SIAM J. COMPUT.
Vol. 26, No. 2, pp. 418–455, April 1997

007

BOUNDED CONCURRENT TIME-STAMPING*

DANNY DOLEV† AND NIR SHAVIT‡

Abstract. We introduce concurrent time-stamping, a paradigm that allows processes to temporally order concurrent events in an asynchronous shared-memory system. *Concurrent time-stamp systems* are powerful tools for concurrency control, serving as the basis for solutions to coordination problems such as mutual exclusion, ℓ-exclusion, randomized consensus, and multiwriter multireader atomic registers. Unfortunately, all previously known methods for implementing concurrent time-stamp systems have been theoretically unsatisfying since they require unbounded-size time-stamps—in other words, unbounded-size memory.

This work presents the first bounded implementation of a concurrent time-stamp system, providing a modular unbounded-to-bounded transformation of the simple unbounded solutions to problems such as those mentioned above. It allows solutions to two formerly open problems, the bounded-probabilistic-consensus problem of Abrahamson and the *fifo*-ℓ-exclusion problem of Fischer, Lynch, Burns and Borodin, and a more efficient construction of *multireader multiwriter* atomic registers.

Key words. atomic registers, serialization, concurrency, time-stamping, distributed computing, parallel computing

AMS subject classifications. 68Q22, 05C90, 05C99

PII. S0097539790192647

1. Introduction.

A *time-stamp system* is like a ticket machine at an ice cream parlor. People's requests to buy the ice cream are time-stamped based on a numbered ticket (label) taken from the machine. In order to know the order in which requests will be served, a person need only scan through all the numbers and observe the order among them. A *concurrent* time-stamp system (CTSS) is a time-stamp system in which any process can either take a new ticket or scan the existing tickets simultaneously with other processes. A CTSS is required to be *wait-free*, which means that a process is guaranteed to finish any of the two above-mentioned label-taking or scanning tasks in a finite number of steps, even if other processes experience stopping failures. Wait-free algorithms are highly suited for fault-tolerant and real-time applications (see Herlihy [Her91]).

Concurrent time-stamping is the basis for simple solutions to a wide variety of problems in concurrency control. Examples of such algorithms include Lamport's *first-come first-served* mutual exclusion [Lam74], Vitanyi and Awerbuch's construction of a multireader multiwriter (MRMW) atomic register [VA86], Abrahamson's randomized consensus [Abr88], and Fischer, Lynch, Burns, and Borodin's *fifo*-ℓ-exclusion problem [FLBB79, FLBB89] (also see [AD*94]).

* Received by the editors December 10, 1990; accepted for publication (in revised form) May 18, 1995. A preliminary version of this paper appeared in *Proc. 21st Annual ACM Symposium on Theory of Computing*, ACM, New York, 1989, pp. 454–465.

http://www.siam.org/journals/sicomp/26-2/19264.html

† IBM Almaden Research Center, K53/802, 650 Harry Road, San Jose, CA 95120-6099 (dolev@almaden.ibm.com) and Institute of Mathematics and Computer Science, Hebrew University, Givat-Ram, Jerusalem 91906, Israel (dolev@cs.huji.ac.il).

‡ Department of Computer Science, Hebrew University, Givat-Ram, Jerusalem 91906, Israel. Current address: Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel (shanir@cs.tau.ac.il). The research of this author was supported by a Libnitz Foundation Scholarship, the Israeli Communications Ministry Award, NSF contract CCR-8611442, ONR contract N0014-85-K-0168, DARPA contract N00014-83-K-0125, and a special grant from IBM. Parts of this research were also conducted while this author was visiting the Theory of Distributed Systems group at MIT, AT&T Bell Laboratories, and the IBM Almaden Research Center.

Unfortunately, the only formerly known implementation of the CTSS paradigm using read/write registers was a version of Lamport's "bakery algorithm," which uses labels of unbounded size [Lam74]. Researchers were thus led to devise complicated problem-specific solutions to show that the above problems are solvable in a bounded way.[1]

In [IL93], Israeli and Li were the first to isolate the notion of bounded time-stamping (time-stamping using bounded-size memory) as an independent concept, developing an elegant theory of bounded *sequential* time-stamp systems. Sequential time-stamp systems prohibit concurrent operations. This work was continued in several interesting papers on sequential systems with weaker ordering requirements by Li and Vitanyi [LV87], Cori and Sopena [CS93], and Saks and Zaharoglou [SZ91].

This paper introduces the *concurrent time-stamping* paradigm and provides the first bounded construction of a concurrent time-stamp system. It provides a modular unbounded-to-bounded transformation, enabling the design of simple unbounded concurrent-time-stamp-based algorithms to problems such as those mentioned above, with the knowledge that each unbounded solution immediately implies a bounded one. Our work allows solutions of the above flavor to two formerly open problems, the bounded-randomized-consensus problem of [Abr88] (which requires one to solve the randomized-consensus problem of [CIL87] without using an atomic coin-flip operation) and the *fifo*-ℓ-exclusion problem of [FLBB79, FLBB89] (see [AD*94] for details). A bounded CTSS solution to the former problem is given in [Sha90], and in [AD*94], Afek et al. use a CTSS to provide the first bounded solution to the latter problem.[2]

Though one might think that the price of introducing a modular unbounded-to-bounded transformation would be a blowup in memory size or number of operations, this is hardly the case. For an n-process system, the construction presented in this paper requires only n registers of $O(n)$ bits each, meeting the lower bound of [IL93] for sequential-time-stamp-system construction. The time complexity is $O(n)$ operations for an update and $O(n^2 \log n)$ for a scan. (Like the unbounded algorithm, the scan consists only of read operations, i.e., no writes.)

One example of the efficiency of the CTSS solutions is given by the famous problem of multireader multiwriter atomic register construction. A simple solution based on transforming the unbounded protocol of Vitanyi and Awerbuch [VA86] using our construction (see [Sha90, G92]) has the same space complexity of the [PB87, Sch88] algorithm, yet it has a better time complexity—$O(n)$ memory accesses for a write, $O(n \log n)$ for a read, as compared with $O(n^2)$ for either in the former solutions. Our implementation is the only known bounded construction of an MRMW atomic register from single-writer multireader (SWMR) atomic registers where the implementation of the MRMW read operation does not require a process to perform an SWMR write. The importance of the readers-do-not-write property was first raised by Lamport in [Lam86a], where he showed the impossibility of a bounded construction where readers do not write of a single-writer single-reader (SWSR) atomic register from SWSR regular ones. Moreover, as explained in [AD*94], this property is important when defining liveness conditions such as *first-come first-enabled* for problems like ℓ-exclusion.

The structure of our presentation is as follows. We begin by describing concurrent time-stamping (sections 2 and 3), first formally using Lamport's axiomatic approach

[1] See [And89a, Blo88, BP87, CIL87, Dij65, DGS88, FLBB79, FLBB89, Kat78, Lam74, Lam77, Lam86b, LH89, LV87, **?**, Ray86, Pet81, Pet83, PB87, VA86].

[2] The only prior known solutions to the *fifo*-ℓ-exclusion problem [DGS88, Pet88] achieve weaker forms of fairness than the original *test-and-set*-based solution of [FLBB79].

[Lam86c, Lam86a] and then informally through a simple unbounded-memory implementation. In sections 4.3 and 4.4, the bounded wait-free CTSS implementation is described. Section 5 provides the final details of the formal specification and the main parts of the proof of the bounded CTSS implementation are presented. Section 6 describes the implications of a bounded CTSS construction on various interprocess-communication problems and gives a summary of research following our work. For berevity, some of the more tedious parts of the correctness proof have been omitted and can be found in [Sha90].

2. A concurrent time-stamp system. The following is a formal definition of a CTSS for a system of processes numbered $1, \ldots, n$. It uses the axiomatic specification formalism of Lamport [Lam86c, Lam86a]. The reader may benefit by checking how the formal properties described below are met by the unbounded implementation described in the next section.

A CTSS is a problem specification with an operational interface. A CTSS that permits n concurrent operations has $2n$ operation types, specifically, $labeling_i(\ell_i)$ and $scan_i(\bar{\ell}, \prec)$ for $i \in \{1, \ldots, n\}$. A $labeling_i$ operation associates an input *value*, ℓ_i, taken from any domain $\mathcal{D}$ with a label.[3] We call ℓ_i the *labeled-value* of operation $labeling_i$. In an application such as an atomic-register construction, the labeled-value would be the value written to the register, while in a mutual-exclusion-type application, where the input values are unimportant, it would be null. A $scan_i$ operation returns as output a pair $(\bar{\ell}, \prec)$, where the *view* $\bar{\ell} = \{\ell_1, \ldots, \ell_n\}$ is an indexed set of labeled-values (one per process) and $\prec$ is a *total order* on these indexes.

Assume that each process' program consists of these two operations, whose execution generates a sequence of *elementary operation executions*, totally ordered by the *precedes* relation (of [Lam86c, Lam86a], denoted "$\longrightarrow$") and where any number of scan operation executions are allowed between any two labeling operation executions. The following,

$$L_i^{[1]} \longrightarrow S_i^{[1]} \longrightarrow L_i^{[2]} \longrightarrow L_i^{[3]} \longrightarrow S_i^{[2]} \longrightarrow S_i^{[3]} \longrightarrow S_i^{[4]} \longrightarrow \cdots,$$

is an example of such a sequence by process i, where $L_i^{[k]}$ denotes process i's kth execution of a labeling operation and $S_i^{[k]}$ is the kth execution of a scan operation. (The superscript $[k]$ is used for notation and is not visible to the processes.) The labeled-value input in each labeling operation execution $L_i^{[k]}$ is denoted by $\ell_i^{[k]}$.[4] A *global-time model* of operation executions is assumed, implying that for any two operation executions, $a \longrightarrow b$ or $b \dashrightarrow a$. (For more details, see section 5.1.)

The elementary operation executions of a CTSS must have following set of properties.

P1: *ordering*. There exists an irreflexive total order $\Longrightarrow$ on the set of all labeling operation executions such that we have the following:

a: *precedence*. For any pair of labeling operation executions $L_p^{[a]}$ and $L_q^{[b]}$ (where p and q are possibly the same process), if $L_p^{[a]} \longrightarrow L_q^{[b]}$, then $L_p^{[a]} \Longrightarrow L_q^{[b]}$.

b: *consistency*. For any scan operation execution $S_i^{[k]}$ that returns $(\bar{\ell}, \prec)$, $p \prec q$ if and only if $L_p^{[a]} \Longrightarrow L_q^{[b]}$.

[3] In order correctly handle initial conditions, the value domain $\mathcal{D}$ must specify some initial value.

[4] In order for a unique labeled-value $\ell_i^{[k]}$ to be associated with each label operation execution $L_i^{[k]}$, the reader can think of $\ell_i^{[k]}$ as a triplet $\langle \ell_i^{[k]}, i, k \rangle$, where the second and third fields are dummy indexes used only for purposes of the specification.

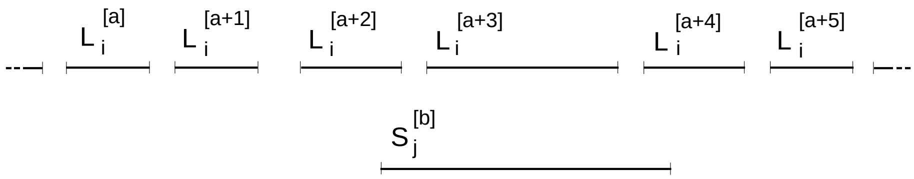

FIG. 1. *Regularity.*

Property P1 formalizes the idea that a CTSS can be envisioned as a black box, inside of which hides a mechanism (a logical clock) associating causally ordered time stamps—from an infinite totally ordered range—with each of the labeled-values entered in labeling operations, and where scanning is like peeping into this black box, each scan returning a view of a part of this hidden ordering.[5] The black box metaphor is used to stress that it suffices to know of the existence of such a total ordering $\Longrightarrow$, while the ordering itself need not be known.

One should bear in mind that the asynchronous nature of the operations allows situations where a scan operation execution overlaps many consecutive labeling operation executions of other processes. Also, several consecutive scans could possibly be overlapped by a single labeling operation execution. It is therefore important that a requirement be made that the view $\bar{\ell}$ returned by $S_i^{[k]}$ be a meaningful one, namely, that it reflect the ordering among labeling operation executions immediately before or concurrent with the scan, and not just any possible set of labeled-values. (In the example of Figure 1, any of the labeled-values $\ell_x^{[a+1]}$ through $\ell_x^{[a+4]}$ can be returned by $S_i^{[k]}$, but not those preceding or following them.) This will eliminate uninteresting trivial solutions and introduce a measure of liveness into the system. This requirement is formalized in the following definition, where $\dashrightarrow$ is the *can affect* relation of [Lam86c, Lam86a].

P2: *regularity.* For any labeled-value $\ell_p^{[a]}$ in $\bar{\ell}$ of $S_i^{[k]}$, $L_p^{[a]} \dashrightarrow S_i^{[k]}$, and there is no $L_p^{[b]}$ such that $L_p^{[a]} \longrightarrow L_p^{[b]} \longrightarrow S_i^{[k]}$.

Although such a *regular* concurrent time-stamp system as P1–P2 would suffice for some applications (as in Lamport's "bakery algorithm" [Lam74]), a more powerful *monotonic* concurrent time-stamp system will be needed in applications such as the *multireader multiwriter* atomic register construction (as in [LV87, VA86]). zTo this end, the following third property is added.

P3: *monotonicity.* For any labeled-value $\ell_p^{[a]}$ in $\bar{\ell}$ of $S_i^{[k]}$, there does not exist an $S_j^{[k']}$ with a labeled-value $\ell_p^{[b]}$ in its view $\bar{\ell}'$, such that $S_i^{[k]} \longrightarrow S_j^{[k']}$ and $L_p^{[b]} \longrightarrow L_p^{[a]}$ (possibly $i = j$).

Monotonicity is the property that in the unbounded natural-number CTSS can be described by saying that the labels of any one process, as read by increasingly later scans, are "monotonically nondecreasing." In other words, later scans cannot read labels smaller than those read by earlier ones. It is important to note, however, that P3 does not imply that labeling and scan operation executions of all processes are serializable, that is, appear to happen atomically. (Figure 2 shows two scan operations that meet property P3 that cannot be serialized.) It does, however, imply the serializability of the scan operation executions of all processes relative to the labeling operation executions of any *one* process.

[5] Notice that there is no requirement that labeled-values returned by different scans must be comparable.

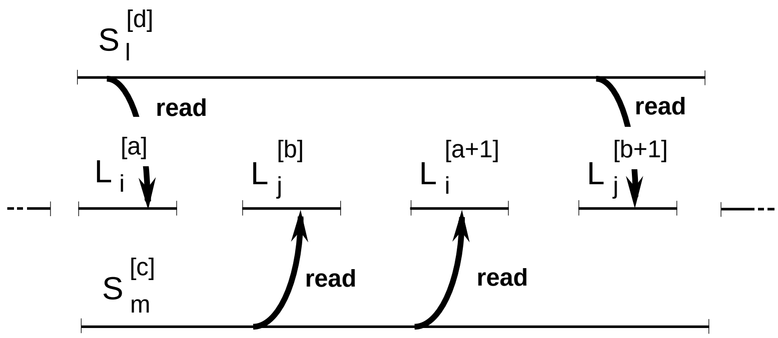

FIG. 2. *Monotonicity does not imply atomicity.*

Property P4 is an extension of part of the regularity property to the $\Longrightarrow$ order.[6] Properties P3 and P4 together imply that all *scan* operations that consider only the "largest" value, where "largest" is based on the $\prec$ ordering, can be serialized with respect to all labeling operations.

P4: *order regularity.* For any labeled-value $\ell_p^{[a]}$ in $\bar{\ell}$ of $S_i^{[k]}$, $S_i^{[k]} \longrightarrow L_q^{[b]}$ implies that $L_p^{[a]} \Longrightarrow L_q^{[b]}$.

3. Unbounded concurrent time-stamping. The basic communication primitive used in our implementations is a single writer multireader atomic register. Our goal is to design an implementation that is *wait-free* [Her91, AG90]: each process' *scan* or *label* operation execution consists of a bounded number of SWMR register operations independently of the pace or type of operations carried out by other processes. Wait-free constructions of SWMR atomic registers from weaker primitives have been shown in [BP87, IL93, Lam86d, SAG94, New87].

We begin with the following simple implementation of a CTSS using SWMR registers of unbounded size. The concurrent time stamp system will consist of n SWMR atomic registers v_i, $i \in \{1..n\}$. Each v_i is written by process i and read by all. Each *labeling*$_i$ operation writes ℓ_i to register v_i. In our implementation, ℓ_i is a data type consisting of two fields, a labeled-value, denoted *value*(ℓ_i), and its associated label, denoted *label*(ℓ_i). Each *label*(ℓ_i) is a pair of the form $(number_i, i)$, where $number_i$ is a natural number and $i \in \{1..n\}$ is the id of the process writing ℓ_i.

A process i collects the labels and values of other processes by performing a *collect* operation, a reading of all the registers v_j, $j \in \{1..n\}$, once each, in some arbitrary order. The collect operation returns an indexed set $\ell = \{\ell_1, \ldots, \ell_n\}$, that is, one value and associated label per process. The collected elements in ℓ are ordered by $ord(\ell)$, an ordering on their indexes in $\{1..n\}$, such that i is smaller than j if and only if the label $(number_i, i)$ is lexicographically smaller than the label $(number_j, j)$. Figure 3 provides the pseudocode of the *labeling* and *scan* operations for a process i.

To understand how property P1 is met, consider that if the labeling operation execution of ℓ_i by a process i completely preceded the labeling operation execution of ℓ_j by j, then it must be that j chose a label with $number_j > number_i$ since j collected ℓ_i. If they are concurrent, at worst they might both collect the same maximal label and choose $number_i = number_j$, in which case they are ordered by their ids. Thus

[6] The need for property P4 in applications such as the multireader multiwriter atomic register construction of [LV87, VA86] was discovered by Gawlick [G92].

procedure *labeling*(*val*);
  begin
    $\ell :=$ *collect*;
    $v_i := (val,\ (\max_{j\in\{1..n\}} number_j + 1, i))$;
  end;
function *scan*;
  begin
    $\ell :=$ *collect*;
    return $(\{value(\ell_1) \dots value(\ell_n)\}, ord(\ell))$;
  end;

FIG. 3. *The unbounded* natural-number-*based implementation.*

the lexicographic order on the labels defines a linearization order [HW88] on the concurrent labeling operation executions, that is, an order $\Longrightarrow$ by which they can be thought of as happening sequentially in time. The reader can convince herself that properties P2–P4 follow directly from the use of SWMR atomic registers in the implementation.

It is important to note that the actual $label(\ell_1) \dots label(\ell_n)$ used in computing $ord(\ell)$ are hidden from the user (scan operations do not return them), and there is thus no way to compare the order among a pair of values returned by different scans.

4. A bounded concurrent time-stamp system.

4.1. Labels and precedence. The bounded implementation presented will be of the exact same form as the unbounded natural-number-based one. The concurrent time-stamp system will consist of n SWMR atomic registers v_i, $i \in \{1..n\}$, each v_i written by process i and read by all. Each value ℓ_i written to register v_i consists, just as in the unbounded case, of two fields, a *labeled-value*, to which the input of a labeling operation is written, and an associated *label*.

