

# The power of logical clock abstractions\*

Ajay D. Kshemkalyani

Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607, USA (e-mail: ajayk@cs.uic.edu)

Received: March 15, 2002 / Accepted: January 15, 2004  
 Published online: June 9, 2004 – © Springer-Verlag 2004

**Abstract.** Vector and matrix clocks are extensively used in asynchronous distributed systems. This paper asks, “how does the clock abstraction generalize?” To address this problem, the paper motivates and proposes logical clocks of arbitrary dimensions. It then identifies and explores the conceptual link between such clocks and knowledge. It establishes the necessary and sufficient conditions on the size and dimension of clocks required to attain any specified level of knowledge about the timestamp of the most recent system state for which this is possible without using any messages in the clock protocol. The paper then gives algorithms to determine the timestamp of the latest system state about which a specified level of knowledge is attainable in a given system state, and to compute the timestamp of the earliest system state in which a specified level of knowledge about a given system state is attainable. The results are applicable to applications that deal with a certain class of properties, identified as monotonic properties.

**Keywords:** Causality – Clocks – Concurrency – Distributed system – Knowledge – Logical time – Time

## 1 Introduction

Logical clocks are used extensively by distributed applications that need to maintain logical time in an asynchronous message-passing system. Logical clocks track causality which helps to determine the extent of the past execution that could possibly affect actions at any process. Logical time and causality are important tools in the design of distributed applications, as elegantly expressed by Schwarz and Mattern [21] and Fidge [8]. In this paper, we define logical clock abstractions and analyze their power.

A distributed system  $(N, L)$  can be modeled as a network consisting of  $N$ , a set of processes that communicate by asynchronous message-passing over  $L$ , a set of links. An execution

of the system generates events at each process. An event may be a *send* event, a *receive* event, or an *internal* event. The (potential) *causality relation*  $\prec$  on the set of events  $H$  in a system execution was defined by Lamport [15] as follows: event  $e \prec e'$  if and only if (i) events  $e$  and  $e'$  occur at the same process and  $e$  occurs before  $e'$ , (ii) event  $e$  is the event at which a message is sent and event  $e'$  is the event at which that message is received by a different process, or (iii) there exists some event  $e''$  such that  $e \prec e''$  and  $e'' \prec e'$ .

Scalar, vector, and matrix clocks are the only clock systems that have been developed in the literature. Each process  $i$  maintains a local clock  $Clk_i$  to track logical time. Lamport defined scalar clocks with the property that for events  $e$  and  $f$ ,  $e \prec f$  implies  $Clk(e) < Clk(f)$  [15]. Such clocks consist of a single integer. Mattern [16] and Fidge [7] independently formalized vector clocks which have the property that  $e \prec f$  if and only if  $Clk(e) < Clk(f)$ . Some example areas that use vector clocks are checkpointing and recovery, garbage collection, causal memory, detecting global predicates, distributed discrete-event simulation, virtual time, multicast and group communication algorithms, causal ordering, maintaining consistency of replicated files, taking efficient snapshots of a system, global time approximation, termination detection, bounded multiwriter construction of shared variables, mutual exclusion, debugging, managing drift between physical clocks, synchronizing processes, and defining concurrency measures. There are many good references for the use of vector clocks in these areas [7, 8, 16, 21].

A vector clock consists of  $n = |N|$  integers, with the  $<$  operator on vectors defined as follows:  $V_1 \leq V_2$  if and only if, for all  $k \in N$ ,  $V_1[k] \leq V_2[k]$ ;  $V_1 < V_2$  if and only if  $V_1 \leq V_2$  and  $V_1 \neq V_2$ . The following rules are used by a process to maintain its clock using the *vector clock protocol*.

- VC1. Before process  $i$  executes an internal event, it does the following.
  - $Clk_i[i] = Clk_i[i] + d$  ( $d > 0$ ).
- VC2. Before process  $i$  executes a send event, it does the following.
  - $Clk_i[i] = Clk_i[i] + d$  ( $d > 0$ ).
 Send message timestamped by  $Clk_i$ .
- VC3. When process  $j$  receives a message with timestamp  $T$  from process  $i$ , it does the following.

\* This material is based on work supported by the National Science Foundation under Grant No. 9875617.  
 A conference version appeared in [14] and the full technical report is in [13].

