Atomic Shared Register Access by Asynchronous Hardware
(Detailed Abstract)

Paul M.B. Vitányi

Massachusetts Institute of Technology, Laboratory for Computer Science
Cambridge, Massachusetts
and
Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands

Baruch Awerbuch

Massachusetts Institute of Technology, Department of Mathematics
and Laboratory for Computer Science, Cambridge, Massachusetts

ABSTRACT

The contribution of this paper is two-fold. First, we describe two ways to construct multivalued atomic n-writer n-reader registers. The first solution uses atomic 1-writer 1-reader registers and unbounded tags. The other solution uses atomic 1-writer n-reader registers and bounded tags. The second part of the paper develops a general methodology to prove atomicity, by identifying a set of criteria which guaranty an effective construction for the required atomic mapping. We apply the method to prove atomicity of the two implementations for atomic multiwriter multireader registers.

1. Introduction

When two processors communicate, they may do so using a shared memory, i.e., one processor writes the message in it and the other processor reads the message from it. Less obvious is the case when the processors communicate by message passing. In [Lamport1986] it is pointed out that the message is put in a buffer at the receiver's end, and a flag is set. The receiver periodically tests the flag, and fetches the contents of the buffer when the flag indicates that the buffer contains a message. Logically, the flag register is a shared register between the sender who can write it, and the receiver who can read it. Thus, concurrent reading and writing of a shared register is more basic than mutual exclusion, semaphores and the like, which require interprocess communication. Solutions to the problem of concurrent reading and writing which rely on mutual exclusion or any other method which serializes the concurrent actions by executing them in sequence (and therefore having one action wait for another) only shift the problem to another level. Another issue in favor of truly concurrent reading and writing is variation in speed of different technologies. E.g., when a Cray XMP communicates with an IBM PCjr, the Cray is several orders of magnitude faster, and solutions requiring exclusive access to the shared register would slow the faster machine down to the speed of the slower one. We analyse the problem of how to implement a shared register which can be written and read by all participating processors in a truly concurrent fashion. I.e., without any restrictions to prevent simultaneous access and making no assumptions about the relative durations of the read's and write's, or about the actual timing of the lower level constituent actions. The implementations preserve the property that the read's and write's *seem* to take place in an indivisible instant of time each, and in a particular order. Each action can be thought to take place in an *atomic* grain of time, and this time atom is situated somewhere in the action's finite time span as used by the executing processor in reality. This latter condition ensures that an external observer of the processors

The work of the first author was supported in part by the Office of Naval Research under Contract N00014-85-K-0168, by the Office of Army Research under Contract DAAG29-84-K-0058, by the National Science Foundation under Grant DCR-83-02391, and by the Defense Advanced Research Projects Agency (DARPA) under Contract N00014-83-K-0125. The work of the second author was supported in part by a Chaim Weizmann Postdoctoral Fellowship and by the Air Force Office of Scientific Research under Contract TNDGAFOSR-86-0078.

cannot find contradictions; the actions *may* have taken place in this order according to the observed results, i.e. *external consistency*. Moreover, the sequence of values written and read by the successive atomic actions in the particular order have the *register property*: an atomic read following an atomic write, without other atomic write's in between, returns the value that atomic write wrote, i.e. *internal consistency*. A sequence of such seemingly atomic read's and write's, which are both externally consistent and internally consistent is called an *atomic run*. A shared register, such that all system executions of read's and write's by the participating processors are atomic runs, is called an *atomic* register.[1] We construct multivalued atomic registers which can be read and written asynchronously by many processors. The solutions do not require one process to wait for another. This rules out any solution using mutual exclusion, synchronization, execution rounds, and so on. The roots of the problem under consideration are hardware design issues of concurrent accesses to registers by asynchronous components, and asynchronous interprocess communication.

Results. The result of the paper is two-fold. First, we describe two ways to construct multivalued atomic n-writer n-reader registers. The first solution uses multivalued atomic 1-writer 1-reader registers and unbounded tags. The other solution uses multivalued atomic 1-writer n-reader registers and bounded tags. The second part of the paper develops a general methodology to prove atomicity, by identifying a set of criteria which guaranty an effective construction for the required 'atomic mapping,' cf. below. We apply the method to prove atomicity of the two implementations for atomic multiwriter multireader registers.

Tag Size. We call the number of bits needed to represent the values, which have to be written into a register, the *range* of that register. The first construction below solves the problem how to implement an atomic n-reader n-writer register R using a matrix of n^2 atomic 1-reader 1-writer registers $R_{i,j}$. (Such registers exist in the sense that Lamport [Lamport1986] has exhibited an implementation from existing hardware components.) In this solution, the range V' of the constituent registers $R_{i,j}$ is bounded by $V' \leq V + \log T$, where T equals the maximum number of actions issued to R, and V is the range of the register R being constructed. The second solution below solves the same problem, starting from atomic 1-writer n-reader registers, where the range of values written into the constituent reqisters is bounded by $V' \leq V + 4n^2 \log n$. At the cost of some complication this bound may be improved to $V' \leq V + 4n \log n$.

The number T of *tag* bits used by a solution is $V' - V$. A solution uses *bounded tags* (i.e., $V' - V < \infty$) if the range of constituent registers is bounded as a function of the range of the constructed register and the number of processors, otherwise it is said to use *unbounded tags*. In the unbounded tag solution above, the largest tag which ever needs to be used is bounded by the total number of read and write actions of the constituent processors in the lifetime of the system. Since it is not likely that the number of such actions will exceed, say 2^{100}, the cross-over point where the unbounded tags solution becomes superior to the bounded tags solution is for a number of participating processors between 10 and 20. (Note, that atomic 1-writer n-reader registers using bounded tags, have not yet been constructed.)

Related Work. Motivation and explanation of the defined concepts and their relevance to current computing issues can be found in [Lamport1986, Misra1986]. In the former paper, Lamport gives an implementation of a multivalued atomic 1-writer, 1-reader register. In [Peterson1983] an atomic 1-writer n-reader m-valued register is constructed using $n+2$ *safe* n-reader m-valued registers (i.e., registers that return the correct value if no Write overlaps the Read, and some value from the correct domain otherwise), $2n$ atomic, boolean, 1-reader registers, and two atomic, boolean, n-reader registers. (It is not known how to construct the last type.) Misra [Misra1986] has investigated axioms for the design of multiwriter registers. Bloom [Bloom1986] has constructed an atomic 2-writer, n-reader register from atomic 1-writer n-reader registers. Related research includes [Lamport1977]. Our idea of using ticket algorithms was inspired by the example of [Fischer1985], but is otherwise unrelated. We use the 'global time' model of [Lamport1986, Lamport1986 .b....].

