Simple Atomic Snapshots
A Linear Complexity Solution With Unbounded Time-Stamps*

Lefteris M. Kirousis[†] Paul Spirakis[†‡] Philippas Tsigas[§]

Abstract

Let $X_1, \ldots, X_c$ be variables which together constitute a composite register. These variables are shared by a number of processes which operate in a totally asynchronous and wait-free manner. An operation by a process on the composite register is either a write to one of the variables or a read of the values of *all* variables. All operations are required to be atomic, i.e. an execution of any number of them (including reads) must be linearizable. in a way consistent with the values returned by the reads. In a single reader composite register no two reads can concurrently access the composite register. We give a new protocol implementing a single reader composite register for the case when there is a single writer per variable. Our construction uses time-stamps that may take values as large as the number of operations performed. The advantages of our construction over previous (bounded time-stamps) solutions are: (i) Both the protocol and its formal correctness proof are easy to understand. (ii) The time complexity of an operation of our construction (i.e. the number of its sub-operations) and the number of the subregisters used in our construction are at most *equal* to the number of processes that can concurrently access the composite register.

Keywords: Distributed Computing, Wait-Free Registers, Snapshot Problem.

1 Introduction

Recently, a number of constructions have been proposed for a shared, array-like variable (called **a composite register**) that is comprised of a number of variables $X_1, \ldots, X_c$ (the **components**), so that each X_k, $k = 1, \ldots, c$, can be written to by a set of processes (the **writers** of this component) and the values of *all* components can be read in a single atomic operation by one or more processes (the **readers**). All processes are assumed to operate totally asynchronously, while all operations (i.e. either writes to a component or multi-reads) are required to be executed in a wait-free manner. The building blocks of these constructions are atomic, single-component variables, called **subregisters**. A read or a write operation on

*This research was partially supported by the ESPRIT II Basic Research Actions of the EU under contracts no. 3075 and 7141 (projects ALCOM and ALCOM II). To appear in *Information Processing Letters* (IPL). A preliminary version of this paper was presented at the International Conference on Computing and Information 1991 (ICCI'91), Lecture Notes in Computer Science Vol.497, pp. 582-587, Springer-Verlag, 1991.

[†]Department of Computer Engineering and Informatics, University of Patras, Rio, 265 00 Patras, Greece; Computer Technology Institute, P.O. Box 1122, 261 10 Patras, Greece; email: ⟨lastname⟩@cti.gr

[‡]Courant Institute of Mathematical Sciences, New York University, New York, 10012 NY, USA.

[§]MPI für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany, email: tsigas@mpi-sb.mpg.de. Much of the work described here was completed while he was at the University of Patras and CTI.

the level of the composite register (a high-level operation) may have as sub-operations both reads and writes on the level of subregisters (low-level operations). A construction of a composite register is said to be **wait-free** iff any high-level read or write operation is guaranteed to finish in a finite number of steps, regardless of the speed of the other processes that may access the object concurrently. Informally, a wait-free composite register (**snapshot**) gives the means to processes to take an instantaneous picture of memory components while are have being concurrently updated by other processes. A composite register is one of the most powerful synchronization abstract data types that can be implemented in a wait-free manner using as building blocks only read and write objects.

A composite register is characterized by the number of readers that can concurrently read it, the number of writers that can concurrently write to the same component, the number of components, as well as the number of bits per component. The general case is the n-reader, m-writer per component, c-component, b-bit per component composite register. The problem which we study in this paper is the wait-free implementation of a single-reader, c-component, 1-writer per component, b-bit per component composite register using one component registers. It must be pointed out that the case of multi-reader is rather different in nature than the single-reader case, in the sense that it has an increased difficulty in achieving sequential consistency among read "results" by different processes.