- $(k \in N) \text{Clk}_j[k] = \max(\text{Clk}_j[k], T[k]);$
- $\text{Clk}_j[j] = \text{Clk}_j[j] + d \quad (d > 0);$
- deliver the message.

A canonical version of these clocks uses  $d = 1$ . With canonical vector clocks,  $\text{Clk}(e)[i] = |\{e_i \mid e_i \preceq e\}|$ , where  $e_i$  denotes an event at process  $i$ .

Charron-Bost showed that to capture the property “ $e \prec f$  if and only if  $\text{Clk}(e) < \text{Clk}(f)$ ”, vector clocks must have a size equal to the dimension of the partial order  $(H, \prec)$ , which can be as large as  $n$  [4]. Hence, clocks of size  $n$  are used.

Matrix clocks, first proposed by Wu and Bernstein [24], contain vector clock information about other processes’ views of the system execution. A matrix clock is an array of size  $n \times n$ .  $\text{Clk}_i[j, k]$  is  $i$ ’s knowledge of process  $j$ ’s knowledge of process  $k$ ’s local (scalar) clock value. Some example areas that use matrix clocks are designing fault-tolerant protocols and distributed database protocols [12, 24], including protocols to discard obsolete information in distributed databases [20], and protocols to solve the replicated log and replicated dictionary problems [24]. Recently, matrix clocks have also been used to implement the disconnected mode of operation in mobile databases [11]. The following rules are used by a process to maintain its clock using the *matrix clock protocol*.

- MC1. Before process  $i$  executes an internal event, it does the following.
- $\text{Clk}_i[i, i] = \text{Clk}_i[i, i] + d \quad (d > 0).$
- MC2. Before process  $i$  executes a send event, it does the following.
- $\text{Clk}_i[i, i] = \text{Clk}_i[i, i] + d \quad (d > 0).$
- Send message timestamped by  $\text{Clk}_i$ .
- MC3. When process  $j$  receives a message with timestamp  $T$  from process  $i$ , it does the following.
- $(k \in N) \text{Clk}_j[j, k] = \max(\text{Clk}_j[j, k], T[i, k]);$
  - $(l \in N \setminus \{j\}) (k \in N),$   
 $\text{Clk}_j[l, k] = \max(\text{Clk}_j[l, k], T[l, k]);$
  - $\text{Clk}_j[j, j] = \text{Clk}_j[j, j] + d \quad (d > 0);$
  - deliver the message.

Due to the importance of vector and matrix clocks, many efficient implementation designs for the *clock protocols* have also been studied by varying the assumptions about the system or the executions [12, 17, 19, 22, 23].

The processes in the system collectively run a clock protocol to maintain logical time. In order to abstract clock protocols, we make the following observations about vector and matrix clock protocols, and about applications such as those mentioned above, that use scalar and matrix clocks. First, the *clock protocols* do not introduce any additional protocol events but rather perform some actions as specified by rules VC1–VC3 and MC1–MC3 atomically with the events of the *underlying execution*. Thus, the clock protocols’ actions are superimpositions on the underlying execution events. Second, the protocol execution does not generate any messages. Rather, the protocol information exchanged by the processes is done via *piggybacking* the clock value (i.e., *timestamp*) on the messages generated by the underlying execution. By piggybacking timestamps, information about clocks is exchanged using the *ambient message-passing* and without using any *protocol messages*. Third, as the execution progresses and newer clock values are generated, information about the clock values is

*continually diffused* by each process, using only messages of the underlying execution, whenever sent. Fourth, the clock protocols are *inhibition-free*, i.e., they do not prevent events that would otherwise occur in the underlying system execution [5]. Fifth, it follows from the first, second, and fourth observations that the clock protocols do not alter the structure of the partial order  $(H, \prec)$ .

We now give some definitions to introduce our problem informally. The projection of an execution on a process identifies a local execution at that process. The union of the events in some prefix of the local execution at each process is a *cut* of the execution. Each cut in an execution can be assigned a vector timestamp (defined later in Definition 1). A *global state* of the system is the collection of values of the local variables at each process after the execution of the events in some cut [3]. A *cut is consistent* if it contains a send event for each message for which the receive event also belongs to the cut. Each global state is identified by its corresponding cut. A *global state is consistent* if it corresponds to a consistent cut. It has been shown by Mattern [16] that for any given (terminating) execution, the set of all global states, which corresponds to the set of all cuts, *Cuts*, forms a distributive lattice ordered by the subset relation  $\subseteq$ ; the set of all consistent global states is its sublattice. Also, given any execution, there exists an isomorphism between  $(\text{Cuts}, \subseteq)$ , and its  $(T^1, <)$ , the set of its vector timestamps ordered by  $<$ .

