BOUNDARY 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 multireader atomic registers.

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

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

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 multireader (MWM) 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]).
Fortunately, 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 \[\text{Lam74}\]. Researchers were thus led to devise complicated problem-specific solutions to show that the above problems are solvable in a bounded way.\(^1\)

In \[\text{IL93}\], 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 \[\text{LV87}\], Cori and Sopena \[\text{CS93}\], and Saks and Zaharoglou \[\text{SZ91}\].

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 \[\text{Abr88}\] (which requires one to solve the randomized-consensus problem of \[\text{CIL87}\] without using an atomic coin-flip operation) and the fifo-\(\ell\)-exclusion problem of \[\text{FLBB79, FLBB89}\] (see \[\text{AD*94}\] for details). A bounded CTSS solution to the former problem is given in \[\text{Sha90}\], and in \[\text{AD*94}\], 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 \[\text{IL93}\] 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 \[\text{VA86}\] using our construction (see \[\text{Sha90, G92}\]) has the same space complexity of the \[\text{PB87, Sch88}\] 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 \[\text{Lam86a}\], 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 \[\text{AD*94}\], this property is important when defining liveness conditions such as first-come first-enabled for problems like \(\ell\)-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.\(^1\)

\(^1\) See \[\text{And89a, Blo88, BP87, CIL87, Di65, DG88, FLBB79, FLBB89, Kat78, Lam74, Lam77, Lam86b, LH89, LV87, ?, Ray86, Pet81, Pet83, PB87, VA86}\].

\(^2\) The only prior known solutions to the fifo-\(\ell\)-exclusion problem \[\text{DGS88, Pet88}\] achieve weaker forms of fairness than the original test-and-set-based solution of \[\text{FLBB79}\].
[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 brevity, 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, . . . , 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, labelingi(ℓ) and scani(ℓ, ∗) for i ∈ {1, . . . , n}. A labelingi operation associates an input value, ℓi, taken from any domain D with a label.3 We call ℓi the labeled-value of operation labelingi. 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 scani operation returns as output a pair (ℓ, ∗), where the view ℓ = {ℓ1, . . . , ℓn} is an indexed set of labeled-values (one per process) and ∗ 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 “→”) and where any number of scan operation executions are allowed between any two labeling operation executions. The following,

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

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, \( a \rightarrow b \) or \( b \rightarrow a \). (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 \( \Rightarrow \) 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 \( L_p^{[a]} \rightarrow L_q^{[b]} \), then \( L_p^{[a]} \Rightarrow L_q^{[b]} \).

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

---

3 In order correctly handle initial conditions, the value domain \( D \) 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 \((\ell^{[k]}, i, k)\), where the second and third fields are dummy indexes used only for purposes of the specification.
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. The black box metaphor is used to stress that it suffices to know of the existence of such a total ordering \( \Rightarrow \), 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 \( \ell \) returned by \( S[k] \) 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 \( \ell_{a+1} \) through \( \ell_{a+4} \) can be returned by \( S[k] \), 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 \( - \text{ is the can affect relation of} \ [Lam86c, Lam86a] \).

P2: regularity. For any labeled-value \( \ell_{a} \) in \( \ell \) of \( S[k] \), \( L_{a} \rightarrow S[k] \), and there is no \( L_{b} \) such that \( L_{a} \rightarrow L_{b} \rightarrow S[k] \).

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]). To this end, the following third property is added.

P3: monotonicity. For any labeled-value \( \ell_{a} \) in \( \ell \) of \( S[k] \), there does not exist an \( S[j] \) with a labeled-value \( \ell_{b} \) in its view \( \ell \) such that \( S[k] \rightarrow S[j] \) and \( L_{b} \rightarrow L_{a} \) (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.

\[5 \text{ Notice that there is no requirement that labeled-values returned by different scans must be comparable.} \]
Property P4 is an extension of part of the regularity property to the \( \Rightarrow \) 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: \textit{order regularity}. For any labeled-value \( \ell_p^{[a]} \) in \( \ell \) of \( S_i^{[k]} \), \( S_i^{[k]} \rightarrow L_q^{[b]} \) implies that \( L_p^{[a]} \Rightarrow L_q^{[b]} \).

3. \textbf{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 \textit{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 \textit{labeling}, operation writes \( \ell_i \) to register \( v_i \). In our implementation, \( \ell_i \) is a data type consisting of two fields, a labeled-value, denoted \textit{value}(\( \ell_i \)), and its associated label, denoted \textit{label}(\( \ell_i \)). Each \textit{label}(\( \ell_i \)) is a pair of the form \( (\text{number}_i, i) \), where \text{number}_i is a natural number and \( i \in \{1..n\} \) is the id of the process writing \( \ell_i \).

A process \( i \) collects the labels and values of other processes by performing a \textit{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 \( \ell \) are ordered by \textit{ord}(\( \ell \)), an ordering on their indexes in \( \{1..n\} \), such that \( i \) is smaller than \( j \) if and only if the label \( (\text{number}_i, i) \) is lexicographically smaller than the label \( (\text{number}_j, j) \). Figure 3 provides the pseudocode of the \textit{labeling} and \textit{scan} operations for a process \( i \).

To understand how property P1 is met, consider that if the labeling operation execution of \( \ell_j \) by a process \( i \) completely preceded the labeling operation execution of \( \ell_j \) by \( j \), then it must be that \( j \) chose a label with \text{number}_j > \text{number}_i \) since \( j \) collected \( \ell_i \). If they are concurrent, at worst they might both collect the same maximal label and choose \text{number}_i = \text{number}_j \), in which case they are ordered by their ids. Thus

\[^6\] The need for property P4 in applications such as the multireader multewriter atomic register construction of [LV87, VA86] was discovered by Gawlick [G92].
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 \( \ell_i \) 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.

\[ \ell := \text{collect}; \]
\[ v_i := (val, (\max_{j \in \{1..n\}} \text{number}_j + 1, i)); \]

end;

function scan;
begin
\[ \ell := \text{collect}; \]
return \( \{ \text{value}(\ell_1) \ldots \text{value}(\ell_n) \}, \text{ord}(\ell) \); 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 \( \Rightarrow \) 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 \( \text{label}(\ell_1) \ldots \text{label}(\ell_n) \) used in computing \( \text{ord}(\ell) \) 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.

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 \( \ell_{[k]} \) will represent only the label field written in a labeling operation execution \( L_{[k]} \). 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 \( \prec \) 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 \( \prec \) 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 \( \prec \) among them, are defined in terms of a precedence graph\(^7\) \( (V, \prec) \). Each possible label is a node in this graph. The order among the labels in any two registers is the order \( \prec \) 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 $\alpha$-composition, $G \circ_\alpha H$, of two graphs $G$ and $H$, where $\alpha$ 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 $T^n = (V, \prec)$ 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 $\ell_i$.

The label can be viewed as a string $\ell_i[n..1]$ of $n$ digits, where each $\ell_i[k] \in \{1, \ldots, 5\}$ is the digit of the corresponding node in $T^2$, replaced by a $T^k$ subgraph during the $k$th step of the inductive construction above. The digit $\ell_i[n]$ is always 1, representing the complete $T^n$ graph, and if in $\ell_i$, $\ell_i[k] = 2$, 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 $\ell_i$, 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 $\ell_i$ 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 $i$th 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 $\ell$ of labels. The labeling operation of a process $i$ is of the form described in Figure 5, where $L : V^n \times \{1..n\} \mapsto V$ is a labeling function, returning a label value $\ell_i$ “greater than” all other label values.\(^8\) This is the same form as the natural number CTSS, where the labeling function $L$ returns $\max_{j \in \{1..n\}}\text{number}_j + 1, i$. However, the interpretation of being “greater than” is not as straightforward as in the natural-number case.