Multi-reader composite register implementations have been proposed: i) by Afek et al. in [1] (bounded memory and quadratic number of sub-operations per operation), Anderson in [2] (bounded memory and exponential number of sub-operations per operation) and by Attiya and Rachman in [5] (bounded memory and $O(n \log n)$ sub-operations per operation); these implementations solve the general case of the problem (multi-reader and multi-writer), ii) by Aspnes and Herlihy in [3] (unbounded memory and quadratic number of sub-operations per operation); this is an implementation for the multi-reader, single writer per component composite register and uses unbounded time-stamps. These constructions are all for the multi-reader problem; however, the corresponding time complexities (i.e. number of sub-operations per operation) remain the same even if they are restricted to the case of a single reader and single writer per component composite register.

Because of the significance of the problem much work has been done in the general area of composite register constructions. Kirousis, Spirakis, and Tsigas in [10] proposed an implementation for the single-reader, multi-writer per component composite register (bounded memory, linear time complexity for a read operation and constant for a write operation); in that work, first an implementation using unbounded number of subregisters is proposed and subsequently it is shown that bounded number of subregisters are sufficient. Dwork et al. in [7] give a linear time-lapse snapshot object. Attiya, Herlihy, and Rachman in [4] give an $O(n \log^2 n)$ implementation that uses $Test\&Set$ registers, and an $O(n)$ implementation that uses dynamic $Test\&Set$ registers. Israeli, Shaham, and Shirazi in [9], by modifying and extending the construction presented here, give a general technique to transform any snapshot construction that requires $O(f(n))$ operations per read or write, into a construction that requires $O(f(n))$ operations per read, and only linear number of operations per write; in the same paper they also give a general technique to transform any snapshot construction that requires $O(f(n))$ operations per read or write, into a construction that requires $O(f(n))$ operations per write. Hoepman and Tromp in [8] showed that the complexity of the general composite register implementation reduces to the complexity of the implementation of a composite register whose components are binary variables. Chandy and Lamport in [6] considered a closely related problem in the message–passing model.

In this paper we use time-stamping and we present a solution for the case of single-reader, single-writer per component composite register. Besides its contribution to the design of other wait-free algorithms (cf. [9]), our construction has the nice feature that the number of sub-operations per operation as well as the number of (multi-reader) subregisters used are at most *equal* to the total number of processes that can concurrently access the composite register. We believe that the resulting construction is simple and transparent, while its formal correctness proof is elegant and easy to understand. The time-stamps used may take values that are at most equal to the number of operations in a run, therefore it is enough to assume that the subregisters of our construction can "hold" a number of bits that is linear in the logarithm of the number of operations. Given the word-length that is achieved with today's technology, it seems that "unbounded" time-stamps in this sense is a fair price to pay for the simplicity attained.

2 Model and Definitions

A formalism for the problem of composite registers can be found either in [2] or in [1]. In [11] one can find a formalism for the notion of atomic registers and the global time assumption that we adopt. Our notation is compatible with these formalisms. We assume that each operation Op has a time interval $[s_{Op}, f_{Op}]$ on a linear time axis. Think of s_{Op} and f_{Op} as the starting and finishing times of Op. Moreover, we assume that there is a precedence relation on operations which is a strict partial order (denoted by '$\rightarrow$'). Semantically, $a \rightarrow b$ means that operation a ends before operation b starts. If two operations are incomparable under $\rightarrow$, they are said to **overlap**. If $a \rightarrow b$, then for any sub-operations s and t of a and b, respectively, we have that $s \rightarrow t$.

A construction of a composite register is comprised of: (i) a description of the set of the subregisters and their initial values and (ii) procedures (protocols) that describe a high-level operation in terms of its sub-operations on the subregisters. A construction, apart from the shared variables (i.e. the subregisters), may make use of **local** variables as well. The local variables might be *static* i.e. retain their values between invocations of the corresponding procedures. We adopt the convention to denote shared variables with capital letters and local variables with lower case letters.