Let us analyze the manner in which the applications listed earlier use the vector and matrix clocks. Each application tracks a specific property of interest. Such a property is parameterized by some function of the global state and the structure of the property is such that the cardinality of facts about this property is bounded by the number of global states. Facts about this property can be deduced directly from the timestamps exchanged by the clock protocol. Some examples of properties of interest to applications are as follows. (i) The property “execution has progressed at least up to event  $x$  at (a specific) process  $i$ ” is useful in debugging. (ii) The property “all logs up to global state  $s$  can be discarded” is useful in checkpointing. (iii) The property “global virtual time has advanced to at least time  $t$ ” is useful for tracking global virtual time and for distributed simulations. (iv) The property “information of the execution at least up to event  $x$  at process  $i$  has reached all processes and hence process  $i$  can discard this execution prefix from its log” is useful for efficiently discarding local logs to solve the replicated log problem. For each such property, the various facts about it as the execution progresses are related by a semantic inclusion relation “ $\sqsubseteq$ ” (if  $\phi \sqsubseteq \psi$ , then  $\psi$  semantically includes  $\phi$ ). For certain properties such as (i) and (iii) above, the semantic inclusion relation on the facts forms a total order. For other properties such as (ii) and (iv) above, the semantic inclusion relation on the facts forms a partial order that is isomorphic to the lattice of global states. For all the above properties, each fact is stable, i.e., once it becomes true, it remains true thenceforth. To characterize both (i) the stability of facts, and (ii) the semantic inclusion relation on the facts as an execution progresses, we term such facts as *monotonic facts* about the property of interest. The property of interest is termed as a *monotonic property of interest* and has the characteristic that multiple (stable) facts about it can be true in the same execution. Henceforth, we implicitly assume that such a property is parameterized by some function of the global state

and the structure of the property is such that the cardinality of facts about this property is bounded by the number of global states.

For each of the applications considered above, we saw that the set of monotonic facts ordered by  $\sqsubseteq$  is a lattice. Generalizing this, we have that for any execution and a monotonic property of interest, the set of monotonic facts ordered by  $\sqsubseteq$  is a lattice and there is a semantically greatest fact in each global state. Further, for any execution, there is a homomorphism from  $(Cuts, \subset)$  to  $(\mathcal{M}, \sqsubseteq)$ , where  $\mathcal{M}$  is the set that contains the greatest monotonic fact of interest at each state in  $(Cuts, \subset)$ . Combining this homomorphism with the isomorphism between  $(\mathcal{T}^1, <)$  and  $(Cuts, \subset)$  for that execution gives a homomorphism from  $(\mathcal{T}^1, <)$  to  $(\mathcal{M}, \sqsubseteq)$ , formalized as Axiom 1. The results of this paper hold for any application for which Axiom 1 is valid.

**Axiom 1** *For any given execution and any monotonic property of interest, the semantically greatest monotonic fact in a global state can be uniquely identified by the vector timestamp of that global state.*

Hence, we treat the semantically greatest fact in a state as synonymous to that state, and identify the fact by the vector timestamp of that state.

Vector clocks can be thought of as imparting knowledge to a process: when  $V[i] = x$  at process  $h$ , process  $h$  knows that process  $i$  has executed at least  $x$  events. Matrix clocks impart one more level of knowledge: when  $M[i, j] = x$  at process  $h$ , process  $h$  knows that process  $i$  knows that process  $j$  has executed at least  $x$  events. Vector and matrix clocks are convenient because they are updated without sending any additional messages; knowledge is imparted via the *inhibition-free ambient message-passing* that (i) *eliminates protocol messages* by using piggybacking, and (ii) *continually diffuses* the latest knowledge using only messages, whenever sent, by the underlying execution.

This paper asks the question: “*how does this clock abstraction generalize?*” The problem is cast in terms of knowledge which provides a rigorous framework for studying this problem. Formalizing the levels of knowledge in a distributed system simplifies the analysis of various problems and the design of various protocols to solve the problems [6, 10]. Knowledge analysis also provides a formal method for reasoning about distributed protocols and executions of asynchronous distributed systems [2]. A good reference work on the theory of knowledge in distributed systems is a textbook by Fagin et al. [6].