Note. In what follows, almost all of the discussion involves only the label field of v_i and not its labeled-value field. In order to simplify the exposition, we choose, with few exceptions, to ignore the existence of the labeled-value field and deal only with the associated label field. Thus, for example, the notation $\ell_i^{[k]}$ will represent only the label field written in a labeling operation execution $L_i^{[k]}$. We trust that the interested reader will be able to add the relevant operations regarding the labeled-value, as in the unbounded implementation in section 3.

Let V denote the range of possible labels and $\precneq$ denote an irreflexive and antisymmetric relation among them. In the unbounded natural-number implementation of a CTSS, V is just the unbounded size set of pairs of natural numbers and integers in $\{1..n\}$ and $\precneq$ is the lexicographic total ordering among them. In the following sections, the set of possible label values V of the implementation, together with a relation $\precneq$ among them, are defined in terms of a *precedence graph*[7] $(V, \precneq)$. Each possible label is a node in this graph. The order among the labels in any two registers is the order $\precneq$ established by the edges of the precedence graph. A tournament is a complete directed graph. The precedence graph representing labels of the natural-number-based implementation is an acyclic tournament of unbounded size, i.e., a total

[7] The elegant idea of defining the labels and ordering as a tournament graph was introduced by Israeli and Li in [IL93].

order. The definition of the precedence graph will provide the basis for describing the implementation of the labeling and scan operations.

4.2. A bounded precedence graph. The following is the description of the *precedence graph* T^n (see Figure 4). Unlike the unbounded precedence graph defined by the natural numbers, T^n contains cycles.

Define "A *dominates* B in G," where A and B are two subgraphs of a graph G (possibly single nodes), to mean that every node of A has edges directed to every node of B. Define the following generalization of the composition operator of [IL93]. The α-*composition*, $G \circ_\alpha H$, of two graphs G and H, where α is a subset of the nodes of G, is the following noncommutative operation:

> Replace every node $v \in \alpha$ of G by a copy of H (denoted H_v), and let H_v (or v) dominate H_u in $G \circ_\alpha H$ if v dominates u in G.

Define the graph T^2 to be the following graph of five nodes: a cycle of three nodes $\{3, 4, 5\}$, where 3 dominates 5, which dominates 4, which in turn dominates 3, all dominating the nodes $\{2, 1\}$, and where node 2, in turn, dominates node 1.

Define the graph T^k (a tournament) inductively as follows:

1. T^1 is a single node.
2. $T^k = T^2 \circ_\alpha T^{k-1}$, where $\alpha = \{5, 4, 3, 1\}$ and $k > 1$.

The graph $T^n = (V, \overset{\nu}{\prec})$ is the precedence graph to be used in the implementation of the labeling and scan algorithms of a concurrent time-stamp system for n processes. For any process i, each node in T^n corresponds to a uniquely defined label value ℓ_i. The label can be viewed as a string $\ell_i[n..1]$ of n digits, where each $\ell_i[k] \in \{1, \dots, 5\}$ is the digit of the corresponding node in T^2, replaced by a T^k subgraph during the kth step of the inductive construction above. The digit $\ell_i[n]$ is always 1, representing the complete T^n graph, and if in ℓ_i, $\ell_i[k] = 2$, then $\ell_i[j] = 1$ for all $j \in \{k-1..1\}$ (since node 2 is never expanded in the induction step). Therefore, given any label ℓ_i, the T^k subgraph of T^n in which its corresponding node is located is identified by the corresponding prefix $\ell_i[n..k]$.

To assure that based on the graph T^n a total ordering among the label values returned by a scan can be established, we need to break symmetry among processes having the same label. Thus the label ℓ_i is assumed to be concatenated with the id of process i, where label and id are lexicographically ordered. (In terms of the graph T^n, this amounts to no more than assuming that each T^1 graph consists of a total order tournament of n nodes, each process i always choosing the ith node in the order. For simplicity, this point is not further elaborated upon in what follows.)

4.3. The labeling operation. Recall that the *collect* operation by any process i is a reading of all the registers v_j, $j \in \{1..n\}$, once each, in an arbitrary order, returning a set ℓ of labels. The *labeling* operation of a process i is of the form described in Figure 5, where $\mathcal{L} : V^n \times \{1..n\} \mapsto V$ is a *labeling function*, returning a label value ℓ_i "greater than" all other label values.[8] This is the same form as the natural number CTSS, where the labeling function $\mathcal{L}$ returns $(\max_{j\in\{1..n\}} number_j + 1, i)$. However, the interpretation of being "greater than" is not as straightforward as in the natural-number case.

The definition of the labeling function $\mathcal{L}(\ell, i)$ presented below is based on a recursively defined function $\mathcal{L}^k(G, \ell, \ell_{\max})$, which, given a T^k subgraph G of T^n, a set of labels ℓ, and a "maximal" label $\ell_{\max} \in \ell$ in T^k, returns the label of a node in G

[8] Initially, all labels are on node 111..11, the node dominated by all others in T^n.

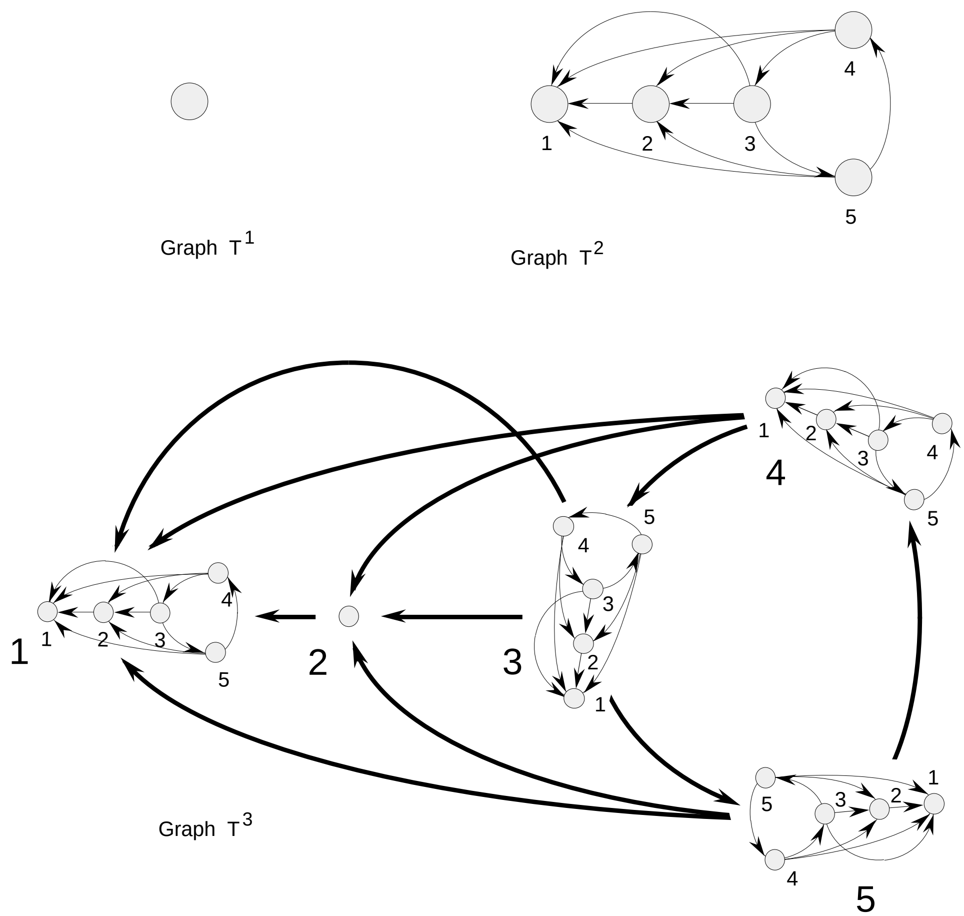

FIG. 4. *The precedence graph.*

procedure *labeling*(*val*);
    begin
        $\ell := collect$;
        $v_i := (val, \mathcal{L}(\ell, i))$;
    end;

FIG. 5. *The labeling operation.*

that is "greater than" the other labels. The reader may benefit by going through Examples 4.1, 4.2, and 4.3 before or during the reading of the code in Figure 6. For simplicity, and since the collected set of labels ℓ remains unchanged in $\mathcal{L}(\ell, i)$ once it is collected (similarly for the variable $\ell_{\max}$ once it is computed), it is treated as a global variable and is not passed as a parameter in all of the utility functions used by $\mathcal{L}(\ell, i)$. The following functions are used in defining $\mathcal{L}$:

num_labels(G)—a function that, for the given label set ℓ, returns how many of the labels are in subgraph G;

function $\mathcal{L}\,(\ell, i)$;
  function $\mathcal{L}^k(G)$;
  begin
  1: **if** $k\!=1$ **then return** G;
  2: **if** $\ell_{\max}[n..k] \neq G$
    then return $\mathcal{L}^{k-1}(G.1)$;
  3: **if** $\ell_{\max}[n..k\!-\!1] = G.2$
    then return $\mathcal{L}^{k-1}(G.3)$;
  4: **if** $k > 2$ **then**
    if $\ell_{\max}[k\!-\!2] \in \{2, 3, 4, 5\}$ **and**
     $(\ell_i[n..k\!-\!1] \neq \ell_{\max}[n..k\!-\!1])$
    then return $\mathcal{L}^{k-1}(G.dom(\ell_{\max}[k\!-\!1]))$;
  5: **if** $(num\_labels(\ell_{\max}[n..k\!-\!1]) < k\!-\!1)$ **or**
    $((num\_labels(\ell_{\max}[n..k\!-\!1]) = k\!-\!1)$ **and**
    $(\ell_i[n..k\!-\!1] = \ell_{\max}[n..k\!-\!1]))$
    then return $\mathcal{L}^{k-1}(G.\ell_{\max}[k\!-\!1])$
    else return $\mathcal{L}^{k-1}(G.dom(\ell_{\max}[k\!-\!1]))$;
  end $\mathcal{L}^k$;
begin
  $\ell_{\max} := max(dominating\_set(\ell, \ell_i))$;
  return $\mathcal{L}^n(T^n)$;
end $\mathcal{L}$;

FIG. 6. *The labeling function.*

$dom(x)$—a function that, for a given digit $x \in \{1..5\}$ representing a node in the graph T^2, returns the next dominating node, namely, $dom(1) = 2$, $dom(2) = 3$, $dom(3) = 4$, $dom(4) = 5$, and $dom(5) = 3$;

$dominating\_set(\hat{\ell}, \ell_i)$—a function that, for a set of labels $\hat{\ell} \subseteq \ell$ and a label $\ell_i \in \hat{\ell}$, returns a subset of labels $\{\ell_j \in \hat{\ell} \,|\, \ell_i \overset{\vee}{\prec} \ell_j\} \cup \{\ell_i\}$; and

$max(\hat{\ell})$—a function that, for a set of labels $\hat{\ell} \subseteq \ell$, returns a label

$$(\ell_x \in \hat{\ell} \,:\, |dominating\_set(\hat{\ell}, \ell_x)| \leq |dominating\_set(\hat{\ell}, \ell_j)|,\ \forall \ell_j \in \hat{\ell}\,),$$

the maximal label, i.e., the one least dominated within this set.

Define $G.x$ to be the *concatenation* of string G and digit x. Figure 6 is thus the definition of the labeling function $\mathcal{L}(\ell, i)$, where the parameter subgraphs G are identified with the relative label prefixes and T^n is identified with the label 1. To give the reader some intuition about the properties of the labeling operation, let it be assumed that one can talk about the values of the labels of all processes at "points in time." To show how the labeling operation executions allow us to define the order $\Longrightarrow$, we will first argue informally that they meet a much simpler requirement, namely, that at any point in time, the following hold:

R1. The labels reflect the precedence among nonconcurrent labeling operation executions.

R2. The subgraph of the precedence graph T^n induced by the labeled nodes (those whose corresponding label is written in some v_i) contains no cycle.

Since T^n is a tournament, R2 implies that at any point in time, all labels are totally ordered. One should notice that these two requirements are easily met by the unbounded implementation since for any $n - 1$ nodes, one can always choose a

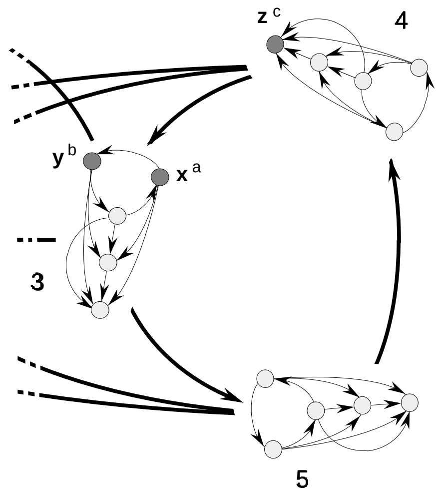

Fig. 7. *Starting state for the examples.*

dominating node in an unbounded total order graph in order to maintain R1, and this will never impair R2 because the graph does not contain cycles.

Let us begin by showing that the labeling operation executions maintain the following two "invariants" at any point in time:

(1) There are labels on at most two of the three nodes in any cycle of any subgraph T^k. (The cycle consists of "supernodes" $\{3, 4, 5\}$, called supernodes since they are actually T^{k-1} subgraphs.)

(2) There are no more than k labels in the cycle of any subgraph T^k.

Maintaining the second invariant is the key to maintaining the first, and the first implies R2.

The manner by which the invariance of (1) and (2) is preserved is explained via several examples. In these examples, T^3 is a precedence graph for a system of three processes x, y, and z. As shown in Figure 7, all of the examples start at a point in time where $\ell_y^{[b]} = 134$, $\ell_x^{[a]} = 135$, and $\ell_z^{[c]} = 141$, that is, all labels are totally ordered by $\overset{v}{\prec}$. In the figure, a label such as $\ell_x^{[a]} = 135$ is denoted by shading node 135 and denoting it with the mark x^a.

Example 4.1. Assume that the following sequence of labeling operation executions occur sequentially. Process y performs $L_y^{[b+1]}$, reading $\ell_x^{[a]}$, $\ell_y^{[b]}$, and $\ell_z^{[c]}$ and moving based on $\mathcal{L}(\ell, y)$ to $\ell_y^{[b+1]} = 142$. Process z performs $L_z^{[c+1]}$, reading the new label $\ell_y^{[b+1]}$. It thus moves to the T^2 subgraph 14, following the rule that the node chosen should be the "lowest node dominating all other nodes with labels." This is actually the most basic rule implied by the definition of $\mathcal{L}$. The move to a dominating node is intended to meet R1.

Processes y and z can continue forever to choose $\ell_y^{[b+2]} = 144$, $\ell_z^{[c+2]} = 145$, $\ell_y^{[b+3]} = 143, \ldots$ (that is, move in the cycle of 14), maintaining the above invariants, because the T^2 graph is a precedence graph for two processes. If at some point x moves, in $L_x^{[a+1]}$ it will read the labels of both z and y as being in the T^2 subgraph 14. A T^2 subgraph is a precedence graph able to accommodate two labels and no more.

Since *num_labels* $('14') = 2$ in $L_x^{[a+1]}$, that is, there are already two labels in the T^2 subgraph, by line 5 of $\mathcal{L}(\ell, i)$, x will move to $\ell_x^{[a+1]} = 151$, and so on. □

The reader can convince herself that following any labeling operation execution $L_z^{[c]}$ by some process z, the above invariants hold. Furthermore, for the set of labels of processes y ($y \neq z$) that were read in $L_z^{[c]}$'s collect operation (denoted $read(L_z^{[c]})$), it is the case that

$$(\forall \ell_y^{[b]} \in read(L_z^{[c]}))\,(\ell_y^{[b]} \not\succ \ell_z^{[c]}). \tag{3}$$

This invariant—that the new label chosen is greater than all those read—is the basis for meeting requirement R1.

As seen in the following example, in the concurrent case, more than k labels may move into the same T^k subgraph at the same time. It is thus not immediately clear why the second invariant holds.

Example 4.2. Assume that the following sequence of labeling operation executions occur concurrently. Processes x and y begin performing $L_x^{[a+1]}$ and $L_y^{[b+1]}$ concurrently, reading $\ell_x^{[a]}$, $\ell_y^{[b]}$, and $\ell_z^{[c]}$ and computing $\mathcal{L}$ such that $\ell_x^{[a+1]} = \ell_y^{[b+1]} = 142$. If they then continue to complete their operations by writing their labels, though they choose the same node, they were concurrent and can be ordered by relative id. If any of them were to continue to perform a new labeling operation, since *num_labels*$('14') > 2$, it would choose label 151, not entering the cycle. However, let us suppose that they do not both complete writing their labels, that is, x stops just before writing $\ell_x^{[a+1]}$ to v_x, while y writes $\ell_y^{[b+1]} = 142$. Process z then performs $L_z^{[c+1]}$, reading the new label $\ell_y^{[b+1]}$ and the old label $\ell_x^{[a]}$, thus moving to $\ell_z^{[c+1]} = 143$. Processes y and z continue to move into and in the cycle of the T^2 subgraph 14 since they continue to read x's old label. Then at some point, x completes $L_x^{[a+1]}$, and there are three labels in 14 (two of them in the cycle). However, if x now performs a new labeling $L_x^{[a+2]}$, it will read the labels of both x and y as being in 14. Since *num_labels*$('14') > 2$, by line 5 of $\mathcal{L}(\ell, i)$, x will move to $\ell_x^{[a+2]} = 151$, not entering the cycle. □

If nodes 1 and 2 did not exist in a T^2 subgraph (that is, each T^2 subgraph was a cycle of three nodes), a process' first move into T^k would be onto a node of the cycle. The reader can verify that the sequence of operations in Example 4.2, given that T^2 is just a cycle, would cause the labels of x, y, and z to end up each on a different node of the cycle, contradicting the first invariant. Based on the existence of nodes 1 and 2, this does not occur.

The following is intended to explain to the reader why for a given level k ($k = 2$ in the example), even if more than k processes move into a T^k subgraph *without* reading one another's labels, at most k of them will enter the cycle in T^k. The reason is the following well-known *flag principle*[9]:

> If there are $k+1$ people, each of which first raises a flag and then counts the number of raised flags, at least one person must see $k+1$ flags raised.

By the definition of the labeling function $\mathcal{L}$, each process moving into the cycle of a T^k subgraph must first move to either supernode 1 or 2 in T^k, and only then can it perform a labeling into the cycle. The move to 1 or 2 is the raising of the flag, and the move into the cycle is the counting of all flags.

[9] The proof follows since the last person to start counting flags must have seen $k+1$ flags raised.

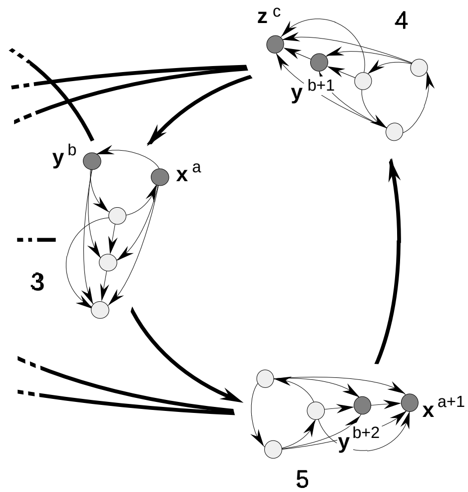

FIG. 8. *A collect returning a cycle.*

Example 4.3 below, which is depicted in Figure 8, shows that even though by the above there are at most k labels at a time in any T^k subgraph, the sets of labels *read* in a labeling operation execution may contain cycles.

