w2$ then

if $w2 \leftarrow rri$ for some $i \neq j$ then $M := (rri, val@i)$ else $M := (w2, val@w2)$;

4) Write $(R1, M), \ldots, (Rn, M), (W1, M)$; *first*:=**true**; and output *val*of M.

Remark. The reader executes the "goto 2" in Case 1.a at most once.

Lemma 2 (legality). *The readers and the writer play a legal pebble game on this register.*

Corollary. Protocol 1 implements an atomic multreader register.

Proof. Since the number of tickets is $2n+2$, while there are at most $2n+1$ tickets forbidden, the writer always selects a new node in step 1. It is left to prove that such a new node is not dominated or occupied by readers' pebbles. We do induction on the moves by the writer.

Base. The writer and readers start from the same start node. In the initial move preceding all other moves, the writer goes by definition of G to a node which is undominated by readers.

Induction. By contradiction, let there be a first move $k > 1$ such that the writer goes to a node dominated or occupied by a reader. The writer avoids nodes dominated or occupied by readers pebbles it has observed. Hence he moves to a target node which is also the target node, or is dominated by a target node, of a reader's move which was in progress when the writer observed this reader's pebble. (To observe another's pebble means scanning a joint register.)

Roughly, the burden of avoiding a move to a node dominating a writer pebble, or occupied by a writer pebble, falls mainly on the readers. The readers anounce a sequence of destinations, alternated with pebble observations, before selecting a target. The writer avoids moving to an anounced reader destination.

The reader can only move its pebble to dominating nodes in G in Case 2 of Step 3 in its protocol. Let m be the tag selected in step 3 of the reader protocol, i.e., $M = (m, .)$ at the beginning of step 4.

Subcase a. $w1 = w2$ and $m := w2$. We are only concerned with the reader and writer's actions to joint variables. The reader's target node is $w2$, and the writer's target node is w, $w \leftarrow w2$ or $w = w2$. W.l.o.g., let the reader execute step i at time t_i, and the writer execute step i at time T_i. Now $T_1 < t_2$ since the writer has not seen the reader's anounced target node $w1$. On the other hand, $T_2 > t_3$ since the reader has not seen the writer's target node w. Hence, since the reader sees the writer's pebble on node $w2$ at time t_3, $w2 \leftarrow w$ (or $w2 = w$). However, since $w1 = w2$, this implies a cycle of length 1 or 2 in G, which is a contradiction.

Subcase b. $w1 = w2$ and $m = rrj$ with $w2 \leftarrow rrj$. I.e., the reader's target node $m = rrj$, and the writer's target node is $w2$, $w2 \leftarrow m$. Now reader j has put its pebble on m because it selected that target node because it was pebbled by the writer or by another reader. If by another reader then this reader put its pebble on the node because he saw the writer's pebble or the pebble of another reader, and so on. Because of the base case, this chain of readers must end with a first reader which selected node m by Subcase a. According to the analysis of subcase a, the writer does not move to target $w2$ while the first reader moves to m. Now the second reader in the chain moves to m by Subcase b, i.e., to the white pebble of reader one. Let the times the steps are executed be denoted by t_i, t_i' and T_i for the first reader, second reader and writer, respectively. Let furthermore $t+_i$ denote the steps of the next move by the first reader (to a node other than m). In multiple steps we indicate the substeps by superscripts, e.g., in Step 1 of the writer we indicate the substeps by $T_1^{R1}, T_1^{R2}, \ldots, T_1^{Rn}$.