2. Formalism and Problem Statement

An *action* or *operation execution* is a *Read* or a *Write*.

Let S be a finite set of *actions*

$$S = \{a_1, a_2, \ldots, a_m\}$$

or an infinite set of *actions*

$$S = \{a_1, a_2, \cdots\}$$

and let R be the set of nonnegative real numbers.

$s: S \to R$ maps each action $a \in S$ to a *start* time $s(a)$.

$f: S \to R$ maps each action $a \in S$ to a *finish* time $f(a)$, $f(a) > s(a)$.

f and s are such that $u(v) \neq x(y)$ for $u, x \in \{s, f\}$, $v, y \in S$ and $v \neq y$ or $u \neq x$. (*Each time instant harbors only one start or finish.*)

$|\{s(a): s(a) < c, a \in S, c \in R\}| < \infty$. (*At any time, only a finite number of actions has started.*)

$\pi: S \to S$ is a *reading* mapping which maps an action a to an action $\pi(a)$. Many actions can be mapped to a single action.

A *run* is a fourtuple $\rho = (S, s, f, \pi)$.

A *register* mapping $REG: \{(S, s, f)\} \to \{\pi\}$ associates a reading mapping π with each triple (S, s, f). (*Each register implementation induces a register mapping.*)

The *set* of all runs associated with a register REG is:

$$\{\rho : \rho = (S, s, f, REG(S, s, f))\}$$

A *shrinking* mapping $\sigma: R \to R$ is a 1:1 mapping which associates new start and finish times with each action $a \in S$ such that

$$s(a) \leq \sigma(s(a)) \leq \sigma(f(a)) \leq f(a)$$

(*Intuition: external consistency is maintained under this condition.*)

A shrinking mapping is *serial* if for all $a, b \in S$, $a \neq b$, it holds that

$$\sigma(s(b)), \sigma(f(b)) \notin [\sigma(s(a)), \sigma(f(a))]$$

A serial shrinking mapping σ is *consistent* with a run ρ if π's reading and writing order is consistent with the obvious reading and writing order induced by σ.

A run ρ is *atomic* if there exists a serial shrinking mapping which is consistent with it. (*Intuition: atomicity implies internal consistency.*)

A register is *atomic* if each of its runs is atomic.

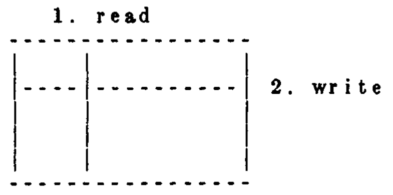
For an atomic register it *seems* that all actions take place at nonoverlapping intervals as defined by the shrinking mapping consistent with a run. For convenience, we may think these intervals shrunk to single points, so that we can conceive of the actions as being executed in an indivisible grain of time.

Problem. Given simple atomic registers, implement more general ones. In particular, (a) given atomic n-reader 1-writer registers implement atomic n-reader n-writer registers, or better, (b) given atomic 1-reader 1-writer registers (Lamport registers) implement atomic n-reader n-writer registers. (All registers multivalued.)

To *implement* an atomic n-reader n-writer register R from other atomic registers means the following. There is a set of processors $1, \ldots, n$ and a set of registers $R_1, \ldots, R_m$. Every one of the processors i issues Read and Write actions to a conceptual common register R. Every such high-level action will be implemented by many low level reads and writes to the constituent registers $R_1, \ldots, R_m$ of R. Each processor executes a Protocol which specifies this implementation. The *input* to the Protocol consists of start points of Read actions issued by the readers, start points of Write actions issued by the writer and the values written in Write actions. The *output* of the Protocol consists of endpoints of Read actions issued by the readers, endpoints of Write actions issued by the writers and the values returned in Read actions. The Protocol together with registers $R_1, \ldots, R_m$ implements an atomic register R if the register mapping induced by R is atomic.

3. First Solution

Consider the following architecture, consisting of n processors $1, \ldots, n$, and n^2 atomic registers, $R_{i,j}$, each having one read-terminal and one write-terminal. Each i is connected to the write-terminal of each $R_{i,j}$ and to the read-terminal of each $R_{j,i}$ $(1 \leq i, j \leq n)$. The n^2 registers form a matrix R with register $R_{i,j}$ the element in the ith row and the jth column. A processor i can write all registers in row i and read all registers in column i. Each register $R_{i,j}$ can hold any tag in $N \times \{1, \ldots, n\}$, N the set of natural numbers.



Informally, the Read and Write protocols are as follows (suppose i does the Read or Write):

1. For all j in $\{1, \ldots, n\}$, read the contents of register $R_{j,i}$, in one read each. Determine the lexicographically highest tag $(t_{\max}, m)$ held by any register $R_{j,i}$, and set own tag equal $(t_{\max}+1, i)$ for a Write and to $(t_{\max}, m)$ for a Read.
2. For all j in $\{1, \ldots, n\}$, write a new contents consisting of a new *tag* and *value* to register $R_{i,j}$, in one write each. For a Write the new value is the value v which has to be

written, for a Read the new value is the value v_m held by a register $R_{j,i}$, which held the lexicographical maximal tag $(t_{\max}, m)$.

The atomic subactions of a Read or Write by processor i consist of first reading all atomic registers in its associated column, and then writing all atomic registers in its row. The order within the reading phase and within the writing phase is arbitrary, symbolized by the "for all j in *SET*" construct.

4. Second Solution (Struldbrugg Protocol[2])

The architecture consists of n atomic 1-writer n-reader registers, $R_1, \ldots, R_n$. There are n processors $1, \ldots, n$, each i is connected to the write-terminal of R_i and to a read-terminal of each R_j $(1 \leq j \leq n)$. The n registers form an array A with register R_i the ith element. A processor i can write register R_i and read all registers in A.

4.1. Outline of the Protocol

On the first approach, the protocol is similar to the unbounded tags case. Namely, the writer draws a new "ticket" every time it writes, and the reader returns the value of the most recent "ticket". Every writer draws from a separate pool of tickets. Thus, each processor (c.q. register) has its *own* tickets. The main idea is to maintain a bounded number of tickets, and keeping track of the ordering in which they were issued. However, to facilitate the presentation, we assume that every time that a certain ticket is redrawn, a special unbounded *draw number* is attached to that ticket, which is incremented by 1 with every draw. Later, we show how to get rid of those draw numbers, showing that at any given time, all existing tickets with the same name have necessarily the same draw number.