A **reading function** π_R for a register R is a function that assigns a write operation w to each read operation r on R, such that the value returned by r is the value written by w. It is assumed that there exists a write operation, which initializes the register R, that precedes all other operations on R. Similarly, a reading function π_k for a component k of the composite register is function that assigns to each high-level read r a high-level write w to X_k so that the value returned by r is written to X_k by w.

A **run** on a register is an execution of an arbitrary number of operations according to its protocol.

Following [11] we distinguish the single-component registers (i.e. the subregisters that are the building blocks of composite register constructions) according to the consistency guaranteed in presence of concurrent reads and writes. For all these cases we assume that for every run on a register R there exists a write operation which precedes all the other operations. This write operation writes the initial value to subregister R.

• A run on a register R is **safe** if every read operation r that is not concurrent with any write operation returns a value which is equal to the value written by a write operation that

directly precedes r (a write operation w directly precedes r if there is no other write operation u such that $w \rightarrow u \rightarrow r$). A read operation that overlaps a write operation returns one of the possible values of the register. The register is called safe if all its runs are safe.

• A run on a register R is **regular** if every read operation r returns a value which is equal to the value written by a write operation that is either concurrent with r or directly precedes r. A register is regular if all its runs are regular.

• A run on a register R is **atomic** if there is a total order $\Rightarrow$ on the set of all the operations of the run such that: (i) The total order $\Rightarrow$ extends the precedence relation $\rightarrow$, (ii) every read operation r returns a value which is equal to the value written by a write operation that directly precedes r in the total ordering $\Rightarrow$. A register is atomic if all its runs are atomic.

Obviously, an atomic register is regular, and a regular one is safe.

A run on a *composite* register is **atomic** if the partial order $\rightarrow$ on its operations can be extended to a strict *total* order $\Rightarrow$ and if for each component k there is a reading function π_k, such that for all high-level reads r: (i) $\pi_k(r) \Rightarrow r$ and (ii) there is no write w to X_k such that $\pi_k(r) \Rightarrow w \Rightarrow r$. A composite register is atomic if all its runs are atomic.

3 The Protocol

We first give an atomicity criterion for single-reader, single-writer per component, c-component, composite register constructions; this theorem will help in giving the intuition of the algorithm with the informal description that follows, and will also be used in order to prove the correctness of our construction.

Theorem 1 *In the case of a single-reader, single-writer per component, c-component, composite register a run is atomic if for each k, where $1 \leq k \leq c$, there exists a reading function π_k, whose domain is the set of high-level reads and whose range is a set of high-level writes to component k, such that the following four conditions hold:*

(F) Future *For any high-level read r and for any component k, it is not the case that: $r \rightarrow \pi_k(r)$.*

(P) Past *For any high-level read r, for any component k, and for any high-level write w to component k it is not the case that: $\pi_k(r) \rightarrow w \rightarrow r$.*

(N-O) New-Old Inversion *For all high-level reads r_1, r_2, and for any component k, it is not the case that: $(r_1 \rightarrow r_2$ and $\pi_k(r_2) \rightarrow \pi_k(r_1))$.*

(In-C) Inconsistency *For any high-level read r, for all components k, l and for any high-level write w to component k, it is not the case that: $\pi_k(r) \rightarrow w \rightarrow \pi_l(r)$.*

This theorem is essentially the restriction to the single-reader, single-writer per component, composite register case of the atomicity criterion for the general case given in [2], so we omit its proof. □