Example 4.3. Process z begins performing $L_z^{[c+1]}$, reading $\ell_x^{[a]} = 135$. Process y then performs $L_y^{[b+1]}$, reading $\ell_x^{[a]}$, $\ell_y^{[b]}$, and $\ell_z^{[c]}$ and moving to $\ell_y^{[b+1]} = 142$. Process x performs $L_x^{[a+1]}$, reading the new label $\ell_y^{[b+1]}$ and $\ell_z^{[c]}$ and thus, by line 5 of $\mathcal{L}$, moving to $\ell_x^{[a+1]} = 151$. Process y then performs $L_y^{[b+2]}$, reading $\ell_x^{[a+1]}$ and moving to $\ell_y^{[b+2]} = 152$. Finally, process z reads $\ell_y^{[b+2]}$. It thus read $\ell_x^{[a]} = 135$, $\ell_y^{[b+2]} = 152$, and $\ell_z^{[c]} = 141$, three labels on a cycle. $\square$

In order to select a label that dominates all others, z must establish where the "maximal label" among them is. To overcome the problem that the labels read form cycles (as in the example above), the labeling function $\mathcal{L}(\ell, z)$ does not take into account "old values" such as $\ell_x^{[a]}$; it considers only the labels that dominate the current label $\ell_z^{[c]}$.

In order to maintain the first invariant, z should move to $\ell_z^{[c+1]} = 131$ to dominate the current labels of both x and y without moving directly into the cycle. However, there is seemingly a problem since z did not read the label $\ell_x^{[a+1]} = 151$; so how can it know that there are already two labels in the T^2 subgraph 15? The solution is based on the fact that z can indirectly deduce the existence of $\ell_x^{[a+1]} = 151$. By the first invariant, in all of the cycle of T^3, there are at most three labels. In order to move to $\ell_y^{[b+1]} = 152$, y must have read some label in node 151 of the T^2 subgraph 15. By simple elimination, this must be a label of x. This rule is maintained by the application of line 4 in $\mathcal{L}^k$.

If the above scenario had occurred in the cycle of a T^k graph, where $k > 3$, then in order to allow the same reasoning as above, it would have to be that z's reading $\ell_y^{[b+2]} = 152$ (or $\ell_y^{[b+2]} \in \{153, 154, 155\}$) would imply that there are $k-2$ labels apart from that of y in the T^{k-1} subgraph 15. It would thus have to be that if $\ell_y^{[b+2]}$ is

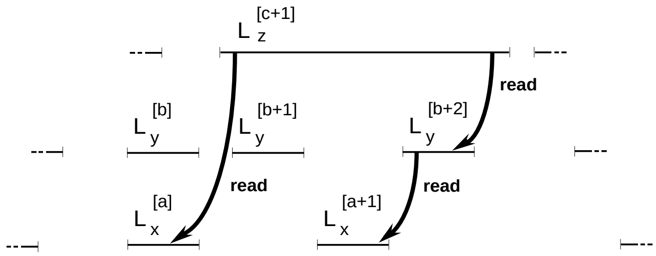

FIG. 9. *The observed relation.*

on supernode 2 in 15, it already established the existence of $k-2$ (and not just one) other labels in supernode 1.

It is for this purpose that supernode 1 of any T^k graph, where $k > 2$, is *not* a single node but a T^{k-1} subgraph. This creates a situation whereby as long as there are $k-1$ or fewer labels in T^k, all labels enter and move around in supernode 1. Supernode 2 can be chosen in $L_y^{[b+2]}$ only if $k-1$ labels were established by it as being in supernode 1 (i.e., supernode 1 is full). Since supernode 2 is a "bridge" that some process must "cross" (choose) before any process can move into the cycle, the above reasoning for z holds in case it read $\ell_y^{[b+2]} \in \{152, 153, 154, 155\}$.

Although the above invariants hold, it follows from Example 4.3 that the property that the chosen new label is greater than all those read, true for sequential labeling operation executions, does not hold in the concurrent case. Fortunately, there is a similar property that does hold, a property that will prove important in the implementation of the scan. Recall that $\mathit{read}(L_y^{[b]})$ denotes the set of label values read in the collect of $L_y^{[b]}$. Let us define the following *observed* relation among labeling operation executions to be the transitive closure of the *read* relation.

DEFINITION 4.1. *A labeling operation execution* $L_x^{[a]}$ *is* observed *by* $L_y^{[b]}$ (*denoted* $L_x^{[a]} \xrightarrow{\text{obs}} L_y^{[b]}$) *if* $\ell_x^{[a]} \in \mathit{read}(L_y^{[b]})$ *or there exists an* $L_z^{[c]}$ *such that* $\ell_z^{[c]} \in \mathit{read}(L_y^{[b]})$ *and* $L_x^{[a]} \xrightarrow{\text{obs}} L_z^{[c]}$.

DEFINITION 4.2. *Let the* maximal observed *set* $\mathit{max\_obs}(L_x^{[a]})$ *be defined as*

$$\{L_y^{[b]} \mid y \in \{1..n\},\ y \neq x,\ L_y^{[b]} \xrightarrow{\text{obs}} L_x^{[a]} \text{ and } (\forall L_y^{[b']}) \text{ (if } L_y^{[b]} \longrightarrow L_y^{[b']}\text{, then } L_y^{[b']} \not\xrightarrow{\text{obs}} L_x^{[a]})\}.$$

It thus consists of the "latest" of labeling operation executions observed for each process. In a concurrent execution, instead of invariant (3) stating that the new label chosen is greater than all the labels read, it is the case that

$$(\forall \ell_y^{[b]} \in \mathit{max\_obs}(L_x^{[a]}))\,(\ell_y^{[b]} \npreceq \ell_x^{[a]}). \tag{3$'$}$$

The new label chosen is greater than the latest of those observed for each process. As shown in Figure 9, for the labeling $L_z^{[c+1]}$ of Example 4.3, although z read $\ell_x^{[a]} = 143$ and $\ell_z^{[c+1]} \npreceq \ell_x^{[a]}$, it is the case that its maximal observed label is $\ell_x^{[a+1]}$, and $\ell_x^{[a+1]} \npreceq \ell_z^{[c+1]}$.

Finally, the following is the irreflexive total order $\Longrightarrow$ on the labeling operation executions as required by property P1.

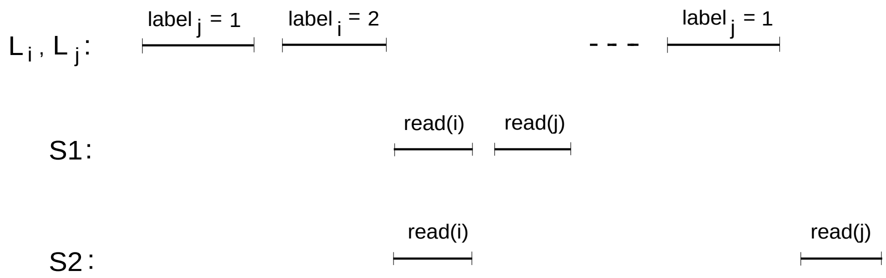

FIG. 10. *Ambiguity given bounded labels.*

DEFINITION 4.3. *Given any two distinct labeling operation executions* $L_x^{[a]}$ *and* $L_y^{[b]}$, $L_x^{[a]} \Longrightarrow L_y^{[b]}$ *if either*

1. $L_x^{[a]} \overset{\text{obs}}{\dashrightarrow} L_y^{[b]}$ *or*
2. $L_x^{[a]} \overset{\text{obs}}{\not\dashrightarrow} L_y^{[b]}$, $L_y^{[b]} \overset{\text{obs}}{\not\dashrightarrow} L_x^{[a]}$, *and* $\ell_x^{[a]} \overset{\text{v}}{\prec} \ell_y^{[b]}$.

Since with every $L_x^{[a]}$ there is an associated label $\ell_x^{[a]}$, $\Longrightarrow$ can be seen as a "lexicographical" order on pairs $(L_x^{[a]}, \ell_x^{[a]})$. The first element in the pair is ordered by $\overset{\text{obs}}{\dashrightarrow}$, a partial order that is consistent with the ordering $\longrightarrow$. (If $L_x^{[a]} \longrightarrow L_y^{[b]}$, then in $L_y^{[b]}$, y read $\ell_x^{[a]}$ or a later label.) The second element is ordered by $\overset{\text{v}}{\prec}$, an irreflexive and antisymmetric relation. Parts of the rather involved reasoning as to why the "static" relation $\overset{\text{v}}{\prec}$ on the labels completes the "dynamic" partial order $\overset{\text{obs}}{\dashrightarrow}$ to a total order on all labeling operation executions are provided in section 5.9. The main difficulty is in establishing transitivity. The intuition as to why $\Longrightarrow$ is transitive is based on the fact that "at any point in time," the current labels of all processes are totally ordered, that is, no three labels are on three different supernodes of a cycle in any T^k subgraph. The reader is encouraged to try to bring about a scenario where there are three labeling operation executions such that

$$L_x^{[a]} \Longrightarrow L_y^{[b]} \Longrightarrow L_z^{[c]} \Longrightarrow L_x^{[a]}$$

while keeping in mind that $\overset{\text{obs}}{\dashrightarrow}$ is transitive. It will become clear that this requires that at some point in time, there will be three labels of x, y, and z on three different supernodes of a cycle in some T^k subgraph, a contradiction.

4.4. The scan operation. The scan operation returns a pair $(\bar{\ell}, \prec)$. In the scan operation of the unbounded label implementation, the linearization order among the labeling operation executions can be determined just by reading the labels since the order among any two operations is just the order among their associated labels. However, as Example 4.4 shows, if labels are taken from a bounded range (and therefore the same labels are repeatedly used), a process scanning the labels concurrently with ongoing labeling operations cannot deduce the order $\Longrightarrow$ from the order of the labels alone.

Example 4.4. In Figure 10, segments represent operation-execution intervals, where time runs from left to right. Two processes i and j perform labeling operations sequentially, j followed by i, followed by many labelings, until eventually the labels are used, and j, for example, uses the same label as before. A third process z performs a scan concurrently with the labelings, reading $label_i$ and then $label_j$. $S1$ and $S2$

represent possible executions of this same scan, the only difference being that many labeling operations of other processes occurred between the reads in $S2$. In both the case where the scan is of the form $S1$ and the case where it is of the form $S2$, the values collected are $label_i = 2$ and $label_j = 1$, where the order among the labels is, say, $1 < 2$. However, in the case of $S1$, j's labeling preceded i's, while in $S2$, i's labeling preceded j's. Thus the order of the labels is not the order among the labeling operations. □

However, we do wish to provide the exact form of solution as in the unbounded case, where just by reading the labels, the scanning process can return a set of labels and the order among them. From Example 4.4, it should be clear that the order $\prec$ returned by the scan cannot be the order $\Longrightarrow$ among the associated labels of labeled-values in $\bar{\ell}$. Nevertheless, the requirement of property P1b is that $\prec$ be consistent with $\Longrightarrow$ for the set of labeling operation executions of labeled-values in $\bar{\ell}$. The key to the solution is to perform many collections of labels and then, based on the properties proven in what follows, return n of them for which $\prec$ can be determined.

The scan algorithm thus consists of two main steps, a sequence of $8n \log n$ *collect* operations[10] and an analysis phase of the collected labels to select a set $\bar{\ell}$ and an order $\prec$.

The $8n \log n$ *collect* operations are logically divided into n phases, where each phase consists of $\log n$ levels, each of eight collects. We use the notation $\ell^{\,c,m,k}, c \in \{1..8\}$, $m \in \{1..\lceil \log n \rceil\}$, and $k \in \{1..n\}$, to denote variables, each holding a set of labels $\{\ell_1^{c,m,k}, \ldots, \ell_n^{c,m,k}\}$ collected in the cth *collect* operation execution of the mth *level* of the kth *phase*. Let $half(r)$ and $other\_half(r)$ be complementary functions that, for a given set r, return two disjoint subsets $r1$ and $r2$ such that $r1 \cup r2 = r$ and $-1 \le ||r1|| - ||r2|| \le 1$.

The scan algorithm, presented in Figure 11, returns the indexed set of labeled-values $\bar{\ell}$, one of each process, and an ordering $\prec$ on their indexes. This order is represented by the vector $O[1..n]$, holding a *permutation* of the indexes in $\{1..n\}$, the number in the ith position representing the ith largest element in the order. The scan algorithm begins with a sequence of $8n\lceil \log n \rceil$ collect operation executions, for which the returned labels are all saved in the variables $\ell^{c,m,k}, c \in \{1..8\}$, $m \in \{1..\lceil \log n \rceil\}$, and $k \in \{1..n\}$. The remainder of the algorithm defines how to choose n of these labels, one per process, for which $\prec$ (i.e., $\Longrightarrow$) can be established. The following is an outline of how this selection process is performed. A formal proof of its correctness can be found in section 5.

By the order of label collection, the labels read in phase $k = 1$ are the earliest to have been collected and those for $k = n$ the last. Notice that from the $8\lceil \log n \rceil$ collected label sets of each phase, the algorithm selects one label. The selected label in the kth phase will be the k largest in the order $\prec$. As it turns out, in order to be able to show that it is the kth largest, it suffices that the following condition holds (slightly abusing notation in the definition).

Condition 1. For the label $\ell_s^{8,\lceil \log n \rceil,k}$, collected in the $\lceil \log n \rceil$th level of the kth phase, and any label $\ell_y^{8,1,k}$ of a process $y \in R$, collected in the first level of the kth phase, it is the case that $L_y^{8,1,k} \Longrightarrow L_s^{8,\lceil \log n \rceil,k}$.

To prove that this condition suffices, let it be shown that if it is maintained, the labeling operation execution of a label returned in a phase $k' < k$ precedes (in the

[10] Note that, as mentioned in section 1, the *scan* algorithm requires a scanning process *only to read* other's labels and does not require it to write.

```
function scan;
    function select(m,k,r);
    begin
        if ||r|| = 1 then return (x : x ∈ r);
          else
            x := select(m−1, k, half(r));
            y := select(m−1, k, other_half(r));
            if (∃c1, c2 ∈ {1..8}) (c1 < c2) ∧ (ℓ_x^{c1,m,k} ⋠ ℓ_y^{c2,m,k})
              then return y
              else return x
        fi fi;
    end select;
begin
    R := {1..n};
    ℓ̄ := ∅;
    for k := 1 to n do
        for m := 1 to ⌈log n⌉ do
            for c := 1 to 8 do
                ℓ^{c,m,k} := collect
    od od od;
    for k := n downto 1 do
        s := select(⌈log n⌉, k, R);
        ℓ̄ := ℓ̄ ∪ {value(ℓ_s^{8,⌈log n⌉,k})};
        O[s] := k;
        R := R − {s};
    od;
    return (ℓ̄, O);
end scan;
```

FIG. 11. *The scan algorithm.*

ordering $\Longrightarrow$) that of the label returned in phase k. The following shows that this is the case for the labels $\ell_x^{8,\lceil \log n\rceil,k}$, $\ell_y^{8,\lceil \log n\rceil,k-1}$, and $\ell_z^{8,\lceil \log n\rceil,k-2}$ returned in phases k, $k-1$, and $k-2$, respectively. The same line of proof can be extended inductively to all $k' < k$.

By Condition 1, $L_y^{8,1,k} \Longrightarrow L_x^{8,\lceil \log n\rceil,k}$. Since the read of $\ell_y^{8,1,k}$ was performed after that of $\ell_y^{8,\lceil \log n\rceil,k-1}$, either the label of the same labeling operation execution was read in both cases or $L_y^{8,\lceil \log n\rceil,k-1} \Longrightarrow L_x^{8,\lceil \log n\rceil,k}$. By similar reasoning, $L_z^{8,\lceil \log n\rceil,k-2} \Longrightarrow L_y^{8,\lceil \log n\rceil,k-1}$, which by the transitivity of $\Longrightarrow$ establishes $L_z^{8,\lceil \log n\rceil,k-2} \Longrightarrow L_x^{8,\lceil \log n\rceil,k}$.

It remains to be shown that the label returned in any phase, determined by the *select* function, meets Condition 1. The *select* function is a recursively defined "winner-take-all"-type algorithm among the processes in R. In any given phase, R is the set of processes for which a label has not been selected in earlier phases. The *select* function returns the id of the "winner," a process s that meets Condition 1. At any level m of the application of *select*(m, k, r), the winners of the selections at level $m-1$ are paired up, and from each pair one "winner" process is selected to be passed on to the $(m+1)$th level of selection. After at most $\lceil \log ||R|| \rceil$ levels, s, the winner of all selections, is returned.

Based on the definition of the *select* function, maintaining the following condition two suffices to assure that the label of the process s returned by *select*(m,k,r) meets Condition 1.

Condition 2. Of the two processes x and y in the application of *select* at level m of phase k, the one returned, say x, is such that $L_y^{1,m,k} \Longrightarrow L_x^{8,m,k}$, where $\ell_y^{1,m,k}$ and $\ell_x^{8,m,k}$, respectively, are the labels associated with these labeling operation executions.

Maintaining Condition 2 suffices for the following reason. If at level m process x was selected between x and y and at level $m-1$ process y was selected between y and z, by the same line of proof as above, from $L_y^{1,m,k} \Longrightarrow L_x^{8,m,k}$ and $L_z^{1,m-1,k} \Longrightarrow L_y^{8,m-1,k}$, it follows that $L_z^{8,m-2,k} \Longrightarrow L_x^{8,m,k}$. By induction, this implies Condition 1.

It remains to be shown that Condition 2 can be met. Recall Example 4.4, which implies that it is impossible to establish the order $\Longrightarrow$ among two labeling operation executions from the order among their associated labels alone. To overcome this problem, instead of attempting to decide the order between two given labeling operation executions, the algorithm will choose a pair out of several given labeling operation executions for which the order $\Longrightarrow$ can be determined. Thus to allow the *select* operation at level m of phase k to choose a "winner" process, say x, for which $L_y^{1,m,k} \Longrightarrow L_x^{8,m,k}$, labels of x and y from eight consecutive collects will be analyzed.

Let it first be shown that if the following condition holds for y, namely, if it is the case that

$$[\textit{Condition } 3.] \qquad (\exists c1, c2 \in \{1..8\})\,(c1 < c2) \wedge (\ell_x^{c1,m,k} \not\preceq \ell_y^{c2,m,k}),$$

then $L_x^{c1,m,k} \Longrightarrow L_y^{c2,m,k}$. (Because of the order of label collecting, this will imply $L_x^{1,m,k} \Longrightarrow L_y^{8,m,k}$.) Assume by way of contradiction that $L_x^{c1,m,k} \Longrightarrow L_y^{c2,m,k}$. Since $\ell_x^{c1,m,k} \not\preceq \ell_y^{c2,m,k}$, it must be by the definition of $\Longrightarrow$ that $L_y^{c2,m,k} \xrightarrow{\text{obs}} L_x^{c1,m,k}$. It cannot be that $\ell_y^{c2,m,k} \in max\_obs(L_x^{c1,m,k})$ since by the properties of the labeling scheme, for the label $\ell_y^{[b]} \in max\_obs(L_x^{c1,m,k})$, $\ell_y^{[b]} \not\preceq \ell_x^{c1,m,k}$. Thus there must be a different labeling operation execution $\ell_y^{[b]} \in max\_obs(L_x^{c1,m,k})$, $L_y^{c2,m,k} \longrightarrow L_y^{[b]}$. This label $\ell_y^{[b]}$ was already observed (i.e., must have been written) before the end of the read of $\ell_x^{c1,m,k}$. Thus $\ell_y^{[b]}$ or a label later than it must have been read instead of $\ell_y^{c2,m,k}$ in the *collect* $c2$ of level m in phase k, a contradiction.