The main contributions of the paper are outlined as follows. The paper motivates and proposes logical clocks of arbitrary dimensions. It then identifies and explores the conceptual link between such clocks and knowledge. It establishes the necessary and sufficient conditions on the size and dimension of clocks required to attain any specified level of knowledge about the timestamp of the most recent global state for which this is possible without using protocol messages. The paper then gives algorithms to determine the timestamp of the latest global state about which a specified level of knowledge is attainable in a given global state, and to compute the timestamp of the earliest future state in which a specified level of knowledge about a given global state is attainable.

In addition to the main results, this paper also makes three other contributions. First, the paper directly addresses the concluding thesis of Schwarz-Mattern [21] that “... None of the presented (causality-tracking or timestamping) schemes is sufficiently mature to serve as a *general-purpose mechanism* for the analysis of causality. ... The problem of anticipating the relevant behavior, assigning meaningful semantics to general global predicates, and finding correct and efficient algorithms for their detection remains a challenge. ... Anyhow, the holy grail of causality analysis has not been found yet.” We expect that the results give a better understanding and insight into tracking causality. Second, we define a new variant of levels of knowledge, that is tailored for asynchronous message-passing systems and is based on consistent cuts. This variant is useful to applications reasoning with monotonic properties. Third, a consequence of this new definition of levels of knowledge is that classical problems such as the “muddy children” [10] and “cheating husbands” [9] problems have easily understandable solutions in asynchronous message-passing systems that use our system model. Hitherto, the synchronous execution model had been widely (but not exclusively) assumed in the theory of knowledge. All the classical examples in this theory had been explained in the synchronous execution model [6, 9, 10].

Section 2 describes the distributed system model and introduces the necessary formalism about knowledge and about monotonic properties which are the kinds of properties that can use the results. Section 3 examines the relation between attaining levels of knowledge and message chains in distributed asynchronous executions. Using the formalism about knowledge, the paper then formulates specific problems about logical clock abstractions. Section 4 identifies the properties that a logical clock abstraction should possess and proposes the  $\alpha$ -dimensional clock. This section then proves that the new  $\alpha$ -dimensional clock satisfies the properties that any logical clock abstraction should possess. Section 5 describes the levels of knowledge that can be inferred from the  $\alpha$ -dimensional timestamps and formally answers the problems posed in Sect. 3. Section 6 gives the conclusions.

## 2 System model and logic

### 2.1 System model

A distributed system can be modeled by a network  $(N, L)$ , where  $N$  is the set of processes that communicate by message-passing over  $L$ , the set of logical communication links. We assume an *asynchronous (distributed) system*, i.e., there is no global clock or shared memory, relative process speeds are independent, and message delivery times are finite but unbounded. We assume the network is fully connected and reliable, and messages can be delivered out of order. Let  $|N|$  be denoted by  $n$  and let the processes be numbered 1 to  $n$ .

The notion of an event at a process is primitive. Let  $\mathcal{H} = (\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_n)$  be the vector of sets of possible events, where events in the possibly infinite set  $\mathcal{H}_i$  correspond to operations performed by process  $i$ . An event  $e$  at process  $i$  is denoted  $e_i$  and can be (i) an *internal event*, (ii) a *send* event, denoted  $send(m)$ , at which message  $m$  is sent by process  $i$ , or (iii) a *receive* event, denoted  $receive(m)$ , at which message  $m$  is received by process  $i$ . The notation  $m_i$  denotes that message

$m$  was sent by process  $i$ ; the subscript is omitted when it is not important. The notation  $e_i^j$  refers to the  $j$ th event at process  $i$ . The null event  $e_i^0$  at each process  $i$  is introduced to simplify later formalisms. We denote the initial set of null events as  $H^0$ . We also require that for  $x > 0$  and for all  $i$  and  $j$  in  $N$ ,  $e_i^0 \prec e_j^x$ . The *local history* of process  $i$  in a *local execution*, denoted  $H_i$ , is any (possibly infinite) subset of events  $\{e_i^0, e_i^1, e_i^2, e_i^3, \dots\}$  of  $\mathcal{H}_i$ , that is totally ordered by the order of occurrence of the events at the process. The *system execution*  $H$  is the set of events  $\bigcup_{i \in N} H_i$ . The events in a given system execution are related by the *causality relation*  $\prec$ , defined in Sect. 1, and form an irreflexive partially ordered set  $(H, \prec)$ .