The definition of the labeling function $L(\ell, i)$ presented below is based on a recursively defined function $L^k(G, \ell, \ell_{\max})$, which, given a $T^k$ subgraph $G$ of $T^n$, a set of labels $\ell$, 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.

```plaintext
procedure labeling(val);
begin
  ℓ := collect;
  vi := (val, L(ℓ, 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 L(ℓ, i) once it is collected (similarly for the variable ℓ_{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 L(ℓ, i). The following functions are used in defining 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 $\ell_{\text{max}}[n..k] \neq G$  
then return $\mathcal{L}^{k-1}(G.1)$;  
3: if $\ell_{\text{max}}[n..k-1] = G.2$  
then return $\mathcal{L}^{k-1}(G.3)$;  
4: if $k > 2$ then  
if $\ell_{\text{max}}[k-2] \in \{2,3,4,5\}$ and  
$(\ell_i[n..k-1] \neq \ell_{\text{max}}[n..k-1])$  
then return $\mathcal{L}^{k-1}(G.\text{dom}(\ell_{\text{max}}[k-1]))$;  
5: if (num labels($\ell_{\text{max}}[n..k-1]) < k-1$) or  
((num labels($\ell_{\text{max}}[n..k-1]) = k-1$) and  
$(\ell_i[n..k-1] = \ell_{\text{max}}[n..k-1])$)  
then return $\mathcal{L}^{k-1}(G.\ell_{\text{max}}[k-1])$  
else return $\mathcal{L}^{k-1}(G.\ell_{\text{max}}[k-1]))$;  
end $\mathcal{L}^k$;  
begin  
$\ell_{\text{max}} := \text{max(}\text{dominating set}(\ell,\ell_i))$;  
return $\mathcal{L}^n(T^n)$;  
end $\mathcal{L}$;  

**Fig. 6.** The labeling function.

$\text{dom}(x)$—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, $\text{dom}(1) = 2$, $\text{dom}(2) = 3$, $\text{dom}(3) = 4$, $\text{dom}(4) = 5$, and $\text{dom}(5) = 3$;  

$\text{dominating set}(\hat{\ell},\ell_i)$—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 $\{\ell_j \in \hat{\ell} | \ell_i \preceq \ell_j\} \cup \{\ell_i\}$; and  

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

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

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 $\Rightarrow$, 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
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 \( \ell^{[b]}_y = 134, \ell^{[a]}_x = 135, \) and \( \ell^{[c]}_z = 141 \), that is, all labels are totally ordered by \( \preceq \). In the figure, a label such as \( \ell^{[a]}_x = 135 \) 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 \( L^{[b+1]}_y \), reading \( \ell^{[a]}_x, \ell^{[b]}_y, \) and \( \ell^{[c]}_z \) and moving based on \( L(\ell, y) \) to \( \ell^{[b+1]}_y = 142 \). Process \( z \) performs \( L^{[c+1]}_z \), reading the new label \( \ell^{[b+1]}_y \). 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 \( L \). The move to a dominating node is intended to meet R1.

Processes \( y \) and \( z \) can continue forever to choose \( \ell^{[b+2]}_y = 144, \ell^{[c+2]}_z = 145, \ell^{[b+3]}_y = 143, \ldots \) (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 \( L^{[a+1]}_x \) 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 \( \text{num labels}(14') = 2 \) in \( L^x_{\ell+1} \), that is, there are already two labels in the \( T^2 \) subgraph, by line 5 of \( L(\ell, i) \), \( x \) will move to \( \ell^x_{\ell+1} = 151 \), and so on. \( \square \)

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 \( \text{read}(L_z^c) \)), it is the case that

\[
(\forall \ell^y_{[b]} \in \text{read}(L_z^c)) (\ell^y_{[b]} \lt \ell^c_{[c]}).
\]

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 \( \ell^x_{[b]} \), \( \ell^y_{[b]} \), and \( \ell_x^{c} \) and computing \( L \) 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 \( \text{num labels}(14') > 2 \), 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 \( \ell_x^{a+1} \), 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 \( \text{num labels}(14') > 2 \), by line 5 of \( L(\ell, i) \), \( x \) will move to \( \ell_x^{a+2} = 151 \), not entering the cycle. \( \square \)

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 \( L \), 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.
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 $\ell_z^{[a]} = 135$. 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 $\ell_z^{[a+1]}$ 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.

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
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 property that the chosen new label is greater than all those read, it is the case that \( \ell_z \) 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 \( L_x^{[a]} \triangleright L_y^{[b]} \)) if \( \ell_x^{[a]} \in \text{read}(L_y^{[b]}) \) or there exists an \( L_z^{[c]} \) such that \( \ell_z^{[c]} \in \text{read}(L_y^{[b]}) \) and \( L_x^{[a]} \triangleright L_z^{[c]} \).

**Definition 4.2.** Let the maximal observed set \( \text{maxobs}(L_x^{[a]}) \) be defined as

\[
\{ L_y^{[b]} \mid y \in \{1..n\}, y \neq x, L_y^{[b]} \triangleright L_x^{[a]} \text{ and } \\
(\forall L_y^{[b']}) (\text{if } L_y^{[b]} \rightarrow L_y^{[b']}, \text{then } L_y^{[b']} \triangleright L_x^{[a]}) \},
\]

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

\[(3') (\forall \ell_y^{[b]} \in \text{maxobs}(L_x^{[a]})) (\ell_y^{[b]} \triangleright \ell_x^{[a]}).\]

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

Finally, the following is the irreflexive total order \( \Rightarrow \) on the labeling operation executions as required by property P1.

![Fig. 9. The observed relation.](image-url)
Definition 4.3. Given any two distinct labeling operation executions $L_x^a$ and $L_y^b$, $L_x^a \rightarrow L_y^b$ if either
1. $L_x^a \rightarrow_L L_y^b$ or
2. $L_x^a \rightarrow_L L_y^b$, $L_y^b \rightarrow_L L_x^a$, and $\ell_x^a \prec_L \ell_y^b$.

Since with every $L_x^a$ there is an associated label $\ell_x^a$, $\rightarrow_L$ can be seen as a “lexicographical” order on pairs $(L_x^a, \ell_x^a)$. The first element in the pair is ordered by $\rightarrow_L$, a partial order that is consistent with the ordering $\rightarrow$. (If $L_x^a \rightarrow_L L_y^b$, then in $L_y^b$, $y$ read $\ell_x^a$ or a later label.) The second element is ordered by $\prec_L$, an irreflexive and antisymmetric relation. Parts of the rather involved reasoning as to why the “static” relation $\prec_L$ on the labels completes the “dynamic” partial order $\rightarrow_L$ 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 $\rightarrow_L$ 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

$$L_x^a \rightarrow_L L_y^b \rightarrow L_x^c \rightarrow L_x^a$$

while keeping in mind that $\rightarrow_L$ 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 $\rightarrow$ 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 $\text{label}_1 = 2$ and $\text{label}_2 = 1$, 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 $\Rightarrow$ among the associated labels of labeled-values in $\bar{\ell}$. Nevertheless, the requirement of property P1b is that $\prec$ be consistent with $\Rightarrow$ 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 $\ell$ 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 $\{\ell^{c,m,k}_1, \ldots, \ell^{c,m,k}_n\}$ collected in the $c$th collect operation execution of the $m$th level of the $k$th 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 $-1 \leq ||r1|| - ||r2|| \leq 1$.