3.1 Informal Description

The first three conditions of Theorem 1 constitute a correctness criterion for atomic (single-component) register constructions. The fourth condition is the essence of the snapshot problem; this condition disallows read operations from obtaining inconsistent views of the values of the components.

```
type Rtype = record val : valtype; tag : integer end;
var  B : array[1..c] of array[1..c] of Rtype;
     /* Shared variables declaration; initially, they hold the same arbitrary value*/

procedure reader /*returns array[1..c] of valtype*/
var  mr : array[1..c] of array[1..c] of Rtype;
     a : array[1..c] of valtype;
     col, row : 1..c;
begin
     for row := 1 to c do read mr[row] := B[row] od;
     for col := 1 to c do
          select row such that (∀l : 1 ≤ l ≤ c)(mr[row][col].tag ≥ mr[l][col].tag);
          a[col] := mr[row][col].val;
     od;
     return (a[1], ..., a[c]);
end

procedure writer /* writes u : valtype on component i*/
var  mw : array[1..c] of array[1..c] of Rtype;
     b : array[1..c] of Rtype;
     col, row : 1..c;
begin
     for row := 1 to c do read mw[row] := B[row] od;
     for col := 1 to c do
          select row such that (∀l : 1 ≤ l ≤ c)(mw[row][col].tag ≥ mw[l][col].tag);
          b[col].val := mw[row][col].val;
          b[col].tag := mw[row][col].tag;
     od;
     b[i].tag := b[i].tag + 1;
     b[i].val := u;
     write B[i] := [b[1], ..., b[c]];
end
```

Figure 1: The protocol.

Our construction uses an idea which has also been used in the Vitányi-Awerbuch matrix (see [12]). However, instead of having the classical Vitányi-Awerbuch matrix of subregisters, we keep an one-dimensional array of subregisters, each entry of which carries the information included in a "snapshot" of the multiple components. Besides the "snapshot", each entry of the array carries a sequence of time-stamps, each one corresponding to a different component. So, the basic idea of our construction is that the writer of each component takes a view of the values of the other components and propagates it to the reader by writing these values together with the value it has to write to its component. The reader, during each read r, collects the views propagated by the writers and combines them so that for all components k, l, either $\pi_k(r)$ and $\pi_l(r)$ directly precede one the other or $\pi_k(r)$ and $\pi_l(r)$ overlap. During

each write operation to the ith component, the respective writer appends an increasing time-stamp value to the value it writes. The time-stamp values are propagated together with the values of the respective components. This is done so that the reader is able to know the precedence relation between write operations to the same component.

The architecture of the construction is the following: for each component $k = 1, \ldots, c$ we introduce an atomic subregister $B[k]$ which can be written by the writer of the corresponding component and can be read by all processes. We call these subregisters **B**uffers. A buffer holds an **array**$[1..c]$ of records. The ith entry of this array holds a value corresponding to the ith component of the composite register and a time-stamp (tag) appended to this value.

The reader and writer procedures are shown in Figure 1. The protocol's behaviour is:

• The writer of the ith component ($i = 1, \ldots, c$) reads sequentially $B[1], \ldots, B[c]$. Thus, since each $B[i]$ is an array$[1..c]$, the writer obtains a matrix mw (**w**riter's **m**atrix) of dimension (c, c). Each column of this matrix corresponds to a component. Moreover, the entries of the column are values (with an attached tag) of high-level writes to the corresponding component. The writer now selects an entry with a maximum tag for each column $col = 1, \ldots, i-1, i+1, \ldots, c$ and stores the value and tag of this entry in the local variable $b[col]$. On the other hand, for $col = i$ (i.e. the value of col corresponding to the component where this writer writes) the writer selects again from column i an entry with maximum tag, increments this tag by one and stores into $b[i]$ the value that it must write to component i (say, this value is u) together with the incremented tag. Finally, the writer writes the array b to $B[i]$.

• The reader acts similarly to the writer. A read operation first reads sequentially $B[1], \ldots, B[c]$. Thus, it obtains a matrix mr (**r**eader's **m**atrix) of dimension (c, c); next, for each column, col ($col = 1, \ldots, c$), of the matrix mr it selects an entry with maximum tag and stores the value of this entry in the local variable $a[col]$; then it returns the values currently stored in a.

Initially, the subregisters $B[1], \ldots, B[c]$ hold the same arbitrary value. The local variables are arbitrarily initialized.

