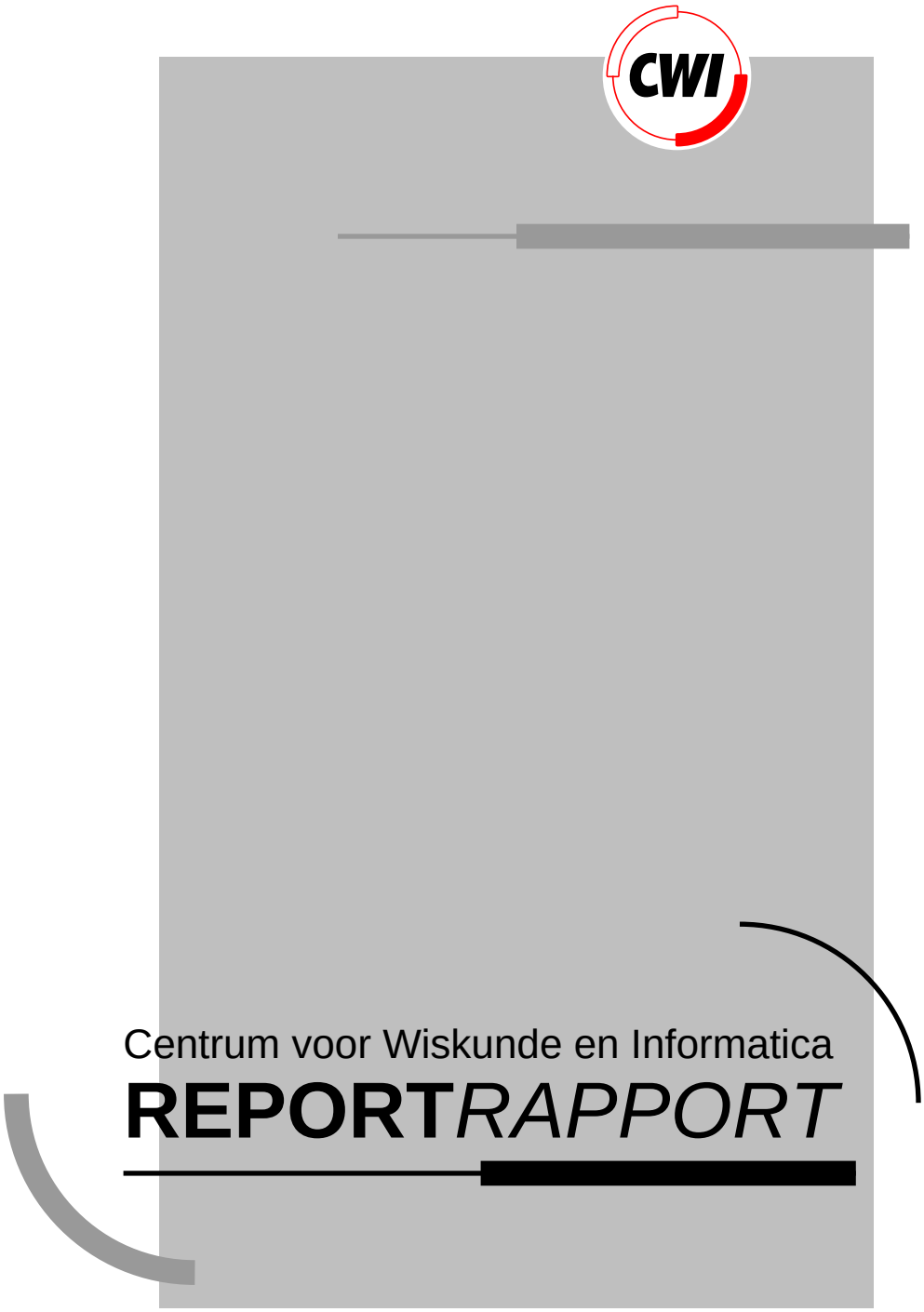

Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

Optimal Multi-Writer Multi-Reader Atomic Register

A. Israeli, A. Shaham

Computer Science/Department of Algorithmics and Architecture

CS-R9235 1992

Optimal Multi-Writer Multi-Reader Atomic Register

Amos Israeli
CWI
P.O. Box 4079, 1009 AB
Amsterdam, The Netherlands
AND
Dept. of Electrical Engineering
Technion — Israel

Amnon Shaham
CWI
P.O. Box 4079, 1009 AB
Amsterdam, The Netherlands
AND
Dept. of Computer Science
Technion — Israel

Abstract. This paper addresses the wide gap in space complexity of atomic, multi-writer, multi-reader register implementations. While the space complexity of all previous implementations is linear, the lower bounds are logarithmic. We present two implementations which close this gap: The first implementation uses multi-reader physical registers while the second uses single-reader physical registers. Both implementations are optimal with respect to the two most important complexity criteria: Their space complexity is *logarithmic* and their time complexity is *linear*.

1991 Mathematics Subject Classification: 68M10, 68Q22, 68Q25.
CR Categories: B.3.2, B.4.3, D.4.1, D.4.4.
Keywords and Phrases: Shared Register, Concurrent Reading and Writind, Atomicity, Multiwriter Register.
Note: This work is partially supported by NWO through NFI Project ALADDIN under Contract number NF 62-376. A preliminary version of this paper was presented in the *11th Annual ACM Symposium on Principles of Distributed Computing*, August 1992, Vancouver, Canada.

1 Introduction

At the most basic level of interprocessor communication, data is transferred via *registers* — memory devices which support *read* and *write* operations. Each register can store any element from its set of *permitted values*; it has a set of *writers* — processors that can write values in the register, and a set of *readers* — processors that can read values from the register. A write operation takes some permitted value as a parameter and stores it in the register; a read operation returns a (permitted) value stored in the register. The stored and returned values should satisfy some consistency guarantees

which depend on the type of the register. Lamport in [La86a, La86b] was the first to formalize the notion of a register. He defined three types of registers in terms of the consistency guarantees they provide: *safe, regular* and *atomic*. An *atomic register* supports read and write as atomic, indivisible, operations. Safe and regular registers provide inferior consistency guarantees which are not discussed in this paper. The execution of an operation is called an *action*. Under the global time model, which we assume throughout this paper, each action has a starting time and an ending time. The interval between the starting and ending times of an action a is called the *execution interval* of a. Each writer or reader executes actions in a serial manner, but we assume absolutely no synchronization among different processors, thus actions of distinct processors may be executed in overlapping time-periods.

In a situation in which some specific type of register is not available at the local hardware store, one may resort to *implementing* the required register using some available registers. For the user, such implementation is a black box which behaves exactly according to its specifications. Formally, an *implementation* of a *logical* register using a set of *physical* registers, consists of a hardware arrangement of the physical registers and two programs that are called the *writer* protocol and the *reader* protocol. Both programs are composed of operations of the physical registers (which are called *physical* operations) and constitute the operations of the logical register (which are called the *logical* operations). The set of processors is partitioned into logical writers and logical readers, that is, writers and readers of the logical register. For simplicity we assume that each processor is either a logical writer or a logical reader though in reality a processor can function as both.

An implementation is *correct* if the assumption that all physical registers provide their consistency guarantees implies that every execution of the logical register satisfies its consistency guarantees. For an *atomic* logical register this requirement materializes as follows: For every logical execution E, there exists an assignment of a serialization time for every logical action in E, that induces a serialization of all logical actions in E. Under this serialization every read action returns the value written by the write action which is the most recent preceding write. To avoid trivial solutions it is required that the reader and writer protocols are *wait-free* and that each action is serialized within its execution interval.

Throughout this paper w and r are used for the number of writers and readers, respectively, and $n = w + r$ is the total number of processors in the system. A register with w writers and r readers is denoted as a (w, r)-register. In [La86a, La86b] Lamport presented five implementations of various logical registers with a single writer. When some of these implementations are combined, they form an implementation of an atomic $(1, 1)$-register with any value set, from binary, safe, $(1, 1)$-registers. Several papers, motivated by the work of Lamport, studied the intriguing problem of implementing atomic, multi-writer, multi-reader registers. The simplest such implementation was presented by Vitanyi and Awerbuch in [VA86]. They implement an atomic, (w, r)-register, using atomic, $(1, 1)$-physical registers. In this implementation the physical registers are divided into two fields: a *value* field and a *label* field. The value field stores the last logical value that was written by the its owner [1], if the owner is a logical writer; if the owner is a logical reader then the value field holds the last value returned by the owner. The only operations which access the value field are copying a value to this field or from it. The label field stores all the coordination information needed for the implementation. We call implementations in which each register is divided into a value field and a label field and in which the only permitted operations accessing the value field are copy operations, *label based* implementations. We rename the label field to be called the *coordination* field and reserve the term *label* for the most basic coordination unit. In our implementations each coordination field consists of several labels. The complexity of a label-based implementation is measured by several criteria. The two most important criteria are:

1. **Space Complexity -** The maximal size of a coordination field of any physical register. (This criterion is often called *label-size*).

[1] The writer of every physical register is called its *owner*.

2. Time Complexity - The maximal number of physical actions executed during a single logical read or write operation.

In the [VA86] implementation labels are time-stamps. This causes its main drawback, namely: *unboundedness*. The actual size of a label in any logical action is logarithmic in the number of write actions performed prior to that action. The time complexity of this implementation is linear in n, the total number of processors.

Several researchers have devised bounded label-based implementations for atomic, multi-writer, multi-reader registers, using single-writer, multi-reader physical registers: The first implementation was proposed in [VA86] and was found to be erroneous. The second implementation was proposed by Peterson and Burns in [PB87] — this implementation has a bug which was discovered and corrected by Schaffer in [Sc89]. In this implementation the space complexity is $O(w)$ and the time complexity is $O(w^2)$. Israeli and Li, in [IL87], suggested *bounded time-stamps* as a bounded primitive to capture the temporal relationship among asynchronous processors. Using this method they devised an implementation which runs in linear time and with $O(n)$ space complexity. Other implementations with an inferior complexity are proposed by Abraham in [Ab91a, Ab91b]. The work of Li, Tromp and Vitanyi in [LTV90] presents an implementation using atomic $(1,1)$-physical registers. The space complexity is $O(n)$ and the time complexity is linear. Since the implementation of [LTV90] uses inferior physical registers its complexity is superior to all aforementioned implementations.

Thus far all proposed bounded concurrent implementations have a space complexity which is *linear* in the number of writers, and in some cases even in the total number of processors. Some explanation for this phenomena was suggested by Israeli and Li in [IL87] who defined the class of *Binary Comparison Protocols* (or in short *BCP*) as the class of label-based implementations in which the labels of every two processors can be compared to find the most recent label among the two. They showed that the space complexity of any *BCP* implementation is at least linear in the number of writers. Later Li and Vitanyi in [LV90] have pointed out that though the original [VA86] implementation is *BCP* as well as all bounded implementations, in principal an implementation of an atomic register does not *have to* be *BCP*. To demonstrate this, they presented a *sequential* implementation (in sequential implementations the *logical* actions are executed sequentially, without overlapping) with $O(\log w)$ space complexity. The sequential implementation of Li and Vitanyi uses the *id*s of the processes, or in other words, is not anonymous. For an anonymous implementation their method requires labels of size $2\log n$, since the processor's *id*s are added to the labels. Later it was proven by Cori and Sopena in [CS90] that an anonymous implementation for w writers should have at least $2w-1$ distinct labels. They also devised a sequential register with exactly $2w-1$ labels which improved the space complexity of the sequential [LV90] implementation by a constant. Recently it was proven by Tromp in [T92] that the sum of sizes of coordination fields in nonanonymous implementations is at least $n\log n$ which translates to $\log n$ size coordination fields assuming that the size of all coordination fields is the same.

These results leave an *exponential* gap between the lower and upper bounds on the space complexity of atomic register implementation. While the lower bounds take into account only to the combinatorial properties of keeping track of the last value (and therefore hold for sequential implementations as well), an actual *concurrent* implementation should also deal with concurrency problems; all existing implementations require *linear* space to do that. The significance of this gap is further emphasized when one takes a closer look at the *unbounded* implementation of Vitanyi and Awerbuch in [VA86]. For polynomial length executions the space complexity of this implementation is *logarithmic*, A linear space complexity is reached only in executions of exponential length. In other words: The bounded protocols supersede the unbounded protocol only in *exponentially* long executions. In polynomial length executions, which are often viewed as a better model for real-life situations, ensuring the theoretical boundedness requires an *exponential* overhead.

The natural question arising here is: *Is this cost necessary?* The answer is *negative* as we show

by presenting two bounded, concurrent, label-based implementations for atomic, multi-writer, multi-reader register, with *logarithmic* space complexity. The first implementation uses atomic, $(1,n)$-physical registers [2]. The second implementation uses atomic, $(1,1)$-physical registers. These implementations are the first to break the linear space barrier for atomic multi-writer registers; moreover, by the lower bound proven by Cori and Sopena in [CS90] and Tromp in [T92], the logarithmic space complexity is optimal. The time complexity of both implementations is linear, which is obviously optimal. Both implementations are *self-stabilizing*: Regardless of the system's *initial* state, it eventually reaches a *legitimate* state — a state which can be reached in a legally initialized system. In such an arbitrary initial state the processors may be in arbitrary states (e.g. in the middle of some logical action) and the physical registers may hold arbitrary values.

To represent temporal relations among protocol executions we use precedence graphs. These graphs which were introduced by Israeli and Li in [IL87], are time-dependent graphs whose nodes and edges at any given time are determined by the labels stored in the processors' registers at that time. Label ℓ_1 precedes ℓ_2 if there is a directed edge (ℓ_2, ℓ_1) in the precedence graph. Following [ILV87] we use *dynamic precedence trees* in which the outdegree of every node is at most 1; each label points to at most one other preceding label. At any given time the precedence graph is a forest of *intrees* — trees whose edges are directed towards the root. For each individual label *lb* the set of labels whose temporal relationship with *lb* can be found by direct comparison includes the labels whose edges point to *lb*, and the single label to which *lb*'s edge points. Hence our protocols are indeed not BCP. Each path of a precedence intree is ordered temporally but labels on distinct paths are in general not comparable. The paths of any precedence intree are ordered lexicographically, by the *id*s of (the processors which generated) their nodes. The most preceded path in the precedence forest is called the *frontal branch* of the forest. Roughly speaking the frontal branch consists of nodes which are generated by recent write actions and the last node on this branch is the last serialized write action.

The rest of this paper is organized as follows: In Section 2 we formally define the model of computation and the implementation problem. In Section 3 we explain the data structure used by our protocols and present a sequential implementation. The sequential implementation serves as an exposition for the ideas which are later used in the concurrent implementations. The $(1,n)$ and the $(1,1)$ implementations are presented in Section 4 and Section 5 respectively.

2 The model

In this section we define the model of computation and the atomic register implementation problem.

A system consists of a set of processing entities called *processors*, and a set of memory entities called *registers*. Processors access registers by executing *read* and *write operations*. Each operation is associated with a processor that can *execute* the operation and with a register which is *accessed* by the operation. Each register has a set of *permitted values*, a set of *writer* processors and a set of *reader* processors. A *write* operation to register REG is executed by some writer of REG, it gets a permitted value of REG as an input parameter and stores the value in REG. Analogously, a *read* operation from register REG is executed by some reader of REG, it retrieves the (permitted) value stored in REG and returns it as an output parameter. Processors are defined by their *programs* whose building blocks are *instructions*; each instruction starts with a number of internal computations which is succeeded by at most one operation. Each program has a distinguished instruction called the program's *initial instruction*. Each register has a distinguished value called the register's *initial value*.

The execution of an operation is called an *action*. Under the global time model, each action has a starting time and an ending time. The time interval between the starting and ending times of action a is the *execution interval* of a. Actions whose execution intervals are overlapping are called

[2] Since we assume that a writer always "knows" the values it writes we count it among the readers of its register.

overlapping actions. Each processor is assumed to have a *program counter* which at any time points to the next instruction to be executed by the processor. When the system is initialized all program counters point to the processors' respective initial instructions, and each register holds its initial value. The actions of a processor are determined by its program and by the values returned by earlier read actions; since processors are *serial* entities the actions of a processor never overlap. Thus, an *execution* of processor P is a *sequence* of non-overlapping actions of P, with their starting and ending times and their corresponding values.

Let REG be some register. A set A of REG's actions can be *serialized*, if every action a, $a \in A$, can be assigned a distinct point in time which lies within its execution interval, called the *serialization time* of a, such that the following *serialization condition* is satisfied:

Any read action returns the value of the write action last serialized before it.