The scan algorithm, presented in Figure 11, returns the indexed set of labeled-values $\ell$, 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 $i$th position representing the $i$th 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., $\Rightarrow$) 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 $k$th 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 $k$th largest, it suffices that the following condition holds (slightly abusing notation in the definition).

**Condition 1.** For the label $\ell^{8,\lceil \log n \rceil,k}_s$, collected in the $\lceil \log n \rceil$th level of the $k$th phase, and any label $\ell^{y,1,k}_{s'}$ of a process $y \in R$, collected in the first level of the $k$th phase, it is the case that $\ell^{8,1,k}_s \Rightarrow \ell^{y,\lceil \log n \rceil,k}_{s'}$.

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 \textit{scan};
  function \textit{select}(m,k,r);
  begin
    if $|r| = 1$ then return $(x : x \in r)$;
    else
      $x := \text{select}(m-1,k, \text{half}(r))$;
      $y := \text{select}(m-1,k, \text{other half}(r))$;
      if $(\exists c1, c2 \in \{1..8\} \ (c1 < c2) \land (\ell_{x^{c1,m,k}} \preceq \ell_{y^{c2,m,k}}))$ then return $y$
        else return $x$ fi fi:
  end \textit{select};
  begin
    $R := \{1..n\}$;
    $\bar{\ell} := \emptyset$;
    for $k := 1$ to $n$ do
      for $m := 1$ to $\lceil \log n \rceil$ do
        for $c := 1$ to $8$ do
          $\ell^{c,m,k} := \text{collect}$
        od od od;
    for $k := n$ downto $1$ do
      $s := \text{select}(\lceil \log n \rceil, k, R)$;
      $\bar{\ell} := \bar{\ell} \cup \{\text{value}(\ell^{s,\lceil \log n \rceil,k})\}$;
      $O[s] := k$;
      $R := R \setminus \{s\}$;
    od;
    return $(\bar{\ell}, O)$;
  end \textit{scan};

**Fig. 11.** The \textit{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, \(L_{y^{8,\lceil \log n \rceil,k}} \Longrightarrow L_{x^{8,\lceil \log n \rceil,k}}\). Since the read of \(\ell_{y^{8,\lceil \log n \rceil,k}}\) 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 \textit{select} function, meets Condition 1. The \textit{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 \textit{select} function returns the id of the “winner,” a process \(s\) that meets Condition 1. At any level \(m\) of the application of \(\text{select}(m, k, r)\), 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 \([\log ||R||]\) levels, \(s\), the winner of all selections, is returned.

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

Condition 2. Of the two processes \(x\) and \(y\) in the application of \(\text{select}\) at level \(m\) of phase \(k\), the one returned, say \(x\), is such that \(L^1_{y, m, k} \implies L^8_{x, m, k}\), where \(L^1_{x, m, k}\) and \(L^8_{x, m, k}\) 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 \(x\) and \(z\), by the same line of proof as above, from \(L^1_{y, m, k} \implies L^8_{x, m, k}\) and \(L^1_{z, m-1, k} \implies L^8_{y, m-1, k}\), it follows that \(L^1_{x, m, k} \implies L^8_{x, m, k}\). 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 \(\implies\) 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 \(\implies\) can be determined. Thus to allow the \(\text{select}\) operation at level \(m\) of phase \(k\) to choose a “winner” process, say \(x\), for which \(L^1_{y, m, k} \implies L^8_{x, m, k}\), 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

\[
(\exists c_1, c_2 \in \{1..8\}) \ (c_1 < c_2) \land (L^{c_1, m, k} \not\subset L^{c_2, m, k}),
\]

then \(L^{c_1, m, k} \implies L^{c_2, m, k}\). (Because of the order of label collecting, this will imply \(L^{c_1, m, k} \implies L^{c_2, m, k}\).) Assume by way of contradiction that \(L^{c_1, m, k} \implies L^{c_2, m, k}\). Since \(L^{c_1, m, k} \not\subset L^{c_2, m, k}\), it must be by the definition of \(\implies\) that \(L^{c_2, m, k} \not\subset L^{c_1, m, k}\). It cannot be that \(L^{c_2, m, k} \in \max \text{obs}(L^{c_1, m, k})\) since by the properties of the labeling scheme, for the label \(L^{c_2, m, k} \in \max \text{obs}(L^{c_1, m, k})\), \(L^{c_2, m, k} \not\subset L^{c_1, m, k}\). Thus there must be a different labeling operation execution \(L^{c_2, m, k} \in \max \text{obs}(L^{c_1, m, k})\), \(L^{c_2, m, k} \not\subset L^{c_1, m, k}\). This label \(L^{c_2, m, k}\) was already observed (i.e., must have been written) before the end of the read of \(L^{c_1, m, k}\). Thus \(L^{c_2, m, k}\) or a label later than it must have been read instead of \(L^{c_2, m, k}\) in the collect \(c_2\) 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 \(L^{c_1, m, k} \implies L^{c_2, m, k}\) 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, \((\exists c_1, c_2 \in \{1..8\}) \ (c_1 < c_2) \land (L^{c_1, m, k} \not\subset L^{c_2, m, k})\). Therefore, it must be that there are four nonconsecutive collects of \(L^{c_1, m, k}\), \(c_1 \in \{1, 3, 5, 7\}\), and four nonconsecutive collects of \(L^{c_2, m, k}\), \(c_2 \in \{2, 4, 6, 8\}\), such that the labels \(L^{c_1, m, k}\), \(c_1 \in \{1, 3, 5, 7\}\), are all different from one another and the labels \(L^{c_2, m, k}\), \(c_2 \in \{2, 4, 6, 8\}\), are all different from one another. The reason is that if any two of them, say \(L^{c_1, m, k}\) and \(L^{c_2, m, k}\), are the same, then in order for Condition 3 not to hold for \(x\) \(c_1 = 4\) and \(c_2 = 3\), it must be that \(L^{c_1, m, k} \not\subset L^{c_2, m, k}\). However, since \(L^{c_2, m, k}\) and \(L^{c_2, m, k}\) are the same, it would follow that \(L^{c_1, m, k} \not\subset L^{c_2, m, k}\), and Condition 3 would hold for \(y\), a contradiction.

To complete the proof, it remains to be shown that if the labels \(L^{c_1, m, k}\), \(c_1 \in \{1, 3, 5, 7\}\), are all different from one another and the labels \(L^{c_2, m, k}\), \(c_2 \in \{2, 4, 6, 8\}\),
are all different from one another, then \( L^{1,m,k}_y \Rightarrow L^{8,m,k}_x \). 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 \( L^{8,m,k}_x \) occurred completely after the initial labeling of \( y \), that is, after \( L^{1,m,k}_y \Rightarrow L^{8,m,k}_x \) (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 \( B \), “\( A \rightarrow B \)” stands for “\( A \) precedes \( B \)” and “\( A \rightarrow B \)” stands for “\( A \) can causally affect \( B \)”.

A system execution is a triple \((\varphi, \rightarrow, \cdots \cdots)\), where \( \varphi \) is a set of operation executions and \( \rightarrow \) and \( \cdots \cdots \) are binary relations over \( \varphi \). Lamport offers the following axioms:

A1. \( \rightarrow \) is an irreflexive transitive relation.
A2. If \( A \rightarrow B \), then \( A \cdots B \) and \( B \cdots /\ A \).
A3. If \((A \rightarrow B \) and \( B \cdots C) \) or \((A \cdots B \) and \( B \rightarrow C) \), then \( A \cdots C \).
A4. If \( A \rightarrow B \cdots C \rightarrow D \), then \( A \rightarrow D \).
A5. For any \( A \), the set of \( B \) such that \( A \cdots B \) 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 \( \varphi \) be a collection of nonempty subsets of \( \mathcal{E} \). For \( A \) and \( B \) in \( \varphi \), define \( A \rightarrow B \) if and only if \((\forall a \in A) (\forall b \in B) (a < b)\) (in the sense of \( \mathcal{E} \)) and \( A \rightarrow B \) 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 \( A \rightarrow B \rightarrow C \cdots D \), then \( A \cdots D \).