It is easy to check from the similarity of the reader's and writer's procedures that the writer of the ith component takes a "snapshot" of the other $c-1$ components and writes this "snapshot" (tags included) together with the value u and its attached tag to $B[i]$.

The following two observations give intuitively the reason for the correctness of the construction (in particular why the fourth condition of Theorem 1 is satisfied).

(i) Each write operation w to component k propagates (in $B[k]$) for any component l a value written by a write operation that either directly precedes or overlaps w.

(ii) Each write operation w' to component l, which propagates (in $B[l]$) for a component k the value v written by the write operation w to component k, will propagate for all the other components either the same values as w propagated or values that "overwrote" them (values associated with greater tag).

3.2 Proof of Correctness

The proof is given by a series of lemmas.

Lemma 1 *The protocol satisfies condition (F).*

Proof Because $B[k]$ is atomic for $1 \leq k \leq c$, it follows from the protocol that for all high-level reads r and for all $k = 1, \ldots, c$ the final step of $\pi_k(r)$ occurs before the final step of r. □

Lemma 2 *The protocol satisfies condition (P).*

Proof Assume, towards a contradiction, that there is a high-level read r, a component k of the composite register and a high-level write w to this component such that $\pi_k(r) \rightarrow w \rightarrow r$. Assume that the tag that $\pi_k(r)$ attaches to its value is α. Since $\pi_k(r) \rightarrow w$, w attaches to its value a tag β such that $\beta > \alpha$. Since $w \rightarrow r$, the reader will read a value from $B[k]$ written by a high-level write w_1 to component k such that either $w \rightarrow w_1$ or $w_1 = w$. Thus, the reader will store in column k and row k of its matrix mr a value written by w_1 with tag greater than or equal to β. Therefore, it will not return the value written by $\pi_k(r)$, since this value has smaller tag than the tag of the value written by w_1, a contradiction. □

Lemma 3 *The protocol satisfies condition (N-O).*

Proof Assume, towards a contradiction, that there are two high-level reads r_1 and r_2 and a component k of the composite register such that: $r_1 \rightarrow r_2$ and $\pi_k(r_2) \rightarrow \pi_k(r_1)$. Since $B[k]$ is atomic, the last sub-operation of $\pi_k(r_1)$ (which is its unique sub-write) occurs before the last sub-operation of r_1. Since $r_1 \rightarrow r_2$, it follows that $\pi_k(r_2) \rightarrow \pi_k(r_1) \rightarrow r_2$, which is a contradiction according to the previous lemma. □

Lemma 4 *The protocol satisfies condition (In-C).*

Proof Assume, towards a contradiction, that there are two components k, l of the composite register, a high-level read r, and a high-level write w to component k such that $\pi_k(r) \rightarrow w \rightarrow \pi_l(r)$. By this hypothesis, the reader during r returns as value for component l a value written by the high-level write $\pi_l(r)$. This high-level write, at its last subwrite on $B[l]$, besides writing its "own" value with a tag to $B[l][l]$, it writes a value and a tag on $B[l][k]$ as well. Since $w \rightarrow \pi_l(r)$, the latter tag is greater than or equal to the tag that w attaches to its value to component k and, therefore (since $\pi_k(r) \rightarrow w$), it is strictly greater than the corresponding tag of $\pi_k(r)$. Now, since r reads $\pi_l(r)$, the tag that r uses for choosing a value for component k is greater than or equal to the tag of $B[l][k]$, so r cannot read $\pi_k(r)$, a contradiction. □

From the above results and by Theorem 1 we have that:

Theorem 2 *A single-reader, c-component, single-writer per component atomic composite register can be constructed using c atomic, $(c+1)$-reader, single-writer, subregisters. The number of sub-operations for a read operation is c, while a write operation has $(c+1)$ sub-operations.*