It remains to be shown that if Condition 3 does not hold for y, it is the case that $L_y^{1,m,k} \Longrightarrow L_x^{8,m,k}$, and x can be correctly returned. Assume by way of contradiction that Condition 3 does not hold for y. By the same arguments as above, it cannot be that Condition 3 holds for x, that is, $(\exists c1, c2 \in \{1..8\})$ $(c1 < c2) \wedge (\ell_y^{c1,m,k} \not\preceq \ell_x^{c2,m,k})$. Therefore, it must be that there are four nonconsecutive collects of $\ell^{c1,m,k}$, $c1 \in \{1,3,5,7\}$, and four nonconsecutive collects of $\ell^{c2,m,k}$, $c2 \in \{2,4,6,8\}$, such that the labels $\ell_y^{c1,m,k}$, $c1 \in \{1,3,5,7\}$, are all different from one another and the labels $\ell_x^{c2,m,k}$, $c2 \in \{2,4,6,8\}$, are all different from one another. The reason is that if any two of them, say $\ell_y^{3,m,k}$ and $\ell_y^{5,m,k}$, are the same, then in order for Condition 3 not to hold for x $c1 = 4$ and $c2 = 3$, it must be that $\ell_x^{4,m,k} \not\preceq \ell_y^{3,m,k}$. However, since $\ell_y^{3,m,k}$ and $\ell_y^{5,m,k}$ are the same, it would follow that $\ell_x^{4,m,k} \not\preceq \ell_y^{5,m,k}$, and Condition 3 would hold for y, a contradiction.

To complete the proof, it remains to be shown that if the labels $\ell_y^{c1,m,k}$, $c1 \in \{1,3,5,7\}$, are all different from one another and the labels $\ell_x^{c2,m,k}$, $c2 \in \{2,4,6,8\}$, are all different from one another, then $L_y^{1,m,k} \Longrightarrow L_x^{8,m,k}$. The situation above is such that during the eight collect operations, each of the processes x and y executed a new labeling operation at least three times. It can be formally shown[11] that after x and y moved at least three times, the third new labeling operation execution $L_x^{8,m,k}$ occurred completely after the initial labeling of y, that is, after $L_y^{1,m,k} \longrightarrow L_x^{8,m,k}$ (see Figure 13 in section 5.8). The scan thus takes $O(n^2 \log n)$ read operations.

As a final comment, note that for algorithms where only the maximum label is required and not a complete order among all returned labels (as in the construction of an MRMW atomic register or solutions to the *mutual exclusion* problem), only one phase of label collection is required, that is, only $8 \log n$ collects.[12]

5. Correctness proof.

5.1. A short review of Lamport's formal theory.

This is a minimal outline (due to Ben-David [Ben88]) of Lamport's formalism, on which the correctness proof in this chapter is based. The reader is encouraged to consult [Lam86c, Lam86d, Lam86a, Lam86b] for an elaborate presentation and discussion.

Lamport bases his formal theory on two abstract relations over operation executions. For operation executions A and B, "$A \longrightarrow B$" stands for "A precedes B" and "$A \dashrightarrow B$" stands for "A can causally affect B."

A *system execution* is a triple $\langle \varphi, \longrightarrow, \dashrightarrow \rangle$, where φ is a set of *operation executions* and $\longrightarrow$ and $\dashrightarrow$ are binary relations over φ. Lamport offers the following axioms:

A1. $\longrightarrow$ is an irreflexive transitive relation.

A2. If $A \longrightarrow B$, then $A \dashrightarrow B$ and $B \not\dashrightarrow A$.

A3. If ($A \longrightarrow B$ and $B \dashrightarrow C$) or ($A \dashrightarrow B$ and $B \longrightarrow C$), then $A \dashrightarrow C$.

A4. If $A \longrightarrow B \dashrightarrow C \longrightarrow D$, then $A \longrightarrow D$.

A5. For any A, the set of B such that $A \not\longrightarrow B$ is finite.

An intuition for these axioms can be gained by considering the following model for it. Let $\mathcal{E}$ be a partially ordered set of events and let φ be a collection of nonempty subsets of $\mathcal{E}$. For A and B in φ, define $A \longrightarrow B$ if and only if $(\forall a \in A)$ $(\forall b \in B)$ $(a < b)$ (in the sense of $\mathcal{E}$) and $A \dashrightarrow B$ if and only if $(\exists a \in A)$ $(\exists b \in B)$ $(a < b)$. A straightforward checking shows that such models satisfy axioms A1–A4 and also the following axiom:

A4*. If $A \dashrightarrow B \longrightarrow C \dashrightarrow D$, then $A \dashrightarrow D$.

This last axiom was suggested by Abraham[13] in [AB87], where a completeness theorem was proven for the above-mentioned class of models with respect to axioms {A1, A2, A3, A4, A4*}. An important class of models is obtained when $\mathcal{E}$ is a linear (total) ordering. In such a case, the system satisfies an additional axiom:

Global time. For all A and B, it is the case that either $A \dashrightarrow B$ or $B \longrightarrow A$ but not both.

The above axioms can be extended to nonterminating operation executions as described in [Lam86c]. Added on top of these axioms are the communication axioms, in our case axioms B0–B5 of [Lam86d], for communication via shared registers. These axioms formalize the behavior of a single-writer multireader atomic register. In a few words, axioms B0–B4 define what constitutes *regular* register behavior, namely, that reads can return only values that

[11] This claim is *not* true if fewer than three new labelings took place.

[12] The number of collects in each phase can be lowered to $5 \log n$ if one gives up the property that the order of reads in a collect be arbitrary.

[13] Ben-David was later informed that this result was obtained independently by Anger.

- were actually written,
- were written before the end of the read, and
- were not overwritten before the beginning of read.

Axiom B5 is added to these, which restricts the allowed behavior of the register by requiring that reads and writes be linearizable. Such a register that abides by axioms B0–B5 is called *atomic* since, in effect, its behavior is equivalent to one in which all reads and writes are "atomic," that is, occur in nonoverlapping intervals of time.

5.2. Proof outline. The proof will follow Definition 8 of [Lam86a], namely, that a system $\mathbf{S}$ *implements* a system $\mathbf{H}$ if there is a mapping $m : \mathbf{S} \mapsto \mathbf{H}$ such that for every system execution $\langle \varphi, \longrightarrow, \dashrightarrow \rangle$ in $\mathbf{S}$, $\langle \varphi, \longrightarrow, \dashrightarrow \rangle$ implements $m(\langle \varphi, \longrightarrow, \dashrightarrow \rangle)$. The definition of a system execution used in what follows is that of [Lam86a] under the assumption of global time. Theorem 5.1 below establishes the correctness of the implementation.

THEOREM 5.1. *The system defined by the labeling and scan procedures implements a concurrent time-stamp system.*

In order to prove the theorem correct, the systems involved need to be formally defined and a mapping between them must be established.

5.3. System definitions. The labeling and scan procedures of the previous sections define a system $\mathbf{S}$, the set of all system executions that consist of reads and writes of the single-writer multireader atomic registers $v_1, \ldots, v_n$, such that the only operations on these registers are the ones indicated by the scan and labeling algorithms. Formally, $\mathbf{S}$ contains all system executions $\langle \varphi, \longrightarrow, \dashrightarrow \rangle$ such that we have the following:

1. φ consists of reads and writes of single-writer multireader atomic registers $v_1, \ldots, v_n$ (with register axioms B0–B5 restricting such read and write operations [Lam86b]).

2. Each v_x is written by process x and read by all processes in $\{1..n\}$, where $r_x^{[k]}(y)$ ($w_x^{[k]}(x)$) denote the kth read (respectively, write) of v_x by process y (respectively, x).

3. The read and write operation executions of a process x are totally ordered by $\longrightarrow$.

4. For any process z and any x and y:

(a) If the read operation $r_x^{[k]}(z)$ occurs, then $r_y^{[k]}(z)$ occurs, $r_x^{[k-1]}(z) \longrightarrow r_y^{[k]}(z)$, and if for some $w_z^{[k']}(z)$, $r_x^{[k]}(z) \longrightarrow w_z^{[k']}(z)$, then $r_y^{[k]}(z) \longrightarrow w_z^{[k']}(z)$.

(b) For any two writes $w_z^{[k]}(z)$ and $w_z^{[k+1]}(z)$, there exists a set of read operation executions

$$\mathcal{R}_{k+1} = \{r_x^{[\alpha]}(z) \,|\, w_z^{[k]}(z) \longrightarrow r_x^{[\alpha]}(z) \longrightarrow w_z^{[k+1]}(z),\, \alpha \in \{0, 1, \ldots\}\},$$

of reads of v_x such that $||\mathcal{R}_{k+1}|| \bmod (8n\lceil \log n \rceil) = 1$.

(c) For every $r_x^{[k]}(z)$, $r_x^{[k]}(z) \in \mathcal{R}_r$, for some r.

This fourth condition formalizes some of the semantics of labeling and scan procedures. It states that every read is part of a *collect* operation consisting of a sequence of reads, one of each register, each collection ending before the next begins, and that reads and writes are bunched in groups of either $8n \log n$ collects or a collect followed immediately by a write.

The following is a formal definition of $\mathbf{H}$, a concurrent time-stamp system.

DEFINITION 5.1. *A* concurrent time-stamp system *is a set of system executions* $\langle \psi, \longrightarrow, \dashrightarrow \rangle$ *that have properties* P0–P4.

Properties P1–P4 are as defined earlier, and the following is the definition of P0.

P0. The set of operation executions on the CTSS is the set $\psi = \bigcup_i \psi_i$, where each ψ_i, the set of operation executions by process i, is as follows:

- *A finite or infinite set of labeling operation executions* $\{L_i^{[1]}, L_i^{[2]}, \ldots\}$: A unique *labeled-value* $\ell_i^{[k]}$ is associated with each $L_i^{[k]}$. The set of possible labeled-values can be from any range. For example, if an atomic register is to be implemented, the labeled-value can be the value written to the register. Given that the value $\ell_i^{[k]}$ may repeatedly appear, in order that a unique labeled-value be associated with each $L_i^{[k]}$, let $\ell_i^{[k]}$ be the triplet $\langle \ell_i^{[k]}, i, k\rangle$, where i and k are dummy fields and only $\ell_i^{[k]}$ is visible to the user. There is thus a one-to-one mapping from labeled-values to labeling operations.
- *A finite or infinite set of scan operation executions* $\{S_i^{[1]}, S_i^{[2]}, \ldots\}$: A *view* $\bar{\ell} = \{\ell_1^{[k_1]}, \ldots, \ell_n^{[k_n]}\}$ is returned by each scan, with different labeled-values associated with labeling operation executions of different processes.
- *An initial labeling operation execution* $L_i^{[0]}$ *with labeled-value* $\ell_i^{[0]}$.

The initial labeling $L_i^{[0]} \longrightarrow S_j^{[k]}$ for any i, j, and k. (This is the same as assuming that there is some initial labeled-value for any process i that a scan will obtain if it preceded any labeling operation of i.) All operation executions in ψ_i are totally ordered by $\longrightarrow$, that is, they occur sequentially.

5.4. The mapping. By Definition 8 of [Lam86a], to show that the labeling and scan procedures implement a CTSS, a mapping m from $\mathbf{S}$ to $\mathbf{H}$ must be defined. In the definition of the labeling and scan procedures, for each system execution $\langle \varphi, \longrightarrow, \dashrightarrow\rangle$ of $\mathbf{S}$, the set of operation executions $m(\varphi)$ of $m(\langle \varphi, \longrightarrow, \dashrightarrow\rangle)$ is the following *higher-level view* of $\langle \varphi, \longrightarrow, \dashrightarrow\rangle$:

1. Each *labeling* operation execution $L_i^{[k]}$ consists of a set $r_1^{[k']}(i), \ldots, r_n^{[k']}(i)$ of reads followed by a write $w_i^{[k]}(i)$, where $k' = \max\{\alpha \,|\, r_j^{[\alpha]}(i) \longrightarrow w_i^{[k]}(i)\}$.
2. Each *scan* operation execution by process i is a set of reads

$$\{r_j^{[\alpha]}(i) \,|\, j = 1..n,\ \alpha = k..k + 8n\lceil \log n\rceil \text{ and}$$
$$(\neg \exists w_i^{[k']}(i),\ r_j^{[\alpha]}(i) \longrightarrow w_i^{[k']}(i) \longrightarrow r_j^{[\alpha+1]}(i))\},$$

all in φ and no element of which is part of another *scan* or *labeling*.

The set $m(\varphi)$ meets conditions H1 and H2 of Definition 4 of [Lam86a], that is, each of its elements is a finite and nonempty set of elements of φ and each element of φ belongs to a finite, nonzero number of elements of $m(\varphi)$. It is thus a higher-level view of φ. (In fact, this implies that the labeling and scan operations as implemented are wait-free since waiting means that a higher-level operation takes an infinite number of lower-level ones.) To complete the description of the mapping m, the precedence relations $\xrightarrow{H}$ and $\overset{H}{\dashrightarrow}$ must be defined so that $m(\langle \varphi, \longrightarrow, \dashrightarrow\rangle)$ is defined as $\langle m(\varphi), \xrightarrow{H}, \overset{H}{\dashrightarrow}\rangle$.

By choosing $\xrightarrow{H}$ and $\overset{H}{\dashrightarrow}$ to be the induced relations $\xrightarrow{*}$ and $\overset{*}{\dashrightarrow}$ as defined by equation 2 of [Lam86a] (by equation 2, choosing the induced precedence relations $\xrightarrow{*}$ and $\overset{*}{\dashrightarrow}$ for $\xrightarrow{H}$ and $\overset{H}{\dashrightarrow}$ simply means that the ordering among the higher-level *scan* and *labeling* operation executions is that of the reads and writes implementing them), axioms A1–A5 are met, implying that $\langle m(\varphi), \xrightarrow{H}, \overset{H}{\dashrightarrow}\rangle$ is indeed a system execution. Since condition H3 of Definition 5 of [Lam86a] is satisfied by the induced precedence relations,[14] $\langle \varphi, \longrightarrow, \dashrightarrow \rangle$ implements $\langle m(\varphi), \xrightarrow{\mathrm{H}}, \overset{\mathrm{H}}{\dashrightarrow} \rangle$.

Having defined the system $\langle m(\varphi), \xrightarrow{\mathrm{H}}, \overset{\mathrm{H}}{\dashrightarrow} \rangle$, it remains to be shown that it is indeed a CTSS, that is, is in $\mathbf{H}$. This amounts to showing that $\langle m(\varphi), \xrightarrow{*}, \overset{*}{\dashrightarrow} \rangle$ satisfies properties P0–P4.

5.5. Properties P0 and P2–P4. The proof that $\langle m(\varphi), \xrightarrow{*}, \overset{*}{\dashrightarrow} \rangle$ meets property P0 follows by applying equation 2 of [Lam86a] to $\langle \varphi, \longrightarrow, \dashrightarrow \rangle$ (again, this amounts to defining the high-level order among *scan* and *labeling* operation executions to be that among the reads and writes implementing them) and observing the following:

1. The labeled-value $\ell_i^{[k]}$ associated with each labeling operation $L_i^{[k]}$ is just the labeled-value part written to v_i by the write $w_i^{[k]}(i)$ of process i. (Recall that there is also a label part of v_i.)
2. Any labeled-value returned by a scan is the result of some write $w_i^{[k]}(i)$.
3. The initial labeling $L_i^{[0]}$ is the write of some initial labeled-value and label 11..1 to register v_i.

The proof that $\bar{\ell}$ and $\succ$ are a *view* and an *irreflexive total order* on its elements follows from the definition of the *scan procedure*. Since v_i, $i \in \{1..n\}$ are SWMR atomic registers, applying equation 2 of [Lam86a] together with register axioms B0–B5 to $\langle \varphi, \longrightarrow, \dashrightarrow \rangle$ yields the proof that $\langle m(\varphi), \xrightarrow{*}, \overset{*}{\dashrightarrow} \rangle$ satisfies properties P2, P3, and P4. The details are left to the reader.

To simplify the presentation, for the remainder of this section, we use the notation $\ell_i^{[k]}$ to denote the label part of the value written to register v_i in the labeling operation execution $L_i^{[k]}$. We will use the notation $value(\ell_i^{[k]})$ to refer to the labeled-value part.

5.6. Properties of the observed relation. As part of the notation used in what follows, $r_j(L_i^{[k]})$ and $w(L_i^{[k]})$ will denote, respectively, the reading of v_j and writing of v_i during a labeling operation execution $L_i^{[k]}$. Also, let $m(\varphi)^L \subseteq m(\varphi)$ denote the set of all labeling operation executions in $m(\varphi)$. To prove that $\langle m(\varphi), \xrightarrow{*}, \overset{*}{\dashrightarrow} \rangle$ meets property P1, the relation $\Longrightarrow$ on the labeling operation executions in $m(\varphi)$ should be shown to be an irreflexive total order. The definition of this relation (Definition 4.3) is based on that of the relation $\overset{\mathrm{obs}}{\dashrightarrow}$ (Definition 4.1).

The following lemma establishes the properties of $\overset{\mathrm{obs}}{\dashrightarrow}$, later used to establish the properties of $\Longrightarrow$.

LEMMA 5.1. *The relation $\overset{\mathrm{obs}}{\dashrightarrow}$ is an irreflexive partial order on the labeling operation executions in $m(\varphi)$, such that for any two labeling operations $L_i^{[a]}$ and $L_j^{[b]}$, if $L_i^{[a]} \xrightarrow{*} L_j^{[b]}$, then $L_i^{[a]} \overset{\mathrm{obs}}{\dashrightarrow} L_j^{[b]}$.*

Proof. Since $r_j(L_i^{[a]}) \longrightarrow w_i(L_i^{[a]})$ for any j, it follows that $\overset{\mathrm{obs}}{\dashrightarrow}$ is irreflexive. The rest of the proof is based on the three claims below.

CLAIM 5.1.1 (transitive). *For any three labeling operation executions $L_i^{[a]}$, $L_j^{[b]}$, and $L_k^{[c]}$, if $L_i^{[a]} \overset{\mathrm{obs}}{\dashrightarrow} L_j^{[b]} \overset{\mathrm{obs}}{\dashrightarrow} L_k^{[c]}$, then $L_i^{[a]} \overset{\mathrm{obs}}{\dashrightarrow} L_k^{[c]}$.*

Proof. The proof is by induction on the length of the minimal production sequence of the production of $L_j^{[b]} \overset{\mathrm{obs}}{\dashrightarrow} L_k^{[c]}$. If $||L_j^{[b]} \overset{\mathrm{obs}}{\dashrightarrow} L_k^{[c]}|| = 1$, then by definition $r_j(L_k^{[c]}) = \ell_j^{[b]}$ and $L_i^{[a]} \overset{\mathrm{obs}}{\dashrightarrow} L_j^{[b]}$, which by the definition of $\overset{\mathrm{obs}}{\dashrightarrow}$ implies that

[14] Definition 5 of [Lam86a] states that a lower-level system execution $\langle \varphi, \longrightarrow, \dashrightarrow \rangle$ implements a higher-level one $\langle \psi, \xrightarrow{\mathrm{H}}, \overset{\mathrm{H}}{\dashrightarrow} \rangle$ if ψ is a higher-level view of φ and condition H3 holds, that is, for any G, $H \in \psi$, if $G \xrightarrow{*} H$, then $G \xrightarrow{\mathrm{H}} H$.