It still remains to keep track of tickets with different names. For that purpose, the tag called *view* in R_i is an array which contains in the ith position the name of the current ticket of R_i, and in position j contains the ticket which was register R_j's current ticket at the time it was last polled by i. Before i issues a new ticket in a Write, it looks at the views kept at each register, and then selects as the new ticket for the ith entry of its own new view a ticket which does not appear as the ith entry in any of the polled views. There is one-to-one correspondence between the value written by the write of i, the ticket (including draw number), and the view in R_i whose ith entry is that ticket. The jth entry $(j \neq i)$ of the new view consists of the jth entry of the current view polled in register R_j.

Readers read all views from all registers and determine the processor which wrote 'latest,' by comparing the views. Namely, if the ith entry of the ith view occurs as the ith entry of the jth view, then the jth view is *deo volente* written later than the ith view.

The method as described does not suffice, because it cannot keep track of views which are in "transit", i.e. views that have been read, but the action who read them has not terminated yet. For instance, when there is a very long read, which overlaps many actions of some processor, say 100 write actions, it is impossible to predict which one of those 100 views will be actually read by the reader. However, the writer should avoid picking a ticket that appears in a view which is read, since the new ticket will be confused with its old version. This complication seems to force us to use unbounded histories as tags in the registers.

The following mechanism, referred to as *Send-Receive* mechanism, solves this problem. Effectively, in the situation where a long action overlaps many short actions of another processor, it makes it appear that the long action polled the result of the first short action among overlapped ones, or the one preceding it.

4.2. Preliminaries

Every variable mentioned in the algorithms is said to be *owned* by a processor. A variable, owned by processor i, is represented by a certain field in its register R_i. By construction of the array A, only the processor i can write this variable. Whenever the processor writes a variable it owns, it does so to the associated field in its register. Every processor can read the value of this variable from that register.

The algorithm revolves around the sending of messages between senders and receivers and the views held by the readers and writers.

4.3. The Send-Receive Procedure

The procedure involves two parties, the *sender* S and the *receiver* R. The procedure involves the following parameters: Message, Input, Output, Received, Bits. The Message and Input are fields owned by the sender, and the Bits and Received are fields owned by the receiver. Message and Received are arrays. Message[R] indicates entry R in the array. In fact, only entries Message[R] and Received[S] are modified by the procedure between S and R.

Informally, the sender sends a new message to the receiver if the previously sent message has been received or if the receiver is "idle". To indicate whether the receiver is idle or not, the receiver writes this information onto *Bits*. The Bits consist of a Flag and an Alternatingbit. The contents of Flag is 0 if receiver is idle and 1 otherwise. The contents of Alternatingbit (0 or 1) is complemented with each action of the receiver.

Upon invocation of the Send procedure, the sender reads Bits, Received[S] from the receiver. If Flag=0 or Received[S]=Message[R] or Alternatingbit has changed, then the sender writes the value of Input into Message[R].

Upon invocation of the Receive procedure, the receiver reads Message[R] and Input from the sender. If Received[S] already equals Message[R], then the value of the Input is assigned to Output Otherwise, the value of Message[R] is assigned to Output and also written into Received[S].

4.4. The Read and Write Protocols

Upon the invocation of a Write, the writer writes 1 in its Flag field, to indicate action in progress. It complements the value of its Alternatingbit field. Then the writer calls the Send procedure, for every other register R_j as receiver, with parameter Input=View, and Bits=Flag, Alternatingbit. The writer continues by calling the Receive procedure, for every other register R_j as sender, with parameters Input=View and Output=seen[j]. The writer then discredits all its own tickets occurring in any field of any register, i.e., Message, View and Received. The new ticket drawn by the writer is an arbitrary own ticket which is not discredited. For convenience, we imagine the value which is the input to the Write attached to this new ticket. Then a new view is composed, the jth entry consisting of the jth entry of the view, i.e. seen[j], which is the output of the Received procedure for R_j as sender, $j \neq i$. The newly drawn own ticket is the ith entry of the new view. Then the new view is written to the own field View. Finally, the writer writes 0 to its Flag field, to indicate the end of Write.

Upon invocation of the Read, the reader writes 1 in its Flag field, to indicate action in progress, and complements the Alternatingbit field. Then it calls the Receive procedure, with parameters Input=View and Output=seen[j], for every other register R_j as sender. Seen[i] is the view in its own View field. From the received views, seen[1] , . . . , seen[n], the reader selects the most recent one, i.e., the unique view seen[m] of which entry m does not occur as entry m of any seen[j] ($j \neq m$), m maximal. It then returns the value associated with the mth entry of seen[m], and writes 0 to its Flag field, to indicate the end of Read.

Convention. Variables without capitals are local variables of the executing processor. Variables with capitals are fields in registers, which are read and written by the executing processor in its own register, and are 'Read-from-X' by the executing processor from another register X. All 'Read-from-X' operation executions in a Read or Write are performed in *one* lower level read of register X. Each operation execution of a Read or Write starts with *one* lower level write to the own register to set Bits, continues with *one* lower level read of each other register to obtain data (the processor either does a lower level read to its own register or just remembers what it wrote last), and ends with *one* lower level write to the own register to set Flag to 0 and write the appropriate fields. Note that all the registers involved, i.e., the elements of A, are atomic.