A set of actions that access REG, is an *execution* of REG, if the set satisfies its *consistency guarantee.* If REG is atomic then its consistency guarantees state that each of its executions can be serialized. A sequence of actions E is a *system execution* if for every processor P, the set of actions of P in E is an execution of P, and for every register REG, the set of actions of REG in E is an execution of REG. Those executions are called the induced subexecutions of P and REG, respectively, under E. Two executions of processor P, $a_1, a_2, \ldots$ and $b_1, b_2, \ldots$ are *equivalent*, if for all i, $i > 0$ a_i and b_i are executions of the same operation o_i. If o_i is a write operation then the written values are equal; if o_i is a read operation then the read values are equal; thus the only differences allowed between the two executions are in the starting and ending time of their actions. Two system executions E_1 and E_2 are equivalent if for every processor P and for every register REG, the induced subexecution of P and REG under E_1 and under E_2 are equivalent.

An *implementation* of a *logical* register is a system whose registers are called *physical* registers. The operations accessing the physical registers are called *physical operations.* A *logical operation* (either read or write) is defined by a program, called *protocol*, whose building blocks are physical operations. The set of processors is partitioned into *writers* and *readers.* The program of (all) writers is called the *writer protocol* and the program of (all) readers is called the *reader protocol.* The writer protocol has one input parameter which is the value that should be stored in the logical register. The reader protocol should be exited through the *return* instruction. The return instruction does not contain any physical operation but rather returns a value which is the result of the logical read action. In label-based implementations the only instructions that access the value field of a register are copying a permitted value from this field to a local value-variable or from a local value-variable to that field. No other operations that access the value variables are allowed. In label-based implementations the input parameter to the writer protocol is not specified explicitly and is assumed to be the value corresponding to the label generated by the protocol. The return instruction returns a label whose corresponding value, which is not specified explicitly, is the returned logical value.

A *logical action* a is an execution of a protocol. Such a logical action is a sequence of physical actions called the *physical actions* of a. The starting time of a is the starting time of the first physical action of a. The ending time of a is the ending time of the last physical action of a. A program is *wait-free* if all its executions consist of a bounded number of physical actions, where the bound may depend on the number of processors in the system. To avoid implementations which use mutual-exclusion techniques, that almost eliminate the system's concurrency, we require that the logical operations are wait-free. A *logical execution* is a set of logical actions of the system's processors. Every logical execution E *induces* a *corresponding* physical execution which contains all the physical actions of the logical actions of E. An implementation is *atomic* if all its logical executions are serializable. An implementation is *sequential* if every sequential logical execution (in which no two logical actions overlap) is serializable.

In systems all of whose registers are atomic, every system execution is equivalent to (at least one) sequential execution in which every action a is serialized at a distinct point in time called the *occurrence*

time of a. When such a system implements a logical register every logical action is equivalent to a sequence of instantaneous physical actions where sequences that correspond to overlapping logical actions might be interleaved. Since every system execution is equivalent to some serial execution, any property which is correct with respect to all serial executions, is correct with respect to all executions. To prove the correctness of such an implementation, we assume that every processor continuously executes logical operations. A *schedule* is a sequence of processor names. Every schedule s naturally induces a serial execution in which processors execute (physical) actions in the order in which they appear in s. Since our implementations are label-based, the execution induced by s does not depend on the logical values written and read, but only on the schedule s. The correctness of the implementation is proven by showing that for every schedule s, the logical execution corresponding to the physical execution induced by s, is serializable.

3 The Precedence Trees

The three implementations presented in this paper are label-based. Temporal relations among logical actions and among the labels corresponding to these logical actions are represented by use of *precedence graphs* which were originally proposed by [IL87]. These are time dependent directed graphs whose nodes and edges are encoded by the labels generated during the processors' actions. The semantics of the precedence graph is: If there is an edge from label ℓ_1 to ℓ_2 then the action that generates ℓ_1 is serialized after the action that generates ℓ_2. We use *dynamic precedence trees* which are similar to those used in [ILV87]. In this section we describe the structure of the precedence trees used by all the implementations, and outline a simple sequential implementation of a multi-writer, multi-reader, register using single-writer, multi-reader registers. The sequential implementation does not improve upon the complexity of previously known sequential implementations. It is brought here as an exposition for the ideas which enable the concurrent implementations.

3.1 Data Structure

The writers of the implemented logical registers are denoted by $\mathcal{W}_1, \mathcal{W}_2, \ldots, \mathcal{W}_w$. Execution number a of the writer protocol by $\mathcal{W}_i$ is denoted by L_i^a. Each individual execution of the writer protocol, L_i^a, *generates* a single label which is denoted by ℓ_i^a; i and a are called the *id* and the index of L_i^a (and of ℓ_i^a), respectively. Label ℓ_i^a is stored in the register of $\mathcal{W}_i$, (or in its registers in the $(1,1)$ implementation) at the end of L_i^a.

All our protocols share an identical structure of the labels which is described below: Each label encodes a node and a potential edge emanating from that node, where the outgoing edge is directed either towards the label itself to form a self loop, or towards a node encoded by another label. For convenience we identify the label ℓ_i^a with its node and denote the node by ℓ_i^a as well. Node ℓ_i^a is specified by the number i, which is called the *id* of the node, and by its *address* which is an integer. Since the *id* of all nodes of $\mathcal{W}_i$ is i, the *id* of a node is omitted from its encoding in ℓ_i^a, hence the node of ℓ_i^a is encoded by the *address* field which is denoted by $\ell_i^a.address$. The edge of ℓ_i^a is encoded by the *edge* field which is denoted by $\ell_i^a.edge$. The edge field stores the *id* and *address* of the node to which the potential edge is directed, in two subfields, which are denoted by $\ell_i^a.edge.id$ and $\ell_i^a.edge.address$, respectively. The potential edge emanating from ℓ_i^a *exists* in some precedence graph G if for some label ℓ_j^b in G, it holds that $\ell_i^a.edge = (j, \ell_j^b.address)$. In case this equality does not hold for any node (label) in G, there is no edge outgoing from ℓ_i^a in G. Our protocols make sure that in any precedence graph the aforementioned equality holds for at most one label, hence each label ℓ_i^a encodes at most one outgoing edge which is denoted by $e_i^a = (\ell_i^a, \ell_j^b)$, where ℓ_j^b is the node towards which e_i^a is directed.

We require that $\ell_i^a.edge.id \leq i$, thus the only type of cycles that we permit are self loops. Consequently the nodes (labels) on each directed path of the precedence graph are ordered in increasing order

of their *id*s from the root to the leaves. The writers' *id*'s lie between 1 and w. In case $\ell_i^a.edge.id = 0$ we say that e_i^a exists and is directed to the *virtual root label* ℓ_0^0. Since the outdegree of each node is at most 1, every precedence graph is a forest of labeled intrees — trees whose edges are directed towards the root. From now on we assume that the node set of each precedence graph includes the root label ℓ_0^0. An edge (ℓ_j^b, ℓ_i^a) reflects the fact that L_j^b is serialized after L_i^a. If two edges (ℓ_j^b, ℓ_i^a) and (ℓ_k^c, ℓ_i^a) enter the same node ℓ_i^a, then the serialization of actions L_j^b and L_k^c cannot be determined by (the precedence relation induced by) the edges of the precedence tree. In this case we serialize these actions by an ascending order of their *id*s. (higher *id*s are serialized *before* lower *id*s.) Later we define and use the *history graph* of an execution whose nodes are all labels generated during the execution. In this graph it is necessary to serialize labels with equal *id*s whose edges enter the same node. Since the actions of each individual processor are temporally ordered by their indices, a label with a lower index is serialized before a label with a higher index. These requirements are formally accommodated by the following lexicographic ordering of labels: Let ℓ_i^a and ℓ_j^b be two labels; ℓ_i^a *locally precedes* ℓ_j^b if either $j < i$ or if $i = j$ and $a < b$. Though the *locally precedes* relation is defined for any pair of distinct labels, it reflects a precedence relation *only* among labels whose edges enter the same node in the precedence graph.

Let G be some precedence graph. The *frontal branch* of G is a path which is defined recursively as follows: The first node in the frontal branch is the virtual root label ℓ_0^0; the second node of the frontal branch of G is a node whose edge enters the root and which is *locally preceded* by all other nodes in G whose edges enter ℓ_0^0. In general if α is a prefix of the frontal branch of G whose last node is ℓ_i^a then the next node in the frontal branch is the label whose edge enters ℓ_i^a and which is locally preceded by all other labels whose edges enter ℓ_i^a. The last node in the frontal branch of G (whose indegree is 0), is called the *last* node of G. Let $e_1 = (\ell_i^a, \ell_k^c)$ and $e_2 = (\ell_j^b, \ell_k^c)$ be two edges in some precedence graph G, such that e_1 belongs to the frontal branch of G (that is ℓ_j^b locally precedes ℓ_i^a). In this case we say that e_1 *excludes* e_2 from the frontal branch of G. A pictorial description of a precedence tree appears in Figure 1, where the edges of the frontal branch appear as solid arrows and the rest of the edges appear as dotted arrows. The basic idea in all the implementations is: *At any time t the labels stored*

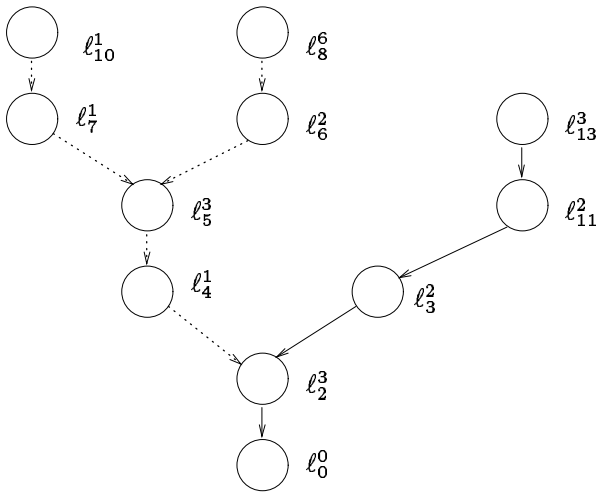

FIGURE 1. A precedence intree

in the registers encode a precedence graph whose last node belongs to the last write action completed until t.

3.2 Sequential Implementation

In the sequential implementation each writer writes in a $(1,n)$-register called R_i, which can be read by all processors, writers and readers. (Since the implementation is sequential we need not require that the registers are atomic.) The structure of R_i is: $R_i = (value, current)$. Since the value field is not used by the protocol we assume from now on that each label is written with the corresponding value and omit the values from the protocol's description. In this implementation readers do not write. The *current* labels at time t are the labels stored in the processors' registers at time t. The *collect* operation consists of reading the labels of all writers and computing the precedence graph. This precedence graph is called the *current graph*, its nodes are the root and all the current labels, that is $w+1$ nodes.

Let α be a path in a precedence tree. The *i-prefix* of α, denoted by α/i, is the prefix of α which contains all nodes whose *id* is less then i. The protocol for $\mathcal{W}_i$, $1 \leq i \leq w$, works as follows: $\mathcal{W}_i$ collects the labels of all writers, computes the frontal branch and chooses a new label which replaces its previous active label. The edge of the new label is directed towards the last node in the i-prefix of the frontal branch of the current graph. The new edge excludes the rest of the previous frontal branch from the new frontal branch. At the same time $\mathcal{W}_i$ invalidates all edges which are directed towards its previous label, by choosing a new address which is not the address of the *edge* field of any writer. The reader protocol is simply to collect the current graph and to return its last node. The register of $\mathcal{W}_i$, R_i, is initialized to $((i,0),0)$. At the initial state all edges of the current graph are self-loops and its initial frontal branch contains only the virtual root label ℓ_0^0. The initial logical value is the value corresponding to ℓ_0^0 which can be chosen freely from the permitted values of the logical register.

Correctness of the sequential implementation is straight forward and is left to the reader. In order to allow a writer to invalidate (at most) $w-1$ edges of the precedence tree, it should have at least w possible addresses. Therefore for each writer, the size of the *edge* field is $2\log w$, the size of the address field is $\log w$. The protocol's space complexity which in this case is equal to the size of one label is therefore $3\log w$. Every execution of the writer protocol consists of $w+1$ physical actions assuming that each writer reads its own register; every execution of the reader protocol consists of w actions, hence the time-complexity of the seqential implementation is $O(w)$.

4 Multi-Writer out of Multi-Reader Registers

In this section we present a concurrent implementation of a (w,r)-atomic register using physical $(1,n)$-atomic registers. In this implementation communication is again one-sided, readers do not write. An important design decision in this implementation is to serialize all logical write actions *independently* of the scheduling of the logical read actions. Serialization of the logical read actions is then done in accordance with the serialization of the write actions. Therefore this section is divided into two subsections: The first subsection presents the writer protocol and the serialization scheme for the logical write actions, and continues with a proof of the correctness of this serialization scheme. The second subsection presents the reader protocol and the serialization scheme for the logical read actions, and then proceeds to prove the correctness of the serialization scheme for the read actions. Together they imply the correctness of the entire implementation.

4.1 The Writer Protocol

4.1.1 Description

The writer protocol is obtained by adjusting the sequential writer protocol to the concurrent environment while keeping its basic ideas and data structure. The structure of the labels in this implementation is identical to the structure of labels in the sequential implementation and once more they

encode a labeled intree as a precedence graph whose last node corresponds to the last completed write action. In a concurrent environment however, a writer cannot simply collect a precedence graph and replace its current label by a new label which becomes the last in the frontal branch of the collected graph. If the protocol were to work in this fashion a new label ℓ_i^a might be *pointed at* by a label of another write action which is completed at an earlier time, immediately when ℓ_i^a is written. In this way the precedence graph may not reflect the temporal order of actions. This problem does not rise in the sequential implementation which supports only non-overlapping logical actions. To overcome this problem the coordination field of a writer's register in this implementation consists of two labels called *new* and *current*. During L_i^a, the *new* field holds a tentative choice for ℓ_i^a, the next label of $\mathcal{W}_i$. The final ℓ_i^a is written in the *current* field in the last physical action of L_i^a. The details of this mechanism are explained below. The structure of R_i, the register of $\mathcal{W}_i$ is: $R_i = (new, current)$, and the space complexity is equal to the size of two labels rather than one.

Similar to the sequential implementation the register of $\mathcal{W}_i$, R_i, is initialized to $((i,0),0,(i,0),0)$. That is, at the initial state all edges of the current graph are self loops, its initial frontal branch contains only the root label ℓ_0^0 and the initial logical value is the value corresponding to ℓ_0^0, which can be chosen freely from the set of all permitted values. Let ℓ_i^a and ℓ_j^b be two labels, the fact that ℓ_i^a and ℓ_j^b are actually the same label, i.e. $i = j$ and $a = b$, is denoted by $\ell_i^a \equiv \ell_j^b$. Since the processors cannot compute the relation $\equiv$, the protocol uses the relation $\ell_j^b \simeq \ell_j^c$ which denotes the fact that $\ell_j^b.edge = \ell_j^c.edge$ and $\ell_j^b.address = \ell_j^c.address$.

begin

Collect G_i	$r_i^a[1], \ldots, r_i^a[w]$
$last :=$ *id* of the last node of B_i/i	
$new.address :=$ select (G_i)	
$new.edge := (last, \ell_{last}.address)$	
$R_i :=$ **write** $(new, current)$	d_i^a
$n\ell_{last} :=$ **read** $R_{last}.current$	$\tilde{r}_i^a[last]$
if $\ell_{last} \simeq n\ell_{last}$	
then	
/* *connect:* Direct e_i^a towards *last* */	
$current := new$	
else	
/* *loop:* Direct e_i^a toward ℓ_i^a */	
$current.edge := (i, new.address)$	
$current.address := new.address$	
endif	
$R_i :=$ **write** $(new, current)$	w_i^a

end

FIGURE 2. The protocol for $\mathcal{W}_i$ in the $(1,n)$ implementation

The writer protocol appears in Figure 2. We now describe the protocol assuming that L_i^a is executed: First $\mathcal{W}_i$ executes the procedure *collect* in which it reads the registers of all other writers and computes a graph, denoted by G_i^a. The labels of G_i^a are those read from the *current* fields of all registers. Execution of *collect* takes w atomic physical actions which are denoted by $r_i^a[1]$ ... $r_i^a[w]$. During collection of G_i^a some writers may change their labels, therefore G_i^a is not necessarily equal to the current graph at any time during its collection. The frontal branch of G_i^a is denoted by B_i^a. The node with the maximal *id* in B_i^a/i is the *last* node of B_i^a/i. Let ℓ_{last} be the last node of B_i^a/i. After G_i^a is

collected, $\mathcal{W}_i$ chooses a tentative *new* label whose edge is directed towards ℓ_{last} and whose address is obtained by the function *select*; this function ensures that the *new* label is not the head of any (existing or potential) edge encoded by any *current* or *new* label read during L_i^a. In this way all edges entering ℓ_i^{a-1} are invalidated. In addition the function *select* ensures that $new.address \neq \ell_i^{a-1}.address$ even if no other edge is directed towards ℓ_i^{a-1}; thus a writer never uses the same address twice in a row. The chosen label is declared by $\mathcal{W}_i$ by writing it into the *new* field of R_i while the *current* label is not changed. The declaring write action is denoted by d_i^a. Following d_i^a, $\mathcal{W}_i$ rereads the register of $\mathcal{W}_{last}$, the writer whose label is *last* in B_i^a/i. This second read action is denoted by $\tilde{r}_i^a[last]$. The label read in action $\tilde{r}_i^a[last]$ is called the *target* label of L_i^a. The logical write action L_i^a is concluded by a final physical write action which is denoted by w_i^a. In this action $\mathcal{W}_i$ replaces its *current* label as follows: Let ℓ_{last} and $n\ell_{last}$ be the two labels read by $\mathcal{W}_i$ in actions $r_i^a[last]$ and $\tilde{r}_i^a[last]$ respectively. If $\ell_{last} \simeq n\ell_{last}$ then *new* label is assigned to *current*, in this case we say that L_i^a *connects*. If however $\ell_{last} \not\simeq n\ell_{last}$ then *current* is chosen such that its edge is a self-loop directed towards ℓ_i^a itself. In this case we say that L_i^a *loops*.