$L_i^{[a]} \overset{\text{obs}}{\dashrightarrow} L_k^{[c]}$. Assume that the induction hypothesis holds for every $r' < r$. Let $||L_j^{[b]} \overset{\text{obs}}{\dashrightarrow} L_k^{[c]}|| = r$. By definition, there exists an $L_j^{[b']}$ such that $L_i^{[a]} \overset{\text{obs}}{\dashrightarrow} L_j^{[b]} \overset{\text{obs}}{\dashrightarrow} L_j^{[b']}$, where $||L_j^{[b]} \overset{\text{obs}}{\dashrightarrow} L_j^{[b']}|| = r-1$ and $r_j(L_k^{[c]}) = \ell_j^{[b']}$. By the induction hypothesis, $L_i^{[a]} \overset{\text{obs}}{\dashrightarrow} L_j^{[b']}$, which by definition implies that $L_i^{[a]} \overset{\text{obs}}{\dashrightarrow} L_k^{[c]}$. □

COROLLARY 5.1. *If $L_i^{[a]} \overset{\text{obs}}{\dashrightarrow} L_j^{[b]}$, then there exists a read $r_i(L_\alpha^{[\beta]}) = \ell_i^{[a]}$ such that $w(L_i^{[a]}) \dashrightarrow r_i(L_\alpha^{[\beta]}) \longrightarrow w(L_j^{[b]})$ for some α and β (possibly $\alpha = j$ and $\beta = b$).*

Proof. The proof is by induction on the length of the minimal production of the observation sequence, as in the previous claim. If $||L_i^{[a]} \overset{\text{obs}}{\dashrightarrow} L_j^{[b]}|| = 1$, then by definition $\alpha = j$ and $\beta = b$. Assume that the induction hypothesis holds for every $r' < r$. By the minimality of the production sequence, there must exist an $L_\alpha^{[\beta]}$, $r_i(L_\alpha^{[\beta]}) = \ell_i^{[a]}$, such that $||L_\alpha^{[\beta]} \overset{\text{obs}}{\dashrightarrow} L_j^{[b]}|| = r-1$. By the induction hypothesis, there exist α' and β' such that

$$w(L_\alpha^{[\beta]}) \dashrightarrow r_\alpha(L_{\alpha'}^{[\beta']}) \longrightarrow w(L_j^{[b]}),$$

where possibly $\alpha' = j$ and $\beta' = b$. By the definition of the labeling operation, $r_i(L_\alpha^{[\beta]}) \longrightarrow w(L_\alpha^{[\beta]})$, and so

$$r_i(L_\alpha^{[\beta]}) \longrightarrow w(L_\alpha^{[\beta]}) \dashrightarrow r_\alpha(L_{\alpha'}^{[\beta']}) \longrightarrow w(L_j^{[b]}),$$

which by axiom A4 implies $r_i(L_\alpha^{[\beta]}) \longrightarrow w(L_j^{[b]})$. Since $r_i(L_\alpha^{[\beta]}) = \ell_i^{[a]}$, it follows by atomic register axiom B5 that $w(L_i^{[a]}) \dashrightarrow r_i(L_\alpha^{[\beta]}) \longrightarrow w(L_j^{[b]})$. □

CLAIM 5.1.2 (antisymmetric). *For any two distinct labeling operation executions $L_i^{[a]}$ and $L_j^{[b]}$, if $L_i^{[a]} \overset{\text{obs}}{\dashrightarrow} L_j^{[b]}$, then $L_j^{[b]} \not\overset{\text{obs}}{\dashrightarrow} L_i^{[a]}$.*

Proof. Assume by way of contradiction that $L_i^{[a]} \overset{\text{obs}}{\dashrightarrow} L_j^{[b]}$ and $L_j^{[b]} \overset{\text{obs}}{\dashrightarrow} L_i^{[a]}$. Thus by Corollary 5.1, for some α, β, γ, and δ (possibly $\alpha = j$, $\beta = b$, $\gamma = i$, or $\delta = a$),

$$w(L_i^{[a]}) \dashrightarrow r_i(L_\alpha^{[\beta]}) \longrightarrow w(L_j^{[b]}) \text{ and}$$
$$w(L_j^{[b]}) \dashrightarrow r_j(L_\gamma^{[\delta]}) \longrightarrow w(L_i^{[a]}).$$

Since this implies

$$w(L_i^{[a]}) \dashrightarrow r_i(L_\alpha^{[\beta]}) \longrightarrow w(L_j^{[b]}) \dashrightarrow r_j(L_\gamma^{[\delta]}),$$

by axiom A4*, $w(L_i^{[a]}) \dashrightarrow r_j(L_\gamma^{[\delta]})$. By $w(L_i^{[a]}) \dashrightarrow r_j(L_\gamma^{[\delta]})$ and $r_j(L_\gamma^{[\delta]}) \longrightarrow w(L_i^{[a]})$, a contradiction to the axiom of global time is derived. □

CLAIM 5.1.3 (consistent). *If $L_x^{[a]} \overset{*}{\longrightarrow} L_y^{[b]}$, then $L_x^{[a]} \overset{\text{obs}}{\dashrightarrow} L_y^{[b]}$.*

Proof. If $x = y$, then $r_x(L_y^{[a+1]}) = \ell_x^{[a]}$, and by induction, for $b > a$, $L_x^{[a]} \overset{\text{obs}}{\dashrightarrow} L_y^{[b]}$. If $x \neq y$, then by register axioms B0–B4, since $w(L_x^{[a]}) \longrightarrow r_x(L_y^{[b]})$, either $r_x(L_y^{[b]}) = \ell_x^{[a]}$ (implying $L_x^{[a]} \overset{\text{obs}}{\dashrightarrow} L_y^{[b]}$) or there exists an $L_x^{[a']}$, $L_x^{[a]} \overset{*}{\longrightarrow} L_x^{[a']}$, where $r_x(L_y^{[b]}) = \ell_x^{[a']}$, which by the transitivity of $\overset{\text{obs}}{\dashrightarrow}$ (Claim 5.1.1) implies $L_x^{[a]} \overset{\text{obs}}{\dashrightarrow} L_y^{[b]}$. This completes the proof of Lemma 5.1. □

The following lemma formalizes the property that whenever a new label $\ell_x^{[a]}$ is selected, it is greater (by the ordering $\preceq$) then the latest label observed in $L_x^{[a]}$ for any process $y \neq x$.

LEMMA 5.2. *For any labeling operation execution* $L_x^{[a]}$, *it is the case that*

$$(\forall \ell_y^{[b]} \in max\_obs(L_x^{[a]}))\,(\ell_y^{[b]} \overset{\vee}{\prec} \ell_x^{[a]}).$$

To simplify the exposition, the the proof is differed to section 5.9, where it is joined with the proof of Claim 5.3.2.

5.7. Property P1a. The following lemma asserts that $\Longrightarrow$ meets part a of property P1.

LEMMA 5.3. *The relation* $\Longrightarrow$ *is an irreflexive total order on the labeling operation executions in* $m(\varphi)$ *such that for any two labeling operations* $L_i^{[a]}$ *and* $L_j^{[b]}$, *if* $L_i^{[a]} \xrightarrow{*} L_j^{[b]}$, *then* $L_i^{[a]} \Longrightarrow L_j^{[b]}$.

Proof. By Definition 4.3, the relation $\Longrightarrow$ is irreflexive and total, and is consistent with the ordering $\xrightarrow{*}$ among the labeling operation executions in $m(\varphi)$. The following two claims complete the proof by showing that it is also antisymmetric and transitive.

CLAIM 5.3.1 (antisymmetric). *For any two distinct labeling operation executions* $L_i^{[a]}$ *and* $L_j^{[b]}$, *if* $L_i^{[a]} \Longrightarrow L_j^{[b]}$, *then* $L_i^{[a]} \not\Longleftarrow L_j^{[b]}$.

Proof. Assume by way of contradiction that for two distinct labeling operation executions, $L_i^{[a]} \Longrightarrow L_j^{[b]}$ and $L_i^{[a]} \Longleftarrow L_j^{[b]}$. Since $\xrightarrow{\text{obs}}$ is antisymmetric, it is not the case that both $L_i^{[a]} \xrightarrow{\text{obs}} L_j^{[b]}$ and $L_j^{[b]} \xrightarrow{\text{obs}} L_i^{[a]}$ hold. Thus if $L_i^{[a]} \xrightarrow{\text{obs}} L_j^{[b]}$, $L_j^{[b]} \not\Longrightarrow L_i^{[a]}$ even if $\ell_j^{[b]} \overset{\vee}{\prec} \ell_i^{[a]}$, a contradiction. Thus it must be the case that $\ell_j^{[b]} \overset{\vee}{\prec} \ell_i^{[a]}$ and $\ell_i^{[a]} \overset{\vee}{\prec} \ell_j^{[b]}$, which contradicts the definition of the ordering $\overset{\vee}{\prec}$ of the labels. $\square$

CLAIM 5.3.2 (transitive). *For any three labeling operation executions* $L_i^{[a]}$, $L_j^{[b]}$, *and* $L_k^{[c]}$, $L_i^{[a]} \Longrightarrow L_j^{[b]} \Longrightarrow L_k^{[c]}$ *implies* $L_i^{[a]} \Longrightarrow L_k^{[c]}$.

Due to its extreme length and to simplify the presentation, the proof is deferred to section 5.9.

This completes the proof of Lemma 5.3. $\square$

5.8. Property P1b. It remains to be proven that $\langle m(\varphi), \xrightarrow{*}, \dashrightarrow^{*} \rangle$ meets part b of property P1, that is, for any scan operation execution $S_i^{[k]}$ that returns $(\bar{\ell}, \prec)$, where

$$value(\ell_x^{[a]}),\ value(\ell_y^{[b]}) \in \bar{\ell},$$

it is the case that $x \prec y$ if and only if $L_x^{[a]} \Longrightarrow L_y^{[b]}$. Since both $\prec$ and $\Longrightarrow$ are irreflexive total orders, it suffices to show the "only if" direction. By the definition of the scan implementation, the returned order $\prec$ among the indexes of labeled-values in $\bar{\ell}$ is just the ordering among the collection phases in which they were selected. Thus it suffices to prove that in any scan operation execution $S_i^{[k]}$ that returns $(\bar{\ell}, \prec)$, if $value(\ell_x^{[a]})$ was returned in phase k' and $value(\ell_y^{[b]})$ was returned in phase k, where $k' < k$, then $L_x^{[a]} \Longrightarrow L_y^{[b]}$. This is captured by the following lemma (slightly abusing notation).

LEMMA 5.4. *In any scan operation execution, for any* i *such that* $O[i] < O[j]$, *where* $value(\ell_i^{8,\lceil \log n \rceil, O[i]})$, $value(\ell_j^{8,\lceil \log n \rceil, O[j]}) \in \bar{\ell}$, *it is the case that* $L_i^{8,\lceil \log n \rceil, O[i]} \Longrightarrow L_j^{8,\lceil \log n \rceil, O[j]}$.

Proof. The general outline of the proof is as follows. Recall that a phase of the scan execution consists of $8 \log n$ collect operation executions, where each consecutive eight of them are called a *level* in the phase. The "first" level is the earliest collected and the "$\log n$th" is the latest. The proof begins with Claim 5.4.1, which states that the relation between two labels in any two collects can be extended to the collects preceding and following them. Then in Claims 5.4.3, 5.4.4, and 5.4.5, it is shown that among the labels of any eight collects in a level of the scan, two labels can be chosen for which the order $\Longrightarrow$ is known. Based on Claim 5.4.1 and the transitivity of $\Longrightarrow$, the results of comparing labels of x and y in one level and y and z in a lower level are extended to relate those of x and z, allowing us to show (Claim 5.4.6) that for any k and R, if s is returned by *select*$(\lceil \log n \rceil, k, R)$, then $L_i^{1,1,k} \Longrightarrow L_s^{8,\lceil \log n \rceil,k}$ for all $i \in R - \{s\}$. Finally, transitivity is used again to prove Lemma 5.4, that is, that the results of different phases (*select* executions) are comparable and that the order $\Longrightarrow$ among the labels returned is the order of the phases.

To simplify the presentation, in what follows, indexes will be dropped when it is clear from the context what they should be. This will include the index of the process i performing the *scan* or *collect* operation. The notation C_w will denote $C_i^{[w]}$, the wth collect operation execution performed during a given scan. A label associated with $L_x^{[a]}$, read in any C_w, will be denoted by $\ell_{x,w}^{[a]}$ or ℓ_x^w, and the labeling operation execution $L_x^{[a]}$ itself will be similarly denoted by $L_{x,w}^{[a]}$ or L_x^w.

The following claim will be used to assert that the relation between two labels in any two collects can be extended to the collects preceding and following them. More specifically, this claim asserts that if the label of x is ordered before that of y, where x's label was collected in a collect C_{w+1}, earlier than C_{w+2} in which y's was collected, then any label of x collected in collect C_w that precedes collect C_{w+1} must be ordered before that of y and, similarly, any label of y from collect C_{w+3} must be ordered after that of x.

CLAIM 5.4.1. *If $C_w \xrightarrow{*} C_{w+1} \xrightarrow{*} C_{w+2}$ (or $C_{w+1} \xrightarrow{*} C_{w+2} \xrightarrow{*} C_{w+3}$) and if for some $\ell_{x,w+1}^{[a]}$ and $\ell_{y,w+2}^{[b]}$, $L_x^{[a]} \Longrightarrow L_y^{[b]}$, then for any $\ell_{x,w}^{[c]}$ (similarly, $\ell_{y,w+3}^{[d]}$), it is the case that $L_x^{[c]} \Longrightarrow L_y^{[b]}$ (similarly, $L_x^{[a]} \Longrightarrow L_y^{[d]}$).*

Proof. For any x, if $a \neq c$, that is, if they are of different labeling operations, then it must be the case that $L_x^{[c]} \xrightarrow{*} L_x^{[a]}$. The reason is that if this were not the case, then since $\ell_{x,w}^{[c]}$ was read in C_w and $L_x^{[a]} \xrightarrow{*} L_x^{[c]}$, by atomic register axiom B5, it could not be the case that $\ell_{x,w+1}^{[a]}$ was read in the later collect C_{w+1}, a contradiction. By Definition 4.3, it is thus the case that $L_x^{[c]} \Longrightarrow L_x^{[a]} \Longrightarrow L_y^{[b]}$, which by transitivity (Claim 5.3.2) implies $L_x^{[c]} \Longrightarrow L_y^{[b]}$. By a similar proof, $L_x^{[a]} \Longrightarrow L_y^{[d]}$. ◻

CLAIM 5.4.2. *For any eight collect operation executions of level m of phase k in a given scan operation execution, if the condition*

$$(\exists c1, c2 \in \{1..8\})\ (c1 < c2) \wedge (\ell_x^{c1,m,k} \preceq \ell_y^{c2,m,k}),$$

holds, then $L_x^{1,m,k} \Longrightarrow L_y^{8,m,k}$, and otherwise $L_y^{1,m,k} \Longrightarrow L_x^{8,m,k}$.

Proof. The following claim (Claim 5.4.3) establishes that there are three complementary conditions (one of the three must always hold) on the labels in the eight collects:

1. There are a label of y and a label of x where the label of y was collected in a later collect than that in which x was collected and where the label of y is greater (by $\prec$) than the label of x.

2. This is the first condition with the roles of x and y reversed.

3. The labels of x and y have each changed at least three times during these eight collect operation executions.

The claims that follow show that if the first condition holds, $L_x^{1,m,k} \Longrightarrow L_y^{8,m,k}$, and if one of the other two holds, then $L_y^{1,m,k} \Longrightarrow L_x^{8,m,k}$. More formally, we have the following.

CLAIM 5.4.3. *For the* 16 *labels* $\ell_x^{c1,m,k}$ *and* $\ell_y^{c2,m,k}$, $c1$, $c2 \in \{1..8\}$, *collected in level* m *of phase* k *of a scan operation execution, one of the following three conditions must hold:*

1. $(\exists c1, c2 \in \{1..8\})\ (c1 < c2) \wedge (\ell_x^{c1,m,k} \overset{\vee}{\prec} \ell_y^{c2,m,k})$.
2. $(\exists c1, c2 \in \{1..8\})\ (c1 < c2) \wedge (\ell_y^{c1,m,k} \overset{\vee}{\prec} \ell_x^{c2,m,k})$.
3. *The four labels* $\ell_x^{2,m,k}$, $\ell_x^{4,m,k}$, $\ell_x^{6,m,k}$, *and* $\ell_x^{8,m,k}$ *differ from one another according to* $\prec$, *and the four labels* $\ell_y^{1,m,k}$, $\ell_y^{3,m,k}$, $\ell_y^{5,m,k}$, *and* $\ell_y^{7,m,k}$ *also differ from one another according to* $\prec$.

Proof. Let it be shown that if condition 3 does not hold, then either condition 1 or 2 holds. If condition 3 does not hold, then either

$$(\exists c1, c2 \in \{1..8\})\ (c1 + 1 < c2) \wedge (\ell_x^{c1,m,k} = \ell_x^{c2,m,k})$$

or

$$(\exists c1, c2 \in \{1..8\})\ (c1 + 1 < c2) \wedge (\ell_y^{c1,m,k} = \ell_y^{c2,m,k})$$

Note that labels of the same process can be the same, as denoted by the equivalence sign, though by definition those of different processes always differ by $\overset{\vee}{\prec}$. Without loss of generality, assume that the first condition holds. Then by definition, there must exist a label $\ell_y^{c,m,k}$, $c1 < c < c2$. If $\ell_x^{c1,m,k} \overset{\vee}{\prec} \ell_y^{c,m,k}$, then condition 1 holds and the claim is proven. Thus it must be the case that $\ell_y^{c,m,k} \overset{\vee}{\prec} \ell_x^{c1,m,k}$. However, since $\ell_x^{c1,m,k} = \ell_x^{c2,m,k}$, it is the case that $\ell_y^{c,m,k} \overset{\vee}{\prec} \ell_x^{c2,m,k}$, and condition 2 holds. ∎

By direct application of Claim 5.4.1, the following claim (Claim 5.4.4) implies that if condition 1 of Claim 5.4.3 holds, then $L_x^{1,m,k} \overset{\vee}{\prec} L_y^{8,m,k}$, and similarly, if condition 2 holds, then $L_y^{1,m,k} \overset{\vee}{\prec} L_x^{8,m,k}$. (This follows by exchanging the roles of x and y in Claim 5.4.4 below.)

CLAIM 5.4.4. *If* $\ell_x^{c1,m,k} \overset{\vee}{\prec} \ell_y^{c2,m,k}$, *then* $L_x^{c1,m,k} \Longrightarrow L_y^{c2,m,k}$.