A *cut* of an execution  $H$  is a subset of the execution such that it is the union of the events in an arbitrary prefix (of the total order) of the local execution of each process. A cut  $C$  is a *consistent cut* if  $\text{send}(m)$  belongs to  $C$  whenever  $\text{receive}(m)$  belongs to  $C$ . A consistent cut  $C$  has the property that it is downward-closed with respect to  $\prec$ , i.e.,  $e \in C \implies (\forall e' (e' \prec e \implies e' \in C))$ . For any cut  $C$ , the set consisting of the latest event in  $C$  at each process is termed as the *front* of the cut, denoted  $F(Cut)$ . The *(local) state of a process* after event  $e_i^x$ , denoted  $s_i^x$ , is the value of its variables at this point in the execution. Thus, the process state is a function of the local history. The *(global) state of the system* is a collection of local states, one per process. Given an execution, for every cut, there is a corresponding global state and a corresponding front, and vice versa. We will refer to a global state as occurring at the corresponding cut, meaning, the state of each process after the execution of events in that cut. A *global state is consistent* if its corresponding cut is consistent. Given a cut  $Cut$ , its projection  $Cut_i$  on process  $i$  is the subset of  $Cut$  that contains only all its events that are at process  $i$ . Given a cut  $Cut$ , its projection  $F_i(Cut)$  on process  $i$  is the element of  $F(Cut)$  at process  $i$ .

Vector clocks can be used to assign timestamps to cuts. For a cut  $Cut$ , we define its timestamp  $T(Cut)$  such that the  $i$ th component of the timestamp is the  $i$ th component of the timestamp of  $F_i(Cut)$ .

**Definition 1 (Vector timestamp of a cut)** The vector timestamp of a cut  $Cut$  is defined as  $T(Cut) =_{def} (i \in N) T(Cut)[i] = T(F_i(Cut))[i]$

We define the *timestamp of a global state* and the *timestamp of the front of a cut* to be the timestamp of the corresponding cut.

Mattern [16] showed that for any given execution, the set of all cuts (for a terminating execution) forms a distributive lattice ordered by the subset relation; the set of all consistent cuts is its sublattice. For any execution, observe that there exists an isomorphism between  $(Cuts, \subset)$ , the lattice of cuts ordered by the subset relation, and  $(T^1, <)$ , the set of vector timestamps ordered by  $<$ . The sublattice of consistent cuts is not visible to any process, but gives the possible consistent cuts which could have occurred, consistent with the local states of individual processes. Our results consider such an execution.

For an event  $e$  in an execution,  $\downarrow e$  is the maximal set of events that happen before or equal  $e$ .

**Definition 2 (Past of an event  $e$  in execution  $H$ )**

$\downarrow e =_{def} \{e' \mid e' \preceq e \text{ and } e, e' \in H\}$

The cut  $\downarrow e$  has a unique maximal event  $e$  and is downward-closed in  $(H, \prec)$ . As the set of all downward-closed cuts forms

a lattice, therefore for any arbitrary set of events  $X$ , the sets  $\bigcap_{x \in X} \downarrow x$  and  $\bigcup_{x \in X} \downarrow x$ , which we also denote by  $\cap_{\downarrow} X$  and  $\cup_{\downarrow} X$ , respectively, are also downward-closed cuts. This property will be used in Sects. 5.2 and 5.3 by setting  $X$  to  $F(Cut)$  to prove Theorems 7 and 8, respectively.

Based on Definition 2 and the semantics of the intersection operation, it is easy to show the following lemma. The proof is left to the reader.

**Lemma 1**  $e \in \cap_{\downarrow} X \iff \forall x \in X, e \preceq x$

From Lemma 1, it follows that  $\cap_{\downarrow} X$ , the maximal set of events that causally precede every  $x \in X$ , represents the maximum cut with the following property. Any fact that is true in the global state after this cut can be known in the local state of each process  $i$  after any event  $x_i \in X$  has occurred, if adequate information is propagated on messages. The propagation of information on messages is studied in Sect. 3.