The function *select* considers at most $2w-1$ addresses, thus $2w$ possible addresses are needed in order to ensure that *select* can always find an unused address. Therefore, the size of the *edge* field in each label is is $\log w + O(1)$ and the size of the address field is also $\log w + O(1)$. The protocol's space complexity which in this case is equal to the size of two labels is therefore $O(\log w)$. Every execution of the writer protocol consists of $w+2$ physical actions assuming that each writer reads its own register; every execution of the reader protocol consists of $3w$ actions, hence the time-complexity of this implementation is $O(w)$.

4.1.2 Serialization Scheme for Logical Write Actions

The atomicity of an implementation is proved by showing that for every sequential physical execution, the induced logical execution can be serialized. A sequential execution is entirely determined by its schedule; the complete asynchrony of the system means that the only points in time which can be used to serialize a logical action, are the occurrence times of the physical actions, and whenever some flexibility is possible, the intervals between these occurrence times. For this reason we sometimes use the name of an action to denote its occurrence time. Whenever we say that some property p holds at action a we mean that p holds right after the occurrence time of a. In the following subsections we fix some arbitrary sequential physical execution with respect to which all definitions and proofs are being made, since this fixed execution is arbitrary, the results hold for every system execution of the implementation. The logical write actions are serialized by an explicit assignment of *serialization time* for each action. This assignment is done using a *History graph* — a precedence graph whose structure reflects the execution of the system. The history graph which is not computable by the processors plays a key-role in the correctness proof of our protocols.

DEFINITION 1: Let E be an execution of the system. Let τ be the minimal time interval between any two successive physical actions in E. The *History graph* of E, H, is a time dependent graph which is constructed incrementally during the execution, as follows:

1. H^0 is the *History graph* at time 0 (before the execution begins). It contains only the *root* node — ℓ_0^0.

2. Let L_i^a be an arbitrary logical write action. Let t be the time when w_i^a occurs, The graph H^t is obtained from $H^{t-\tau}$ by adding the node, ℓ_i^a, and the directed edge e_i^a as follows: If L_i^a connects then e_i^a is directed towards the target label of L_i^a, otherwise (L_i^a loops) e_i^a is a self-loop. According to our convention of using an action's name to denote its occurrence time, H^t is also denoted by $H^{w_i^a}$.

At any time t, the history graph H^t is a precedence forest which consists of a single precedence tree and some disjoint self-loops. The frontal branch of H^t is denoted by B_H^t. In order to serialize the logical actions we first partition the set of logical write actions into two subsets, *good* and *bad*, where a good write action is a write action whose label is last in B_H^t at the time t that it joins the history graph.

DEFINITION 2: Let L_i^a be an arbitrary logical write action Action. L_i^a is *good* if ℓ_i^a is *last* in $B_H^{w_i^a}$. A logical write action which is not good, is *bad*.

Obviously if L_i^a loops then it is bad, but there are many cases in which L_i^a connects and it is also bad. Using the partition of logical write actions to good and bad we assign a serialization time to every logical write action. The serialization time establishes a total order on the set of logical write actions:

DEFINITION 3: Let t be the occurrence time of w_i^a. Define the *serialization time* of L_i^a as follows:

1. If L_i^a is good, that is, ℓ_i^a is the last node of B_H^t, then L_i^a is serialized at t.
2. If L_i^a is bad and ℓ_j^b is the last node of B_H^t then L_i^a is serialized before L_j^b and after any other physical action which precedes w_j^b. In this case we say that L_i^a is *serialized by* L_j^b. In case two bad write actions, L_i^a and L_j^b, are serialized by the same good write action L_k^c, L_i^a and L_j^b are serialized by an ascending order of their *id*s.

Let t be the occurrence time of an arbitrary physical action. Under the defined serialization time the last node in B_H^t is the most recent write action, that is the value of the logical register.

4.1.3 Correctness of the Serialization Scheme for Logical Write Actions

In the correctness proof we have to prove that the serialization time satisfies the serialization requirements for atomic registers. In Theorem 9 we prove that every logical write action is serialized within its execution interval. In Theorem 11 we prove that all graphs collected by the writers are precedence graphs. We start the correctness proof with some technical lemmas.

LEMMA 1: Let ℓ_i^a be a node in H^t. If $\ell_i^a \notin B_H^t$, then for every $t' > t$, $\ell_i^a \notin B_H^{t'}$.

Proof: If L_i^a loops then ℓ_i^a is not connected to the root at any time $t' > t$. Assume that L_i^a connects and denote the path in H^t from ℓ_i^a to the root by α. Since $\ell_i^a \notin B_H^t$, let (ℓ_k^c, ℓ_j^b) be the first edge which is in α but not in B_H^t and let (ℓ_ℓ^d, ℓ_j^b) be the edge excluding (ℓ_k^c, ℓ_j^b) from B_H^t. No edge of H is ever deleted, therefore at any time $t' > t$, (ℓ_ℓ^d, ℓ_j^b) excludes the suffix of α, and in particular ℓ_i^a, from $B_H^{t'}$. □

Let a and b be two physical atomic actions; the fact that a occurs before b is denoted by $a \rightarrow b$.

LEMMA 2: If (ℓ_i^a, ℓ_j^b), $i \neq j$, is an edge in H then w_j^b occurs before w_i^a (i.e. $w_j^b \rightarrow w_i^a$).

Proof: According to the definition of the history graph, (ℓ_i^a, ℓ_j^b), $i \neq j$, is an edge in H only if the *id* of the last node of B_i^a/i is equal to j, and the label read in $\tilde{r}_i^a[last]$ is ℓ_j^b. Therefore $w_j^b \rightarrow \tilde{r}_i^a[j]$. Since $\tilde{r}_i^a[j] \rightarrow w_i^a$, we get $w_j^b \rightarrow w_i^a$. □

LEMMA 3: If $\ell_i^a \in B_H^t$ then L_i^a is good.

Proof: By Lemma 1 $\ell_i^a \in B_H^{w_i^a}$. Lemma 2 implies that for any edge (ℓ_j^b, ℓ_i^a) in H, $w_i^a \rightarrow w_j^b$. Hence the indegree of ℓ_i^a in $H^{w_i^a}$ is 0, and ℓ_i^a is last in $B_H^{w_i^a}$. By definition L_i^a is good. □

LEMMA 4:

(a) The sequence of node *ids* along any path of the history graph from the leaf towards the root is strictly decreasing.

(b) Every path of the *history graph* contains at most one label of every writer.

Proof: **(a)** According to the protocol e_i^a is directed towards the last node of B_i^a/i. Since B_i^a/i contains only nodes whose $id < i$, the proof follows. **(b)** is implied immediately by **(a)**. □

Due to concurrency it is not guaranteed that the edges of a graph collected by any writer or reader belong to the history graph. A weaker relationship between the edges of the collected graphs and the edges of the history graph is depicted in the next lemma:

LEMMA 5: If (ℓ_i^a, ℓ_j^b) is an edge in G_k^c then there exists an integer r, $r \geq 0$, such that (ℓ_i^a, ℓ_j^{b+r}) is an edge in H.

Proof: The lemma holds trivially if $i = j$. Assume $i > j$, by the definition of the history graph, there exists some integer r such that (ℓ_i^a, ℓ_j^{b+r}) is an edge in H, and ℓ_j^{b+r} is the current label of $\mathcal{W}_j$ at $\tilde{r}_i^a[j]$. To prove the lemma we have to show that $r \geq 0$. Assume by way of contradiction that $r < 0$. Since (ℓ_i^a, ℓ_j^b) and (ℓ_i^a, ℓ_j^{b+r}) are both edges of some precedence graph, it holds that $\ell_j^b.address = \ell_j^{b+r}.address$, thus if $r < 0$ then $r < -1$, because a writer never uses the same address twice in a row. Therefore $w_j^{b+r} \rightarrow w_j^{b-1} \rightarrow r_j^b[i]$. Consider actions d_i^a and $r_j^b[i]$: In d_i^a, $\mathcal{W}_i$ declares that e_i^a may be directed towards the *current* label of $\mathcal{W}_j$ read in $r_i^a[j]$, and in $r_j^b[i]$, $\mathcal{W}_j$ reads the register of $\mathcal{W}_i$. We first show that the case $r_j^b[i] \rightarrow d_i^a$ is impossible. In this case we get that $w_j^{b+r} \rightarrow w_j^{b-1} \rightarrow r_j^b[i] \rightarrow d_i^a \rightarrow \tilde{r}_i^a[j]$, and in particular $w_j^{b+r} \rightarrow w_j^{b-1} \rightarrow \tilde{r}_i^a[j]$. Therefore ℓ_j^{b+r} is not the current label of $\mathcal{W}_j$ when $\tilde{r}_i^a[j]$ occurs, and the edge (ℓ_i^a, ℓ_j^{b+r}) does not belong to H, contradiction.

The proof is completed by showing that the case $d_i^a \rightarrow r_j^b[i]$ is also impossible. First we prove that under this assumption, either the *new* label or the *current* label read by $\mathcal{W}_j$ in action $r_j^b[i]$, is ℓ_i^a. By our assumption $d_i^a \rightarrow r_j^b[i]$. By the protocol $r_j^b[i] \rightarrow w_j^b$. Since ℓ_j^b is a node in G_k^c it holds that $w_j^b \rightarrow r_k^c[j]$. Since all graphs and in particular G_k^c are collected in ascending order we get that $r_k^c[j] \rightarrow r_k^c[i]$. Thus we get that $d_i^a \rightarrow r_j^b[i] \rightarrow r_k^c[i]$. Since $\ell_i^a \in G_k^c$, ℓ_i^a is the *current* label of $\mathcal{W}_i$ at $r_k^c[i]$. Therefore during the interval $[d_i^a\ ,\ r_k^c[i]]$, ℓ_i^a appears either as the *new* label of $\mathcal{W}_i$ or as its *current* label. In this case the definition of the function *select* implies that the *new.address*, selected during L_j^b satisfies $\ell_j^b.address \neq \ell_i^a.edge.address$. On the other hand the fact that (ℓ_i^a, ℓ_j^b) is an edge of G_k^c, implies that $\ell_i^a.edge.address = \ell_j^b.address$, a contradiction. □

The fact that every label $\ell_j^b \in B_H^t$ is in B_i^a and every label $\ell_k^c \in B_i^a$ is in B_H^t, is denoted by $B_H^t \equiv B_i^a$. In the next lemma we prove that if ℓ_i^a belongs to the frontal branch of the history graph at some arbitrary time t then ℓ_i^a is the current label of $\mathcal{W}_i$ at t.

LEMMA 6: If $\ell_i^a \in B_H^t$ for some time t, then ℓ_i^a is the current label of $\mathcal{W}_i$ at t.

Proof: Assume by way of contradiction that there are labels which do not satisfy the lemma. Let ℓ_i^a be the label with the lowest *id* among these labels, that is, there exists some time t_0, such that w_i^{a+1} occurs before t_0, while $\ell_i^a \in B_H^{t_0}$. Under these conditions Lemma 1 implies that $\ell_i^a \in B_H^{w_i^{a+1}}$. By Lemma 4(b), the *ids* along any path of the history graph are distinct, therefore $\ell_i^{a+1} \notin B_H^{w_i^{a+1}}$. Let ℓ_m^e be the last node of B_i^{a+1}/i. Since $\ell_i^{a+1} \notin B_H^{w_i^{a+1}}$, ℓ_m^e is not the last node of $B_H^{w_i^{a+1}}/i$ and therefore $B_H^{w_i^{a+1}}/i \not\equiv B_i^{a+1}/i$.

Now we show that $B_H^{w_i^{a+1}}/i$ is a subgraph of G_i^{a+1}, that is, every label of $B_H^{w_i^{a+1}}/i$ is also a label of G_i^{a+1}. Let ℓ_m^e, $m < i$, be an arbitrary label in $B_H^{w_i^{a+1}}/i$. Since $\ell_i^a \in B_H^{w_i^{a+1}}$, Lemma 2 implies that $w_m^e \rightarrow w_i^a$ and ℓ_m^e is on the path from ℓ_i^a to the root. Therefore, for any t during the interval $[w_i^a\ ,\ w_i^{a+1}]$ it holds that $\ell_m^e \in B_H^t$. Since $w_i^a \rightarrow r_i^{a+1}[m] \rightarrow w_i^{a+1}$ we get that $\ell_m^e \in B_H^{r_i^{a+1}[m]}$. By the minimality of i, ℓ_m^e is the *current* label of $\mathcal{W}_m$ at $r_i^{a+1}[m]$, and therefore $\ell_m^e \in G_i^{a+1}$.

Let ℓ_j^b, $j < i$, be the last node in the common prefix of B_i^{a+1}/i and $B_H^{w_i^{a+1}}/i$. Such a node always exists because the root, ℓ_0^0, belongs to both branches. Now we show that ℓ_j^b is not last in B_i^{a+1}/i. Assume by way of contradiction, that ℓ_j^b is last in B_i^{a+1}/i. In this case, a node with *id* smaller then i and with an edge directed towards ℓ_j^b does not exist in G_i^{a+1}. Since $B_H^{w_i^{a+1}}/i$ is a subgraph of G_i^{a+1}, there is no incoming edge to ℓ_j^b in $B_H^{w_i^{a+1}}/i$, that is ℓ_j^b is last in $B_H^{w_i^{a+1}}/i$. Since ℓ_j^b is the last node in the common prefix of B_i^{a+1}/i and $B_H^{w_i^{a+1}}/i$ and since ℓ_j^b is last in B_i^{a+1}/i and in $B_H^{w_i^{a+1}}/i$ we get that $B_i^{a+1}/i \equiv B_H^{w_i^{a+1}}/i$, a contradiction.

Since ℓ_j^b is not last in B_i^{a+1}/i, let (ℓ_k^c, ℓ_j^b), $j < k < i$, be the edge incoming to ℓ_j^b in B_i^{a+1}/i. Since ℓ_j^b is the last node in the common prefix of B_i^{a+1}/i and $B_H^{w_i^{a+1}}/i$, $\ell_k^c \notin B_H^{w_i^{a+1}}/i$. Now we show that (ℓ_k^c, ℓ_j^b) is an edge in $H^{w_i^{a+1}}$. By Lemma 5 there exists some r, $r \geq 0$ such that (ℓ_k^c, ℓ_j^{b+r}) is an edge in $H^{w_i^{a+1}}$. Since $\ell_j^b \in B_H^{w_i^{a+1}}$, the minimality of i implies that ℓ_j^b is the current label of $\mathcal{W}_j$ at w_i^{a+1}. Therefore $r = 0$, which implies that (ℓ_k^c, ℓ_j^b) is an edge in $H^{w_i^{a+1}}$ (but not in $B_H^{w_i^{a+1}}$). Since (ℓ_k^c, ℓ_j^b) is not in $B_H^{w_i^{a+1}}$, let (ℓ_ℓ^d, ℓ_j^b), $\ell < i$, be the edge excluding (ℓ_k^c, ℓ_j^b) from $B_H^{w_i^{a+1}}/i$. By the definition of B_H^t, ℓ_k^c locally precedes ℓ_ℓ^d. Since $B_H^{w_i^{a+1}}/i$ is a subgraph of G_i^{a+1}, (ℓ_ℓ^d, ℓ_j^b) is an edge in G_i^{a+1}. Since ℓ_k^c locally precedes ℓ_ℓ^d we get that (ℓ_ℓ^d, ℓ_j^b) excludes (ℓ_k^c, ℓ_j^b) from B_i^{a+1}, contradiction to the assumption that (ℓ_k^c, ℓ_j^b) is an edge in B_i^{a+1}/i. The lemma follows. □

The next corollary forms a tighter relationship between edges of G whose head belong to the frontal branch of the history graph and the corresponding edges of H.