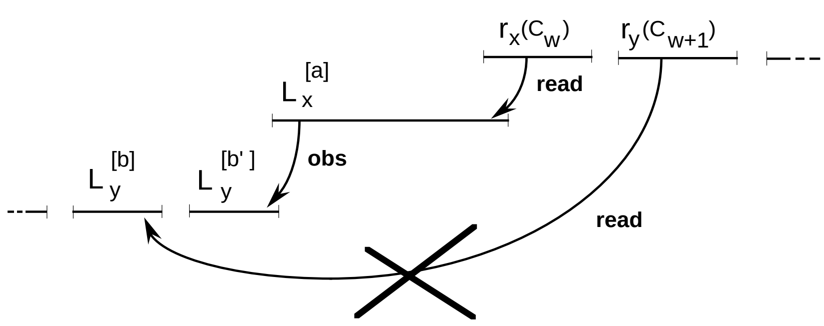

FIG. 12. *Greater and later implies precedence.*

Proof. For simplicity, let $(c1, m, k) = w$ (the label $\ell_x^{c1,m,k}$ read is $\ell_{x,w}^{[a]}$, that is, of labeling operation execution $L_x^{[a]}$) and $(c2, m, k) = w+1$ (similarly, $\ell_y^{c2,m,k}$ is $\ell_{y,w+1}^{[b]}$). The outline of the proof appears in Figure 12. Assume by way of contradiction that $\ell_{x,w}^{[a]} \overset{\vee}{\prec} \ell_{y,w+1}^{[b]}$ and $L_y^{[b]} \Longrightarrow L_x^{[a]}$. By Definition 4.3, it must be that $L_y^{[b]} \xrightarrow{\text{obs}} L_x^{[a]}$. By

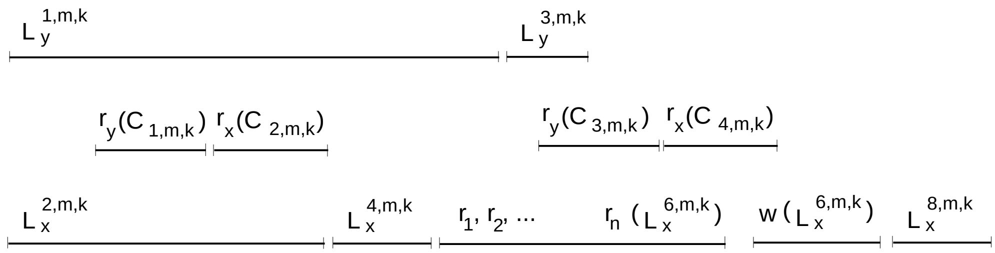

FIG. 13. *Three labeling "moves" are necessary.*

Lemma 5.2, it cannot be that $L_y^{[b]} \in max\_obs(L_x^{[a]})$. Thus there must exist an $L_y^{[b']}$, $b < b'$, such that $L_y^{[b']} \in max\_obs(L_x^{[a]})$. By Corollary 5.1, since $L_y^{[b']} \overset{\text{obs}}{\dashrightarrow} L_x^{[a]}$, there must exist some $r_y(L_\alpha^{[\beta]}) = \ell_y^{[b']}$ such that

$$w(L_y^{[b']}) \dashrightarrow r_y(L_\alpha^{[\beta]}) \longrightarrow w(L_x^{[a]}),$$

where possibly $\alpha = x$ and $\beta = a$. Since $r_y(C_{w+1}) = \ell_y^{[b]}$ and $r_y(L_\alpha^{[\beta]}) = \ell_y^{[b']}$, $b < b'$, by atomic register axiom B5, it must be that $r_y(C_{w+1}) \dashrightarrow r_y(L_\alpha^{[\beta]})$. Similarly, since $\ell_x^{[a]}$ was read in C_w, it must be by axiom B5 that $w(L_x^{[a]}) \dashrightarrow r_x(C_w)$. Thus

$$r_y(C_{w+1}) \dashrightarrow r_y(L_\alpha^{[\beta]}) \longrightarrow w(L_x^{[a]}) \dashrightarrow r_x(C_w),$$

which by axiom A4* of [AB87] implies that $r_y(C_{w+1}) \dashrightarrow r_x(C_w)$, a contradiction to $C_w \xrightarrow{*} C_{w+1}$. □

To complete the proof, it remains to be shown that if the first two conditions of Claim 5.4.3 do not hold (in which case the third one does), it is the case that $L_y^{1,m,k} \Longrightarrow L_x^{8,m,k}$. One can intuitively think of this claim as stating that if each of the processes x and y "moved" (chose a new label) three times, the original labeling operation of y—before the three new ones—must have been completely before the latest labeling operation of x and so precedes it by $\Longrightarrow$. The reason that one needs three "moves" to assure this property becomes clear from the proof. The example in Figure 13 shows why if fewer "moves" are made by each, the property does not hold.

CLAIM 5.4.5. *If the four labels $\ell_x^{2,m,k}$, $\ell_x^{4,m,k}$, $\ell_x^{6,m,k}$, and $\ell_x^{8,m,k}$ differ from one another according to $\prec$ and the four labels $\ell_y^{1,m,k}$, $\ell_y^{3,m,k}$, $\ell_y^{5,m,k}$, and $\ell_y^{7,m,k}$ also differ from one another according to $\prec$, then $L_y^{1,m,k} \Longrightarrow L_x^{8,m,k}$.*

Proof. By serialization axiom B5 of reads and writes from the atomic registers v_x and v_y, it must be the case that

$$L_y^{1,m,k} \xrightarrow{*} w(L_y^{3,m,k}) \dashrightarrow r_y(C_{3,m,k}) \longrightarrow r_x(C_{4,m,k}) \dashrightarrow w(L_x^{6,m,k}) \xrightarrow{*} L_x^{8,m,k}.$$

By applying axiom A4 twice, it follows that $L_y^{1,m,k} \xrightarrow{*} L_x^{8.m,k}$, which by Definition 4.3 implies that $L_y^{1,m,k} \Longrightarrow L_x^{8.m,k}$. □

This completes the proof of Claim 5.4.2. □

The following claim proves the correctness of the recursive procedure *select*.

CLAIM 5.4.6. *For any k and R, if s is returned by select$(\lceil \log n \rceil, k, R)$, then $L_i^{1,1,k} \Longrightarrow L_s^{8,\lceil \log n \rceil,k}$ for all $i \in R - \{s\}$.*

Proof. First, observe that for $||r||$, the size of the input set of $\mathit{select}(m,k,r)$, it follows by simple induction (given that initially $||r|| \leq n$) that $m \geq \lceil \log ||r|| \rceil$. The proof of the claim will thus be by induction on $||r|| \in \{1..\lceil R \rceil\}$.

For $||r|| = 1$, the claim follows vacuously. For $||r|| = 2$, since $m \geq \lceil \log ||r|| \rceil = 1$, the claim follows from Claim 5.4.2. Assume that the claim holds for $||r|| < t$, and let the claim be proven for $||r|| = t$. Since $||\mathit{half}(r)||, ||\mathit{other\_half}(r)|| \leq t/2$, by the induction hypothesis applied to $\mathit{select}(\lceil \log t \rceil - 1, k, \mathit{half}(r))$ and $\mathit{select}(\lceil \log t \rceil - 1, k, \mathit{other\_half}(r))$, it follows that

$$(\forall i \in \mathit{half}(r))\ (L_i^{1,1,k} \Longrightarrow L_x^{8,\lceil \log t \rceil - 1,k}) \quad \text{and}$$
$$(\forall i \in \mathit{other\_half}(r))\ (L_i^{1,1,k} \Longrightarrow L_y^{8,\lceil \log t \rceil - 1,k}).$$

By Claim 5.4.2, without loss of generality, it can be assumed that $L_y^{1,\lceil \log t \rceil,k} \Longrightarrow L_x^{8,\lceil \log t \rceil,k}$ in $\mathit{select}(\lceil \log t \rceil, k, r)$. Thus since $C_{8,\lceil \log t \rceil - 1,k} \xrightarrow{*} C_{1,\lceil \log t \rceil,k}$, by Claim 5.4.1 and the above,

$$L_i^{1,1,k} \Longrightarrow L_y^{8,\lceil \log t \rceil - 1,k} \Longrightarrow L_y^{1,\lceil \log t \rceil,k} \Longrightarrow L_x^{8,\lceil \log t \rceil,k}$$

for every $i \in \mathit{other\_half}(r)$. By transitivity (Claim 5.3.2), it is the case that $L_i^{1,1,k} \Longrightarrow L_x^{8,\lceil \log t \rceil,k}$ for every $i \in \mathit{other\_half}(r)$. Similarly, by Claim 5.4.1 and the above, given that $C_{8,\lceil \log t \rceil - 1,k} \xrightarrow{*} C_{1,\lceil \log t \rceil,k}$, it follows that

$$L_i^{1,1,k} \Longrightarrow L_x^{8,\lceil \log t \rceil - 1,k} \Longrightarrow L_x^{8,\lceil \log t \rceil,k}$$

for every $i \in \mathit{half}(r)$. Again by transitivity, it is the case that $L_i^{1,1,k} \Longrightarrow L_x^{8,\lceil \log t \rceil,k}$ for every $i \in \mathit{half}(r)$, and the claim follows. $\square$

Based on the above claims, the proof can be completed by showing that in any scan operation execution, for any i such that $O[i] < O[j]$, where $\mathit{value}(\ell_i^{8,\lceil \log n \rceil,O[i]})$, $\mathit{value}(\ell_j^{8,\lceil \log n \rceil,O[j]}) \in \bar{\ell}$, it is the case that $L_i^{8,\lceil \log n \rceil,O[i]} \Longrightarrow L_j^{8,\lceil \log n \rceil,O[j]}$. The proof is by induction on k, where $O[i] := k$ in phase k of a scan operation execution. For $k = n$, since there exists no $k', k < k'$, there is no $O[i] < O[j]$, and the claim holds vacuously. Assume that for some $k < n$, the claim holds for all $k', k < k' \leq n$. Let it be proven for k.

Since $k < n$, there is an $O[\alpha] = k+1$ for some $\alpha \in \{1..n\} - \{i\}$ (possibly $\alpha = j$), where $\mathit{value}(\ell_\alpha^{8,\lceil \log n \rceil,k+1}) \in \bar{\ell}$ of the scan operation execution, that is, the returned labeled-value for process α. By Claim 5.4.6,

$$L_i^{1,1,k+1} \Longrightarrow L_\alpha^{8,\lceil \log n \rceil,k+1}$$

for $i \in R$. By Claim 5.4.1, since $C_{8,\lceil \log n \rceil,k} \xrightarrow{*} C_{1,1,k+1}$, it is the case that

$$L_i^{8,\lceil \log n \rceil,k} \Longrightarrow L_\alpha^{8,\lceil \log n \rceil,k+1}.$$

If $\alpha = j$ the lemma follows. If not, by the induction hypothesis, it follows that for any $O[j]$, $k+1 < O[j]$,

$$L_i^{8,\lceil \log n \rceil,k} \Longrightarrow L_\alpha^{8,\lceil \log n \rceil,k+1} \Longrightarrow L_\alpha^{8,\lceil \log n \rceil,O[j]}.$$

By the transitivity of $\Longrightarrow$ (Claim 5.3.2), it then follows that $L_i^{8,\lceil \log n \rceil,O[i]} \Longrightarrow L_j^{8,\lceil \log n \rceil,O[j]}$. $\square$

5.9. Proof of precedence and transitivity. To complete the proof of Theorem 5.1, it remains to be proven that Lemma 5.2 and Claim 5.3.2 hold.

5.9.1. Preliminaries. Given that the definitions of both the graph T^n and the labeling function $\mathcal{L}$ are inductive on k, the first two parts of the following definition simply define the notation to be used in relating labels. The third part is the notion of *inside*(X). X identifies a specific labeling operation execution. In this labeling operation execution, the label chosen was in a certain T^k subgraph on level k. X also identifies this T^k subgraph. The set of labeling operation executions in *inside* are those performed inside T^k from the latest time the process moved into T^k and up to its labeling operation execution X. The *min* is simply the earliest in a sequence of labeling operation executions. For example, *min*(*inside*(X)) is the first among the moves since the process performing X entered T^k.

DEFINITION 5.2. *For $k \in \{1..n\}$ and $\xrightarrow{*}$, the ordering on labeling operation execution, we have the following notation:*

- *Let $\ell_y^{[b]} \overset{k}{=} \ell_x^{[a]}$ denote that $\ell_y^{[b]}[n..k{-}1] = \ell_x^{[a]}[n..k{-}1]$ for $k \geq 2$.*
- *Let $\ell_y^{[b]} \overset{k}{\neq} \ell_x^{[a]}$ (similarly, $\ell_y^{[b]} \overset{k}{\prec} \ell_x^{[a]}$) denote that $\ell_y^{[b]}[n..k] \overset{k+1}{=} \ell_x^{[a]}[n..k]$ and $\ell_y^{[b]}[k{-}1] \neq \ell_x^{[a]}[k{-}1]$ (similarly, $\ell_y^{[b]}[k{-}1]$ is dominated by $\ell_x^{[a]}[k{-}1]$).*
- *Let inside($\ell_x^{[a]}[n..k]$) be a set of operation executions*

$$\{L_x^{[\alpha]} \mid \alpha = a \textit{ or } L_x^{[\alpha]} \xrightarrow{*} L_x^{[a]} \textit{ and } \ell_x^{[\alpha]} \overset{k+1}{=} \ell_x^{[a]} \textit{ and}$$
$$(\forall L_x^{[a']})\ (\textit{if } L_x^{[\alpha]} \xrightarrow{*} L_x^{[a']} \xrightarrow{*} L_x^{[a]}, \textit{ then } \ell_x^{[a']} \overset{k+1}{=} \ell_x^{[a]})\}.$$

- *Let the min of a set of labeling operation executions totally ordered by $\xrightarrow{*}$ be the least element in the ordering.*

If $\ell_x^{[a-1]} \overset{k}{=} \ell_x^{[a]}$, $k = 2$ (the same label by the same process), then let the convention be that $\ell_x^{[a-1]} \overset{k}{\neq} \ell_x^{[a]}$, where $k = 1$ (and similarly for any two equal labels of different labelings by the same process).

5.9.2. The order of induction. The proof of Claim 5.3.2 and Lemma 5.2 will proceed by induction on the system execution $\langle m(\varphi)^L, \xrightarrow{*}, \dashrightarrow^{*} \rangle$ consisting of all labeling operation executions in $m(\varphi)^L$. (Recall that $m(\varphi)^L$ is the set of labeling operation executions in $m(\varphi)$.) The induction base will be the subexecution $m(\varphi)^{L'} = \{L_1^{[0]}, \ldots, L_n^{[0]}\}$ of $m(\varphi)^L$. The induction will proceed to larger subexecutions $m(\varphi)^{L'}$, where $m(\varphi)^{L'} \subseteq m(\varphi)^L$. The subexecution in each step of the induction will include one $L_i^{[a]} \in m(\varphi)^L$ more than its preceding one. The induction order on $\langle m(\varphi)^L, \xrightarrow{*}, \dashrightarrow^{*} \rangle$ is thus that $m(\varphi)^{L'} \cup \{L_i^{[a]}\}$ follows $m(\varphi)^{L'}$, where $L_i^{[a]} \in m(\varphi)^L - m(\varphi)^{L'}$, if for any $L_j^{[b]} \in m(\varphi)^L - m(\varphi)^{L'}$, it is the case that either

- $L_i^{[a]} \overset{\text{obs}}{\dashrightarrow} L_j^{[b]}$ or
- $L_j^{[b]} \overset{\text{obs}}{\not\dashrightarrow} L_i^{[a]}$ and for $\ell_i^{[a-1]} \overset{k}{\prec} \ell_i^{[a]}$ and $\ell_j^{[b-1]} \overset{k'}{\prec} \ell_j^{[b]}$, it is the case that $k' > k$ or that $k' = k$ and $j > i$.

The order is thus to add the labeling operation executions that observed a greater part of the execution later, and if no such labeling operation execution can be identified, settle on choosing the one that was a move (a change in the label) on the lowest-level k.

To see that the above defines a total order of induction, note that $\overset{\text{obs}}{\dashrightarrow}$ is a partial order, and if two labels are not related by $\overset{\text{obs}}{\dashrightarrow}$, they are ordered by the order $<$ on the level in the graph in which they made their last move and by the id if they have the same level. Since $<$ together with the id forms a total order that is independent of the partial order $\overset{\text{obs}}{\dashrightarrow}$, the above order of induction is total.

5.9.3. The induction hypothesis. The induction hypothesis consists of I1 $\wedge$ I2 $\wedge$ I3 $\wedge$ I4, where I1–I4 are as follows:

I1. For any $L_y^{[b]} \in max\_obs(L_x^{[a]})$, it is the case that $\ell_y^{[b]} \overset{V}{\preccurlyeq} \ell_x^{[a]}$.

I2. The relation $\Longrightarrow$ is transitive.

I3. For any $L_x^{[a]}$ and $L_y^{[b]}$, where

- $\ell_y^{[b]}[k-1]$, $\ell_x^{[a]}[k-1] \in \{3,4,5\}$ and
- $\ell_y^{[b]} \overset{k}{\neq} \ell_x^{[a]}$, $k \geq 2$,

if there exist labeling operation executions $\ell_x^{[a-1]}$ and $\ell_y^{[b-1]}$, where

- $\ell_x^{[a-1]} \overset{k}{\neq} \ell_x^{[a]}$ and
- $\ell_y^{[b-1]} \overset{k}{\neq} \ell_y^{[b]}$,

then either $L_x^{[a]} \overset{\text{obs}}{\dashrightarrow} L_y^{[b]}$ or $L_y^{[b]} \overset{\text{obs}}{\dashrightarrow} L_x^{[a]}$.

I4. 1. If $\ell_x^{[a]}[k-1] \in \{2,3,4,5\}$, $k > 2$, then there are at least $k-1$ labels $L_y^{[b]} \in max\_obs(L_x^{[a]})$ such that $\ell_y^{[b]} \overset{k+1}{=} \ell_x^{[a]}$;

2. if there exists an $L_x^{[a1]} \in inside(\ell_x^{[a]}[n..k])$, $\ell_x^{[a1]}[k-1] \in \{4,5\}$ (possibly $a = a1$), then there are exactly $k-1$ labels $L_y^{[b]} \in max\_obs(L_x^{[a]})$ such that $\ell_y^{[b]} \overset{k+1}{=} \ell_x^{[a]}$ and $\ell_y^{[b]}[k-1] \in \{3,4,5\}$; and

3. if $\ell_x^{[a]} \overset{k}{\preccurlyeq} \ell_x^{[a-1]}$ ($\ell_x^{[a]}[k-1]$, $\ell_x^{[a-1]}[k-1] \in \{3,4,5\}$), then for any $L_y^{[b]} \in max\_obs(L_x^{[a]})$, where $\ell_y^{[b]} \overset{k+1}{\neq} \ell_x^{[a-1]}$ and $\ell_y^{[b]}[k-1] \in \{3,4,5\}$, it is the case that $\ell_x^{[a-1]} \overset{k}{\preccurlyeq} \ell_y^{[b]} \overset{k}{\preccurlyeq} \ell_x^{[a]}$.