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 \( A \rightarrow B \) or \( B \rightarrow A \) 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 S implements a system H if there is a mapping \( m : S \mapsto H \) such that for every system execution \( \langle \phi, \rightarrow, \cdots \rangle \) in S, \( \langle \phi, \rightarrow, \cdots \rangle \) implements \( m(\langle \phi, \rightarrow, \cdots \rangle) \).

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 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, S contains all system executions \( \langle \phi, \rightarrow, \cdots \rangle \) such that we have the following:

1. \( \phi \) 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 \( k \)th 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 \( \rightarrow \).
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, \( r_x^{[k-1]}(z) \rightarrow r_y^{[k]}(z) \), and if for some \( w_x^{[k']} (z) \), \( r_x^{[k']} (z) \rightarrow w_x^{[k']} (z) \), then \( r_y^{[k']} (z) \rightarrow w_x^{[k']} (z) \).
   b) For any two writes \( w_x^{[k]} (z) \) and \( w_x^{[k+1]} (z) \), there exists a set of read operation executions
      \[ \mathcal{R}_{k+1} = \{ r_x^{[a]} (z) | w_x^{[k]} (z) \rightarrow r_x^{[a]} (z) \rightarrow w_x^{[k+1]} (z), \alpha \in \{0, 1, \ldots\} \}, \]
      of reads of \( v_x \) such that \( ||\mathcal{R}_{k+1}|| \mod (8n \log n) = 1 \).
   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 H, a concurrent time-stamp system.

**Definition 5.1.** A concurrent time-stamp system is a set of system executions \( \langle \psi, \rightarrow, \cdots \rangle \) 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 \( \psi_i \), 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 \( L_i^{[k]} \) 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 \( L_i^{[k]} \) may repeatedly appear, in order that a unique labeled-value be associated with each \( L_i^{[k]} \), let \( L_i^{[k]} \) be the triplet \( (\ell_i^{[k]}, i, k) \), 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 \( \ell = \{ \ell_i^{[k]}_1, \ldots, \ell_i^{[k]}_n \} \) is returned by each scan, with different labeled-values associated with labeling operations of different processes.

- An initial labeling operation execution \( \ell_i^{[0]} \) with labeled-value \( \ell_i^{[0]} \). The initial labeling \( \ell_i^{[0]} \rightarrow S_j^{[k]} \) 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 precedes any labeling operation of \( i \).) All operation executions in \( \psi_i \) are totally ordered by \( \rightarrow \), 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 \( S \) to \( H \) must be defined. In the definition of the labeling and scan procedures, for each system execution \( \langle \varphi, \rightarrow, \ldots \rangle \) of \( S \), the set of operation executions \( m(\varphi) \) of \( m(\langle \varphi, \rightarrow, \ldots \rangle) \) is the following higher-level view of \( \langle \varphi, \rightarrow, \ldots \rangle \):

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 \( k' = \max \{ \alpha | r_j^{[\alpha]}(i) \rightarrow w_i^{[k]}(i) \} \).

2. Each scan operation execution by process \( i \) is a set of reads

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

all in \( \varphi \) 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 \( \varphi \) and each element of \( \varphi \) belongs to a finite, nonzero number of elements of \( m(\varphi) \). It is thus a higher-level view of \( \varphi \). (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 \( \preceq^{\cdot \cdot} \) and \( \preceq^{\cdot \cdot} \) must be defined so that \( m(\langle \varphi, \rightarrow, \ldots \rangle) \) is defined as \( m(\varphi), \preceq^{\cdot \cdot}, \preceq^{\cdot \cdot} \).

By choosing \( \preceq^{\cdot \cdot} \) and \( \preceq^{\cdot \cdot} \) to be the induced relations \( \rightarrow^{\cdot \cdot} \) and \( \rightarrow^{\cdot \cdot} \) as defined by equation 2 of [Lam86a] (by equation 2, choosing the induced precedence relations \( \rightarrow^{\cdot \cdot} \) and \( \rightarrow^{\cdot \cdot} \) for \( \preceq^{\cdot \cdot} \) and \( \preceq^{\cdot \cdot} \) 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 \( m(\varphi), \preceq^{\cdot \cdot}, \preceq^{\cdot \cdot} \) is indeed a system execution. Since condition H3 of Definition 5 of [Lam86a] is satisfied by the induced precedence
relations.\textsuperscript{14} \langle \varphi, \rightarrow^*, \rightarrow^* \rangle implements \langle m(\varphi), \rightarrow^*, \rightarrow^* \rangle.  

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

### 5.5. Properties P0 and P2–P4

The proof that \langle m(\varphi), \rightarrow^*, \rightarrow^* \rangle meets property P0 follows by applying equation 2 of [Lam86a] to \langle \varphi, \rightarrow^*, \rightarrow^* \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 \( \ell \) and \( \rightarrow \) 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 \( \langle \varphi, \rightarrow^*, \rightarrow^* \rangle \) yields the proof that \( \langle m(\varphi), \rightarrow^*, \rightarrow^* \rangle \) 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 \( \langle m(\varphi), \rightarrow^*, \rightarrow^* \rangle \) 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 \( \equiv^{obs} \) (Definition 4.1).

The following lemma establishes the properties of \( \equiv^{obs} \), later used to establish the properties of \( \Longrightarrow \).

**Lemma 5.1.** The relation \( \equiv^{obs} \) 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 \( L_i^{[a]} \rightarrow L_j^{[b]} \), then \( L_i^{[a]} \equiv^{obs} L_j^{[b]} \).

**Proof.** Since \( r_j(L_i^{[a]}) \rightarrow w_i(L_i^{[a]}) \) for any \( j \), it follows that \( \equiv^{obs} \) 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 \( L_i^{[a]} \equiv^{obs} L_j^{[b]} \equiv^{obs} L_k^{[c]} \), then \( L_i^{[a]} \equiv^{obs} L_k^{[c]} \).

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

\textsuperscript{14} Definition 5 of [Lam86a] states that a lower-level system execution \( \langle \varphi, \rightarrow^*, \rightarrow^* \rangle \) implements a higher-level one \( \langle \psi, \rightarrow^H, \rightarrow^H \rangle \) if \( \psi \) is a higher-level view of \( \varphi \) and condition H3 holds, that is, for any \( G, H \in \psi \), if \( G \rightarrow^H H \), then \( G \rightarrow^H H \).
This completes the proof of Lemma 5.1. Thus by Corollary 5.1, for some \( \alpha \) selected, it is greater (by the ordering \( \preceq \)) which by axiom A4 implies \( r_y \) any process \( x \) of the observation sequence, as in the previous claim. If \( ||L_{i}[\alpha] \obr L_{j}[\beta]|| = 1 \), then by definition \( \alpha = j \) and \( \beta = b \). Assume this induction hypothesis holds for every \( r' < r \). By the definition of the production sequence, there must exist an \( L_{\alpha}^{b} \), \( r_{i}(L_{\alpha}) = \ell_{i}[\alpha] \) such that \( ||L_{\alpha}^{b} \obr L_{j}[\beta]|| = r - 1 \). By the induction hypothesis, there exist \( \alpha' \) and \( \beta' \) such that