4.5. The Protocol

Procedures

```
Procedure New (Bits)
begin Procedure
    flag := 1;
    alternatingbit := complement (alternatingbit);
    Bits := flag, alternatingbit
end Procedure

Procedure SEND(Input, Bits, R, S)
/* S sends new message to R if the previous message
sent was received or if R is idle */
begin Procedure
    flag, alternatingbit := Read-from-R (Bits);
    received := Read-from-R (Received[S]);
    if received= Message[R] or flag = 0 or alterna-
    tingbit changed
    then  Write (Message[R] := Input)
end Procedure

Procedure RECEIVE(Input, Output, S, R)
/* R receives message from S */
begin Procedure
    message := Read-from-S (Message[R]);
    input := Read-from-S (Input);
    if Received[S] = message
    then Output := input
    else
    begin
        Output := message;
        Write ( Received[S] := message )
    end
end Procedure
```

The Read and Write Protocols

```
Protocol for Write by i
begin
    Write (Bits := New (Bits));
```

```
        for all j unequal i
        begin
              SEND (Input=View, Bits, j, i);
              RECEIVE(Input=View,   Output=seen[j], j,i);
              discredited[j] := Read-from-j (Own tickets in Message, View, Received)
        end
        discredited[i] := Own tickets in Message, View, Received;
        ticket := arbitrary own ticket not occurring in discredited[j] for j=1,...,n;
        Write ( View := ticket + for all j unequal i, entry j of seen[j]);
        Write (Flag := 0)
end

Protocol for Read by i
begin
        Write (Bits := New (Bits));
        for all j unequal i
        begin
              RECEIVE(Input=View,   Output=seen[j], j,i)
        end
        Read ( seen[i] := View );
        determine maximal lexicographic sink in the Seen graph S determined by seen[j], j=1,...,n, (see below);
        output the value associated with this ticket;
        Write (Flag := 0)
end
```

Seen Graph. The Seen graph $G_r=(W_r,E_r)$, associated with a Read r, has as vertices the jth entry of seen[j], that is, the 'own' ticket t_j in register R_j's view as obtained by r, for $j=1, \ldots, n$. There is a directed edge $t_i \rightarrow t_j$ between t_i and t_j if t_i is the ith entry of seen[j].

5. Method and Proofs

The atomic constituent registers of an implementation support the constituent lower level actions of a higher level action of the register constructed by way of solution. The lower level actions start when a processor starts executing them, and end when the processor finishes executing them. Because of atomicity, there exists a point in this interval at which the action *seems* to take place. We use those mathematical fictions - i.e., the instants of time at which the action *may be thought* to take place *instantaneously* - in our formal reasoning. In particular, $s(a)$ is an instant of time during which the first lower level atomic action of a can be thought to take place, and $f(a)$ is an instant of time during which the last such subaction can be thought to happen.

Auxiliary Constructs. Below we need additionally the following *polling* mapping.

$\mu:(S-W)\rightarrow W^n$, $W\subseteq S$, is a partial *polling* mapping which maps an action a to n actions

$$(a_1, \ldots, a_n) = \mu(a)$$

such that a_i is an action of processor i. Here $W\subseteq S$ is the set of Writes and $S-W$ is the set of Reads. The projection $\mu_i(a) = a_i$.

Let $G=(W,E)$ be an *action digraph*, W as above, and a directed edge $(a\rightarrow b)\in E$ $(a\neq b)$ if

1) there exists a $r\in S-W$, such that $a=\mu_i(r)$, for some i, and $b=\pi(r)$, or
2) both b and a are Writes by the same processor, with b the next Write after a.

5.1. Sufficient Conditions for Atomicity

It will be shown that (P1) - (P4) imply atomicity.

(P1) a) For all Reads r, $w=\pi(r)$ is an entry of $\mu(r)$. (I.e., one Write is polled from each processor, and the Write $\pi(r)$, which is selected by Read r, is one of them.)

b) The action digraph is acyclic, i.e. has no directed cycle. (I.e., there is no chain of selected actions which forms a cycle.)

(P2) If $w=\pi(r)$ then $f(w) < f(r)$. (I.e., a Read does not return the value of a Write which finishes later than itself.)

(P3) If $f(w)<s(r)$, w a Write of i, then either $w=\mu_i(r)$ or there is a Write w', $f(w)<s(w')<f(r)$ and $w'=\mu_i(r)$. (I.e., if there is a Write w of i which properly precedes a Read r, then either w is polled by r or a later action w' (of i), which starts before r finishes, is polled by r.)

(P4) For all Reads r, no Write w satisfies $f(\pi(r))<s(w)<f(w)<s(r)$. (I.e., there is no Write properly in between a Read and the Write it selects.)

Lemma 1. *If a run $\rho=(S,s,f,\pi)$, with polling mapping μ and $W\subseteq S$ the set of Writes, satisfies (P1)-(P4) then ρ is atomic.*

Proof. We construct explicitly a serial mapping $\sigma*$ consistent with run ρ. Call a Write $w\in W$ *live* if its value is returned by a Read r, that is, $w=\pi(r)$ for some $r\in S-W$, otherwise *dead*.

(1) Let σ be a shrinking mapping such that, for each Write $w\in W$, $\sigma(s(w))=\sigma(f(w))=f(w)$, and for each Read $r\in S-W$, $\sigma(s(r))=\sigma(f(r))=\max\{s(r), f(\pi(r))\}$. The images of $s(r)$ and $f(r)$ are contained in $[s(r), f(r)]$ by (P2). That is, σ contracts an interval to a point inside the interval.

(2) For any live Write $w\in W$, we define an *influence interval* of that Write (w.r.t. σ) as the maximum interval of time $[\sigma(w), \sigma(r)]$, such that

$w = \pi(r)$. Below '$\cdots$' means a finite nonzero time interval, and its absence a zero time interval. Thus, '$\sigma(\pi(r))\sigma(r)$' means that the interval between '$\sigma(\pi(r))$' and '$\sigma(r)$' has measure 0. Assume below that $w_i = \pi(r_i)$, and that $[\sigma(w_i), \sigma(r_i)]$ are influence intervals, $i = 1,2$.

Remark. We assume some ϵ-interval of measure 0 around each start and finish of a Read or Write. This makes physical sense [Lamport1986]. If mapping σ creates a cluster of Reads and Writes we sort the cluster to put all Reads immediately after the Writes to which they belong, as far as possible. E.g., '$\sigma(w_2)\sigma(r_1)\sigma(r_2)$' is resolved as '$\sigma(w_2)\sigma(r_2)\sigma(r_1)$.'