The induction hypothesis includes four main parts. I1 and I2 are simply Lemma 5.2 and Claim 5.3.2, which are to be proven. However, the proof of these properties is based on several "structural" properties of the labeling operation executions, and these are added in order to strengthen the induction hypothesis.

Property I3 is a weak formulation for the case of any T^k subgraph, $k \geq 2$, of a powerful property that holds in the case of a T^2 subgraph. For $k = 2$, that is, two labels in the cycle of a T^2 subgraph, it is the case that

> among any two labeling operation executions in the cycle, there must be one that observed the other.

Unfortunately, this is not true for any pair of labeling operation executions in a cycle on level $k > 2$. For example, the reader can verify that it is possible that while one process x moves among supernodes 3 and 4 on level k, another process y can concurrently move many times inside supernode 3 (that is, on a level lower than k) with neither x nor y observing a labeling operation execution of the other. However, the property that does hold is that the process x must have observed at least one labeling operation execution by y among those that y executed since it last started choosing labels in supernode 3. (Thus the first move into 3 was definitely observed.) The generalization of this example is formalized by property I3 of the inductive hypothesis.

Property I4 is a collection of three properties that were informally mentioned in section 4.3:

- I4.1 is based on the fact that supernode 1 in any T^k subgraph is a sink in which at least $k-1$ labels must accumulate before a label may be placed on the bridge supernode 2. Because of this accumulation property, any process that performs a labeling operation execution on supernodes $\{2, 3, 4, 5\}$ must have maximally observed at least $k-1$ other labels in the subgraph with him. The maximally observed set of operations of a labeling operation execution $L_x^{[a]}$ ($max\_obs(L_x^{[a]})$) is actually the set of labeling operation executions whose labels, in a sequential execution, could have existed together with $L_x^{[a]}$ at some point in time. Thus I4.1 can be thought of as establishing that if a process completes a labeling operation execution on one of the supernodes $\{2, 3, 4, 5\}$, there are at that point in time at least $k-1$ other labels in the subgraph with him.

- I4.2 is a continuation of the behavior described in I4.1. Again, given that the maximally observed set of operations of a labeling operation execution $L_x^{[a]}$ represents the set of labeling operation executions whose labels, in a sequential execution, could have existed together with $L_x^{[a]}$ at some point in time, I4.2 formalizes the "invariant" that

> at any given time, there cannot be more than k labels in a cycle of a T^k structure.

In addition, not only is it true that there are not more than k, but if any one of these k labels moves inside the cycle, it must maximally observe exactly $k-1$ other labels in the cycle with it.

- Finally, I4.3 strengthens I1 for the particular case in which the new label chosen is dominated by the older label (such as a move from supernode 3 to 5 in the cycle). Based on I1, it could still be that some of the labels maximally observed by the process, though dominated by the new label, are on node 5 together with it. I4.3 establishes that this cannot be the case, that is, all other labels maximally observed in the cycle must be on supernode 4. Property I1 together with I4.3 capture the the "invariant" that

> at any given time, there are never labels on three different nodes of a cycle of a T^k subgraph.

In the next two sections, the induction base and the inductive step are presented.

5.9.4. The induction base.

LEMMA 5.5. *The hypothesis* I1 ∧ I2 ∧ I3 ∧ I4 *holds for* $m(\varphi)^{L'} = \{L_1^{[0]}, \ldots, L_n^{[0]}\}$.

Proof. By definition, initially $max\_obs(L_x^{[a]}) = \emptyset$, and I1 and I4 hold vacuously. Since for any $L_x^{[a]}$, $a = 0$, there does not by definition exist an $L_x^{[a-1]}$, I3 holds vacuously. Also, by definition, for any two labels $\ell_x^{[0]}$ and $\ell_y^{[0]}$, $\ell_x^{[0]} \overset{k1}{\neq} \ell_y^{[0]}$, where $k1 = 1$, and $L_x^{[0]} \overset{\text{obs}}{\not\longleftrightarrow} L_y^{[0]}$. Since $\overset{k1}{\preceq}$ is a total order for level $k1 = 1$, it follows that $\Longrightarrow$ is transitive in $m(\varphi)^{L'}$. $\square$

5.9.5. The induction step.

LEMMA 5.6. *Given that the induction hypothesis* I1 ∧ I2 ∧ I3 ∧ I4 *holds for the system execution* $\langle m(\varphi)^{L'}, \overset{*}{\longrightarrow}, \overset{*}{\dashrightarrow}\rangle$, $m(\varphi)^{L'} \subseteq m(\varphi)^{L}$, *it holds also for* $\langle m(\varphi)^{L'} \cup \{L_x^{[a]}\}, \overset{*}{\longrightarrow}, \overset{*}{\dashrightarrow}\rangle$, *where* $L_i^{[a]} \in m(\varphi)^L - m(\varphi)^{L'}$ *is such that for any* $L_j^{[b]} \in m(\varphi)^L - m(\varphi)^{L'}$, *either*

- $L_i^{[a]} \overset{\text{obs}}{\dashrightarrow} L_j^{[b]}$ *or*
- $L_j^{[b]} \overset{\text{obs}}{\not\longleftrightarrow} L_i^{[a]}$ *and for* $\ell_i^{[a-1]} \overset{k}{\preceq} \ell_i^{[a]}$ *and* $\ell_j^{[b-1]} \overset{k'}{\preceq} \ell_j^{[b]}$, *it is the case that* $k' > k$ *or that* $k' = k$ *and* $j > i$.

The proof of Lemma 5.6 will be separated into several sections. In the following section, several lemmas that will become useful in later sections of the proof are presented and proven. The proof will then proceed by showing that the *maximally observed* set of labeling operation executions by a process x is a good representation of the possible label values that other processes can have given the location of x. In other words, later unobserved labeling operation executions cannot be "far away" from the maximally observed labels, and could definitely not have "cycled around" the current location of x. Based on these established properties, I1, I4, I3, and finally I2 will be proven for the inductive case. The order of presentation of the different lemmas will follow the order of dependency among them.

We make a final important comment: Throughout the proof, unless specifically stated otherwise, $L_x^{[a]}$ will denote the labeling operation execution added in the induction step to form $\langle m(\varphi)^{L'} \cup \{L_x^{[a]}\}, \overset{*}{\longrightarrow}, \overset{*}{\dashrightarrow}\rangle$,

5.9.6. At most k labels in the cycle of a T^k subgraph. In this section, several lemmas are presented, proving a lemma that captures the informal invariant that at any point in time, there can be at most k different labels in the cycle of a T^k subgraph (supernodes $\{3, 4, 5\}$. The following lemma formalizes the notion that "before it can choose a label in the cycle of any T^k subgraph, a process must first raise a flag, that is, choose a label on supernodes 1 or 2 on level k in T^k."

LEMMA 5.7. *For any labeling operation execution $L_x^{[a]}$, if $\ell_x^{[a]}[k-1] \in \{3, 4, 5\}$, $k \geq 2$, then there exists an $L_x^{[a1]} \in inside(\ell_x^{[a]}[n..k])$ such that $L_x^{[a1]}[k-1] \in \{1, 2\}$.*

Proof. Assume by way of contradiction that the claim does not hold. It must thus be that for $L_x^{[a2]} = min(inside(\ell_x^{[a]}[n..k]))$, $\ell_x^{[a2]}[k-1] \in \{3, 4, 5\}$. This implies that there is a labeling operation execution $\ell_x^{[a2-1]} \overset{k+1}{\neq} \ell_x^{[a2]}$. By the definition of $\mathcal{L}$, in order for $\ell_x^{[a2]}[k-1]$ to be in $\{3, 4, 5\}$, it must be that for $\ell_{\max}$, the maximal label in the dominating set read by $L_x^{[a2]}$, we have the following:

- $\ell_{\max} \overset{k+1}{=} \ell_x^{[a2]}$ (as a reminder, this means $\ell_{\max}[n..k] = \ell_x^{[a2]}[n..k]$),
- $\ell_{\max}[k-1] \in \{2, 3, 4, 5\}$, and
- $\mathcal{L}^k(G)$ (the kth level of the recursion in $\mathcal{L}$) was executed for $G = \ell_{\max}[n..k]$

and returned the value $\ell_x^{[a2]}[k-1] = 3$ (as in line 3) or $\ell_x^{[a2]}[k-1] = dom(\ell_{\max}[k-1])$ (as in line 4 or 5).

But this implies that when executing $\mathcal{L}^{k+1}$, it must have been line 4 that was executed because from the above the conditions of lines 1–3 are not met and because

- $\ell_{\max}[k-1] \in \{2, 3, 4, 5\}$, $k \geq 2$, and
- $\ell_{\max}[n..k] = \ell_x^{[a2]}[n..k] \neq \ell_x^{[a2-1]}[n..k]$ ($\ell_x^{[a2-1]}$ is ℓ_i in line 4).

But this implies that $\ell_x^{[a2]}[k] = dom(\ell_{\max}[k])$, that is, x would not execute $\mathcal{L}^k(G)$ for $G = \ell_{\max}[n..k]$ in the first place, a contradiction. $\square$

The following lemma establishes that if in an earlier labeling operation execution a label $\ell_y^{[b]}$ was observed, the current labeling operation execution must read that label for y or a label later than it.

LEMMA 5.8. *If $L_y^{[b]} \overset{\text{obs}}{\dashrightarrow} L_x^{[a-1]}$, it cannot be that $r_y(L_x^{[a]}) = \ell_y^{[b1]}$, where $b1 < b$.*

Proof. By Corollary 5.1, it follows that if $L_y^{[b]} \overset{\text{obs}}{\dashrightarrow} L_x^{[a-1]}$, then there exists a read $r_y(L_\alpha^{[\beta]}) = \ell_y^{[b]}$ such that

$$w(L_y^{[b]}) \dashrightarrow r_y(L_\alpha^{[\beta]}) \longrightarrow w(L_x^{[a-1]}),$$

where possibly $\alpha = x$ and $\beta = a - 1$. Since $w(L_x^{[a-1]}) \longrightarrow r_y(L_x^{[a]})$, it follows that $r_y(L_\alpha^{[\beta]}) \longrightarrow r_y(L_x^{[a]})$. Since, in addition, $w(L_y^{[b1]}) \longrightarrow w(L_y^{[b]})$, by register axiom B5, it cannot be that $r_y(L_x^{[a]}) = \ell_y^{[b]}$. $\square$

The following lemma states that there cannot be a label read by $L_x^{[a]}$ that dominates $\ell_x^{[a-1]}$ on level $k1 > k$, where k is the level such that $\ell_x^{[a-1]} \overset{k}{\neq} \ell_x^{[a]}$.

LEMMA 5.9. *For $\ell_x^{[a]} \overset{k}{\neq} \ell_x^{[a-1]}$, it cannot be that there is an $L_y^{[b]}$ such that*

- $r_y(L_x^{[a]}) = \ell_y^{[b]}$ *and*
- $\ell_x^{[a-1]} \overset{k1}{\overset{\vee}{\prec}} \ell_y^{[b]}$, *where $k1 > k$.*

Proof. By the definition of $\overset{\vee}{\prec}$, it must be that $\ell_x^{[a]} \overset{k1}{\overset{\vee}{\prec}} \ell_y^{[b]}$, where $k1 \geq k$. By the definition of $\mathcal{L}$, either $\ell_{\max} \overset{\vee}{\prec} \ell_x^{[a]}$ or $\ell_{\max}$ is equal to $\ell_x^{[a]}$ (in which case by definition $\ell_{\max}$ is just $\ell_x^{[a-1]}$). The reason is that when executing $\mathcal{L}^{k3}$ for some level $k3$, $\ell_{\max}[k3] = \ell_x^{[a]}[k3]$ or $\ell_{\max}[k3] = dom(\ell_x^{[a]}[k3])$. It thus must be that $max \neq y$. It can either be the case that $\ell_y^{[b]} \overset{\vee}{\prec} \ell_{\max}$ or not.

If indeed $\ell_y^{[b]} \overset{\vee}{\prec} \ell_{\max}$, by the definition of $\overset{\vee}{\prec}$, in order for $\ell_x^{[a]} \overset{\vee}{\prec} \ell_y^{[b]}$, $\ell_y^{[b]} \overset{\vee}{\prec} \ell_{\max}$, and either $\ell_{\max} \overset{\vee}{\prec} \ell_x^{[a]}$ or $\ell_{\max} = \ell_x^{[a]}$, it must be that

$$\ell_x^{[a]} \overset{\vee}{\prec} \ell_y^{[b]} \overset{\vee}{\prec} \ell_{\max} \overset{\vee}{\prec} \ell_x^{[a]},$$

that is, the three labels are also on a cycle. Since by the definition of

$$max(dominating\_set(\ell, \ell_x^{[a-1]})),$$

either $\ell_x^{[a-1]} \overset{\vee}{\prec} \ell_{\max}$ or $\ell_{\max} = \ell_x^{[a-1]}$, it follows that $k \geq k1$, a contradiction.

However, if $\ell_{\max} \overset{\vee}{\prec} \ell_y^{[b]}$, by the definition of $max(dominating\_set(\ell, \ell_x^{[a-1]}))$, it could be only if the labels of y and max were on a cycle on a level $k2$, where $2 \leq k2 \leq k1$ ($k1$ is the level such that $\ell_x^{[a-1]} \overset{k1}{\overset{\vee}{\prec}} \ell_y^{[b]}$).[15] In order for $\ell_{\max} \overset{\vee}{\prec} \ell_x^{[a]}$ or $\ell_{\max} = \ell_x^{[a]}$, together with $\ell_x^{[a]} \overset{\vee}{\prec} \ell_y^{[b]}$, it must be that $\ell_x^{[a]}$ is in the cycle with these two labels. However, this implies $k \geq k1$, a contradiction. $\square$

The following lemma captures the informal invariant that at any point in time, there can be at most k different labels in the cycle of any T^k subgraph. More precisely, it states that for any set of more than k labeling operation executions whose labels are in the cycle of the same T^k subgraph, all could not have been there at the same point in time since at least one of them must have already been observed by the others in a later location outside the cycle.

LEMMA 5.10. *Let $\mathcal{S}^k = \{L_{i_1}^{[a_1]}, L_{i_2}^{[a_2]}, \ldots, L_{i_m}^{[a_m]}\}$, $i_1, \ldots, i_m \in \{1..n\}$, and $i_\alpha \neq i_\beta$ for any $\alpha, \beta \in \{1..m\}$ be the set of labeling operation executions such that for any $L_i^{[a]}$, $L_j^{[b]} \in \mathcal{S}^k$,*

- $\ell_i^{[a]} \overset{k+1}{=} \ell_j^{[b]}$ *and* $\ell_i^{[a]}[k-1]$, $\ell_j^{[b]}[k-1] \in \{3, 4, 5\}$, *and*
- *for* $L_j^{[b1]} \in max\_obs(L_i^{[a]})$ *and* $L_i^{[a1]} \in max\_obs(L_j^{[b]})$, *it is the case that* $b1 \leq b$ *and* $a1 \leq a$.

It must be that $||\mathcal{S}^k|| \leq k$.

Proof. Assume by way of contradiction that $||\mathcal{S}^k|| > k$. By Lemma 5.7, for each $L_i^{[a]} \in \mathcal{S}^k$, $||inside(L_i^{[a]}[n..k])|| \geq 2$, that is, it is included at least two labeling operation executions inside the T^k subgraph that $L_i^{[a]}$ is in. Let us define the relation *not_read_by* between labeling operation executions $L_i^{[a]}, L_j^{[b]} \in \mathcal{S}^k$ to be as follows.

DEFINITION 5.3. $L_i^{[a]}$ *not_read_by* $L_j^{[b]}$ *if* $r_i(L_j^{[b]}) \neq \ell_i^{[a1]}$, $a1 \in \{a-1, a\}$.

[15] The reason for this is that if the two labels are on different supernodes of a cycle, there could be a third label on the other supernode of the cycle, and any one of them could be selected as $\ell_{\max}$.

That is, $L_j^{[b]}$ did not read a label of a labeling operation execution $L_i^{[a]}$ or its preceding operation execution in the T^k that it is in. The contradiction will be derived by showing that there must be at least one labeling operation execution $L_i^{[a]} \in \mathcal{S}^k$ that read at least $k+1$ labels (including its own) in the T^k subgraph that $L_i^{[a]}$ and $L_i^{[a-1]}$ are in. This is the *flag principal* mentioned in section 4.3. Since for each $L_i^{[a]} \in \mathcal{S}^k$, $\ell_i^{[a]} \underline{\overset{k}{=}}\ \ell_i^{[a-1]}$, that is, a move at level k, it must be that when executing $\mathcal{L}^{k+1}$ in $L_i^{[a]}$, line 5 was executed and that *num_labels* $< (k+1)-1$ (at most $k-1$ labels not including its own, or k including it, were read in the T^k subgraph $\ell_i^{[a]}[n..k]$), a contradiction.

Since it was assumed by way of contradiction that there are more than k labeling operation executions in $\mathcal{S}^k$, it must be that each labeling operation execution did not read (*not_read_by*) at least one of the others. Let it first be shown that the relation *not_read_by* is antisymmetric.

CLAIM 5.10.1. *For any $L_i^{[a]}$ and $L_j^{[b]}$ in $\mathcal{S}^k$, if $L_i^{[a]}$ not_read_by $L_j^{[b]}$, then it cannot be that $L_j^{[b]}$ not_read_by $L_i^{[a]}$.*

Proof. For any two labeling operation executions $L_i^{[a]}$ and $L_j^{[b]}$ in $\mathcal{S}^k$, by definition (for $L_j^{[b1]} \in max\_obs(L_i^{[a]})$ and $L_i^{[a1]} \in max\_obs(L_j^{[b]})$, it is the case that $b1 \le b$ and $a1 \le a$,) neither $r_i(L_j^{[b]}) = \ell_i^{[a1]}$, $a1 > a$, nor $r_j(L_i^{[a]}) = \ell_j^{[b1]}$, $b1 > b$. Given $L_i^{[a]}$ *not_read_by* $L_j^{[b]}$, it thus follows by atomic register axiom B5 that $r_i(L_j^{[b]}) \dashrightarrow w(L_i^{[a-1]})$. However, this implies

$$w(L_j^{[b-1]}) \longrightarrow r_i(L_j^{[b]}) \dashrightarrow w(L_i^{[a-1]}) \longrightarrow r_j(L_i^{[a]}).$$

By axiom A4, it follows that $w(L_j^{[b-1]}) \longrightarrow r_j(L_i^{[a]})$, implying that it cannot be that $L_j^{[b]}$ *not_read_by* $L_i^{[a]}$. □

Think of the relation *not_read_by* as the set of edges of a directed graph whose nodes are labeling operation executions, where an edge is directed from $L_i^{[a]}$ to $L_j^{[b]}$ if $L_i^{[a]}$ *not_read_by* $L_j^{[b]}$. Each labeling operation execution in $\mathcal{S} \cup \{L_x^{[a]}\}$ did not read at least one of the others; each node has at least one incoming edge. By known graph-theoretic arguments, this implies that

- there are two nodes that have edges directed one at the other or
- there is at least one node $L_s^{[c]}$ that has a directed path leading from it to every other node in the graph.