## 2.2 Logic and formal semantics

A definition of knowledge requires the identification of an appropriate set of possible worlds, and a family of possible relations between those worlds [6, 10]. Fact  $\phi$  can be a primitive proposition or a formula using the usual logical connectives on primitive propositions and the  $K$  and  $E$  operators. The traditional semantics of knowledge, using the  $K$  and  $E$  operators and based on *timed executions* used by Halpern and Moses [10], is summarized informally as follows. A process  $i$  that knows a fact  $\phi$  is said to have knowledge  $K_i(\phi)$ , and if “every process in the system knows  $\phi$ ”, then the system exhibits knowledge  $E^1(\phi) = \bigwedge_{i \in N} K_i(\phi)$ . A knowledge level of  $E^2(\phi)$  indicates that every process knows  $E^1(\phi)$ , i.e.,  $E^2(\phi) = E(E^1(\phi))$ . Inductively, a hierarchy of levels of knowledge  $E^j(\phi)$  ( $j \in \mathbb{Z}^*$ ) gets defined, where  $\mathbb{Z}^*$  is used to denote the set of whole numbers  $\{0, 1, 2, 3, \dots\}$ . It has been shown that  $E^{k+1}(\phi) \implies E^k(\phi)$ . Each level in the hierarchy represents a different level of group knowledge among the processes. Common knowledge of  $\phi$ , denoted as  $C(\phi)$ , is defined as the knowledge  $X$  which is the greatest fixed point of  $E(\phi \wedge X)$ , and implies the conjunction  $\bigwedge_{j \in \mathbb{Z}^*} E^j(\phi)$ . Common knowledge is not attainable in asynchronous systems because simultaneous action is required for its achievement [10].

In our system model, the possible worlds are the consistent cuts of the set of possible executions in an asynchronous system. Let  $(a, c)$  denote cut  $c$  in asynchronous execution  $a$ . As each cut also identifies a global state,  $(a, c)$  is also used to denote the state of the system after  $(a, c)$ .  $(a, c)_i$  denotes the projection of  $c$  on process  $i$ , and is also used to denote the state of process  $i$  after  $(a, c)$ . Two cuts  $c$  and  $c'$  are indistinguishable by process  $i$ , denoted  $(a, c) \sim_i (a', c')$ , if and only if  $(a, c)_i = (a', c')_i$ . Although the semantics of knowledge are based on *asynchronous executions*, as was done by Panangaden and Taylor [18], instead of on *timed executions* used by Halpern and Moses [10], the semantics are different from Panangaden and Taylor’s [18].

A modal operator that is indexed by process identifiers is  $K_i$ . The operator  $K_i$  is required to satisfy the five axioms of modal logic S5 [6].  $K_i(\phi)$  means “ $\phi$  is true in all possible consistent states that include process  $i$ ’s local state”. Observe that  $K_i(\phi)$  is implicitly quantified over all consistent global

states over all executions, that include  $i$ 's local state. Similar meanings hold for  $E(\phi)$  and  $E^k(\phi)$ , for  $k > 1$ . We also define  $E^0(\phi)$  to be  $\phi$  to simplify the proof of Theorem 7. Formal definitions are given next, using the  $\models$  symbol to denote the “satisfaction” operator.

**Definition 3 (Knowledge in asynchronous systems defined using consistent cuts)**

- $(a, c) \models \phi$  if and only if  $\phi$  is true in cut  $c$  of execution  $a$ .
- $(a, c) \models K_i(\phi)$  if and only if  $\forall(a', c'), ((a', c') \sim_i (a, c) \implies (a', c') \models \phi)$
- $(a, c) \models E^0(\phi)$  if and only if  $(a, c) \models \phi$
- $(a, c) \models E^1(\phi)$  if and only if  $(a, c) \models \bigwedge_{i \in N} K_i(\phi)$
- $(a, c) \models E^{k+1}(\phi)$  for  $k \geq 1$  if and only if  $(a, c) \models \bigwedge_{i \in N} K_i(E^k(\phi))$ , for  $k \geq 1$
- $(a, c) \models C(\phi)$  if and only if  $(a, c) \models X$ , where  $X$  is the greatest fixed point satisfying  $X = E(X \wedge \phi)$

Lemma 1 can be represented using this logic as follows. Assuming that adequate knowledge about local histories is propagated on messages, for any cut  $X$ ,  $(a, X) \models E(\cap_{\downarrow} F(X))$ , meaning that all the processes know after the execution of  $X$  that “events in  $\cap_{\downarrow} F(X)$  have occurred”.