Claim 1. The influence intervals w.r.t. σ can overlap only in the following way:

$$\cdots \sigma(w_1)\sigma(w_2)\,\sigma(r_2) \cdots \sigma(r_1) \cdots \qquad \text{(i)}$$

$$\cdots \sigma(w_1) \cdots \sigma(w_2)\,\sigma(r_2)\sigma(r_1) \cdots \qquad \text{(ii)}$$

$$\cdots \sigma(w_1) \cdots \sigma(w_2)\,\sigma(r_2) \cdots \sigma(r_1) \cdots \qquad \text{(iii)}$$

Proof. Suppose the contrary. By (P2) and the Remark above, the only other possibilities for overlap of the influence intervals are (modulo interchange of elements in a cluster):

$$\cdots \sigma(w_1) \cdots \sigma(w_2) \cdots \sigma(r_1) \cdots \sigma(r_2) \cdots$$

$$\cdots \sigma(w_1) \cdots \sigma(w_2) \cdots \sigma(r_2) \cdots \sigma(r_1) \cdots$$

$$\cdots \sigma(w_1)\sigma(w_2) \cdots \sigma(r_2) \cdots \sigma(r_1) \cdots$$

$$\cdots \sigma(w_1) \cdots \sigma(w_2) \cdots \sigma(r_2)\sigma(r_1) \cdots$$

$$\cdots \sigma(w_1)\sigma(w_2) \cdots \sigma(r_2)\sigma(r_1) \cdots$$

We show that the first situation is impossible. The impossibility of the others follows by the essentially the same argument.

By the construction of σ in (1), $f(w_1), f(w_2) < s(r_1), s(r_2)$. Otherwise, '$\sigma(r_1)$' and '$\sigma(r_2)$' could be shifted further left. Let $w_1 = \mu_i(r_1)$ and $w_2 = \mu_j(r_2)$. By (P3), $\mu_i(r_2) = w_1$ which implies $w_1 \rightarrow w_2$ in E, or $\mu_i(r_2) = w_i$ with $f(w_1) < s(w_i) < f(r_2)$, which implies $w_1 \rightarrow \cdots \rightarrow w_i$ and $w_i \rightarrow w_2$. Again by (P3), $\mu_j(r_1) = w_2$, which implies $w_2 \rightarrow w_1$, or $\mu_j(r_1) = w_j$ with $f(w_2) < s(w_j) < f(r_1)$, which implies $w_2 \rightarrow \cdots \rightarrow w_j$ and $w_j \rightarrow w_1$. All combinations contradict (P1): the acyclicity of the action digraph. End proof of Claim 1.

Claim 2. We can construct $\bar{\sigma} = \sigma' \cdot \sigma$, so that the images of the elements of S under $\bar{\sigma}$ form a set of nonoverlapping influence intervals.

Proof. By Claim 1, σ maps the elements of S such that the only possibilities for overlapping influence intervals under σ which are left are (i) - (iii):

$$\cdots \sigma(w_1)\sigma(w_2)\,\sigma(r_2) \cdots \sigma(r_1) \cdots \qquad \text{(i)}$$

Then '$\sigma(w_1)\sigma(w_2)\,\sigma(r_2)$' is mapped to a single 0-length ϵ-interval, and we can change the mapping so that the image in that ϵ-interval is '$\sigma(w_2)\,\sigma(r_2)\sigma(w_1)$'. Similar for the situation

$$\cdots \sigma(w_1) \cdots \sigma(w_2)\,\sigma(r_2)\sigma(r_1) \cdots \qquad \text{(ii)}$$

The last remaining possibility for overlapping influence intervals is:

$$\cdots \sigma(w_1) \cdots \sigma(w_2)\,\sigma(r_2) \cdots \sigma(r_1) \cdots \qquad \text{(iii)}$$

In this case, $f(w_1) < f(w_2) < s(r_1)$, and $s(r_2) < s(r_1)$. Let w_1 be a Write of i and w_2 be a Write of j. By (P3), either $w_2 = \mu_j(r_1)$, and $w_2 \rightarrow w_1$ is a directed edge in the action digraph G, or there is a $w_j = \mu_j(r_1)$ with $f(w_2) < s(w_j) < f(r_1)$ and both $w_2 \rightarrow \cdots \rightarrow w_j$ and $w_j \rightarrow w_1$ are directed edges in G.

Assume $f(w_1) < s(r_2)$. Then, by (P3), either $w_1 = \mu_i(r_2)$ which implies a directed edge $w_1 \rightarrow w_2$ in G, or $w_i = \mu_i(r_2)$ for $s(w_i) > f(w_1)$, which implies both $w_1 \rightarrow \cdots \rightarrow w_i$ and $w_i \rightarrow w_2$ in G.

All combinations violate the acyclicity of G claimed by (P1). Hence, $s(r_2) < f(w_1)$.

Since $w_1 = \pi(r_1)$, and $f(w_2) < s(r_1)$, by (P4) we have $s(w_2) < f(w_1)$. (w_2 cannot be properly in between $\pi(r_1)$ and r_1.)

Thus the situation is as follows:

```
                                               r_1
                                              *------
         w_1
---------*
           w_2
  ------------*
                r_2
   ----------------*

-----------σ(w_1)--σ(w_2)σ(r_2)---σ(r_1)---
```

We can now shift the pair 'σ(w_2)σ(r_2)' to immediately before 'σ(w_1)'. In this way, we obtain a mapping as follows:

$$\cdots \sigma(w_2)\sigma(r_2)\sigma(w_1) \cdots \sigma(r_1) \cdots$$

This reduces the amount of nesting of the influence intervals. We iterate this procedure until all proper nesting has disappeared, and compose the resulting mapping σ' with the previous mapping σ to $\bar{\sigma} = \sigma' \cdot \sigma$. End proof of Claim 2.

Trivially, we can map all Reads r', with $\pi(r') = \pi(r)$, such that $s(r') < s(r)$ in between the images $\bar{\sigma}(\pi(r))$ and $\bar{\sigma}(r)$.

(3) A dead Write w is one such that for no Read $r \in S-W$ holds $w=\pi(r)$. By (P4), for no dead Write w is there a Read r such that $f(\pi(r))<s(w)<f(w)<s(r)$, $s(r)$ maximal.

So, after having constructed σ as in (1), composed it with a mapping σ' to $\overline{\sigma}$ as in (2), we can finally, by yet another mapping σ'', shift the image of such dead Writes w to just before $\overline{\sigma}(\pi(r))$ or just after $\overline{\sigma}(r)$. The resulting mapping $\sigma^* = \sigma'' \cdot \sigma' \cdot \sigma$ is a serial shrinking mapping of S and is consistent with ρ, i.e., run ρ is atomic. •

5.2. Proof First Solution

We now prove that register R is atomic. That is, for each run ρ of Reads and Writes there exists a mapping σ^* as above.