\[
w(L_{\alpha}^{b}) \rightarrow r_{i}(L_{\alpha}^{b}) \rightarrow w(L_{j}[\beta]),
\]

where possibly \( \alpha' = j \) and \( \beta' = b \). By the definition of the labeling operation, \( r_{i}(L_{\alpha}^{b}) \rightarrow w(L_{\alpha}^{b}), \) and so

\[
r_{i}(L_{\alpha}^{b}) \rightarrow w(L_{\alpha}^{b}) \rightarrow r_{i}(L_{\alpha}^{b}) \rightarrow w(L_{j}[\beta]),
\]

which by axiom A4 implies \( r_{i}(L_{\alpha}^{b}) \rightarrow w(L_{j}[\beta]) \). Since \( r_{i}(L_{\alpha}^{b}) = \ell_{i}[\alpha] \), it follows by atomic register axiom B5 that \( w(L_{i}[\alpha]) \rightarrow r_{i}(L_{\alpha}^{b}) \rightarrow w(L_{j}[\beta]). \)

Claim 5.1.2 (antisymmetric). For any two distinct labeling operation executions \( L_{i}[\alpha] \) and \( L_{j}[\beta] \), if \( L_{i}[\alpha] \obr L_{j}[\beta] \), then \( L_{j}[\beta] \obr L_{i}[\alpha] \).

Proof. Assume by way of contradiction that \( L_{i}[\alpha] \obr L_{j}[\beta] \) and \( L_{j}[\beta] \obr L_{i}[\alpha] \). Thus by Corollary 5.1, for some \( \alpha, \beta, \gamma, \) and \( \delta \) (possibly \( \alpha = j, \beta = b, \gamma = i, \) or \( \delta = a \)),

\[
w(L_{i}[\alpha]) \rightarrow r_{i}(L_{\alpha}^{b}) \rightarrow w(L_{j}[\beta]) \text{ and } w(L_{j}[\beta]) \rightarrow r_{j}(L_{\beta}^{i}) \rightarrow w(L_{i}[\alpha]).
\]

Since this implies

\[
w(L_{i}[\alpha]) \rightarrow r_{i}(L_{\alpha}^{b}) \rightarrow w(L_{j}[\beta]) \rightarrow r_{j}(L_{\beta}^{i}),
\]

by axiom A4*, \( w(L_{i}[\alpha]) \rightarrow r_{i}(L_{\alpha}^{b}) \). By \( w(L_{i}[\alpha]) \rightarrow r_{j}(L_{\beta}^{i}) \) and \( r_{j}(L_{\beta}^{i}) \rightarrow w(L_{i}[\alpha]), \) a contradiction to the axiom of global time is derived.

Claim 5.1.3 (consistent). If \( L_{x}[\alpha] \obr L_{y}[\beta] \), then \( L_{x}[\alpha] \obr L_{y}[\beta] \).

Proof. If \( x = y \), then \( r_{x}(L_{x}[\alpha]) = \ell_{x}[\alpha] \), and by induction, for \( b > a, L_{x}[\alpha] \obr L_{y}[\beta] \). If \( x \neq y \), then by register axioms B0–B4, since \( w(L_{x}[\alpha]) \rightarrow r_{x}(L_{y}[\beta]), \) either \( r_{x}(L_{y}[\beta]) = \ell_{x}[\alpha] \) (implying \( L_{x}[\alpha] \obr L_{y}[\beta] \)) or there exists an \( L_{x}[\alpha], L_{x}[\alpha] \obr L_{x}[\alpha], \) where \( r_{x}(L_{y}[\beta]) = \ell_{x}[\alpha] \), which by the transitivity of \( \obr \) (Claim 5.1.1) implies \( L_{x}[\alpha] \obr L_{y}[\beta] \). This completes the proof of Lemma 5.1.

The following lemma formalizes the property that whenever a new label \( \ell_{x}[\alpha] \) is selected, it is greater (by the ordering \( \preceq \)) then the latest label observed in \( L_{x}[\alpha] \) for any process \( y \neq x \).
Lemma 5.2. For any labeling operation execution \( L^a_i \), it is the case that
\[
(\forall \ell^b_j \in \max \text{obs}(L^a_i)) (\ell^b_j \preceq \ell^a_i).
\]

To simplify the exposition, the the proof is deferred 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 \( \implies \) meets part a of property P1.

Lemma 5.3. The relation \( \implies \) is an irreflexive total order on the labeling operation executions in \( m(\varphi) \) such that for any two labeling operations \( L^a_i \) and \( L^b_j \), if \( L^a_i \implies L^b_j \), then \( L^a_i \neq L^b_j \).

Proof. By Definition 4.3. the relation \( \implies \) is irreflexive and total, and is consistent with the ordering \( \preceq \) 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 \( L^a_i \) and \( L^b_j \), if \( L^a_i \implies L^b_j \), then \( L^a_i \neq L^b_j \).

Proof. Assume by way of contradiction that for two distinct labeling operation executions, \( L^a_i \implies L^b_j \), and \( L^a_i = L^b_j \). Since \( \implies \) is antisymmetric, it is not the case that both \( L^a_i \implies L^b_j \) and \( L^b_j \implies L^a_i \) hold. Thus if \( L^a_i \implies L^b_j \), \( L^b_j \implies L^a_i \) even if \( \ell^b_j \preceq \ell^a_i \), a contradiction. Thus it must be the case that \( \ell^b_j \not\preceq \ell^a_i \) and \( \ell^a_i \not\preceq \ell^b_j \), which contradicts the definition of the ordering \( \preceq \) of the labels. \( \square \)

Claim 5.3.2 (transitive). For any three labeling operation executions \( L^a_i, L^b_j, \) and \( L^c_k \), if \( L^a_i \implies L^b_j \), and \( L^b_j \implies L^c_k \), implies \( L^a_i \implies L^c_k \).

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 \( \langle m(\varphi), \preceq, \preceq, \preceq \rangle \) meets part b of property P1, that is, for any scan operation execution \( \bar{S}_k^i \) that returns \( (\bar{\ell}, \prec) \), where
\[
\text{value}(\ell^a_i), \text{value}(\ell^b_j) \in \bar{\ell},
\]

it is the case that \( x \prec y \) if and only if \( L^a_x \implies L^b_y \). Since both \( \prec \) and \( \implies \) 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 \( \bar{S}_k^i \) that returns \( (\bar{\ell}, \prec) \), if \( \text{value}(\ell^a_i) \) was returned in phase \( k' \) and value \( \text{value}(\ell^b_j) \) was returned in phase \( k \), where \( k' < k \), then \( L^a_x \implies L^b_y \). 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 \( \text{value}(\ell^a_i) \), \( \text{value}(\ell^b_j) \in \bar{\ell} \), it is the case that \( L^a_i \implies L^b_j \).

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 nth” 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 \( \Rightarrow \) is known. Based on Claim 5.4.1 and the transitivity of \( \Rightarrow \), 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(\([\log n]\), \( k \), \( R \)), then \( L^1_1.k \Rightarrow L^s_{\{\log n\}k} \) 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 \( \Rightarrow \) 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 \( u \)th collect operation execution performed during a given scan. A label associated with \( L^m_x[a] \), read in any \( C_w \), will be denoted by \( L^m_x[a] \) or \( L^m_x [a] \), and the labeling operation execution \( L^m_x[a] \) itself will be similarly denoted by \( L^m_w[a] \) or \( L^m_x [a] \).

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 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 \( \Rightarrow \) is known. Based on Claim 5.4.1 and the transitivity of \( \Rightarrow \), 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(\([\log n]\), \( k \), \( R \)), then \( L^1_1.k \Rightarrow L^s_{\{\log n\}k} \) 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 \( \Rightarrow \) among the labels returned is the order of the phases.

Claim 5.4.1. If \( C_w \leftarrow C_{w+1} \leftarrow C_{w+2} \) (or \( C_{w+1} \leftarrow C_{w+2} \leftarrow C_{w+3} \)) and if for some \( L^m_1[a] \) and \( L^m_w[b] \), \( L^m_x[b] \Rightarrow L^m_y[y] \), then for any \( L^m_x[a] \) (similarly, \( L^m_y[d] \)), it is the case that \( L^m_x[a] \Rightarrow L^m_y[y] \) (similarly, \( L^m_x[a] \Rightarrow L^m_y[d] \)).

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

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

\[
(\exists c_1, c_2 \in \{1..8\}) \ (c_1 < c_2) \land (L^c_1 m, k \Leftrightarrow L^c_2 m, k),
\]

holds, then \( L^1_{1,m,k} \Rightarrow L^8_{8,m,k} \) and otherwise \( L^1_{1,m,k} \Rightarrow L^8_{8,m,k} \).