3.3 Using Regular Subregisters

We can easily notice that, provided time-stamps are used, it is enough to assume that $B[k]$ are regular and simulate the atomic subregisters $B[k]$ used in the above construction by using a method given in [11]. In order to adopt this in our protocol we first have to make the reader remember both the values that it last returned and the respective *tag* values. This means that the local array a will have to be extended (its entries will be of *Rtype* instead of *valtype*) and considered static. The reader's protocol must be slightly modified; the assignments to the entries of the extended static array a are conditional, so that its tag fields never decrease. This means that the entry $mr[row][col]$ obtained after the select "statement", together with its tag, replaces the corresponding entry in a ($a[col]$), only if $a[col].tag$ is smaller than the tag of the value selected from mr. No changes have to be done to the writer's protocol. Thus, Theorem 2 can be extended to:

Theorem 3 *A single-reader, c-component, single-writer per component atomic composite register can be constructed using c regular, $(c+1)$-reader, single-writer, subregisters. The number of sub-operations for a read operation is c, while a write operation has $(c+1)$ sub-operations.*

Acknowledgment

We thank Andreas Veneris for many helpful comments on an earlier draft of this paper. We would also like to thank the anonymous referees for their accurate remarks that helped in improving the presentation of the paper.

References

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit Atomic snapshots of shared memory. *In Proc. of the 9th ACM Symp. on Principles of Distributed Computing*, 1990, pp. 1–14.

[2] J.H. Anderson Composite registers. *In Proc. of the 9th ACM Symp. on Principles of Distributed Computing*, 1990, pp. 15–29.

[3] J. Aspnes and M. Herlihy Wait-free data structures in the asynchronous PRAM model. *In Proc. of the 2nd ACM Symp. on Parallel Algorithms and Architectures*, 1990, pp. 340–349.

[4] H. Attiya, M. Herlihy, and O. Rachman Efficient Atomic Snapshots Using Lattice Agreement. In *Proc. of the 6th International Workshop on Distributed Algorithms*, 1992, volume 647 of *Lecture Notes In Computer Science*, pp. 35–53. Springer-Verlag.

[5] H. Attiya and O. Rachman Atomic Snapshots in $O(n \log n)$ Operations. *In Proc. of the 12th ACM Symp. on Principles of Distributed Computing*, 1992, pp. 29–40.

[6] K.M. Chandy and L. Lamport Distributed Snapshots, Determining Global States of Distributed Systems. *ACM Trans. on Computer Systems 3:1, 195*, 63–75.

[7] C. Dwork, M.P. Herlihy, S.A. Plotkin, and O. Waarts Time-Lapse Snapshots, *Proc. of the Israel Symp. on the Theory of Computing and Systems*, 1992, volume 601, of *Lecture Notes In Computer Science*, pp. 154–170. Springer-Verlag.

[8] J.H. Hoepman and J.Tromp Binary Snapshots. In *Proc. of the 7th International Workshop on Distributed Algorithms*, 1993, volume 725 of *Lecture Notes In Computer Science*, pp. 18–25. Springer-Verlag.

[9] A. Israeli, A. Shaham, and A. Shirazi Linear-Time Snapshot Protocols for Unbalanced Systems. In *Proc. of the 7th International Workshop on Distributed Algorithms*, 1993, volume 725 of *Lecture Notes In Computer Science*, pp. 26–38. Springer-Verlag.

[10] L.M. Kirousis, P. Spirakis, and Ph. Tsigas Reading Many Variables in One Atomic Operation: Solutions With Linear Complexity. *IEEE Transactions on Parallel and Distributed Systems*, **5**(7), pp. 688-696, July 1994

[11] L. Lamport On interprocess communication, part i: basic formalism, part ii: basic algorithms, *Distributed Computing* **1**, 77–101.

[12] P. Vitányi and B. Awerbuch Atomic shared register access by asynchronous hardware. *In Proc. 27th IEEE Symp. on Foundations of Computer Science*, 1986, pp. 233–243.