Define the reading mapping π induced by register R such that a Read r is mapped to the Write w whose value it returns. Let ρ be a run (S,s,f,π) of register R. Define the run $\rho' = (S,s,f',\pi)$ with $f' = \sigma \cdot f$, where σ is a shrinking mapping defined as $\sigma(f(w)) = \min(\{f(r): w=\pi(r)\} \cup \{f(w)\})$ for the finish of a Write $w \in W$, and the identity in all other cases. We define μ such that $\mu_i(r)$ is the *last* Write of i of which the tag and value are seen by Read r, directly or indirectly. We use an intermediate mapping γ. Let γ_i map every Read r, say by j, into action a_i which wrote in $R_{i,j}$ the value which r read from $R_{i,j}$. Let w_i be the last Write by i which wrote $R_{i,j}$ before it was read by r. Note, that w_i not necessarily equals a_i. The polling mapping μ is defined by $\mu_i(r) = \pi(\gamma_k(r))$ or $\mu_i(r) = w_i$, whichever is the latest Write by i among $w_i, \pi(\gamma_1(r)), \ldots, \pi(\gamma_n(r))$. (Here, let π map every Write to itself.)

Lemma 2. *The run $\rho' = (S,s,f',\pi)$, with W the set of Writes and μ the polling mapping, satisfies (P1)-(P4).*

Proof. (P1). a) By construction $\pi(r)$ is an entry of $\mu(r)$, for all Reads $r \in S-W$.

b) The action digraph G is a subgraph of the digraph with $N \times \{1, \ldots, n\}$ as the set of vertices (N is the set of natural numbers) and a directed edge $i \rightarrow j$ from vertex i to vertex j if i is lexicographically less than j. Consequently, G is acyclic for ρ and ρ'.

(P2). If $w=\pi(r)$ then by construction of σ we have $\sigma(f(w))<\sigma(f(r))$. (Actually $\leq$, but allow a small ϵ-shift, see Remark in proof Lemma 1.)

(P3) In the definition of μ it holds that $s(\mu_i(r))<f(r)$. In particular (P3) is satisfied.

(P4). Holds for ρ by construction of R and definition of π. I.e., for all Reads $r \in S$, no Write $w \in W$ satisfies $f(\pi(r))<s(w)<f(w)<s(r)$. Let w be a Write by i. Since $\sigma(f(a)) \leq f(a)$ for all $a \in S$, and because of the definition of μ, the only cases in which (P4) can fail to hold for ρ', is that (a) $f(w)>s(r)$, $\sigma(f(w))<s(r)$, and $\sigma(f(\pi(r)))<s(w)$, or (b) $f(\pi(r))>s(w)$, $\sigma(f(\pi(r)))<s(w)$ and $\sigma(f(w))<s(r)$. We analyse case (a). Case (b) is analogous. By definition of σ, there is a Read r' such that $w=\pi(r')$, and $f(r')=\sigma(f(w))<s(r)$. Since $s(w)>\sigma(f(\pi(r)))$, by construction of R, Write w draws a higher ticket than Write $\pi(r)$. By definition of μ, $w=\mu_i(r)$ or there is a later Write w' of i such that $w'=\mu_i(r)$, with a higher ticket than w. But by R's Read Protocol, the tag of $\pi(r)$ is lexicographically at least as high as the tag of $\mu_i(r)$, which is a contradiction. •

Theorem 3. *R is an atomic n-reader n-writer register implemented using n^2 atomic 1-reader 1-writer registers and unbounded tags.*

Proof. Let $\rho=(S,s,f,\pi)$ be a run associated with R. Then $\sigma^* \cdot \sigma$, with σ the shrinking mapping used to obtain ρ' in Lemma 2, and σ^* the shrinking mapping constructed in the proof of Lemma 1, is a serial shrinking mapping which is consistent with ρ, by Lemma's 1 and 2. •

5.3. Proof Second Solution

Lemma 4. (Correctness Send-Receive) *Neither sender S in executing an action a_S nor receiver R in executing an action a_R can observe ticket x of S with draw number p in Message[R] and with draw number q in Receive[S], p unequal q.*

Proof. Suppose S puts x in Message[R] for the first time (draw number 1) in a_S. By the Protocol, x cannot occur in Received[S] until after $f(a_S)$. x stays in Message[R] until it is received by R in an action a_R, and occurs in Received[S] at $f(a_R)$. By the Protocol, x stays in Received[S], and in Message[R], until the end of an action a_S' which polls Received[S] containing x. Subsequent to $f(a_S')$, every next action of S will poll the same view in Received[S], until it is replaced in an action a_R' by another view in Message[R] of S. By the Protocol, x is on the list of discredited tickets of S, until S polls this new view in Received[R]. I.e., at least during the interval $[f(a_S), f(a_R')]$. Hence, by the Write Protocol, x cannot occur in the new view in Message[R]. Subsequent to $f(a_R')$, the initial situation that x occurs neither in Message[R] nor Receive[S] is restored. Consequently, the Lemma follows by induction. •

We now argue the uniqueness of tickets for the Read and Write Procedures.

Lemma 5. (Unique tickets.) *Tickets of i with different draw numbers do not turn up in the same Read or Write.*

Proof. Suppose the contrary, and let the tickets have name x and draw numbers p and q, $p > q$.

Assume the tickets turn up in a *Write* w. Then either a) or b) holds.

a) If w is a Write by i then x is simply not redrawn. The view written by w will not contain own ticket x (as ith entry) at all.

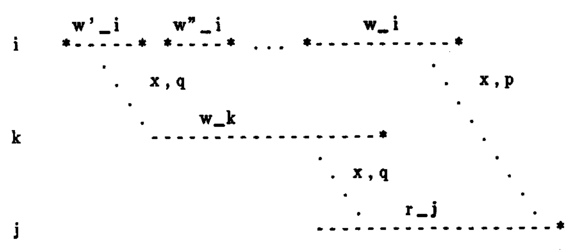
b) If w is a Write by j ($j \neq i$) then at most one x was in seen[i], the other one must be in seen[k], $k \neq i$. However, the ith entry of seen[k] is not used at all in the Write protocol as executed by j - for convenience we can consider it as not having been polled.

Either way, only in case b) is it possible that a ticket x of i turns up in the View of a Write by $j \neq i$. The only other way to obtain x of i is to poll it as the own ticket of a view in register R_i. Therefore Cases I-III below exhaust all possibilities. We do the analysis for a Read receiving the two x's. (The analysis for a Write is the same, but is unnecessary. Viz, already by a) and b) polling tickets of i with the same name and different draw numbers does not matter for Writes.)

We first note that if r_j adopts a ticket from i drawn by w_i then $f(r_j) > f(w_i)$. This is immediate from the Protocol.