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, \( L_x^{1,m,k} \implies L_y^{8,m,k} \), and if one of the other two holds, then \( L_x^{1,m,k} \implies L_y^{8,m,k} \). More formally, we have the following.

**Claim 5.4.3.** For the 16 labels \( \ell_x^{1,m,k} \) and \( \ell_y^{2,m,k} \), \( c_1, c_2 \in \{1..8\} \), collected in level \( m \) of phase \( k \) of a scan operation execution, one of the following three conditions must hold:

1. \( (\exists c_1, c_2 \in \{1..8\}) \ (c_1 < c_2) \land (\ell_x^{c_1,m,k} \preceq \ell_y^{c_2,m,k}) \).
2. \( (\exists c_1, c_2 \in \{1..8\}) \ (c_1 < c_2) \land (\ell_x^{c_1,m,k} \preceq \ell_y^{c_2,m,k}) \).
3. The four labels \( \ell_x^{2,m,k}, \ell_x^{4,m,k}, \ell_x^{6,m,k}, \) and \( \ell_y^{8,m,k} \) differ from one another according to \( \prec \), and the four labels \( \ell_x^{1,m,k}, \ell_x^{3,m,k}, \ell_x^{5,m,k}, \) 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

\[
(\exists c_1, c_2 \in \{1..8\}) \ (c_1 + 1 < c_2) \land (\ell_x^{c_1,m,k} = \ell_x^{c_2,m,k})
\]

or

\[
(\exists c_1, c_2 \in \{1..8\}) \ (c_1 + 1 < c_2) \land (\ell_y^{c_1,m,k} = \ell_y^{c_2,m,k})
\]

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 \( \preceq \). Without loss of generality, assume that the first condition holds. Then by definition, there must exist a label \( \ell_y^{c,m,k} \), \( c_1 < c < c_2 \). If \( \ell_x^{c_1,m,k} \preceq \ell_y^{c,m,k} \), then condition 1 holds and the claim is proven. Thus it must be the case that \( \ell_y^{c,m,k} \preceq \ell_x^{c_1,m,k} \). However, since \( \ell_x^{c_1,m,k} = \ell_x^{c_2,m,k} \), it is the case that \( \ell_x^{c,m,k} \preceq \ell_x^{c_2,m,k} \), 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 \( L_x^{1,m,k} \preceq L_y^{8,m,k} \), and similarly, if condition 2 holds, then \( L_y^{1,m,k} \preceq L_x^{8,m,k} \). (This follows by exchanging the roles of \( x \) and \( y \) in Claim 5.4.4 below.)

**Claim 5.4.4.** If \( \ell_x^{1,m,k} \preceq \ell_y^{2,m,k} \), then \( L_x^{1,m,k} \implies L_y^{2,m,k} \).

**Proof.** For simplicity, let \( (c_1, m, k) = w \) (the label \( \ell_x^{c_1,m,k} \) read is \( \ell_y^{[a]} \), that is, of labeling operation execution \( L_y^{[a]} \)) and \( (c_2, m, k) = w + 1 \) (similarly, \( \ell_y^{c_2,m,k} \) is \( \ell_y^{[b]} \)). The outline of the proof appears in Figure 12. Assume by way of contradiction that \( \ell_x^{[a]} \preceq \ell_y^{[b]} \) and \( L_y \implies L_x^{[a]} \). By Definition 4.3, it must be that \( L_y^{[b]} \implies L_x^{[a]} \). By
Lemma 5.2, it cannot be that \( L_y^{[b]} \in \text{max obs}(L_x^{[a]}) \). Thus there must exist an \( L_y^{[b']} \), \( b < b' \), such that \( L_y^{[b']} \in \text{max obs}(L_x^{[a]}) \). By Corollary 5.1, since \( L_y^{[b']} \not\equiv L_x^{[a]} \), there must exist some \( r_y(L_{\alpha}^{[\beta]}) = \ell_y^{[b']} \) such that

\[
\text{where possibly } \alpha = x \text{ and } \beta = a. \quad \text{Since } r_y(C_{w+1}) = \ell_y^{[b]} \text{ and } r_y(L_{\alpha}^{[\beta]}) = \ell_y^{[b']}, \quad b < b', \text{ by atomic register axiom B5, it must be that } r_y(C_{w+1}) \not\equiv r_y(L_{\alpha}^{[\beta]}). \text{ Similarly, since } \ell_x^{[a]} \text{ was read in } C_w, \text{ it must be by axiom B5 that } w(L_x^{[\alpha]}) \not\equiv r_x(C_w). \text{ Thus }
\]

\[
\text{which by axiom A4 of } [AB87] \text{ implies that } r_y(C_{w+1}) \not\equiv r_x(C_w), \text{ a contradiction to } C_w \not\equiv C_{w+1}. \quad \square
\]

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 \( L_y^{1,m,k} \Rightarrow L_x^{8,m,k} \). 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 \( \Rightarrow \). 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 \( \ell_x^{2,m,k}, \ell_x^{4,m,k}, \ell_x^{6,m,k}, \text{ and } \ell_x^{8,m,k} \) differ from one another according to \( \prec \) and the four labels \( \ell_y^{1,m,k}, \ell_y^{3,m,k}, \ell_y^{5,m,k}, \text{ and } \ell_y^{7,m,k} \) also differ from one another according to \( \prec \), then \( L_y^{1,m,k} \Rightarrow L_x^{8,m,k} \).

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

\[
L_y^{1,m,k} \Rightarrow w(L_y^{3,m,k}) \Rightarrow r_y(C_{3,m,k}) \Rightarrow r_x(C_{4,m,k}) \Rightarrow w(L_x^{6,m,k}) \Rightarrow L_x^{8,m,k}.
\]

By applying axiom A4 twice, it follows that \( L_y^{1,m,k} \Rightarrow L_x^{8,m,k} \), which by Definition 4.3 implies that \( L_y^{1,m,k} \Rightarrow L_x^{8,m,k} \), \( \square \)

This completes the proof of Claim 5.4.2. \( \square \)

The following claim proves the correctness of the recursive procedure \( \text{select} \).

Claim 5.4.6. For any \( k \) and \( R \), if \( s \) is returned by \( \text{select}([\log n], k, R) \), then \( L_i^{1,1,k} \Rightarrow L_i^{8,[\log n],k} \) for all \( i \in R - \{s\} \).
Proof. First, observe that for \(||r||\), the size of the input set of \(select(m, k, r)\), it follows by simple induction (given that initially \(||r||| \leq n\) that \(m \geq \lfloor \log ||r|| \rfloor\). The proof of the claim will thus be by induction on \(||r||| \in \{1..[R]\}\).

For \(||r||| = 1\), the claim follows vacuously. For \(||r||| = 2\), since \(m \geq \lfloor \log ||r|| \rfloor = 1\), 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 \(||\text{half}(r)||\), \(||\text{other half}(r)||\) \leq t/2, by the induction hypothesis applied to \(select(\lfloor |t| - 1 \rfloor, k, \text{half}(r))\) and \(select(\lfloor |t| - 1 \rfloor, k, \text{other half}(r))\), it follows that

\[
(\forall i \in \text{half}(r)) \ (L_i^{1,1,k} \Rightarrow L_i^{8,\lfloor \log t\rfloor \!-\! 1,k}) \quad \text{and} \quad (\forall i \in \text{other half}(r)) \ (L_i^{1,1,k} \Rightarrow L_i^{8,\lfloor \log t\rfloor \!-\! 1,k}).
\]

By Claim 5.4.2, without loss of generality, it can be assumed that \(L_y^{1,\lfloor \log t\rfloor \!-\! 1,k} \Rightarrow L_y^{8,\lfloor \log t\rfloor \!-\! 1,k} \) in select(\(\lfloor |t| \rfloor, k, r\)). Thus since \(C_{8,\lfloor \log t\rfloor \!-\! 1,k} \Rightarrow C_{1,\lfloor \log t\rfloor,k}\), by Claim 5.4.1 and the above,

\[
L_i^{1,1,k} \Rightarrow L_y^{8,\lfloor \log t\rfloor \!-\! 1,k} \Rightarrow L_y^{1,\lfloor \log t\rfloor \!-\! 1,k} \Rightarrow L_x^{8,\lfloor \log t\rfloor \!-\! 1,k}
\]

for every \(i \in \text{other half}(r)\). By transitivity (Claim 5.3.2), it is the case that \(L_i^{1,1,k} \Rightarrow L_x^{8,\lfloor \log t\rfloor \!-\! 1,k} \) for every \(i \in \text{other half}(r)\). Similarly, by Claim 5.4.1 and the above, given that \(C_{8,\lfloor \log t\rfloor \!-\! 1,k} \Rightarrow C_{1,\lfloor \log t\rfloor,k}\), it follows that

\[
L_i^{1,1,k} \Rightarrow L_x^{8,\lfloor \log t\rfloor \!-\! 1,k} \Rightarrow L_x^{8,\lfloor \log t\rfloor \!-\! 1,k}
\]

for every \(i \in \text{half}(r)\). Again by transitivity, it is the case that \(L_i^{1,1,k} \Rightarrow L_x^{8,\lfloor \log t\rfloor \!-\! 1,k}\) for every \(i \in \text{half}(r)\), 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 \(value(t_i^{8,\lfloor \log n\rfloor,O[i]}), value(t_j^{8,\lfloor \log n\rfloor,O[j]}) \in \bar{\ell}\), it is the case that \(L_i^{8,\lfloor \log n\rfloor,O[i]} \Rightarrow L_j^{8,\lfloor \log n\rfloor,O[j]}\). 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 \(value(t_\alpha^{8,\lfloor \log n\rfloor,k+1}) \in \bar{\ell}\) of the scan operation execution, that is, the returned labeled-value for process \(\alpha\). By Claim 5.4.6,