COROLLARY 7: Let (ℓ_i^a, ℓ_j^b) be an edge in G_k^c. If for some t after $r_k^c[i]$, $\ell_j^b \in B_H^t$ then (ℓ_i^a, ℓ_j^b) is an edge in H^t.

Proof: Since (ℓ_i^a, ℓ_j^b) is an edge in G_k^c, Lemma 5 implies that for some r, $r \geq 0$, (ℓ_i^a, ℓ_j^{b+r}) is an edge in H^t. Since $\ell_j^b \in B_H^t$, Lemma 6 implies that ℓ_j^b is the current label of $\mathcal{W}_j$ at t, therefore $r = 0$. □

In the next lemma and in the theorem that follows, we prove that the serialization time of each logical write action lies within its execution interval.

LEMMA 8: Let t_0 be the occurrence time of $r_i^a[j]$. If for some t, $t \geq t_0$, $B_H^t/(j+1) \not\equiv B_i^a/(j+1)$, then there exists a good logical action L_k^c, for some $k \leq j$, such that w_k^c occurs within the time interval starting at $r_i^a[k]$ and ending at t.

Proof: By the assumption of the lemma, $B_H^t/(j+1) \not\equiv B_i^a/(j+1)$. Let ℓ_ℓ^d, $\ell \leq j$, be the node with the maximal *id* in the common prefix of $B_H^t/(j+1)$ and $B_i^a/(j+1)$. First we show that ℓ_ℓ^d is not the last node of $B_H^t/(j+1)$. If ℓ_ℓ^d is the last node of $B_H^t/(j+1)$ then since $B_i^a/(j+1) \not\equiv B_H^t/(j+1)$, ℓ_ℓ^d is not the last node of $B_i^a/(j+1)$. Let $e_1 = (\ell_m^e, \ell_\ell^d)$, $m < j+1$, be the edge entering ℓ_ℓ^d in $B_i^a/(j+1)$. Since $\ell_\ell^d \in B_H^t$, Corollary 7 implies that (ℓ_m^e, ℓ_ℓ^d) is an edge of H^t; since $m < j+1$, either $e_1 \in B_H^t/(j+1)$ or there exists some edge $e_2 \in B_H^t/(j+1)$ excluding e_1 from $B_H^t/(j+1)$, a contradiction to the assumption that ℓ_ℓ^d is the last node of $B_H^t/(j+1)$.

Since ℓ_ℓ^d is not the last node of $B_H^t/(j+1)$, let (ℓ_k^c, ℓ_ℓ^d), $\ell < k \leq j$, be the edge entering ℓ_ℓ^d in $B_H^t/(j+1)$. We now show that L_k^c satisfies the requirements of the lemma; since $\ell_k^c \in B_H^t$ Lemma 3 implies that L_k^c is good. Since $\ell_k^c \in H^t$, it is obvious that w_k^c occurs before t. To complete the proof it remains to show that $r_i^a[k] \rightarrow w_k^c$. Assume by way of contradiction that $w_k^c \rightarrow r_i^a[k]$. We reach a contradiction as follows (see Figure 3): First we show that (ℓ_k^c, ℓ_ℓ^d) is an edge in G_i^a (but not in

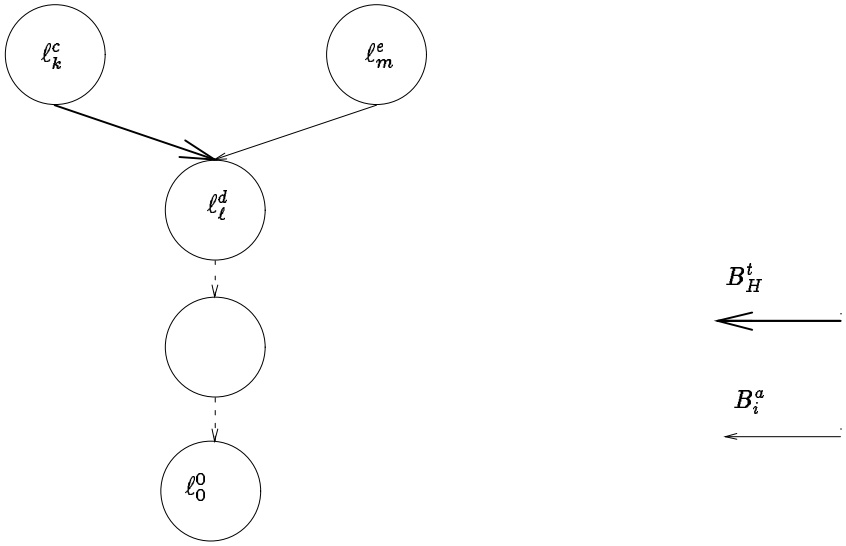

FIGURE 3. The graph for Lemma 8

$B_i^a/(j+1)$). By its definition, ℓ_k^c is a node of B_H^t, therefore by Lemma 6, ℓ_k^c is the current label of $\mathcal{W}_k$ at t. By our assumption $w_k^c \rightarrow r_i^a[k]$, hence ℓ_k^c is a node of G_i^a. Therefore (ℓ_k^c, ℓ_ℓ^d) is an edge in G_i^a. Since $k > \ell$, and ℓ_ℓ^d is the maximal common node in $B_H^t/(j+1)$ and $B_i^a/(j+1)$ we conclude that $\ell_k^c \notin B_i^a/(j+1)$. Let (ℓ_m^e, ℓ_ℓ^d), $\ell < m < k$, be the edge excluding (ℓ_k^c, ℓ_ℓ^d) from $B_i^a/(j+1)$. By Corollary 7, (ℓ_m^e, ℓ_ℓ^d) is an edge in H^t. Since $m < k$, (ℓ_m^e, ℓ_ℓ^d) excludes (ℓ_k^c, ℓ_ℓ^d) from B_H^t, a contradiction to the assumption that (ℓ_k^c, ℓ_ℓ^d) is in $B_H^t/(j+1)$. The lemma follows. □

THEOREM 9: Every logical write action is serialized within its execution interval.

Proof: Let L_i^a be some logical write action. If L_i^a is good then it is serialized at its concluding physical write w_i^a, which is within its execution interval. For the rest of the proof we assume that L_i^a is bad. Let ℓ_j^b be the last node of $B_H^{w_i^a}$. According to the serialization definition for bad write actions, L_i^a is serialized by L_j^b. We have to prove that L_j^b is serialized within the execution interval

of L_i^a, that is $r_i^a[1] \rightarrow w_j^b \rightarrow w_i^a$. Since $\ell_j^b \in H^{w_i^a}$, it is clear that $w_j^b \rightarrow w_i^a$. We now prove that $r_i^a[1] \rightarrow w_j^b$. Action L_i^a is bad, therefore $B_H^{w_i^a}/i \not\equiv B_i^a/i$. By Lemma 8 there exists a good logical action L_k^c, $k < i$, such that w_k^c occurs within the time interval starting at $r_i^a[k]$ and ending at w_i^a. Therefore $r_i^a[k] \rightarrow w_k^c \rightarrow w_i^a$, which implies $r_i^a[1] \rightarrow w_k^c \rightarrow w_i^a$. If $L_j^b \equiv L_k^c$ then the the proof follows. Otherwise, since L_j^b and L_k^c are both good, and since ℓ_j^b is the last node in $B_H^{w_i^a}$, we get that $w_k^c \rightarrow w_j^b$. Therefore, $r_i^a[1] \rightarrow w_k^c \rightarrow w_j^b$, which implies $r_i^a[1] \rightarrow w_j^b$. □

The fact that L_i^a is serialized before L_j^b is denoted by $L_i^a \Rightarrow L_j^b$. The next lemma proves that the history graph is a precedence graph with respect to the relation $\Rightarrow$. Since the history graph is not computable by the processors, the significance of this lemma is reflected in the theorem that follows in which it is proved that the graphs collected by the processors are also precedence graphs with respect to the relation $\Rightarrow$.

LEMMA 10: If (ℓ_i^a, ℓ_j^b), $i > j$, is an edge in H then $L_j^b \Rightarrow L_i^a$.

Proof: By Lemma 2 $w_j^b \rightarrow w_i^a$. Consider the following cases:

Case 1: L_i^a is good. In this case L_i^a is serialized at w_i^a. Since L_j^b is not serialized after w_j^b, and since $w_j^b \rightarrow w_i^a$ it holds that $L_j^b \Rightarrow L_i^a$.

Case 2: L_i^a and L_j^b are both bad. Let ℓ_k^c and ℓ_ℓ^d be the last nodes in $B_H^{w_i^a}$ and $B_H^{w_j^b}$ respectively. In this case L_i^a is serialized by L_k^c and L_j^b is serialized by L_ℓ^d. If $\ell_k^c \equiv \ell_\ell^d$, then by Definition 3, L_i^a and L_j^b are serialized by their *id*s. Since $i > j$ $L_j^b \Rightarrow L_i^a$. Assume $\ell_k^c \not\equiv \ell_\ell^d$. By definition 3, L_ℓ^d is the last good write action serialized before w_j^b and L_k^c is the last good write action serialized before w_i^a. Since $w_j^b \rightarrow w_i^a$ it holds that $w_\ell^d \rightarrow w_k^c$ and therefore $L_\ell^d \Rightarrow L_k^c$. Since L_j^b is serialized by L_ℓ^d and L_i^a is serialized by L_k^c, we get $L_j^b \Rightarrow L_i^a$.

Case 3: L_i^a is bad, L_j^b is good. In this case ℓ_j^b is not the last node of $B_H^{w_i^a}$. Let ℓ_k^c be the last node of $B_H^{w_i^a}$, that is $w_j^b \rightarrow w_k^c \rightarrow w_i^a$. Since L_j^b and L_k^c are both good, and since $w_j^b \rightarrow w_k^c$, $L_j^b \Rightarrow L_k^c$. By Definition 3(2) L_i^a is serialized by L_k^c, therefore we get $L_j^b \Rightarrow L_i^a$.

□

THEOREM 11: If (ℓ_i^a, ℓ_j^b), $i > j$, is an edge in G_k^c then $L_j^b \Rightarrow L_i^a$.

Proof: By Lemma 5, there exists some r, $r \geq 0$, such that (ℓ_i^a, ℓ_j^{b+r}) an edge in H. By Lemma 10 $L_j^{b+r} \Rightarrow L_i^a$. If $r = 0$ then we get $L_j^b \Rightarrow L_i^a$. If $r > 0$ then since $L_j^b \Rightarrow L_j^{b+r}$ we get again that $L_j^b \Rightarrow L_i^a$. □

4.2 The Reader Protocol

4.2.1 Description

Like the writer protocol, the reader protocol is obtained by adjusting the sequential reader protocol to the concurrent environment. Though the basic idea in this implementation is to keep a precedence graph whose last node is the last value written to the logical register, it is not possible to just collect a precedence graph and return the last node in its frontal branch: Due to concurrency, the current graph

and its last node may change during the execution of the collect procedure by the reader. In particular a label of an action which should not be returned by a reader, may appear as last in its collected graph. This happens when some concurrent write actions cause the reader to see some branches of the graph as "hanging in the air". Therefore a single collection is not sufficient. Our protocol collects three precedence graphs and analyzes the differences among them to determine the returned label. The three graphs are denoted by G, $\overleftarrow{G}$, and $\tilde{G}$. Graph $\overleftarrow{G}$ is collected in reverse order — from R_w down to R_1, therefore most of the lemmas proven in the previous section do not hold for $\overleftarrow{G}$. The analysis of the three graphs does not yield an accurate description of the current graph, but rather enables the reader to identify a label which satisfies the requirements for the reader protocol as follows: The identified label is either last in the history graph when the logical read action starts, and hence it is the last logical write action that is serialized before the read action starts, or the identified label is generated by a logical write action serialized within the execution interval of the logical read action. In this case the logical read is serialized immediately after the logical write. The physical actions, executed by $\mathcal{R}_u$, during the reader protocol are denoted by: $r_u[1] \ldots r_u[w]$, in which G is collected, $\overleftarrow{r_u}[w] \ldots \overleftarrow{r_u}[1]$, in which $\overleftarrow{G}$ is collected, and $\tilde{r}_u[1] \ldots \tilde{r}_u[w]$, in which $\tilde{G}$ is collected. The notation $B_i^a \subset B_j^b$ is used when for each $\ell_k^c \in B_i^a$, there is a label $\ell_k^d \in B_j^b$ such that $\ell_k^c \simeq \ell_k^d$. The notation $B_i^a \simeq B_j^b$ is used when $B_i^a \subset B_j^b$ and $B_j^b \subset B_i^a$. The code of the reader protocol appears in Figure 4.

begin
  Collect G_u; Collect $\overleftarrow{G_u}$; Collect $\tilde{G}_u$
  if $(B_u \simeq \tilde{B}_u \not\simeq \overleftarrow{B_u})$ **then**
    $i := \min_j(\ell_j^b (\in G_u) \not\simeq \ell_j^c (\in \overleftarrow{G_u}))$ **and** $(\ell_j^c \not\simeq \ell_j^d (\in \tilde{G}_u)))$
    return ℓ_i^d
  elseif $(B_u \simeq \overleftarrow{B_u} \simeq \tilde{B}_u)$ **then**
    return the label of *last* in $\tilde{B}_u$
  elseif $(B_u \not\simeq \tilde{B}_u)$ **and** $(B_u \subset \tilde{G}_u)$ **then**
    return the label of *last* in $\tilde{B}_u$
  elseif $(B_u \not\simeq \tilde{B}_u)$ **and** $(B_u \not\subset \tilde{G}_u)$ **then**
    $i := \min_j(\ell_j^a (\in B_u) \not\simeq \ell_j^b (\in \tilde{G}_u))$
    return ℓ_i^b
  endif
end

FIGURE 4. The protocol for $\mathcal{R}_u$ in the $(1, n)$ implementation

4.2.2 Serialization Scheme for Logical Read Actions

Throughout this paper S_u^a denotes the a-th execution of the read protocol by $\mathcal{R}_u$. The serialization time of any logical read action is determined by the serialization time of the logical write action whose value is returned by the read action, according to the following proposition:

PROPOSITION 12: If for any logical read action S_u^a the returned label ℓ_j^b satisfies one of the following conditions:

1. L_j^b is the logical write action that is serialized last before S_u^a starts.

2. L_j^b is serialized within the execution interval of S_u^a.

then the implementation is atomic.

To prove the proposition correct we have to show that if for every action in some execution E, one of these conditions holds, then the E is serializable. This is proven by the following serialization scheme:

DEFINITION 4: Let ℓ_i^b be the label returned by S_u^a. Denote by t_s and t_e the occurrence time of $r_u^a[1]$ and $\tilde{r}_u^a[w]$ respectively. The serialization time of S_u^a is defined as follows:

1. If L_i^b is not serialized within the execution interval of S_u^a then S_u^a is serialized at t_s.
2. If L_i^b is serialized within the execution interval of S_u^a then S_u^a is serialized by L_i^b, that is after L_i^b and before any physical action which occurs or any logical write which is serialized after L_i^b. In case two logical actions are serialized by the same logical write action they are serialized by an ascending order of their *id*s.

4.2.3 Correctness of the Implementation

The correctness of the serialization scheme for logical read actions, and the correctness of the entire implementation, is based on Proposition 12 and on the following theorem:

THEOREM 13: Let (t_s, t_e) be the execution interval of S_u, let ℓ be the label returned by S_u, and let L be the logical write action that produced the label ℓ. The label L satisfies one of the following two claims:

1. L is the last logical write action serialized before t_s (and hence ℓ is last in $B_H^{t_s}$), or
2. L is serialized within the interval (t_s, t_e).

Proof: Since ℓ is read during the execution of S_u, it is clear that L terminates before t_e. Therefore, to conclude that (2) holds, it suffices to show that L is serialized after t_s. Consider the following cases (which follow the cases of the protocol):

Case 1: $B_u \simeq \tilde{B}_u \not\simeq \overleftarrow{B}_u$.
Let i be the smallest *id* such that $\ell_i^b \simeq \ell_i^d \not\simeq \ell_i^c$, where ℓ_i^b, ℓ_i^d and ℓ_i^c are nodes in G_u, $\tilde{G}_u$, and $\overleftarrow{G}_u$ respectively. According to the protocol S_u returns ℓ_i^d. Since $\ell_i^b \not\simeq \ell_i^c$ we get that

$$r_u[1] \rightarrow r_u[i] \rightarrow w_i^c \tag{1}$$

Since $\ell_i^c \not\simeq \ell_i^d$ we get that

$$w_i^c \rightarrow r_i^d[1] \tag{2}$$

Therefore from (1) and (2) we get that $r_u[1] \rightarrow r_i^d[1]$. Hence L_i^d is serialized after t_s.