Note, that i's tickets can be received from i by a write of k, put in view of k, send to l and received by l, but are not written by l, for any k and l. Hence, if a read r_j adopts tickets with the same name but different draw numbers from i, the situation must be either Case I, Case II, or Case III below.

Case I.



Claim 1. From $f(w_i{}')$ up till $f(w_k)$, ticket x,q is known to i, since it is either in i's View or in Message[k] of i.

Proof of Claim 1. After being drawn, x,q is written in a field of i's register for the first time at $f(w_i{}')$ in field View of i. By assumption x,q is adopted by k, and by definition of the Received procedure it was either in Message[k] or View of i, at the time when i was polled by k.

Case 1. If it was in Message[k] it will stay there, until i has seen that the view concerned is written in field Received[i] of k, that is, until after $f(w_k)$.

Case 2. Assume x,q was in View of i, when i was polled by w_k, and x,q was adopted by w_k. Then, by definition of Received procedure, Received[i] (in k) equals Message[k] (in i) as read by w_k.

Subcase 2.1. Suppose there is no write $w_i{}''$ with $f(w_i{}'') < f(w_k)$ after $w_i{}'$, then the view containing x,q stays in View of i until $f(w_k)$.

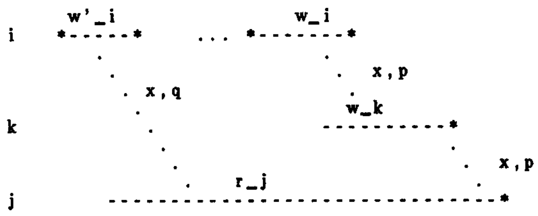
Subcase 2.2. Suppose, there is a next write $w_i{}''$ with $f(w_i{}'') < f(w_k)$ after $w_i{}'$ then, since w_k obtained x,q from View (of i) before $f(w_i{}'')$ by assumption, also $s(w_k) < f(w_i{}'')$. By the Send procedure, the view containing x,q goes into the Message[k] field of i, because of the assumption above that $w_i{}''$ polled Received[i]=Message[k]. Since $s(w_k) < f(w_i{}'')$ this new message of i to k stays in the Message[k] field of i until it is adopted in the Received[i] field of k, which is after $f(w_k)$. End proof of Claim 1.

Claim 2. From $f(w_k)$ up till $f(r_j)$, ticket x,q is in the View field of k, or the Message[j] field of k.

Proof of Claim 2. Instead of proving the Claim for x,q, we can prove it for the (own) ticket y drawn by w_k, because y and x,q are together in the same view written by w_k. The reasoning above applies to the presence of that view. The relation between w_k and r_j with respect to y, is exactly the same as between $w_i{}'$ and w_k with respect to x,q. According to Claim 1, it follows that the view with ticket y stays in View (of k) or Message[j] (of k) from $f(w_k)$ until $f(r_j)$. End proof of Claim 2.

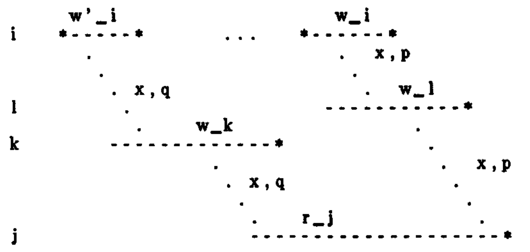
It follows from Claim 1 and Claim 2, that from $f(w_i{}')$, $f(w_i{}') < s(w_i)$, until $f(r_j)$, $f(r_j) > f(w_i)$, i either holds ticket x,q in its own register or polls it from k's register. Namely, if i polled k before $f(w_k)$, then x,q occurred in the register of i after $s(w_i)$ and will stay there until $f(w_i)$. If i polled k after $f(w_k)$ then it will obtain x,q from the register of k. According to the Write Protocol therefore, x is discredited and not drawn in w_i.

Case II.



Since r_j polls x, q from i, by analogous reasoning as above for w_k, in Case I, Claim 1, x, q must be in a field of register of i from $f(w_i')$ up to $f(r_j)$. Hence ticket x is discredited in w_i, and cannot be drawn.

Case III.



By the reasoning in Case I, w_i could not have drawn ticket x again, so this case is impossible as well.

By Cases I - III the lemma is proven. ●

Lemma 6. (Acyclicity) *The Seen digraph constructed from the Views polled by a Read has no directed cycles.*

Proof. By way of contradiction, suppose there is a directed cycle in the Seen graph G_r:

$$x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_l = x_1$$

By construction of the Seen graph we have a directed edge $x \rightarrow y$, when Read r polls views of i and j, written by w_i and w_j, respectively, such that the ith entry of seen[i] is the ticket with name x and draw number p, the ith entry of seen[j] is ticket x with draw number q, and the jth entry of seen[j] is ticket y. By Lemma 5, we have $p = q$. Hence, by construction of A, $f(w_i) < f(w_j)$. Therefore, the contradictory assumption implies $f(w) < f(w)$, for w the Write that drew ticket x_1, which is impossible. Thus, the directed edges in the Seen graph constructed by a read cannot form a directed cycle. ●

Corollary. The Read protocol can determine the maximal lexicographical sink of a Seen graph. That is, for each Read r there exists a $\pi(r)$, and $\pi(r)$ is unique.

Define $\pi(r)$ as the Write whose value is returned by Read r. Define $\mu_i(r)$ as the Write by i of which the tag and value are actually read by r from register R_i.

Lemma 7. (Atomicity) *A run $\rho = (S, s, f, \pi)$, with W the set of Writes, μ the polling mapping and π the reading mapping induced by register A, satisfies (P1)-(P4).*

Proof. (P1). a) By construction of A, for all $a \in S$, $\pi(a)$ is an element of $\mu(a)$.

b) The action digraph $G = (W, E)$ is constructed as follows.

1) If $G_r = (W_r, E_r)$ is the Seen graph associated with a Read $r \in S$, then determine $G_r' = (W_r', E_r')$ as follows. If $x \in W_r$ then $w_x \in W_r'$, with w_x the Write that drew ticket x (with draw number p as polled by r). For all $w \in W_r'$, $w \neq \pi(r)$, there is a directed edge $w \rightarrow \pi(r)$ in E_r'.