\[
L_i^{1,1,k+1} \Rightarrow L_\alpha^{8,\lfloor \log n\rfloor,k+1}
\]

for \(i \in R\). By Claim 5.4.1, since \(C_{8,\lfloor \log n\rfloor,k} \Rightarrow C_{1,1,k+1}\), it is the case that

\[
L_i^{8,\lfloor \log n\rfloor,k} \Rightarrow L_\alpha^{8,\lfloor \log n\rfloor,k+1}.
\]

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

\[
L_i^{8,\lfloor \log n\rfloor,k} \Rightarrow L_\alpha^{8,\lfloor \log n\rfloor,k+1} \Rightarrow L_\alpha^{8,\lfloor \log n\rfloor,O[j]}.
\]

By the transitivity of \(\Rightarrow\) (Claim 5.3.2), it then follows that \(L_i^{8,\lfloor \log n\rfloor,O[i]} \Rightarrow L_j^{8,\lfloor \log n\rfloor,O[j]}\). \(\square\)
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 $L$ 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 $\text{inside}(X)$. $X$ 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 $\text{inside}$ 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 $\text{min}$ is simply the earliest in a sequence of labeling operation executions. For example, $\text{min}(\text{inside}(X))$ is the first among the moves since the process performing $X$ entered $T^k$.

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

- Let $\ell_x^{[a]} \equiv \ell_x^{[k]}$ denote that $\ell_y^{[a]}[1..k-1] = \ell_x^{[a]}[1..k-1]$ for $k \geq 2$.
- Let $\ell_y^{[b]} \neq \ell_x^{[a]}$ (similarly, $\ell_y^{[b]} \preceq \ell_x^{[a]}$) denote that $\ell_y^{[b]}[1..k] = \ell_x^{[a]}[1..k]$ 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 $\text{inside}(\ell_x^{[a]}[1..k])$ be a set of operation executions
  \[
  \{L_x^{[a]} | \alpha = \{a \text{ or } L_x^{[a]} \prec L_x^{[t]} \text{ and } \ell_x^{[a]}[k] = \ell_x^{[t]}[k] \text{ and } \forall \ell_x^{[a]}' \text{ (if } L_x^{[a]} \prec L_x^{[t]} \text{ then } \ell_x^{[a]}'[k+1] = \ell_x^{[t]}'[k])\}\).

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

If $\ell_x^{[a]}[-1] \equiv \ell_x^{[a]}$, $k = 2$ (the same label by the same process), then let the convention be that $\ell_x^{[a]}[-1] \neq \ell_x^{[a]}$, 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 $\langle m(\varphi)^L, \prec, \prec, \prec \rangle$ 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 $\langle m(\varphi)^L, \prec, \prec, \prec \rangle$ 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

- $L_i^{[a]} \prec L_j^{[b]}$ or
- $L_i^{[a]} \prec L_j^{[b]}$ and for $\ell_i^{[a]-1} \prec \ell_j^{[a]}$ and $\ell_j^{[b]-1} \prec \ell_j^{[b]}$, 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 $\prec$ is a partial order, and if two labels are not related by $\prec$, 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 \(<\text{obs}\rangle\), the above order of induction is total.

5.9.3. The induction hypothesis. The induction hypothesis consists of \(I_1 \land I_2 \land I_3 \land I_4\), where \(I_1\)–\(I_4\) are as follows:

I1. For any \(L^{[b]}_x \in \max \text{obs}(L^{[a]}_x)\), it is the case that \(L^{[y]}_y \preceq L^{[a]}_x\).

I2. The relation \(\rightarrow\) is transitive.

I3. For any \(L^{[a]}_x \) and \(L^{[b]}_y\), where

- \(L^{[y]}_y[k-1], L^{[x]}_x[k-1] \in \{3, 4, 5\}\) and
- \(L^{[y]}_y[k] \neq L^{[x]}_x, k \geq 2\),

if there exist labeling operation executions \(L^{[y]}_y[k-1]\) and \(L^{[x]}_x[k-1]\), where

- \(L^{[y]}_y[k-1] \neq L^{[x]}_x[k]\) and
- \(L^{[y]}_y[k] \neq L^{[x]}_x[k]\),

then either \(L^{[a]}_x \preceq L^{[b]}_y\) or \(L^{[b]}_y \preceq L^{[a]}_x\).

I4. 1. If \(L^{[x]}_x[k-1] \in \{2, 3, 4, 5\}, k > 2\), then there are at least \(k-1\) labels \(L^{[y]}_y \in \max \text{obs}(L^{[x]}_x)\) such that \(L^{[y]}_y[k] = L^{[x]}_x[k] \geq 2\), and \(L^{[y]}_y[k-1] \in \{3, 4, 5\}\); and

2. if there exists an \(L^{[a]}_x \in \text{inside}(L^{[x]}_x, k)\), \(L^{[x]}_x[k-1] \in \{4, 5\}\) (possibly \(a = a1\)), then there are exactly \(k-1\) labels \(L^{[y]}_y \in \max \text{obs}(L^{[x]}_x)\) such that \(L^{[y]}_y[k] = L^{[x]}_x[k] \geq 2\), and \(L^{[y]}_y[k-1] \in \{3, 4, 5\}\); and

3. if \(L^{[x]}_x[k-1] \neq L^{[y]}_y[k-1]\), then for any \(L^{[y]}_y \in \max \text{obs}(L^{[x]}_x)\), where \(L^{[y]}_y[k] \neq L^{[x]}_x[k-1]\) and \(L^{[y]}_y[k] \in \{3, 4, 5\}\), it is the case that \(L^{[y]}_y[k-1] \neq L^{[x]}_x[k]\).

The induction hypothesis includes four main parts. \(I_1\) and \(I_2\) 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 \(I_3\) 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 \(k \geq 2\). 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 \(I_3\) of the inductive hypothesis.

Property \(I_4\) is a collection of three properties that were informally mentioned in section 4.3:

- \(I_4.1\) 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 \(I1 \land I2 \land I3 \land I4\) holds for \(m(\phi)^{L'} = \{L_1^{[0]}, \ldots, L_n^{[0]}\}\).

**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 \(\ell_y^{[0]}\), \(\ell_x^{[0]} \neq \ell_y^{[0]}\), where \(k1 = 1\), and \(L_x^{[0]} \not\succ L_y^{[0]}\). Since \(\leq^k\) is a total order for level \(k1 = 1\), it follows that \(\implies\) is transitive in \(m(\phi)^{L'}\). \(\square\)

5.9.5. The induction step.

**Lemma 5.6.** Given that the induction hypothesis \(I1 \land I2 \land I3 \land I4\) holds for the system execution \(\langle m(\phi)^{L'}, \prec, \succ \rangle\), \(m(\phi)^{L'} \subseteq m(\phi)^L\), it holds also for \(\langle m(\phi)^{L'} \cup \{L_j^{[a]}\}, \prec, \succ \rangle\), where \(L_j^{[a]} \in m(\phi)^L - m(\phi)^{L'}\) is such that for any \(L_j^{[b]} \in m(\phi)^L - m(\phi)^{L'}\), either

- \(L_j^{[a]} \succ L_j^{[b]}\) or
- \(L_j^{[a]} \not\prec L_j^{[a]}\) and for \(\ell_i^{[a-1]} \prec \ell_i^{[a]}\) and \(\ell_{i-1}^{[a-1]} \not\prec \ell_i^{[b]}\), 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, $L^a_x$ will denote the labeling operation execution added in the induction step to form $\langle m(\varphi) \cup \{L^a_x\} \rangle$.

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 $L^a_x$, if $L^a_x[\ell_{x}[k-1]] = \{3,4,5\}$, $k \geq 2$, then there exists an $L^a_x \in \text{ins}(\ell^a_x[n..k])$ such that $L^a_x[\ell_{x}[k-1]] = \{1,2\}$.

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

- $\ell_{\text{max}}[k-1] = \ell^a_x[a2]$ (as a reminder, this means $\ell_{\text{max}}[n..k] = \ell^a_x[a2][n..k]$),
- $\ell_{\text{max}}[k-1] \in \{2,3,4,5\}$, and
- $\mathcal{L}^k(G)$ (the $k$th level of the recursion in $\mathcal{L}$) was executed for $G = \ell_{\text{max}}[n..k]$ and returned the value $\ell^a_x[\ell_{x}[a2]][k-1] = 3$ (as in line 3) or $\ell^a_x[\ell_{x}[a2]][k-1] =$ dom($\ell_{\text{max}}[k-1]$) (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

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

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

The following lemma establishes that if in an earlier labeling operation execution a label $\ell^b_y$ was observed, the current labeling operation execution must read that label for $y$ or a label later than it.

**Lemma 5.8.** If $L^b_y \leadsto L^a_x$, it cannot be that $r_y(L^a_x) = \ell^b_y$, where $b1 < b$.

**Proof.** By Corollary 5.1, it follows that if $L^b_y \leadsto L^a_x$, then there exists a read $r_y(L^b_y) = \ell^b_y$ such that

$$w(L^b_y) \rightsquigarrow r_y(L^b_y) \rightarrow w(L^a_x),$$