Case 2: $B_u \simeq \tilde{B}_u \simeq \overleftarrow{B}_u$.
Let ℓ_i^a be the last node of $\tilde{B}_u$. According to the protocol S_u returns ℓ_i^a. Consider the following cases:

Case 2.1: $B_u \not\equiv \tilde{B}_u$

In this case there exist two distinct labels ℓ_j^b and ℓ_j^c $(b < c)$ of the same writer, $\mathcal{W}_j$, such that $\ell_j^b \in B_u$ while $\ell_j^c \in \tilde{B}_u$. Since $B_u \simeq \tilde{B}_u$ $\ell_j^b \simeq \ell_j^c$, and in particular $\ell_j^b.address = \ell_j^c.address$. A writer does not use the same address twice in a row, hence $(b+1) < c$. Therefore $r_u[j] \rightarrow w_j^{b+1} \rightarrow r_j^c[1] \rightarrow w_j^c$, which implies that L_j^c is serialized after t_s. If $\ell_i^a \not\equiv \ell_j^c$ then there is a path from ℓ_i^a to ℓ_j^c in $\tilde{G}_u$ (because ℓ_i^a and ℓ_j^c are both in $\tilde{B}_u$, and ℓ_i^a is last in $\tilde{B}_u$), hence by Theorem 11 $L_j^c \Rightarrow L_i^a$. Therefore L_i^a is serialized after t_s.

Case 2.2: $B_u \equiv \tilde{B}_u$

We first show that B_u is a branch in $H^{r_u[w]}$. Let (ℓ_j^b, ℓ_k^c) be an arbitrary edge in B_u. By Lemma 5 there exists an integer r, $r \geq 0$, such that (ℓ_j^b, ℓ_k^{c+r}) is an edge in $H^{r_u[w]}$. Since $B_u \equiv \tilde{B}_u$, ℓ_k^c is a label in $\tilde{G}_u$, hence it is the current label of $\mathcal{W}_k$ at $\tilde{r}_u[k]$, and therefore $r = 0$. Thus every edge in B_u belongs to $H^{r_u[w]}$. Similarly, the assumptions $B_u \simeq \overleftarrow{B}_u$ (case 2) and $B_u \equiv \tilde{B}_u$ (case 2.2) imply that $B_u \equiv \overleftarrow{B}_u \equiv \tilde{B}_u$. Since $B_u \equiv B_H^{r_u[w]}$ (which is proven in Claim 1) we get that ℓ_i^a is last in $B_H^{r_u[w]}$, which implies that either ℓ_i^a is last in $B_H^{t_s}$ or ℓ_i^a is serialized after t_s.

CLAIM 1: $B_u \equiv B_H^{r_u[w]}$.

Proof of claim: Assume by way of contradiction that $B_u \not\equiv B_H^{r_u[w]}$. In this case at every time t after $r_u[w]$ there exists some edge in H^t that excludes a suffix of B_u from B_H^t. Let $S = \overleftarrow{t}_w < \overleftarrow{t}_{w-1} < \ldots < \overleftarrow{t}_1$ be the sequence of occurrence times of actions $\overleftarrow{r}_u[w]$, $\overleftarrow{r}_u[w-1]$ ... $\overleftarrow{r}_u[1]$ respectively. Define the function $EX(t)$ on S as follows: The value of $EX(t)$ is k where k is the *id* of the label in B_H^t whose edge excludes a suffix of B_u from B_H^t. Note that $EX(\overleftarrow{t}_w) < w$ and $EX(\overleftarrow{t}_1) > 1$. We now show that EX is not increasing in t. Assume that $EX(t) = k$, $EX(t') = \ell$, and $t < t'$. Let α denote the common prefix of B_H^t and B_u, and let β denote the common prefix of $B_H^{t'}$ and B_u. Since $t < t'$, β is a (not necessarily proper) prefix of α. If $\alpha \equiv \beta$ then $\ell \leq k$ and we are done. If β is a proper prefix of α then let (ℓ_j^b, ℓ_i^a) and (ℓ_ℓ^d, ℓ_i^a) be the edges of B_u and $B_H^{t'}$ respectively where (ℓ_ℓ^d, ℓ_i^a) excludes (ℓ_j^b, ℓ_i^a) from $B_H^{t'}$. Obviously $\ell < j < k$.

We now use the function EX to reach a contradiction by showing that under these assumptions $B_u \not\equiv \overleftarrow{B}_u$. Since $\overleftarrow{G}_u$ is collected in reverse order (from w to 1) and EX is not increasing, there exists a time $\overleftarrow{t}_k$, $\overleftarrow{t}_k \in S$, such that $EX(\overleftarrow{t}_k) = k$. Let (ℓ_k^b, ℓ_j^c) be the edge of $B_H^{\overleftarrow{t}_k}$ excluding a suffix of B_u from $B_H^{\overleftarrow{t}_k}$. By this definition, $\ell_j^c \in B_u$, since $B_u \equiv \overleftarrow{B}_u$, $\ell_j^c \in \overleftarrow{B}_u$, hence obviously $\ell_j^c \in \overleftarrow{G}_u$. Since ℓ_k^b belongs to $B_H^{\overleftarrow{t}_k}$, Lemma 6 implies that it is the current label of $\mathcal{W}_k$ at $\overleftarrow{t}_k$ and therefore ℓ_k^b is a node in $\overleftarrow{G}_u$. Since $\ell_k^b \in \overleftarrow{G}_u$ and $\ell_j^c \in \overleftarrow{G}_u$, we get that (ℓ_k^b, ℓ_j^c) is an edge in $\overleftarrow{G}_u$. Since (ℓ_k^b, ℓ_j^c) excludes a suffix of B_u from $B_H^{\overleftarrow{t}_k}$, (ℓ_k^b, ℓ_j^c) excludes a suffix of B_u from $\overleftarrow{B}_u$, contradiction to the assumption that $B_u \equiv \overleftarrow{B}_u$. □

Case 3: $B_u \not\simeq \tilde{B}_u$ **and** $B_u \subset \tilde{G}_u$.

Let ℓ_i^a be the last node of $\tilde{B}_u$. According to the protocol S_u returns ℓ_i^a. In the sequel we show that L_i^a is serialized after t_s. Since $B_u \subset \tilde{G}_u$ and $B_u \not\simeq \tilde{B}_u$ there is a suffix of $\tilde{B}_u$, none of whose labels are in B_u. Let j be the *id* of the minimal label ℓ_j^c in that suffix (see Figure 5), and let ℓ_j^b be the label of $\mathcal{W}_j$ in G_u. By this definition $\ell_j^b \not\simeq \ell_j^c$.

In order to show that L_i^a is serialized after t_s it is enough to show that L_j^c is serialized after t_s, since if $L_i^a \not\equiv L_j^c$ then there is a path from ℓ_i^a to ℓ_j^c in $\tilde{B}_u$, and therefore Theorem 11 implies

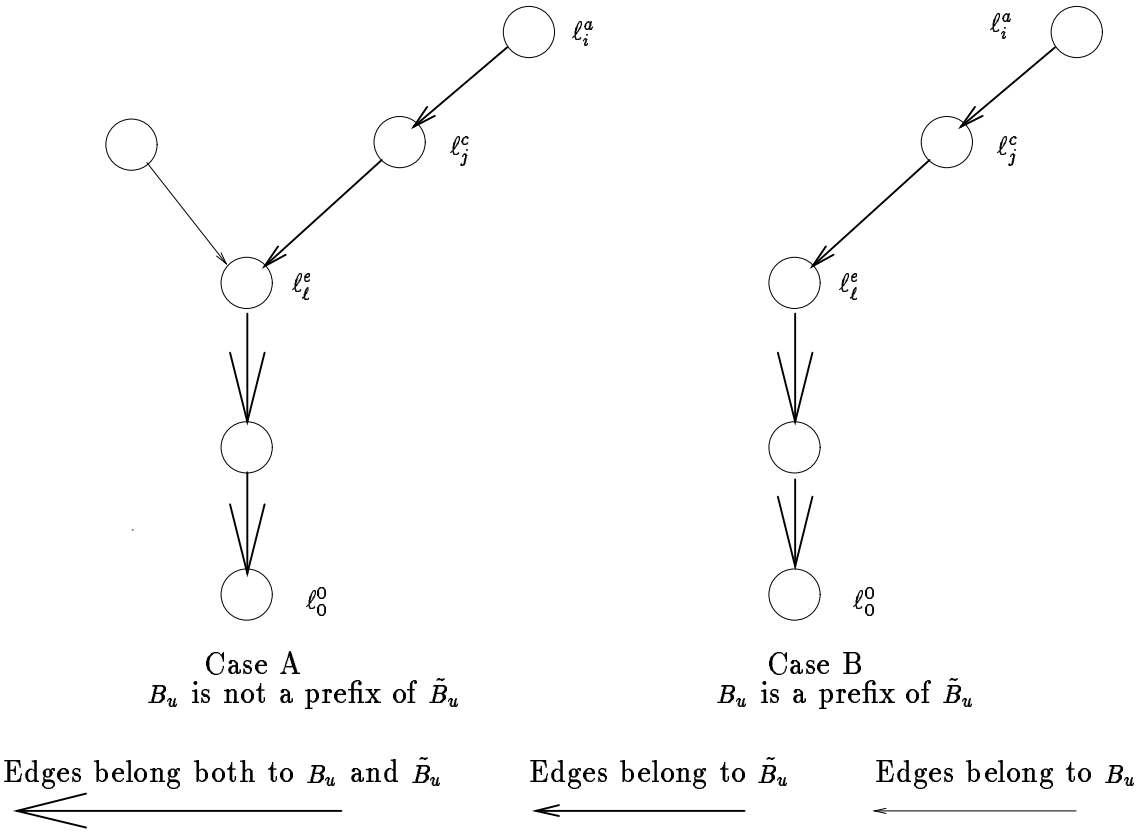

FIGURE 5. Case 3 - Relation between B_u and $\tilde{B}_u$ when $B_u \subset \tilde{G}_u$

that $L_j^c \Rightarrow L_i^a$. We proceed to show that L_j^c is serialized after t_s: Since $\ell_j^b \not\simeq \ell_j^c$ it is clear that $r_u[j] \rightarrow w_j^c$ and therefore w_j^c occurs after t_s. If L_j^c is good then it is serialized at w_j^c and the proof follows. If L_j^c is bad, then let ℓ_k^d be the last node in $B_H^{w_j^c}$. Since L_j^c is bad it is serialized by L_k^d. If L_k^d is serialized after t_s then L_j^c is also serialized after t_s and we are done. We continue the proof assuming that L_k^d is serialized before t_s, that is, $w_k^d \rightarrow r_u[1]$. By this assumption, there are no good writes which are serialized in the interval starting at $r_u[1]$ and ending at $r_u[j]$. In this case, Lemma 8 implies that $B_H^{w_j^c}/(j+1) \equiv B_u/(j+1)$. Consider (ℓ_j^c, ℓ_ℓ^e), the outgoing edge from ℓ_j^c in $\tilde{B}_u$. By Lemma 5 there exists an integer r, $r \geq 0$, such that $(\ell_j^c, \ell_\ell^{e+r})$ is an edge in $H^{w_j^c}$. By Lemma 10 L_j^c is serialized after L_ℓ^{e+r}. We now show that if $r > 0$, then L_ℓ^{e+r} is serialized after t_s, and therefore L_j^c is also serialized after t_s. Later we show that indeed $r > 0$. If $r > 0$ then $r > 1$ since a writer never uses the same address twice in a row. Since ℓ_ℓ^e is the current label read at $\tilde{r}_u[\ell]$, $\tilde{r}_u[\ell] \rightarrow w_\ell^{e+1}$. Since $w_\ell^{e+1} \rightarrow r_\ell^{e+r}[1]$ we get that $\tilde{r}_u[\ell] \rightarrow r_\ell^{e+r}[1]$ which implies that L_ℓ^{e+r} is serialized after t_s. Therefore L_j^c is also serialized after t_s. The proof is completed by showing that $r > 0$. Assume by way of contradiction that $r = 0$, that is (ℓ_j^c, ℓ_ℓ^e) is an edge in $H^{w_j^c}$. Since L_j^c is bad, ℓ_ℓ^e is not last in $B_H^{w_j^c}/j$. Since $B_u/(j+1) \equiv B_H^{w_j^c}/(j+1)$ and $B_u \subset \tilde{G}_u$, the edge excluding (ℓ_j^c, ℓ_ℓ^e) from $B_H^{w_j^c}$ also excludes (ℓ_j^c, ℓ_ℓ^e) from $\tilde{B}_u$, contradiction to the definition of ℓ_j^c.

Case 4: $B_u \not\subseteq \tilde{B}_u$ **and** $B_u \not\subset \tilde{G}_u$.

Let i be the smallest *id* such that ℓ_i^a is in B_u, $\ell_i^b \in \tilde{G}_u$ and $\ell_i^a \not\simeq \ell_i^b$. According to the reader protocol, S_u returns ℓ_i^b. We show that L_i^b is serialized after t_s. Since $\ell_i^b \not\simeq \ell_i^a$, $w_i^a \rightarrow r_u[i] \rightarrow w_i^b \rightarrow \tilde{r}_u[i]$, in particular $r_u[1] \rightarrow w_i^b$. If L_i^b is good, then it is serialized after t_s. Hence for the rest of Case 4, we assume that L_i^b is bad. We show that there exists a good write action L_j^c which is serialized after t_s, and L_i^b is either serialized by L_j^c or it is serialized after L_j^c. Consider the following cases:

Case 4.1: $\ell_i^a \in B_H^{r_u[i]}$.

Since L_i^b is bad, $B_H^{w_i^b}/i \not\equiv B_i^b/i$, hence Lemma 8 implies that there exists a good write action L_j^c such that $j < i$ and $r_i^b[j] \rightarrow w_j^c \rightarrow w_i^b$. Since $w_i^a \rightarrow r_i^b[j]$ we get that $w_i^a \rightarrow w_j^c \rightarrow w_i^b$.

Now we use L_j^c to show that L_i^b is serialized after t_s. Since L_i^b is bad, it is serialized by the label that was last in $B_H^{w_i^b}$. Since $w_j^c \rightarrow w_i^b$, the last label in $B_H^{w_i^b}$ is either ℓ_j^c or the label of another logical write action which is serialized after L_j^c. The proof is completed by showing that L_j^c is serialized after $r_u[i]$, which implies that L_i^b is also serialized after $r_u[i]$ and therefore after t_s. Since L_j^c is good, it is serialized at w_j^c. Therefore it is enough to show that $r_u[i] \rightarrow w_j^c$. Note that since $w_i^a \rightarrow w_j^c$, $\ell_i^a \in H^{w_j^c}$. However, since ℓ_j^c is last in $B_H^{w_j^c}$ and $i > j$, Lemma 4(a) implies that $\ell_i^a \notin B_H^{w_j^c}$. Therefore Lemma 1 implies that for any t after the occurrence time of w_j^c, $\ell_i^a \notin B_H^t$. By the assumption of this case $\ell_i^a \in B_H^{r_u[i]}$ therefore $r_u[i] \rightarrow w_j^c$.

Case 4.2: $\ell_i^a \notin B_H^{r_u[i]}$.

By the definition of i in Case 4, $\ell_i^a \in B_u$. On the other hand, by Case 4.2 $\ell_i^a \notin B_H^{r_u[i]}$, therefore $B_H^{r_u[i]}/(i+1) \not\equiv B_u/(i+1)$. By Lemma 8 there exists a good write action , L_j^c, $j < i$, such that $r_u[j] \rightarrow w_j^c \rightarrow r_u[i]$. L_j^c is good, therefore it is serialized at w_j^c. Since $w_j^c \rightarrow r_u[i] \rightarrow w_i^b$, we get $w_j^c \rightarrow w_i^b$, and therefore L_i^b is serialized either by L_j^c or later. Since w_j^c occurs after t_s, L_i^b is serialized after t_s too.

□

5 Multi-Writer out of Single-Reader Registers

5.1 Description

In this section we present an implementation in which the physical registers are atomic, $(1,1)$-registers. In this implementation readers and writers share the same protocol which is obtained by modifying the writer protocol of the $(1,n)$ implementation. Throughout this section we assume that the *id*s of the readers are larger than the *id*s of the writers. The reader protocol contains an extra *return* statement which terminates its execution. Communication is two-sided, as implied by the fact that the writer and reader protocols are the same and as proved necessary by the recent result of [ITV92].