By Claim 5.10.1 (antisymmetry of *not_read_by*), the former is impossible.[16] The following claim establishes that the labeling operation execution associated with the node $L_s^{[c]}$ from which all other nodes are reachable (note that by assumption, this node has at least one incoming edge and is not a "root") must have read all of them.

CLAIM 5.10.2. *For any subset $\{L_{i_1}^{[a_1]}, L_{i_2}^{[a_2]}, \ldots, L_{i_m}^{[a_m]}\}$ of m labeling operation executions in $\mathcal{S}^k$, where*

$$L_{i_1}^{[a_1]}\ \textit{not\_read\_by}\ L_{i_2}^{[a_2]}\ \textit{not\_read\_by} \cdots \textit{not\_read\_by}\ L_{i_m}^{[a_m]},$$

it is the case that $r_{i_m}(L_{i_1}^{[a_1]}) = L_{i_m}^{[a_m]}$.

[16] Note that if the former does not hold, there is a cycle in the graph. If the relation *not_read_by* were transitive, a cycle would be impossible, and the proof would be complete. However, the reader can verify that this is not the case.

Proof. For any two labeling operation executions $L_i^{[a]}, L_j^{[b]} \in \mathcal{S}^k$, it follows by definition that neither $r_i(L_j^{[b]}) = \ell_i^{[a1]}$, $a1 > a$, nor $r_j(L_i^{[a]}) = \ell_j^{[b1]}$, $b1 > b$.

Let it be proven by induction that $r_{i_{m-1}}(L_{i_m}^{[a_m]}) \dashrightarrow w(L_{i_1}^{[a_1-1]})$. This will imply

$$w(L_{i_m}^{[a_m-1]}) \longrightarrow r_{i_{m-1}}(L_{i_m}^{[a_m]}) \dashrightarrow w(L_{i_1}^{[a_1-1]}) \longrightarrow r_{i_m}(L_{i_1}^{[a_1]}),$$

from which by axiom A4 follows $w(L_{i_m}^{[a_m-1]}) \longrightarrow r_{i_m}(L_{i_1}^{[a_1]})$, implying, $r_{i_m}(L_{i_1}^{[a_1]}) = L_{i_m}^{[a_m]}$, as desired.

The proof that $r_{i_{m-1}}(L_{i_m}^{[a_m]}) \dashrightarrow w(L_{i_1}^{[a_1-1]})$ is by induction on m, the size of the subset of labeling operation executions. For $m = 2$, it follows by definition. Assume it holds for sequences of length $m - 1$, that is,

$$r_{i_{m-2}}(L_{i_{m-1}}^{[a_{m-1}]}) \dashrightarrow w(L_{i_1}^{[a_1-1]}).$$

Since $r_{i_{m-1}}(L_{i_m}^{[a_m]}) \neq \ell_{i_{m-1}}^{[a_{m-1}]}$, it follows that

$$r_{i_{m-1}}(L_{i_m}^{[a_m]}) \dashrightarrow w(L_{i_{m-1}}^{[a_{m-1}-1]}) \longrightarrow r_{i_{m-2}}(L_{i_{m-1}}^{[a_{m-1}]}) \dashrightarrow w(L_{i_1}^{[a_1-1]}).$$

By axiom A4*, it follows that $r_{i_{m-1}}(L_{i_m}^{[a_m]}) \dashrightarrow w(L_{i_1}^{[a_1-1]})$, implying the claim. ∎

Thus the node $L_s^{[c]}$ read at least k labels apart from its own in the T^k subgraph $L_s^{[c]}[n..k]$, providing the desired contradiction. ∎

Based on the above, the following lemma, which is part of the proof of I4.2 for the inductive case, can be shown. As mentioned before, the maximally observed set $max\_obs(L_x^{[a]})$ is actually the set of labeling operation executions whose labels, in a sequential execution, could have existed together with $L_x^{[a]}$ at some point in time. The lemma thus captures the informal notion that if one could look at the cycle of a T^k subgraph at a given point in time in which x had a label $\ell_x^{[a]}$ in it, there would be at most $k-1$ other labels in the cycle together with it.

LEMMA 5.11. *For $\ell_x^{[a]}[k] \in \{3,4,5\}$, there are at most $k-1$ labels $L_y^{[b]}$, where $L_y^{[b]} \in max\_obs(L_x^{[a]})$, such that $\ell_y^{[b]} \overset{k\pm 1}{=} \ell_x^{[a]}$ and $\ell_y^{[b]}[k-1] \in \{3,4,5\}$.*

Proof. For any two labeling operation executions $L_y^{[b]}, L_z^{[c]} \in max\_obs(L_x^{[a]})$, by the definition of $max\_obs(L_x^{[a]})$, neither $r_y(L_z^{[c]}) = \ell_y^{[b1]}$, $b1 > b$, nor $r_z(L_y^{[b]}) = \ell_z^{[c1]}$, $c1 > c$. Also, by definition, for $L_y^{[b]} \in max\_obs(L_x^{[a]})$, neither $r_y(L_x^{[a]}) = \ell_y^{[b1]}$, $b1 > b$, nor $r_x(L_y^{[b]}) = \ell_x^{[a1]}$, $a1 \geq a$. The claim follows from Lemma 5.10 by defining $\mathcal{S}^k$ to be the set of labeling operation executions maximally observed by $L_x^{[a]}$ together with $L_x^{[a]}$ itself. ∎

The completion of the inductive argument involves a proof of properties I1–I4 through rather tedious case analysis. It is ommitted from this manuscript and can be found in [Sha90].

6. Discussion. There are three main types of problems defined in the shared-memory model:

- *waiting problems*, whose solution allows a process to take an infinite number of steps to complete an operation—that is, it could "busy-wait" for some other processes indefinitely;
- *wait-free problems*, whose solution is such that each process is *guaranteed* to complete an operation within a finite number of steps, independently of the pace of other processes; and

- *expected-wait-free problems*, whose solution is such that each process is *expected* (rather than guaranteed) to complete an operation within a finite number of steps, independently of the pace of other processes.

These classes of problems are fundamentally different from one another. However, they have the unifying theme that

> if the requirement that memory size be bounded is dropped, the problems have elegant and simple unbounded solutions based on the use of a CTSS.

The main implication of bounded concurrent time-stamping is that this unifying theme, true under the assumption that memory size can be unbounded, holds true for the bounded-memory case as well.

Based on the use of a bounded CTSS implementation, simple unbounded solutions can be given for what are considered to be core problems in each category and then directly transformed into bounded ones. Examples of problems and algorithms in the first category are the famous first-come first-served mutual-exclusion problem of Lamport [Lam74] (see [Lam86b, Ray86] for complete details) and the *fifo-ℓ*-exclusion problem of [AD*94, FLBB79, FLBB89]. As mentioned earlier, a CTSS-based solution due to Afek et al. can be found in [AD*94].

In the second category, we have Li and Vitanyi's simple version [LV87] of the elegant unbounded Vitanyi–Awerbuch algorithm [VA86] for solving the problem of providing a wait-free construction of an MRMW atomic register from SWMR atomic registers (see also [PB87, IL93, Sch88, **?**]). This algorithm can be immediately transformed into a bounded solution (see [G92]).

In the third category, a version (see [Sha90]) of the algorithm of Abrahamson [Abr88] based on the use of a CTSS can be modularly transformed into a bounded solution to the randomized consensus problem of [CIL87].

6.1. Further related research. The introduction of the concurrent time-stamping paradigm in the conference version of this paper [DS89] has led researchers to devising a series of alternative bounded CTSS algorithms. Israeli and Pinchasov [IP91] have provided a linear-complexity version of our algorithm by dropping the requirement that scan operations do not perform writes. In [DW92], Dwork and Waarts present the most efficient read/write-register-based CTSS construction to date, taking only $O(n)$ time for either a scan or update. They model their bounded construction after a new type of unbounded CTSS construction, where processes choose from "local pools" of label values instead of the simple "global-pool"-based CTSS as in the bakery algorithm [Lam74]. In order to bound the number of possible label values in the local pool of the bounded implementation, they introduce a form of garbage collection on "old" labels. They then prove that the linear-time bounded implementation meets the CTSS axioms of section 2. In [DPHW92], Dwork, Herlihy, Plotkin, and Waarts introduce an alternative linear-complexity bounded CTSS construction that combines a time-lapse snapshot with our bounded CTSS algorithm. The proof of their algorithm leverages the axiomatic proof in this paper by arguing that the executions of their algorithm are a subset of the executions of our algorithm. In [GLS92], Gawlick, Lynch, and Shavit introduce a streamlined version of our CTSS algorithm based on the use of an atomic snapshot primitive [AAD*89, And89a]. A snapshot primitive allows a process P_i to *update* the ith memory location, or *snap* the memory, that is, collect an "instantaneous" view of all n shared-memory locations. By using a snapshot primitive, they limit the number of interleavings that can occur and are able to introduce a considerably simplified version of our labeling algorithm (though a logarithmic factor less efficient) that is tailored to allow a forward-simulation proof [LT87]. An advantage of their algorithm over other solutions is that it is no longer limited to read/write memory, providing a CTSS construction in any computation model whose basic operations suffice to provide a wait-free snapshot implementation, be it single-writer multireader registers [A93], multireader multiwriter registers [ICMT94], consensus objects [CD93], or memory with hardware supported *compare-and-swap* and *fetch-and-add* primitives.

Acknowledgments. We would like to thank Yehuda Afek, Hagit Attiya, Eli Gafni, Rainer Gawlick, Maurice Herlihy, Nancy Lynch, and Mike Merritt for many important conversations and comments. It was a subtle observation of Mike's regarding pairwise consistency among scans that led us to the current CTSS definitions. A subsequent observation by Rainer led us to add property P4 to the CTSS specification.

Finally, the second author would like to thank Nancy Lynch, Baruch Awerbuch, and the members of MIT's Theory of Distributed Systems group for their warm hospitality throughout the writing of this paper.

REFERENCES

[AAD*89] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit, *Atomic snapshots of shared memory*, J. Assoc. Comput. Mach., 40 (1993), pp. 873–890.

[AB87] U. Abraham and S. Ben-David, *Informal and formal correctness proofs for programs (for the critical section problem)*, unpublished manuscript, Technion, Haifa, Israel, 1987.

[Abr88] K. Abrahamson, *On achieving consensus using a shared memory*, in Proc. 7th ACM Symposium on Principles of Distributed Computing, ACM, New York, 1988, pp. 291–302.

[AD*94] Y. Afek, D. Dolev, E. Gafni, M. Merritt, and N. Shavit, *A bounded first-in, first-enabled solution to the ℓ-exclusion problem*, ACM Trans. Programming Lang. Systems, 16 (1994), pp. 939–953.

[A93] H. Attiya and O. Rachman, *Atomic snapshots in $O(n \log n)$ operations*, in Proc. 12th ACM Symposium on Principles of Distributed Computing, ACM, New York, 1993, pp. 29–40.

[AG90] J. Anderson and M. Gouda, *The virtue of patience: Concurrent programming with and without waiting*, Technical Report TR-90-23, Department of Computer Science, University of Texas at Austin, Austin, TX, 1990.

[And89a] J. H. Anderson, *Multi-writer composite registers*, Distrib. Comput., 7 (1994), pp. 175–195.

[Ben88] S. Ben-David, *The global time assumption and semantics for concurrent systems*, in Proc. 7th ACM Symposium on Principles of Distributed Computing, ACM, New York, 1988, pp. 223–231.

[Blo88] B. Bloom, *Constructing two-writer atomic registers*, in Proc. 6th ACM Symposium on Principles of Distributed Computing, ACM, New York, 1987; revised version, IEEE Trans. Commun., 37 (1988), pp. 1506–1514.

[BP87] J. Burns and G. Peterson, *Constructing multi-reader atomic values from non-atomic values*, in Proc. 6th ACM Symposium on Principles of Distributed Computing, ACM, New York, 1987, pp. 221–231.

[CD93] T. D. Chandra and C. Dwork, *Using consensus to solve atomic snapshots*, manuscript, 1993.

[CIL87] B. Chor, A. Israeli, and M. Li, *Wait-free consensus using asynchronous hardware*, SIAM J. Comput., 23 (1994), pp. 701–712.

[CS93] R. Cori and E. Sopena, *Some combinatorial aspects of timestamp systems*, European J. Combin., 14 (1993), pp. 95–102.

[DS89] D. Dolev and N. Shavit, *Bounded concurrent time-stamp systems are constructible*, in Proc. 21st ACM Symposium on Theory of Computing, ACM, New York, 1989, pp. 454–465.

[DW92] C. Dwork and O. Waarts, *Simple and efficient bounded concurrent timestamping, or, bounded concurrent timestamp systems are comprehensible!*, in Proc. 24th

ACM Symposium on Theory of Computing, ACM, New York, 1992, pp. 655–666.

[DPHW92] C. Dwork, M. Herlihy, S. Plotkin, and O. Waarts, *Time lapse snapshots*, in Proc. Israel Symposium on the Theory of Computing and Systems, D. Dolev, Z. Galil, and M. Rodeh, eds., Lecture Notes in Comput. Sci. 601, Springer-Verlag, Berlin, 1992, pp. 154–170.

[DGS88] D. Dolev, E. Gafni, and N. Shavit, *Towards a non-atomic era: ℓ-exclusion as a test case*, in Proc. 20th ACM Symposium on Theory of Computing, ACM SIGACT, ACM, New York, 1988, pp. 78–92.

[Dij65] E. W. Dijkstra, *Solution of a problem in concurrent programming control*, Comm. Assoc. Comput. Mach., 8 (1965), p. 569.

[FLBB79] M. Fischer, N. Lynch, J. Burns, and A. Borodin, *Resource allocation with immunity to limited process failure*, in Proc. 20th Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1979, pp. 234–254.

[FLBB89] M. Fischer, N. Lynch, J. Burns, and A. Borodin, *Distributed fifo allocation of identical resources using small shared space*, ACM Trans. Programming Lang. Systems, 11 (1989), pp. 90–114.

[G92] R. Gawlick, *Concurrent timestamping made simple*, Masters thesis, Technical Report MIT/LCS/TR-556, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1992.

[GLS92] R. Gawlick, N. Lynch, and N. Shavit, *Concurrent time-stamping made simple*, in Proc. Annual Israel Symposium on Theory of Computing and Systems, D. Dolev, Z. Galil, and M. Rodeh, eds., Lecture Notes in Comput. Sci. 601, Springer-Verlag, Berlin, 1992, pp. 171–185.

[Her91] M. P. Herlihy, *Wait-free synchronization*, ACM Trans. Programming Lang. Systems, 13 (1991), pp. 124–149.

[ICMT94] M. Inoue, W. Chen, T. Masuzawa and N. Tokura, *Linear-time snapshot using multi-writer multi-reader registers*, in Workshop on Distributed Algorithms, Springer-Verlag, Berlin, 1994, pp. 130–140.

[HW88] M. P. Herlihy and J. Wing, *Linearizability: A correctness condition for concurrent objects*, ACM Trans. Programming Lang. Systems, 12 (1990), pp. 463–492.

[IL93] A. Israeli and M. Li, *Bounded time stamps*, Distrib. Comput., 6 (1993), pp. 205–209.

[IP91] A. Israeli and M. Pinchasov, *A linear time bounded concurrent timestamp scheme*, Technical Report, Technion, Haifa, Israel, 1991.

[Kat78] H. Katseff, *A new solution to the critical section problem*, in Proc. 10th ACM Symposium on Theory of Computing, ACM, New York, 1978, pp. 86–88.

[Lam74] L. Lamport, *A new solution of Dijkstra's concurrent programming problem*, Comm. Assoc. Comput. Mach., 17 (1974), pp. 453–455.

[Lam77] L. Lamport, *Concurrent reading and writing*, Comm. Assoc. Comput. Mach., 20 (1977), pp. 806–811.

[Lam86a] L. Lamport, *The mutual exclusion problem part* i: *A theory of interprocess communication*, J. Assoc. Comput. Mach., 33 (1986), pp. 313–326.

[Lam86b] L. Lamport, *The mutual exclusion problem part* ii: *Statement and solutions*, J. Assoc. Comput Mach., 33 (1986), pp. 327–348.

[Lam86c] L. Lamport, *On interprocess communication part* i: *Basic formalism*, Distrib. Comput., 1 (1986), pp. 77–85.

[Lam86d] L. Lamport, *On interprocess communication part* ii: *Algorithms*, Distrib. Computing, 1 (1986), pp. 86–101.

[LH89] E. A. Lycklama and V. Hadzilacos, *A first-come-first-served mutual exclusion algorithm with small communication variables*, ACM Trans. Programming Lang. Systems, 13 (1991), pp. 558–576.

[LT87] N. Lynch and M. Tuttle, *Hierarchical correctness proofs for distributed algorithms*, Tecnical Report MIT/LCS/TR-387, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1987.

[LTV96] M. Li, J. Tromp, and P. Vitanyi, *How to share concurrent waitfree variables*, J. Assoc. Comput. Mach., 43 (1996), pp. 723–746 (journal version of [LV87]).

[LV87] M. Li and P. Vitanyi, *A very simple construction for atomic multiwriter registers*, Report, Aiken Computation Laboratory, Harvard University, Cambridge, MA, 1987.

[New87] R. Newman-Wolfe, *A protocol for waitfree atomic multi-reader shared variables*, in Proc. 6th ACM Symposium on Principles of Distributed Computing, ACM, New York, 1987, pp. 232–248.

[PB87] G. L. Peterson and J. Burns, *Concurrent reading while writing* ii: *The multi-writer case*, in Proc. 28th Symposium on Foundations of Computer Science, IEEE

Computer Society Press, Los Alamitos, CA, 1987, pp. 383–392.

[Pet81] G. L. Peterson, *Myths about the mutual exclusion problem*, Inform. Process. Lett., 12 (1981), pp. 115–116.

[Pet83] G. Peterson, *Concurrent reading while writing*, ACM Trans. Programming Lang. Systems, 1 (1983), pp. 46–55.

[Pet88] G. Peterson, personal communication, 1988.

[Ray86] M. Raynal, *Algorithms for Mutual Exclusion*, North Oxford Academic Publishing, Oxford, UK and MIT Press, Cambridge, MA, 1986; originally published as *Algorithmique du Parallélisme*, Dunod Informatique, Paris, 1984 (in French; translated by D. Beeson).

[SAG94] A. Singh, J. Anderson, and M. Gouda, *The elusive atomic register*, J. Assoc. Comput. Mach., 41 (1994), pp. 311–339; original version in Proc. 6th ACM Symposium on Principles of Distributed Computing, ACM, New York, 1987, pp. 206–221.

[Sch88] R. Schaffer, *On the correctness of atomic multi-writer registers*, Bachelor's thesis, Technical Memo MIT/LCS/TM-364, Massachusetts Institute of Technology, Cambridge, MA, 1988.

[Sha90] N. Shavit, *Concurrent time-stamping*, Ph.D. thesis, School of Mathematics and Computer Science, Hebrew University, Jerusalem, 1990.

[SZ91] M. Saks and F. Zaharoglou, *Optimal space distributed move-to-front lists*, in Proc. 10th ACM Symposium on Principles of Distributed Computing, ACM, New York, 1991, pp. 65–73.

[VA86] P. Vitanyi and B. Awerbuch, *Shared register access by asynchronous hardware*, in Proc. 27th Symposium on Foundations of Computer Science, IEEE Computer Society Press, 1986, pp. 233–243.