where possibly $\alpha = x$ and $\beta = a - 1$. Since $w(L^a_x) \rightarrow r_y(L^a_x)$, it follows that $r_y(L^a_x) \rightarrow r_y(L^a_x)$. Since, in addition, $w(L^b_y) \rightarrow w(L^b_y)$, by register axiom B5, it cannot be that $r_y(L^a_x) = \ell^b_y$. \(\square\)
The following lemma states that there cannot be a label read by \( L^a_i \) that dominates \( \ell_x^{[a-1]} \) on level \( k1 > k \), where \( k \) is the level such that \( \ell_x^{[a-1]} \neq \ell_x^a \).

**Lemma 5.9.** For \( \ell_x^a \neq \ell_x^{[a-1]} \), it cannot be that there is an \( L_y^b \) such that

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

**Proof.** By the definition of \( \prec \), it must be that \( \ell_x^a \nleq \ell_y^b \), where \( k1 \geq k \). By the definition of \( \ell \), either \( \ell_{max} \nleq \ell_x^a \) or \( \ell_{max} = \ell_x^{[a-1]} \) (in which case by definition \( \ell_{max} \) is just \( \ell_x^{[a-1]} \)). The reason is that when executing \( L_{k3} \) 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 \( \ell_y^b \prec \ell_{max} \) or not.

If indeed \( \ell_y^b \nleq \ell_{max} \), by the definition of \( \nleq \), in order for \( \ell_x^a \nleq \ell_y^b \), \( \ell_y^b \nleq \ell_{max} \), and either \( \ell_{max} \nleq \ell_x^a \) or \( \ell_{max} = \ell_x^a \), it must be that

\[
\ell_x^a \nleq \ell_y^b \nleq \ell_{max} \nleq \ell_x^a,
\]

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

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

either \( \ell_x^{[a-1]} \nleq \ell_{max} \) or \( \ell_{max} = \ell_x^{[a-1]} \), it follows that \( k \geq k1 \), a contradiction.

However, if \( \ell_{max} \nleq \ell_y^b \), by the definition of \( \text{max(\text{dominating set}(\ell, \ell_x^{[a-1]}))} \), 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 \( \ell_x^{[a-1]} \nleq \ell_y^b \)). In order for \( \ell_{max} \nleq \ell_x^a \) or \( \ell_{max} = \ell_x^a \), together with \( \ell_x^a \nleq \ell_y^b \), it must be that \( \ell_x^a \) is in the cycle with these two labels. However, this implies \( k \geq k1 \), a contradiction.

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 \( S^k = \{ L_i^a[i], L_i^b[i], \ldots, L_i^b[m] \}, i1, \ldots, i_m \in \{ 1..n \} \), and \( i_\alpha \neq i_\beta \) for any \( \alpha, \beta \in \{ 1..m \} \) be the set of labeling operation executions such that for any \( L_i^a, L_j^b \in S^k \),

- \( \ell_i^a[k] \neq \ell_j^b[k] \) and \( \ell_i^a[k-1], \ell_j^b[k-1] \in \{ 3, 4, 5 \} \), and
- for \( L_i^a \in \text{max obs}(L_i^a) \) and \( L_i^a \in \text{max obs}(L_i^a) \), it is the case that \( b1 \leq b \) and \( a1 \leq a \).

It must be that \( \text{card}(S^k) \leq k \).

**Proof.** Assume by way of contradiction that \( \text{card}(S^k) > k \). By Lemma 5.7, for each \( L_i^a \in S^k \), \( \text{card}(L_i^a[1..n,k]) \geq 2 \), 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

\[
\text{not read by \( L_i^a \) between labeling operation executions \( L_i^a, L_j^b \in S^k \) to be as follows.}
\]

**Definition 5.3.** \( L_i^a \) not read by \( L_j^b \) if \( r_i(L_j^b) \neq \ell_i^{a1} \), \( a1 \in \{ a-1, a \} \).

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, \( L_j^1 \) 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 \( L_i^a \in S^k \) 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 \( L_i^a \in S^k \), \( L_i^a \preceq L_i^{a-1} \), that is, a move at level \( k \), it must be that when executing \( L_i^{k+1} \) in \( L_i^a \), line 5 was executed and that \( \text{num labels} < (k + 1) - 1 \) (at most \( k - 1 \) labels not including its own, or \( k \) including it, were read in the \( T^k \) subgraph \( L_i^a[n..k] \)), a contradiction.

Since it was assumed by way of contradiction that there are more than \( k \) labeling operation executions in \( S^k \), 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 \( S^k \), 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 \( S^k \), by definition (for \( L_j^b \in \text{max obs}(L_i^a) \) and \( L_i^a \in \text{max obs}(L_j^b) \), it is the case that \( b_1 \leq b \) and \( a_1 \leq a_1 \) neither \( r_i(L_j^b) = \ell_i[a] \), \( a > a_1 \), nor \( r_j(L_i^a) = \ell_j^b \), \( b_1 > b \). Given \( L_i^a \) not read by \( L_j^b \), it thus follows by atomic register axiom B5 that \( r_i(L_j^b) \rightarrow w(L_i^{a-1}) \). However, this implies

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

By axiom A4, it follows that \( w(L_j^{b-1}) \rightarrow r_j(L_i^a) \), implying that it cannot be that \( L_j^b \) not read by \( L_i^a \). \( \Box \)

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 \( S \cup \{L_i^a\} \) 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 \( L_{i_s}^c \) 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 \( L_{i_s}^c \) 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 \( S^k \), where

\[ L_{i_1}^{a_1} \text{ not read by } L_{i_2}^{a_2} \text{ not read by } \cdots \text{ not read by } L_{i_m}^{a_m}, \]

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

Let it be proven by induction that $r_{i-1}(L_i^{[a_m]}) \rightarrow w(L_i^{[a_{i-1}]})$. This will imply

$$w(L_i^{[a_{m-1}]}) \rightarrow r_{i-1}(L_i^{[a_m]}) \rightarrow w(L_i^{[a_{i-1}]}) \rightarrow r_i(L_i^{[a_i]}),$$

from which by axiom A4 follows $w(L_i^{[a_{m-1}]}) \rightarrow r_i(L_i^{[a_i]})$, implying, $r_i(L_i^{[a_i]}) = L_i^{[a_m]}$, as desired.

The proof that $r_{i-1}(L_i^{[a_m]}) \rightarrow w(L_i^{[a_{i-1}]})$ 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,

$$r_{i-1}(L_i^{[a_{m-1}]}) \rightarrow w(L_i^{[a_{m-1}]}).$$

Since $r_{i-1}(L_i^{[a_{m-1}]}) \neq \ell_i^{[a_{m-1}]}$, it follows that

$$r_{i-1}(L_i^{[a_{m-2}]}) \rightarrow w(L_i^{[a_{m-1}]}) \rightarrow r_{i-2}(L_i^{[a_{m-1}]}) \rightarrow w(L_i^{[a_{i-1}]}).$$

By axiom A4, it follows that $r_{i-1}(L_i^{[a_{m-1}]}) \rightarrow w(L_i^{[a_{i-1}]})$, implying the claim.  

Thus the node $L_i^{[c]}$ read at least $k$ labels apart from its own in the $T^k$ subgraph $L_x^{[c] [n..k]}$, 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 $T^n$ 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 $\ell_x^{[a]} \in \{3, 4, 5\}$, there are at most $k - 1$ labels $L_y^{[b]}$, where $L_y^{[b]} \in \text{max obs}(L_x^{[a]})$, such that $\ell_y^{[b]} = \ell_x^{[a]}$ and $\ell_y^{[b]}[k-1] \in \{3, 4, 5\}$.

**Proof.** For any two labeling operation executions $L_y^{[b]}$, $L_x^{[a]} \in \text{max obs}(L_x^{[a]})$, by the definition of max obs($L_x^{[a]}$), neither $r_y(L_x^{[a]}) = \ell_y^{[b]}$, $b1 > b$, nor $r_x(L_x^{[a]}) = \ell_x^{[a]}$, $c1 > c$. Also, by definition, for $L_y^{[b]} \in \text{max obs}(L_x^{[a]})$, neither $r_y(L_x^{[a]}) = \ell_y^{[b]}$, $b1 > b$, nor $r_x(L_y^{[b]}) = \ell_x^{[a]}$, $a1 \geq a$. The claim follows from Lemma 5.10 by defining $S^k$ 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 II–I4 through rather tedious case analysis. It is omitted 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 \(i\)th 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

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


BOUNDED CONCURRENT TIME-STAMPING