Processor P_i communicates with processor P_j, where $i \neq j$, by writing into a $(1,1)$ atomic register denoted by $R_{i,j}$ from which P_j reads. A physical read action executed by P_i from $R_{j,i}$ is denoted by $r_i[j]$; while $w_i[j]$ denotes a physical write action by P_i to $R_{i,j}$. Logical action number a of P_i, where P_i is either a writer or a reader, is denoted by L_i^a. Since we use $(1,1)$ registers, some single physical actions of the $(1,n)$ implementation are now replaced by n physical actions (for example w_i^a is replaced by $w_i^a[1] \ldots w_i^a[n]$). For reasons we explain later each register contains five successive labels (and their corresponding values). The five labels are called *new*, *current*, *previous*, *old* and *ancient*.

```
begin
    Hand_Shake_Up(i); Hand_Shake_Down(i)
    Collect G_i                                                    r_i^a[1] ... r_i^a[n]
    if L_i is enclosed free then
        ℓ_last := last label in B_i/i
    else
        ℓ_last := label with maximal id enclosed in L_i^a
    endif
    new.address := select(G_i)
    new.edge := (last, ℓ_last.address)
    R_{i,last} := write (new, current, previous, old, ancient)     d_i^a[last]
    new_reg := read R_{last,i}                                     r̃_i[last]
    if (ℓ_last ≃ new_reg.current) then ret_label := current        connect
    else if (ℓ_last ≃ new_reg.previous) then ret_label := previous
    else if (ℓ_last ≃ new_reg.old) then ret_label := old
    else                                                           self-loop
        new.edge := (i, new_address)
        ret_label := ℓ_last
    endif
    for j := 1 to n                                                inform
        R_{i,j} := write (new, current, previous, old, ancient)    i_i^a[1] ... i_i^a[n]
    endfor
    current, previous, old, ancient := new, current, previous, old
    for j := 1 to i
        R_{i,j} := write (new, current, previous, old, ancient)    w_i^a[1] ... w_i^a[i]
    endfor
    for j := i + 1 to n
        end := start_j
        R_{i,j} := write (end, new, current, previous, old, ancient)   w_i^a[i+1] ... w_i^a[n]
    endfor
    if you are a reader (i > w) then return(ret_label)
end
```

FIGURE 6. The protocol for P_i in the $(1,1)$ implementation

```
Procedure Hand_Shake_Up(i)
begin
    for j = (i + 1) to n
        temp := read R_{i,j}                          r_i^a[i+1].c, ..., r_i^a[n].c
        start_j := temp.c
        R_{i,j} := write (start_j)                    w_i^a[i+1].s, ..., w_i^a[n].s
    endfor
end
```

```
Procedure Hand_Shake_Down(i)
begin
    for j = 1 to (i − 1)
        temp := read R_{i,j}                          r_i^a[1].s, ..., r_i^a[i−1].s
        complement := ¬temp.start
        R_{i,j} := write (complement)                 w_i^a[1].c, ..., w_i^a[i−1].c
    endfor
end
```

FIGURE 7. The Hand_Shake procedures

Immediately after the occurrence of $w_i^a[j]$ the *current* label in $R_{i,j}$ holds ℓ_i^a, the label computed by L_i^a, while *previous*, *old* and *ancient* hold ℓ_i^{a-1}, ℓ_i^{a-2} and ℓ_i^{a-3} respectively.

The *serialization interval* of logical action L_i^a is some interval (to be precisely defined later) enclosed within its execution interval and in which its serialization time lies. Our goal in this protocol is to enable each processor to compute its target label after collection of a single precedence graph. In case there exists no lower indexed processor that executes many actions overlapping the collection phase we show that the frontal branch of the history graph at the beginning of L_i^a's serialization interval is a subgraph of G_i^a. In this case the last node of B_i^a/i is a valid target. If however there exists a lower indexed processor that executes many overlapping actions then the last node of B_i^a/i may not be used as a target. In this situation the processor identifies one of these overlapping actions of a lower-indexed processor and uses its label as its target. The overlapping actions identified during execution of L_i^a are called enclosed within L_i^a. The *hand-shake* mechanism enables L_i^a to detect enclosed actions. In section 5.2 we formally define the enclosed relation and give a detailed explanation on how the hand-shake procedures work. The code of the protocol appears in Figures 6 and 7: We now describe the code assuming logical action L_i^a is executed. Action L_i^a is started with execution of procedures *Hand_Shake_Up* and *Hand_Shake_Down*. Procedure *Hand_Shake_Up* enables detection of L_i^a as enclosed by actions of processors with $id > i$. The physical actions executed during *Hand_Shake_Up* are denoted by $r_i^a[i+1].c$, $w_i^a[i+1].s$ ... $r_i^a[n].c$, $w_i^a[n].s$. Procedure *Hand_Shake_Down* enables L_i^a to detect enclosed actions of processors with $id < i$. The physical actions executed during *Hand_Shake_Down* are denoted by $r_i^a[1].s$, $w_i^a[1].c$ ... $r_i^a[i-1].s$, $w_i^a[i-1].c$.

After executing the hand-Shake procedures L_i^a collects G_i^a in actions $r_i^a[1]$ ... $r_i^a[n]$. The graph G_i^a contains $3n+1$ nodes — three nodes per processor and the virtual root node. The three labels of P_j in G_i^a are chosen from the five labels read from $R_{j,i}$ in action $r_i^a[j]$: If in G_i^a there exists a label ℓ_k, $j < k < i$, such that e_k is directed to the *new* label of P_j in $R_{j,i}$ (i.e. $\ell_k.edge = (j, R_{j,i}.new.address)$) then the three chosen labels are *new*, *current* and *previous*. Otherwise the three labels are *current*, *previous* and *old*. After G_i^a is collected, P_i computes a label called ℓ_{last} towards which the edge of its

new tentative label is directed. If L_i^a does not have any enclosed label then ℓ_{last} is the last label in B_i^a/i. Otherwise ℓ_{last} is the enclosed label with the maximal *id*. Then P_i declares its new tentative label to P_{last} in action $d_i^a[last]$. In the next step (action $\tilde{r}_i^a[last]$) P_i re-reads $R_{last,i}$. If ℓ_{last} is $\simeq$ to either *current*, or *previous*, or *old* then L_i^a connects — the tentative *new* label becomes *current*. In this case we say that the *target* label of ℓ_i^a is the label read in $\tilde{r}_i^a[last]$, which is $\simeq$ to ℓ_{last}. Otherwise (L_i^a loops), e_i^a forms a self-loop.

The address of the tentative new label is obtained by the function *select* that considers all five labels collected from each processor. Since every register contains five labels, the addresses of every consecutive five labels are distinct. Therefore, if $\ell_i^a \simeq \ell_i^{a+r}$ and if $r > 0$, then $r > 4$. The *ancient* label is used as follows: Consider an execution E in which the edge outgoing from ℓ_j^b is (ℓ_j^b, ℓ_i^a). In action $w_j^{b+4}[i]$ (ℓ_j^b, ℓ_i^a) is removed from $R_{j,i}$, but it is still present in registers $R_{j,k}$ for $k > i$. At this point, P_i may select the address of ℓ_i^a as an address for some new label ℓ_i^{a+r} $(r > 0)$ where L_i^{a+r} starts after L_j^b terminates. If, after L_i^{a+r} is completed, P_k starts logical action L_k^c and reads both $R_{j,k}$ and $R_{i,k}$ while collecting G_k^c, then P_k adds the edge (ℓ_j^b, ℓ_i^{a+r}) to G_k^c. Since in E, $L_j^b \Rightarrow L_i^{a+r}$, this edge violates the precedence relation. The reason for this problem is the fact that due to the use of $(1,1)$ registers, a label "leaves" the system gradually, first for processors with lower *id*s and then to those with higher *id*s. The ancient label provides a window of time during which the old label leaves the system while reuse of its address is delayed. For this reason the ancient label itself is never chosen as a target for new labels.

Action L_i^a ends with the concluding write stage in which ℓ_i^a is written to each processor in turn. The physical actions in this stage are $w_i^a[1] \ \ldots \ w_i^a[n]$. The fact that the concluding write of the previous implementation is now replaced with n physical actions causes the following problem: If processor P_j directs an edge towards ℓ_i^a before action $w_i^a[k]$, $k > j$, takes place and if at that point P_k reads both $R_{j,k}$ and $R_{i,k}$, P_k may see (ℓ_j^b, ℓ_i^a) as "hanging in the air", while in fact both its endpoints join the history graph at an earlier stage, hence the graph collected by P_k is not updated. To prevent this problem P_i informs all processors that ℓ_i^a is its next label before the concluding write stage. This is done during the inform stage of the protocol, in physical actions $i_i^a[1] \ \ldots \ i_i^a[n]$, in which ℓ_i^a is written in the *new* label field, of $R_{i,1} \ \ldots \ R_{i,n}$ respectively, while all other fields in these registers remain unchanged. If in action L_k^c, P_k sees some edge e_j, $i < j < k$, directed towards a *new* label ℓ_i^a, P_k concludes that ℓ_i^a can safely be added to G_k^c.

During execution of L_i^a, P_i computes a label called *ret_label*. This label is used only if P_i is a reader, in this case the value returned by L_i^a is the value corresponding to *ret_label*; this value also becomes the value corresponding to ℓ_i^a. This is the only place in the implementation in which a value corresponding to one label is copied to another label. Since there are five labels in each register, and each label contains three fields each of which is of size $\log n + O(1)$ bits, the space complexity of the $(1,1)$ implementation is $O(\log n)$. The time complexity is $5n + O(1)$.

5.2 Serialization Scheme

Serialization in the $(1,1)$ protocol is done once more using a history graph. The history graph contains labels of writers and readers. The time ℓ_i^a joins the history graph, which is called the *joining time* of ℓ_i^a should satisfy the following requirements:

1. The joining time of ℓ_i^a lies within the interval $[\ w_i^a[i+1]\ ,\ w_i^a[n]\ ]$.
2. If (ℓ_i^a, ℓ_j^b), $i > j$, is an edge in H then the joining time of ℓ_i^a is after the joining time of ℓ_j^b.

These requirements are fulfilled by the following formal inductive definition:

DEFINITION 5: Let τ be the minimal time interval between any two successive physical actions and let $\epsilon = \tau/n$. Let L_i^a be a logical action where t_1 is the occurrence time of $w_i^a[n]$ and t_2 is the earliest joining time of any label whose target is ℓ_i^a. The *joining time* of ℓ_i^a is defined to be $\min(t_1, t_2 - \epsilon)$. If ℓ_i^a is the *target* label of ℓ_j^b and the joining time of ℓ_i^a is determined according to the joining time of ℓ_j^b then we say that ℓ_i^a joins the history graph by ℓ_j^b.

Using this definition we now define the history graph and the set of good actions:

DEFINITION 6: Let E be an execution of the system. The *History graph* H of E is defined as follows:

1. H^0 is the *History graph* at time 0 (before the execution begins). It contains only the *root* node — ℓ_0^0.
2. Let L_i^a be an arbitrary logical action. Let t be the joining time of ℓ_i^a. H^t is the graph obtained from $H^{t-\epsilon}$ by adding node ℓ_i^a, and the directed edge e_i^a as follows: If L_i^a connects then e_i^a is directed towards the target label of ℓ_i^a. Otherwise (L_i^a loops), e_i^a is a self-loop.

DEFINITION 7: Action L_i^a is *good* if ℓ_i^a is last in B_H^t where t is the joining time of ℓ_i^a.

We now define the *enclosed* relation; we first give an operational definition and then motivate it by some intuitive explanation: To detect enclosed actions, every physical register $R_{i,k}$, $i < k$, is augmented with two *hand-shake* bits called *start* and *end*, while every $R_{i,j}$, $i > j$ is augmented with one *hand-shake* bit called *complement*. Action L_i^a detects an enclosed action L_j^b, $j < i$, as follows: In actions $r_j^b[i].c$ and $w_j^b[i].s$ (executed in procedure *Hand_Shake_Up* during L_j^b) the *complement* bit of $R_{i,j}$ is read and copied to the *start* bit of $R_{j,i}$. Then in action $w_j^b[i]$ this value is copied to the *end* bit of $R_{j,i}$. In contrast, in actions $r_i^a[j].s$ and $w_i^a[j].c$ (executed in *Hand_Shake_Down* during L_i^a) the *start* bit of $R_{j,i}$ is read and its complement is written into the *complement* bit of $R_{i,j}$. If in action $r_i^a[j]$, executed during the collect stage of L_i^a, it is found that the values of the *start* and *end* bits in $R_{j,i}$ are both equal to the *complement* bit in $R_{i,j}$, then L_j^b is enclosed within L_i^a.

DEFINITION 8: Let L_j^b and L_i^a, $j < i$, be some logical actions. Action L_j^b and label ℓ_j^b are *enclosed* within L_i^a if ℓ_j^b is the *current* label in $R_{j,i}$ at $r_i^a[j]$ and the *start* and the *end* bits read in $r_i^a[j]$ are both equal to the *complement* bit in $R_{i,j}$. Action L_i^a is *enclosed free* if it has no enclosed labels.

This operational definition can be intuitively motivated as follows: The protocol is designed so that the serialization time of action L_i^a lies within the interval starting at $r_i^a[1].s$ (the first operation in the *Hand_Shake_Down* procedure) and ending at the joining time of ℓ_i^a. This interval is called the *serialization interval* of L_i^a. If while G_i^a is collected label L_j^b is detected as *enclosed* within L_i^a, then it holds that $r_i^a[1].s \rightarrow r_j^b[1].s$. If under these conditions L_i^a decides to direct e_i^a towards L_j^b then the joining time of ℓ_i^a occurs after the joining time of ℓ_j^b and the serialization interval of L_j^b is enclosed within the serialization interval of L_i^a. This is formally presented in the following proposition:

PROPOSITION 14: If L_j^b is enclosed within L_i^a then $r_i^a[1].s \rightarrow r_j^b[1].s \rightarrow w_j^b[i] \rightarrow r_i^a[j]$.

Proof: Since L_j^b is enclosed within L_i^a the *complement* bit in $R_{i,j}$ is equal to the *start* bit in $R_{j,i}$. Therefore it is clear that $r_i^a[j].s \rightarrow w_j^b[i].s$. Since $r_i^a[1].s \rightarrow r_i^a[j].s$ (if $j \neq 1$) and $w_j^b[i].s \rightarrow r_j^b[1].s$, we get that $r_i^a[1].s \rightarrow r_j^b[1].s$. Since ℓ_j^b is the *current* label in $R_{j,i}$ at $r_i^a[j]$ it holds that $w_j^b[i] \rightarrow r_i^a[j]$, the proof follows. □

Now we can explain why each collected graph contains three labels for each processor: Our goal in this protocol is to enable each processor to compute its target label after collection of a single precedence graph. In case the processor detects some enclosed labels it chooses one of them as its target. Otherwise we require that $B_H^{r_i^a[1]}/i$ should be a subgraph of G_i^a. Assume that $\ell_j^b \in B_H^{r_i^a[1]}/i$. It can be shown that if P_j does not generate any enclosed label then it can produce at most two labels in addition to ℓ_j^b before the occurrence time of $r_i^a[j]$. The first label is produced by an action which starts before the occurrence of $r_i^a[1]$ while the second is finished after the occurrence of $r_i^a[j]$. To ensure that either $R_{j,i}$ consists of some enclosed label or that $\ell_j^b \in G_i^a$ each collected graph contains three labels of every processor.

We now define the serialization time for the logical actions. Logical write actions are serialized independently of the logical read actions. A good write action is serialized at its joining time. A bad write action L_j^b is serialized by the last label in $B_H^t/w+1$ (i.e. the last label of a writer) where t is L_j^b's joining time. In Theorem 21 we prove that each logical write action is serialized within its serialization interval.

DEFINITION 9: Let L_i^a be a logical write action whose joining time is t.

1. If L_i^a is good then L_i^a is serialized at t.
2. If L_i^a is bad and ℓ_j^b is the last node of $B_H^t/(w+1)$ then L_i^a is serialized by L_j^b, that is, before L_j^b and after any other physical action that precedes the joining time of L_j^b. In case several logical write actions are serialized by the same logical action they are serialized in ascending order of their *id*s, starting with the lowest *id* and ending with the highest.

Read actions are serialized according to the labels they return. The serialization time of a logical read action L_i^a is defined as follows:

DEFINITION 10: Let L_i^a be a logical read action whose *ret_label* is ℓ_j^b. If L_j^b is serialized before $r_i^a[1].s$ then L_i^a is serialized at $r_i^a[1].s$. Otherwise, L_i^a is serialized by L_j^b, that is, after L_j^b and before any physical action which occurs after L_j^b, and any logical action which is serialized after L_j^b. In case several logical read actions are serialized by the same logical action they are serialized in ascending order of their *id*s.

Based on this definition it is very easy to show for a logical read action L_i^a that is serialized within its serialization interval. The fact that L_i^a is serialized after $r_i^a[1].s$ follows immediately from Definition 10. In the next proposition it is shown that the serialization time of L_i^a is not later then its joining time and therefore the requirement that L_i^a is serialized within the interval $[r_i^a[1].s\ ,\ t]$, where t is the joining time of ℓ_i^a is preserved. The proof is by induction.

PROPOSITION 15: Let t be the joining time of L_i^a. If ℓ_j^b is the *ret_label* of L_i^a then L_j^b is serialized before t.

5.3 Correctness Proof

We begin this section with several auxiliary lemmas which are used in the proof of Theorem 21 in which it is proved that the serialization definition of the write actions satisfies the requirements.