Now $t'_2 < t'^{R1}_3 < t'^{W}_3$, and $t'^{W}_3 < t+^{R2}_4 < t+^{W}_4$. Up to $t+^{W}_4$, the writer will avoid moving to $w2$, because it observes 1's pebble on the dominating node m. After time $t+^{W}_4$, the writer will observe that 1's pebble has moved from m. But since t'_2, the writer was notified that 2 considers moving to $w2$. So in between $t'_2 < T_1^{R2} < t'^W_4$, the writer will avoid moving to $w2$ as well. Iterating the argument for the successive pairs of reader's moves in the chain shows that the writer avoids moving to $w2$ while reader Rj is moving to m, which is the required contradiction. ●

This method uses $2n$ accesses of subregisters for a write, and at most $3n+3$ accesses of subregisters for a read. Each subregister must hold an integer in between 1 and $2n+2$, except the subregisters in the first column, that must hold two such integers. On the other hand, the subregisters on the main diagonal can be deleted. Hence, each subregister must hold $O(\log n)$ control bits, which sums to a total of $O(n^2 \log n)$ control bits. Using a few $O(n)$ control bit subregisters, we can get the total down to $O(n^2)$ control bits.

6. Protocol 2 (Minimum Total Control Bits)

The same algorithm with only $O(n^2)$ control bits overall can be constructed as follows. Each register owned by the writer contains $2n$ control bits, and each register owned by a reader contains only 2 control bits. The control bits are used to determine the domination relation between readers and the writer. The Protocol stays the same, only the decisions in the protocol are made according to different format data. Since the decisions are isomorphic with that of Protocol 1, the correctness of the new Protocol follows by induction on the total atomic order of the operation executions in each run by the correctness of Protocol 1. Details are given in the full paper.

7. Writing n copies of the value

In the algorithm, each subregister ostensibly contains a copy of the value to be written. This sums up to $O(n^2 \log N)$ bits, for the value ranging from 1 to N. With the following scheme only the registers owned by the writer contain the values. Each register owned by the writer can contain two values. The two fields concerned are used alternatingly. The writer starts its tth write with an extra write (step 0) to all registers it owns, writing the new value in field t mod 2. In step 2 it writes to field t mod 2 (It marks this field as the last one written), and finishes by setting $t := (t+1) \mathrm{mod} 2$. The readers, on the other hand, now write no values, only the tags. If the reader chooses chooses the writer, it takes the value from the marked field; if it chooses a reader, it takes the value from the unmarked field. Since no observed reader can be more than one write ahead of the actually observed write, this is correct.

References

[AKK] B. Awerbuch, L.M. Kirousis, E. Kranakis, P.M.Vitányi, ''On Proving Register Atomicity'', CWI Tech Rept, Amsterdam, May 1987.

[Bl] B. Bloom, ''Constructing Two-writer Atomic Registers,'' PODC-87.

[BP] J.E. Burns and G.L. Peterson, ''Constructing Multi-reader Atomic Values From Non- atomic Values'', PODC-87.

[IL] A. Israeli and M. Li, ''Bounded Time-Stamps'', FOCS-87.

[KKV] L.M. Kirousis, E. Kranakis, P.M.B. Vitányi, ''Atomic Multireader Register'', Information and Computation, to appear. (Also, Proc. 2nd International Workshop on Distributed Computing, Amsterdam, July 1987.)

[La] L. Lamport, ''On Interprocess Communication Parts I and II'', Distributed Computing, Vol. 1, 1986, pp. 77-101.

[Ly] N. Lynch, ''MIT 6.852 Class Notes'', Oct. 3, 1986.

[NW] R. Newman-Wolfe, ''A Protocol for Wait-free, Atomic, Multi-Reader Shared Variables'', PODC-

87.

[PB] G.L. Peterson and J.E. Burns, ‘‘Concurrent reading while writing II: the multiwriter case’’, FOCS-87

[Pe] G.L. Peterson, ‘‘Concurrent reading while writing’’, ACM Transactions on Programming Languages and Systems, vol. 5, No.1, 1983, pp. 46-55.

[SAG] A.K. Singh, J.H. Anderson, M.G. Gouda, ‘‘The Elusive Atomic Register Revisited’’, PODC-87.

[VA] P.M.B. Vitányi and B. Awerbuch, ‘‘Atomic Shared Register Access by Asynchronous Hardware’’, FOCS-86. (Errata, FOCS-87)