2) Note, that if E_r' contains an edge $w \rightarrow w'$ then $f(w) < f(w')$. Consider the set of intervals $I = (W, s, f)$ and draw the directed edges $w \rightarrow w'$ of graph $E_r' = (W_r', E_r')$ as $f(w) \rightarrow f(w')$ in I, for all Reads $r \in S$. The set of all such edges is $E_I \subseteq I \times I$. The resulting graph (I, E_I) does not contain directed cycles. (All edges go from left to right along the real line.)

3) Finally, let the action digraph $G = (W, E)$, with W the set of Writes, have $w \rightarrow w'$ a directed edge in E if there is a directed edge $f(w) \rightarrow f(w')$ in E_I, or if both w and w' are Writes by i and w' is the next Write after w.

By construction the action digraph G has no directed cycles.

(P2). By construction of A it holds that $f(\pi(r)) < f(r)$ for a Read r. (We allow rearrangement of the images of f in an ϵ-interval of measure 0. See Remark in proof Lemma 1.) Namely, the Write protocol writes all fields of the register in one final atomic action. Hence, the result of a Write w (i.e. contents of View, Message, Received and Value fields) can only be polled after $f(w)$.

(P3) By the construction of A and the Protocol, the View field written by a write w of i persists all through time interval $[f(w), f(w')]$, where w' is the Write of i which immediately follows w. A Read r cannot poll a register i outside time interval $[s(r), f(r)]$. Therefore, (P3) easily follows.

(P4). By way of contradiction, suppose for some Read r there is a Write w such that $f(\pi(r)) < s(w) < f(w) < s(r)$. Let w be a Write by j, and let $\pi(r) = \mu_i(r) = w'$. By (P3), $j \neq i$. By construction of A and the Protocol, the View field of R_i persists through interval $[f(w'), s(r)]$. Hence, by the Send-Receive mechanism, seen[i] as polled by w is the same as seen[i] as polled by r. Consequently, the ith entry of seen[i] equals the ith entry of seen[j], as polled by r. Let the name of this ticket be x. The draw number of x is the same for seen[i] and seen[j] by Lemma 5. Let the name of the jth entry in seen[j] be y. Then $x \rightarrow y$ is a directed edge in the Seen graph constructed by r, and x is not a sink, which contradicts $\pi(r) = \mu_i(r)$, by the Read Protocol. •

Theorem 8. *A is an atomic n-reader n-writer register implemented using n atomic n-reader 1-writer registers and bounded tags.*

Proof. Each run ρ associated with A is an atomic run by Lemma's 1 and 7. By Lemma's 4,5 and 6 the draw numbers can be dispensed with: tickets need only have names. The number of discredited tickets in a Write by i equals the number of i's own tickets in the polled n View fields, $n(n-1)$ Received fields, and $n(n-1)$ Message fields. That is, at most $2n^2 - n$. Consequently, each processor needs have only $2n^2 - n + 1$ own tickets. The total tag information one needs to store in each register consists of 1 View field, $n-1$ Received views, $n-1$ Message views, 1 Flag and 1 Alternatingbit. This comes to a total of less than $4n^2 \lceil \log(2n+1) \rceil$ tag bits per register. •

Acknowledgement

Critical remarks of Bard Bloom, Oded Goldreich, Leslie Lamport, Nancy Lynch, Yoram Moses and Mark Tuttle were helpful.

References

Fischer1985.
Fischer, M.J., N.A. Lynch, J.E. Burns, and A. Borodin, "Distributed FIFO allocation of identical resources using small shared space," MIT/LCS/TM-290, Massachusetts Institute of Technology, Cambridge, Mass., October, 1985.

Lamport1977.
Lamport, L., "Concurrent reading and writing," *Comm. Ass. Comp. Mach.*, vol. 20, pp. 806-811, 1977.

Lamport1986.
Lamport, L., "On interprocess communication, Parts I and II," *Distributed Computing*, vol. 1, 1986. (To appear)

Peterson1983.
Peterson, G.L., "Concurrent reading while writing," *ACM Transactions on Programming Languages and Systems*, vol. 5, pp. 46-55, 1983.

Lamport1986.b
Lamport, L., "The mutual exclusion problem, Parts I and II," *J. Assoc. Comp. Mach.*, vol. 33, pp. 313-326, 327-348, 1986.

Misra1986.
Misra, J., "Axioms for memory access in asynchronous hardware systems," *ACM Transactions on Programming Languages and Systems*, vol. 8, pp. 142-153, 1986.

Bloom1986.
Bloom, B., "Constructing two-writer atomic registers," Manuscript, Massachusetts Institute of Technology, Cambridge, Mass., June, 1986.

Footnotes

[1] Suppose we have been able to construct an atomic flip-flop which can be tested (read) by one component (the *reader*) and set (written) by another component (the *writer*). How do we make an atomic register of n bits which can be read by one component and written by another? The writer can never write all flip-flops simultaneously, and neither can the reader read all flip-flops simultaneously. It is already a problem how to ensure that the reader gets either the new or the old value. Worse, if the register contains only values consisting of n-bit words with k bits equal 1, how do we ensure that a read which overlaps a write does not return a word with $\neq k$ bits equal 1. All registers mentioned in this paper are multivalued unless stated otherwise.

[2] I was asked by a person of quality, whether I had seen any of their Struldbruggs or Immortals. ... He told me that [they] happened to be born with a red circular spot in the forehead ... an infallible mark that [they] would never die. The spot ... changed its color; for at twelve years old it became green, so continued till five and twenty, then turned to a deep blue; at five and forty it grew coal black ... but never admitted any further alterations. ... They have no remembrance of anything but what they learned and observed in their youth and middle age, and even that is very imperfect. And for truth or particulars of any fact, it is safer to depend in common traditions than on their best recollections. ... As soon as they have completed the term of eighty years, they are looked on as dead by the law; their heirs immediately succeed to their estates, ... and the poor ones are maintained at the public charge. After that period they are incapable of any employment or trust or profit, ... neither are they allowed to be witnesses in any cause, ... not even for the decision of meres and bounds. ... their memory will not serve them to carry them through from a beginning of a sentence to its end. ... The language of this country always being in flux, the Struldbruggs of one age do not understand those of another, neither are they able after twohundred years to hold any conversation (farther than a few general words) with their neighbors the mortals, and thus lie under the disadvantage of living like foreigners in their own country. This was the account given me of the Struldbruggs, as near as I can remember. [Jonathan Swift, " A Voyage to Laputa, Balnibiri, Glubbdubdrib, Luggnagg and Japan."]