PROPOSITION 16: The last label in B_i^a/i is not a *new* label.

Proof: By the definition of B_i^a/i, the last label of B_i^a/i has no incoming edge from a label whose *id* is smaller then i. Therefore, by the construction of G_i^a, the last label in B_i^a/i is either *current* or *previous* or *old*. □

LEMMA 17: If (ℓ_i^a, ℓ_j^b), $i > j$, is an edge in G_k^c then there exists some integer r, $r \geq 0$ such that (ℓ_i^a, ℓ_j^{b+r}) is an edge in H.

Proof: By the protocol, if (ℓ_i^a, ℓ_j^b) is an edge in G_k^c , where $i > j$, then L_i^a connects, and its target label is some label of P_j, ℓ_j^{b+r}. Obviously $\ell_j^{b+r}.address = \ell_j^b.address$. To prove the lemma we show that $r \geq 0$. Assume by way of contradiction that $r < 0$. Since the addresses of every five consecutive labels are distinct and $\ell_j^{b+r}.address = \ell_j^b.address$, we get that $r < -4$. Label ℓ_i^a is one of the five labels in $R_{i,j}$ throughout the interval $[\ d_i^a[j]\ ,\ w_i^{a+4}[j]\ )$. We prove the lemma by showing that $d_i^a[j] \rightarrow r_j^b[i] \rightarrow w_i^{a+4}[j]$, contradictory to the definition of the function *select* (executed during L_j^b).

First we show that $d_i^a[j] \rightarrow r_j^b[i]$. The old, previous and current labels in $R_{j,i}$ at $w_j^{b-2}[i]$ are ℓ_j^{b-4}, ℓ_j^{b-3} and ℓ_j^{b-2} respectively. Since $r < -4$, ℓ_j^{b+r} is neither *old* nor *previous* nor *current* in $R_{j,i}$ at $w_j^{b-2}[i]$. Since ℓ_j^{b+r} is one of these labels in $R_{j,i}$ at $\tilde{r}_i^a[j]$ we get that $\tilde{r}_i^a[j] \rightarrow w_j^{b-2}[i]$. By the protocol $d_i^a[j] \rightarrow \tilde{r}_i^a[j]$ and $w_j^{b-2}[i] \rightarrow r_j^b[i]$, hence $d_i^a[j] \rightarrow r_j^b[i]$.

The fact that $r_j^b[i] \rightarrow w_i^{a+4}[j]$ is proved as follows:

1. $r_j^b[i] \rightarrow i_j^b[k]$ (according to the protocol)
2. $i_j^b[k] \rightarrow r_k^c[j]$ (since $\ell_j^b \in G_k^c$)
3. $r_k^c[j] \rightarrow r_k^c[i]$ (since $i > j$ and G_k^c is collected in ascending order)
4. $r_k^c[i] \rightarrow w_i^{a+3}[k]$ (since $\ell_i^a \in G_k^c$)
5. $w_i^{a+3}[k] \rightarrow w_i^{a+4}[j]$ (P_i works in sequential manner)

Since the right hand side of each relation is the left hand side of the next one we get $r_j^b[i] \rightarrow w_i^{a+4}[j]$. In conclusion: $d_i^a[j] \rightarrow r_j^b[i] \rightarrow w_i^{a+4}[j]$, a contradiction. □

LEMMA 18: Let L_k^c be an enclosed free action and let t be the joining time of ℓ_k^c. Every node of B_k^c/k belongs to H^t.

Proof: We first show that the last label of B_k^c/k satisfies the lemma: Let ℓ_i^a be the last label of B_k^c/k. By Proposition 16, ℓ_i^a is not the *new* label in $R_{i,k}$ at $r_k^c[i]$. Therefore ℓ_i^a is either *current* or *previous* or *old*. If $w_i^a[n]$ occurs before t, as in the case when ℓ_i^a is either *previous* or *old*, we are done. The only remaining case is when $w_i^a[n]$ occurs after t. In this case ℓ_i^a is *current* in $R_{i,k}$ at $r_k^c[i]$. Since L_k^c is enclosed free the label ℓ_{last} computed during L_k^c is the last label in B_k^c/k, namely ℓ_i^a. We now show that $w_i^a[k] \rightarrow \tilde{r}_k^c[i] \rightarrow w_i^a[n]$, which implies that ℓ_i^a is the target label of ℓ_k^c, that hence its joining time is no later then the joining time of ℓ_i^a, namely t. The first relation is proved as follows: Since ℓ_i^a is *current* in $R_{i,k}$ at $r_k^c[i]$, we get that $w_i^a[k] \rightarrow r_k^c[i]$. According to the protocol $r_k^c[i] \rightarrow \tilde{r}_k^c[i]$ thus $w_i^a[k] \rightarrow \tilde{r}_k^c[i]$. To prove that $\tilde{r}_k^c[i] \rightarrow w_i^a[n]$ note that by Definition 5, the joining time of ℓ_k^c is within the interval $[\ w_k^c[k]\ ,\ w_k^c[n]\ ]$. According to the protocol $\tilde{r}_k^c[i] \rightarrow w_k^c[k]$, hence $\tilde{r}_k^c[i]$ is before t. Since we assume that $w_i^a[n]$ occurs after t, we get that $\tilde{r}_k^c[i] \rightarrow w_i^a[n]$.

To complete the proof it suffices to show that if (ℓ_i^a, ℓ_j^b) is an edge in B_k^c/k, and if the joining time of ℓ_i^a is before t, then the joining time of ℓ_j^b is also before t. By Definition 5 the joining time of ℓ_i^a is after the occurrence time of $w_i^a[i]$. Since according to the protocol $\tilde{r}_i^a[j] \rightarrow w_i^a[i]$ and since the joining time of ℓ_i^a is before t, we get that $\tilde{r}_i^a[j]$ occurs before t. By Lemma 17 there exists some r, $r \geq 0$, such that ℓ_j^{b+r} is the target label of ℓ_i^a. If $r = 0$ then ℓ_j^b is the target label of ℓ_i^a which implies by Definition 5 that the joining time of ℓ_j^b is before t, and we are done. If $r > 0$ then $w_j^b[n] \rightarrow w_j^{b+r}[i] \rightarrow \tilde{r}_i^a[j]$. In this case $w_j^b[n]$ occurs before t and again by Definition 5 the joining time of ℓ_j^b is before t. □

LEMMA 19: If ℓ_j^b is the label with the maximal *id* enclosed in L_i^a, then the joining time of ℓ_j^b is before the joining time of ℓ_i^a.

Proof: By the protocol ℓ_j^b is the label ℓ_{last} computed during L_i^a. By Definition 5 the joining time of ℓ_j^b occurs within the interval $[\ w_j^b[j]\ ,\ w_j^b[n]\ ]$. To prove the lemma we show that either $w_j^b[n] \rightarrow w_i^a[1]$ or ℓ_j^b is the target label of ℓ_i^a and hence, by Definition 5, the joining time of ℓ_j^b is before the joining time of ℓ_i^a. If $w_j^b[n] \rightarrow \tilde{r}_i^a[j]$ then $w_j^b[n] \rightarrow w_i^a[1]$ and we are done. Assume $\tilde{r}_i^a[j] \rightarrow w_j^b[n]$. Since ℓ_j^b is enclosed in L_i^a, Definition 8 implies that $w_j^b[i] \rightarrow r_i^a[j]$. Since $r_i^a[j] \rightarrow \tilde{r}_i^a[j]$ we get $w_j^b[i] \rightarrow \tilde{r}_i^a[j] \rightarrow w_j^b[n]$. Therefore ℓ_j^b is the *current* label in $R_{j,i}$ at $\tilde{r}_i^a[j]$, that is, ℓ_j^b is the target label of ℓ_i^a. □

LEMMA 20: Let t be the joining time of ℓ_i^a.

1. If $B_H^{r_i^a[1].s}/i \equiv B_H^t/i$ then
   - **(1.a)** L_i^a is enclosed free.
   - **(1.b)** $B_H^{r_i^a[1].s}/i$ is a subgraph of G_i^a.
   - **(1.c)** L_i^a is good.
2. For any $t' \geq t$, if $\ell_i^a \in B_H^{t'}$ then $\ell_i^{a+1} \notin H^{t'}$.

Proof: By induction on i the *id* of the labels.

Base $i = 1$:
(1.a) Every logical action executed by P_1 is enclosed free. (1.b) For any t, $B_H^t/1$ contains a single node, namely ℓ_0^0. (1.c) Every action of P_1 is good.
(2) Every label of P_1 excludes its previous label from the frontal branch of the history graph.

Step: Assume correctness for $j < i$. We now prove correctness for i.

Proof of (1.a): Assume by way of contradiction that L_i^a is not enclosed free. Let L_j^b, $j < i$, be the action with maximal *id*, enclosed in L_i^a. By Proposition 14 it holds that $r_i^a[1].s \rightarrow r_j^b[1].s$. Since ℓ_j^b is the label with the maximal *id* enclosed in L_i^a, Lemma 19 implies that the joining time of ℓ_j^b, t', is before t. Hence the interval $[r_j^b[1].s\ ,\ t']$ is enclosed within the interval $[r_i^a[1].s\ ,\ t]$. Since $B_H^{r_i^a[1].s}/i \equiv B_H^t/i$ we get that $B_H^{r_j^b[1].s}/i \equiv B_H^{t'}/i$. By the induction assumption, L_j^b is good. Hence the frontal branch of the history graph is modified at t', the joining time of ℓ_j^b. Since $j < i$, $B_H^{r_j^b[1].s}/i \not\equiv B_H^{t'}/i$, contradiction.

Proof of (1.b): In order to show that $B_H^{r_i^a[1].s}/i$ is a subgraph of G_i^a, we show that every label of $B_H^{r_i^a[1].s}/i$ belongs to G_i^a: First we show that every such label is read during collection of G_i^a either as *new* or as *current* or as *previous*. Let ℓ_j^b be some label in $B_H^{r_i^a[1].s}/i$. Clearly $i_j^b[i] \rightarrow r_i^a[1].s$ and therefore $i_j^b[i] \rightarrow r_i^a[j]$. Since $B_H^{r_i^a[1].s}/i \equiv B_H^t/i$ we get that $\ell_j^b \in B_H^t/i$ which implies, by induction hypothesis (2), $\ell_j^{b+1} \notin H^t$. Therefore $w_j^{b+1}[n]$ occurs after t which implies that $i_j^b[i] \rightarrow r_i^a[j] \rightarrow w_j^{b+1}[n]$. Thus ℓ_j^b is either *new* or *current* or *previous* in $R_{j,i}$ at $r_i^a[j]$. Labels which are either *current* or *previous* belong to G_i^a independently of the rest of the labels in G_i^a. A *new* label belongs to G_i^a only if it has an edge incoming from another legal in G_i^a whose *id* is smaller then i. If the last label of $B_H^{r_i^a[1].s}/i$ is in G_i^a then all the labels in $B_H^{r_i^a[1].s}/i$ which were read as *new* have such an incoming edge. Thus, to show that all the labels of $B_H^{r_i^a[1].s}/i$ are in G_i^a, it suffices to show that the last label of $B_H^{r_i^a[1].s}/i$ is in G_i^a.

Let ℓ_j^b be the last label in $B_H^{r_i^a[1].s}/i$. We now show that $w_j^b[i] \rightarrow r_i^a[j]$ which implies that ℓ_j^b is either *current* or *previous* in $R_{j,i}$ at $r_i^a[j]$, hence $\ell_j^b \in G_i^a$. If the joining time of ℓ_j^b is at $w_j^b[n]$, then $w_j^b[n] \rightarrow r_i^a[1].s$. Since, by the protocol, $r_i^a[1].s \rightarrow r_i^a[j]$, it holds that $w_j^b[i] \rightarrow r_i^a[j]$. Otherwise, ℓ_j^b joins the history graph by another label ℓ_k^c, $j < i \leq k$, whose target label is ℓ_j^b. Since $\ell_j^b \in B_H^{r_i^a[1].s}$ and ℓ_j^b joins the history graph by ℓ_k^c we get that $\ell_k^c \in H^{r_i^a[1].s}$. The proof is completed by the following relations:

1.a. $w_j^b[i] = w_j^b[k]$ — if $k = i$.

1.b. $w_j^b[i] \rightarrow w_j^b[k]$ — according to the protocol, assuming $i < k$.

2. $w_j^b[k] \rightarrow \tilde{r}_k^c[j]$ — since ℓ_j^b is the target label of ℓ_k^c.

3. $\tilde{r}_k^c[j] \rightarrow r_i^a[1].s$ — since $\ell_k^c \in H^{r_i^a[1].s}$.

4. $r_i^a[1].s \rightarrow r_i^a[j]$ — according to the protocol.

which imply $w_j^b[i] \rightarrow r_i^a[j]$. Therefore $\ell_j^b \in G_i^a$, and $B_H^{r_i^a[1].s}/i$ is a subgraph of G_i^a.

Proof of (1.c): Assume by way of contradiction that L_i^a is bad. Since $B_H^{r_i^a[1].s}/i \equiv B_H^t/i$, induction hypothesis (1.a) and (1.b) for i imply that L_i^a is enclosed free and that B_H^t/i is a subgraph of G_i^a. Since L_i^a is enclosed free and since L_i^a is bad, we get that $B_i^a/i \not\equiv B_H^t/i$. Let ℓ_j^b, $j < i$, be the label with the maximal *id* in their common prefix. Label ℓ_j^b is not last in B_i^a/i since B_H^t/i is a subgraph of G_i^a and $B_i^a/i \not\equiv B_H^t/i$. Let (ℓ_k^c, ℓ_j^b), $k < i$, be the incoming edge to ℓ_j^b in B_i^a/i. We now show that (ℓ_k^c, ℓ_j^b) is an edge in H^t: Since L_i^a is enclosed free, Lemma 18 implies that $\ell_k^c \in H^t$. By Lemma 17 there exists some $r \geq 0$, such that (ℓ_k^c, ℓ_j^{b+r}) is an edge in H^t. Since $\ell_j^b \in B_H^t$, induction hypothesis (2) implies that $\ell_j^{b+1} \notin H^t$, that is $r = 0$. Therefore (ℓ_k^c, ℓ_j^b) is an edge in H^t. By the maximality of ℓ_j^b, $(\ell_k^c, \ell_j^b) \notin B_H^t/i$. Let (ℓ_ℓ^d, ℓ_j^b) be the edge excluding (ℓ_k^c, ℓ_j^b) from B_H^t/i. Since B_H^t/i is a subgraph of G_i^a, $(\ell_\ell^d, \ell_j^b) \in G_i^a$, and it excludes (ℓ_k^c, ℓ_j^b) from B_i^a/i, contradiction.

Proof of (2): Assume by way of contradiction that ℓ_i^a does not satisfy the lemma, that is $\ell_i^a \in B_H^{t_0}$ where t_0 is the joining time of ℓ_i^{a+1}. Since $\ell_i^a \in B_H^{t_0}$ it is clear that L_i^{a+1} is bad. We reach the required contradiction by showing that L_i^{a+1} is good. Since the joining time of ℓ_i^a is before $r_i^{a+1}[1].s$ and $\ell_i^a \in B_H^{t_0}$, there are no good actions with *id* smaller then i in the interval $(r_i^{a+1}[1].s, t_0)$. Hence $B_H^{r_i^{a+1}[1].s}/i \equiv B_H^{t_0}/i$ which implies, by (1.c), that L_i^{a+1} is good.

□

The next theorem shows that the protocol satisfies the first design requirement namely that every logical write action is serialized within its serialization interval.

THEOREM 21: Every logical write action L_i^a is serialized within the interval $[\ r_i^a[1].s, w_i^a[n]\ ]$.

Proof: By Definition 9, every logical write action is serialized no later then its joining time. Therefore we only have to show that L_i^a is serialized after $r_i^a[1].s$. Let t be the joining time of ℓ_i^a. If L_i^a is good then it is serialized at t and the theorem follows. Assume L_i^a is bad. By Lemma 20.(1.c) $B_H^{r_i^a[1].s}/i \not\equiv B_H^t/i$, hence there exists a good logical write action L_j^b, whose joining time is after $r_i^a[1].s$ and before t. Therefore L_i^a is serialized either by L_j^b or by another logical write action which is serialized after L_j^b and before t, and the theorem follows. □

The next lemma shows that the protocol satisfies the second design requirement:

LEMMA 22: If L_i^a is enclosed free then $B_H^{r_i^a[1]}/i$ is a subgraph of G_i^a.

Proof: To prove this lemma we show that every label in $B_H^{r_i^a[1]}/i$ belongs to G_i^a. Assume by way of contradiction that there are labels in $B_H^{r_i^a[1]}/i$ that do not belong to G_i^a, and let ℓ_j^b, $j < i$, be the label with maximal *id* among them. Let ℓ_j^{b+r}, ℓ_j^{b+r-1} and ℓ_j^{b+r-2} be the three labels of P_j in G_i^a. By the contradiction assumption $\ell_j^b \notin G_i^a$, therefore either $r < 0$ or $r \geq 3$. We reach the required contradiction by the following case analysis:

Case 1: $r < 0$.

By Definition 5 the joining time of ℓ_j^b is after $i_j^b[i]$. Since $\ell_j^b \in B_H^{r_i^a[1]}/i$, we get that $i_j^b[i] \rightarrow r_i^a[1]$. According to the protocol $r_i^a[1] \rightarrow r_i^a[j]$, hence $i_j^b[i] \rightarrow r_i^a[j]$. Since we assume that $r < 0$, $r_i^a[j] \rightarrow w_j^b[i]$. Therefore $i_j^b[i] \rightarrow r_i^a[j] \rightarrow w_j^b[i]$, and hence ℓ_j^b is the *new* label in $R_{j,i}$ at $r_i^a[j]$. Since $r_i^a[j] \rightarrow w_j^b[i]$ there is no incoming edge to ℓ_j^b in $H^{r_i^a[1]}$ from any label whose $id > i$. Consider the following cases:

Case 1.a: ℓ_j^b **is last in** $B_H^{r_i^a[1]}/i$

Since there is no incoming edge to ℓ_j^b in $H^{r_i^a[1]}$ from any label whose $id > i$ and since ℓ_j^b is last in $B_H^{r_i^a[1]}/i$, we get that ℓ_j^b is last in $B_H^{r_i^a[1]}$. Therefore ℓ_j^b joins the history graph at $w_j^b[n]$, that is $w_j^b[n] \rightarrow r_i^a[1]$, contradiction.

Case 1.b: ℓ_j^b **is not last in** $B_H^{r_i^a[1]}/i$.

Let (ℓ_k^c, ℓ_j^b), $i > k > j$, be the incoming edge to ℓ_j^b in $B_H^{r_i^a[1]}/i$. Since $k > j$ and ℓ_j^b is the label with the maximal *id* that does not satisfy the lemma, $\ell_k^c \in G_i^a$. Since ℓ_j^b is the *new* label in $R_{j,i}$ at $r_i^a[j]$, by the construction of G_i^a, ℓ_j^b joins G_i^a by ℓ_k^c, that is $r \geq 0$, contradiction.

Case 2: $r \geq 3$

In this case we reach a contradiction by showing that the *current* label read in action $r_i^a[j]$, denoted by ℓ_j^c, is enclosed within L_i^a. Note that ℓ_j^c is either ℓ_j^{b+r} or ℓ_j^{b+r-1}. Since ℓ_j^c is the *current* label read in action $r_i^a[j]$, we only have to show that $w_i^a j.c \rightarrow r_j^c[i].c$ Since $\ell_j^b \in B_H^{r_i^a[1]}/i$, Lemma 20(2) implies that $\ell_j^{b+1} \notin H^{r_i^a[1]}$. Therefore $r_i^a[1] \rightarrow w_j^{b+1}[n]$. Since $w_i^a j.c \rightarrow r_i^a[1]$ and since $w_j^{b+1}[n] \rightarrow r_j^{b+2}[i].c$ we get that $w_i^a j.c \rightarrow r_j^{b+2}[i].c$. Since $c \geq b+2$ we get that $w_i^a j.c \rightarrow r_j^c[i].c$. The proof follows.

□

The next theorem shows that all graphs collected by the processors are precedence graphs:

THEOREM 23: If (ℓ_i^a, ℓ_j^b), $i > j$, is an edge in G_k^c then $L_j^b \Rightarrow L_i^a$.

Proof: By Lemma 17 (ℓ_i^a, ℓ_j^{b+r}), $r \geq 0$, is an edge in H. By Definition 5 the joining time of ℓ_j^{b+r} is before the joining time of ℓ_i^a. We show that $L_j^b \Rightarrow L_j^{b+r}$. Since $L_j^b \Rightarrow L_i^a$ the proof follows. Consider the following cases:

Case 1: $i \leq w$
In this case ℓ_j^{b+r} and L_i^a are both write actions. If L_i^a is good or L_i^a and L_j^{b+r} are both bad Definition 9 implies that L_i^a is serialized after L_j^{b+r}. If L_i^a is bad while L_j^{b+r} is good then since (ℓ_i^a, ℓ_j^{b+r}) is an edge in H, there at least one good write action which joins the history graph after L_j^{b+r}. L_i^a is serialized by one of this actions, therefore it is serialized after L_j^{b+r}.

Case 2: $j \leq w < i$
In this case the value written by L_j^{b+r} is returned by L_i^a. By Definition 10, $L_j^{b+r} \Rightarrow L_i^a$.

Case 3: $j > w$
By Definition 10 $L_j^{b+r} \Rightarrow L_i^a$.

□

The next lemma is used in the correctness proof of the reader protocol.

LEMMA 24: Let L_i^a, $i > w$, be an enclosed free read action. If ℓ_j^b, $j \leq w$, is the last label in B_i^a/i and $\ell_j^b \notin B_H^{r_i^a[1]}/i$ then L_j^b is serialized after $r_i^a[1]$.

Proof: Since $\ell_j^b \in B_i^a/(w+1)$ but $\ell_j^b \notin B_H^{r_i^a[1]}/(w+1)$, $B_i^a/(w+1) \not\equiv B_H^{r_i^a[1]}/(w+1)$. Let ℓ_k^c be the label with the maximal id in the common prefix. By Lemma 22 $B_H^{r_i^a[1]}/i$ is a subgraph of G_i^a, hence ℓ_k^c is not last in $B_i^a/(w+1)$ (otherwise $B_i^a/(w+1) \equiv B_H^{r_i^a[1]}/(w+1)$). Let (ℓ_ℓ^d, ℓ_k^c) be the edge incoming to ℓ_k^c in $B_i^a/(w+1)$. Let t be the joining time of ℓ_ℓ^d, we now show that t is after $r_i^a[1]$. Assume by way of contradiction that t is before $r_i^a[1]$, that is, $\ell_\ell^d \in H^{r_i^a[1]}$. By Lemma 17, there exists some r, $r \geq 0$, such that $(\ell_\ell^d, \ell_k^{c+r})$ is an edge in $H^{r_i^a[1]}$. Since $\ell_k^c \in B_H^{r_i^a[1]}/(w+1)$, Lemma 20(2) implies that $\ell_k^{c+1} \notin H^{r_i^a[1]}/(w+1)$. Hence $r = 0$ and (ℓ_ℓ^d, ℓ_k^c) is an edge in $H^{r_i^a[1]}$. Let (ℓ_m^e, ℓ_k^c), $m < \ell$, be the edge in $B_H^{r_i^a[1]}$ excluding (ℓ_ℓ^d, ℓ_k^c) from $B_H^{r_i^a[1]}$. Since $B_H^{r_i^a[1]}/i$ is a subgraph of G_i^a, (ℓ_m^e, ℓ_k^c) is an edge in G_i^a. Therefore (ℓ_m^e, ℓ_k^c) excludes (ℓ_ℓ^d, ℓ_k^c) from B_i^a, contradiction. Hence the joining time and (as we show below) the serialization time of ℓ_ℓ^d is after $r_i^a[1]$. If $L_\ell^d \not\equiv L_j^b$ then there is a path from from ℓ_j^b to ℓ_ℓ^d in G_i^a (because ℓ_j^b and ℓ_ℓ^d are both in B_i^a/i and ℓ_j^b is last in B_i^a/i). By Theorem 23, $L_\ell^d \Rightarrow L_j^b$, therefore L_j^b is serialized after $r_i^a[1]$.

Now we show that L_ℓ^d is serialized after $r_i^a[1]$. If $B_H^{r_i^a[1]}/\ell \not\equiv B_H^t/\ell$, then L_ℓ^d is serialized by the good write action that is serialized last before t, that is after $r_i^a[1]$. Assume that $B_H^{r_i^a[1]}/\ell \equiv B_H^t/\ell$. The following three facts:

1. $B_H^{r_i^a[1]}/i$ is a subgraph of G_i^a,

2. $\ell_k^c \in B_H^{r_i^a[1]}/i$
3. (ℓ_ℓ^d, ℓ_k^c) is an edge in B_i^a/i

imply that ℓ_k^c is the last label in $B_H^{r_i^a[1]}/\ell$ (Otherwise the edge incoming to ℓ_k^c in $B_H^{r_i^a[1]}/\ell$ excludes (ℓ_ℓ^d, ℓ_k^c) from B_i^a/i). Since $B_H^{r_i^a[1]}/\ell \equiv B_H^t/\ell$ it holds that ℓ_k^c is last in B_H^t/ℓ and therefore L_ℓ^d is good. Hence L_ℓ^d is serialized at t, that is after $r_i^a[1]$. The lemma follows. □

THEOREM 25: Let L_i^a be a logical read action, and let L_j^b be the logical write action which wrote the value returned by L_i^a. L_j^b is the most recent write action serialized before L_i^a.

Proof: We prove this theorem by induction on i, the *id* of logical readers.

Base: $i = w + 1$

Let ℓ_j^{b-r}, $r \geq 0$, be ℓ_{last} computed in L_i^a. Consider the following cases:

Case 1: L_i^a is enclosed free

In this case ℓ_j^{b-r} is the last label in B_i^a/i. By Lemma 22, $B_H^{r_i^a[1]}/i$ is a subgraph of G_i^a. We show that either L_j^{b-r} is serialized after $r_i^a[1]$ or L_j^{b-r} is the most recent write action serialized before $r_i^a[1]$. If $\ell_j^{b-r} \notin B_H^{r_i^a[1]}/i$ then by Lemma 24 L_j^{b-r} is serialized after $r_i^a[1]$. If $\ell_j^{b-r} \in B_H^{r_i^a[1]}/i$ then since $B_H^{r_i^a[1]}/i$ is a subgraph of G_i^a, and since ℓ_j^{b-r} is the last label in B_i^a/i, ℓ_j^{b-r} is last in $B_H^{r_i^a[1]}/i$. Hence L_j^{b-r} is the most recent write action serialized before $r_i^a[1]$. Since $L_j^{b-r} \Rightarrow L_j^b$ (if $r > 0$), the proof follows.

Case 2: L_i^a is not enclosed free

In this case, ℓ_j^{b-r} is the label with the maximal *id* enclosed in L_i^a. By Proposition 14, it holds that $r_i^a[1].s \rightarrow r_j^{b-r}[1].s$. By Theorem 21, L_j^{b-r} is serialized after $r_j^{b-r}[1].s$. Hence L_j^{b-r} is serialized after $r_i^a[1].s$. Since $L_j^{b-r} \Rightarrow L_j^b$ (if $r > 0$) L_j^b is serialized after $r_i^a[1].s$. By Definition 10, L_i^a is serialized by L_j^b, that is, L_j^b is the most recent write action serialized before L_i^a.

Step: Assume correctness for m, $w < m < i$. We now prove for i

If ℓ_j^b is the *ret_label* of L_i^a then the proof is identical to the proof of the induction base. Otherwise let ℓ_k^c, $w < k < i$, be the *ret_label* of L_i^a, and let ℓ_k^{c-r}, $r \geq 0$, be the label ℓ_{last} computed during L_i^a. In this case L_k^c is a logical read action which also returns the value written by L_j^b. Consider the following cases:

Case 1: L_i^a is enclosed free

By the induction assumption L_j^b is the most recent write action serialized before L_k^c. If L_k^c is serialized after $r_i^a[1].s$ then since L_i^a is serialized by L_k^c, we are done. Assume that L_k^c is serialized before $r_i^a[1].s$. According to Definition 10, L_i^a is serialized at $r_i^a[1].s$. We claim that there are no good write actions serialized after L_k^c and before L_i^a (that is, before $r_i^a[1].s$), and therefore the proof of this case follows. Assume by way of contradiction that there are such good writes. Let ℓ_ℓ^d, $\ell \leq w$, be the last label in $B_H^{r_i^a[1]}/(w+1)$. By this definition $L_j^b \Rightarrow L_\ell^d$. By Lemma 22 $B_H^{r_i^a[1]}/i$ is a subgraph of G_i^a. Since $\ell_\ell^d \in B_H^{r_i^a[1]}/(w+1)$ we get that $\ell_\ell^d \in G_i^a$. In both of the following cases we reach the required contradiction:

Case 1.1: $\ell_\ell^d \in B_i^a/i$

Since ℓ_k^{c-r} is last in B_i^a/i, there is a path from ℓ_k^{c-r} to ℓ_ℓ^d in G_i^a. Therefore by Theorem 23 $L_\ell^d \Rightarrow L_k^{c-r}$. Since $r \geq 0$ we get that $L_j^b \Rightarrow L_\ell^d \Rightarrow L_k^c$, Hence, L_j^b is not the last write action serialized before L_k^c, contradiction to the induction assumption.

Case 1.2: $\ell_\ell^d \notin B_i^a/i$

In this case there is some label ℓ_m^e , $m < \ell$, that excludes the suffix of $B_H^{r_i^a[1]}/i$ (and ℓ_ℓ^d) from B_i^a/i. Therefore $L_\ell^d \Rightarrow L_m^e$. Since $\ell_m^e \in B_i^a/i$ and ℓ_k^{c-r} is last in B_i^a/i, Theorem 23 implies that $L_m^e \Rightarrow L_k^{c-r}$. Therefore we get $L_j^b \Rightarrow L_\ell^d \Rightarrow L_m^e \Rightarrow L_k^{c-r} \Rightarrow L_k^c$ and in particular $L_j^b \Rightarrow L_m^e \Rightarrow L_k^c$, contradiction to the induction assumption.

Case 2: L_i^a **is not enclosed free**

In this case ℓ_k^{c-r} is the label with the maximal *id* enclosed in L_i^a. By Definition 8 it holds that $r_i^a[1].s \rightarrow r_k^{c-r}[1].s$ therefore L_k^{c-r} is serialized after $r_i^a[1].s$. Since $r \geq 0$ we get that L_k^c is serialized after $r_u^a[1].s$. By the induction assumption, L_j^b is the most recent write action serialized before L_k^c. Since L_i^a is serialized by L_k^c, L_j^b is the most recent write action serialized before L_i^a.

□

ACKNOWLEDGMENTS

We thank Paul Vitányi, for his tireless efforts to convince us that implementations with sublinear space complexity are worth looking at. We also thank Yael Gafni, Arie Rudich comments on an earlier version have helped us in this presentation. Last but surely not least is John Tromp whose continuous help during this work was invaluable in terms of both correctness and style.

References

[Ab91a] U. Abraham, "MultiWriter Atomic Registers and bounded timestamps," preprint.

[Ab91b] U. Abraham, "On Interprocess Communication and the Implementation of a Multi-Writer Atomic Registers," preprint.

[CS90] R. Cori and E. Sopena "Some combinatorial aspects of Time Stamps Systems," submitted to J. Of Algorithms.

[IL87] A. Israeli, and M. Li, "Bounded Time-Stamps," Proceedings of the 28th Annual Symposium on Foundations of Computer Science, 1987, pp. 371-382.

[ILV87] A. Israeli, M. Li, and P. Vitányi, "Simple Multireader registers using Time-Stamp schemes," Report no. CS-R8758 Center for Mathematics and Computer Science, Amsterdam, Holland, November 1987.

[ITV92] A. Israeli, J. Tromp, and P. Vitányi, personal communication.

[La86a] L. Lamport, "On Interprocess Communication. Part I: Basic Formalism," Distributed Computing 1, 2 1986, pp. 77-85.

[La86b] L. Lamport, "On Interprocess Communication. Part II: Algorithms," Distributed Computing 1, 2 1986, pp. 86-101.

[LTV90] M. Li, J. Tromp, and P. Vitányi, "How to Share Concurrent Wait-Free Variables," Tech. Rept. CS-R8916, CWI, April 1989. Submitted to J.ACM/revision. (Prelim. Version in ICALP89)

[LV90] M. Li and P. Vitányi, "Optimality of Wait-Free Atomic Multiwriter Variables," Submitted to Information Processing Letters 1990.

[CS90] R. Cori and E. Sopena "Some combinatorial aspects of Time Stamps Systems," submitted to J. Of Algorithms.

[PB87] G.L. Peterson and J.E. Burns, "Concurrent reading while writing II: the multiwriter case", 28th Annual IEEE Symp. on Foundations of Computer Science, 1987, pp. 383-392.

[Sc89] R.W. Schaffer, "On the Correctness of Atomic Multi-Writer Registers", Technical Report MIT/LCS/TM-364, Laboratory for Computer Science, MIT.

[T92] J. Tromp, "On Update-Last Schemes", preprint.

[VA86] P. Vitányi, and B. Awerbuch, "Atomic Shared Register Access by Asynchronous Hardware," Proceedings of the 27th Annual Symposium on Foundations of Computer Science, 1986, pp.233-243.