As knowledge can be known by a process only in a local state, we say “ $i$  knows  $\phi$  in state  $s_i^x$ ”, denoted  $s_i^x \models \phi$ , to mean  $(\forall(a, c)) ((a, c)_i = s_i^x \implies (a, c) \models \phi)$ . Analogously,  $(\forall(a, c)) ((a, c)_i = s_i^x \implies (a, c) \models K_i(\phi))$  is represented by  $s_i^x \models K_i(\phi)$ . Recall that  $\phi$  can be of the form  $E^k(\psi)$ , for any fact  $\psi$ .

**Definition 4 (Learning)** Process  $i$  learns  $\phi$  in state  $s_i^x$  of execution  $a$  if  $i$  knows  $\phi$  in  $s_i^x$  and, for all states  $s_i^y$  in execution  $a$  such that  $y < x$ ,  $i$  does not know  $\phi$ .

We also say that a process attains  $\phi$  (in some state) if the process learns  $\phi$  in the present or an earlier state. A fact  $\phi$  is attained in an execution  $a$  if  $\exists c, (a, c) \models \phi$ . Observe that a process cannot attain a fact before the fact is attained in an execution. This implies that even though a fact becomes true in an execution, information may need to be propagated for a process to learn the fact.

We now define *local* facts analogous to the definition by Chandy and Misra [2].

**Definition 5 (Local fact)** A fact  $\phi$  is local to process  $i$  in system  $A$  if  $A \models (\phi \implies K_i \phi)$

A fact that is not local is a *global fact*. The state of a process, the local clock value of a process, and the local component of the timestamp of an event at a process are examples of local facts. The global state of a system and the timestamp of a cut are examples of global facts.

Chandy and Misra [2] formalized how processes learn facts, by relating knowledge gain and knowledge loss to message chains in the execution. We modify the definition of a (possibly infinite) message chain used in [2] to adapt it to finite chains which are the chains pertinent to our results.

**Definition 6 (Message chain)** A message chain in an execution is a sequence of messages  $[m_{i_k}, m_{i_{k-1}}, m_{i_{k-2}}, \dots, m_{i_1}]$  such that (i) for all  $0 < j \leq k$ ,  $m_{i_j}$  is sent by process  $i_j$  to process  $i_{j-1}$ , and (ii) for all  $1 < j \leq k$ ,  $\text{receive}(m_{i_j}) \prec \text{send}(m_{i_{j-1}})$ .

A message chain identifies the corresponding process chain  $[i_0, i_1, \dots, i_{k-2}, i_{k-1}, i_k]$ .

The above definition adopts the convention that a process chain lists the processes in the reverse order in which they send the messages in the corresponding message chain. Furthermore, a process chain includes the recipient of the last message sent in the corresponding message chain, and this is the first process in the process chain. A message chain with  $k$  messages thus identifies a process chain with  $k + 1$  processes.

The formalism so far will be used in the results on how processes learn about local components of the clock values of other processes (local facts) and about timestamps of global states (global facts that are collections of local facts). Although the logic defined is over the set of all possible executions, the problems about the logical clocks that we address are meaningful in the context of the partial order of a single execution only.

The proposed definitions of knowledge (Definition 3) are based on consistent cuts in asynchronous executions and are directly useful to applications reasoning about monotonic properties. The only other definition of knowledge in asynchronous systems, that is based on consistent cuts, was given by Panangaden and Taylor [18] and is not well-suited to monotonic properties. They defined  $P_i(\phi)$  to mean “ $\phi$  is true in *some* consistent state in the same asynchronous run, that includes process  $i$ 's local state” and then defined *concurrent* knowledge  $E^c(\phi)$  to be  $\bigwedge_{i \in N} K_i P_i(\phi)$ . Using the conventions of this section, this is expressed as follows.

- $(a, c) \models P_i(\phi)$  if and only if  $\exists(a', c'), ((a', c') \sim_i (a, c) \wedge (a', c') \models \phi)$
- $(a, c) \models E^{c^1}(\phi)$  if and only if  $(a, c) \models \bigwedge_{i \in N} K_i P_i(\phi)$
- $(a, c) \models E^{c^{k+1}}(\phi)$  for  $k \geq 1$  if and only if  $(a, c) \models \bigwedge_{i \in N} K_i P_i(E^{c^k}(\phi))$ , for  $k \geq 1$
- $(a, c) \models C^c(\phi)$  if and only if  $(a, c) \models X$ , where  $X$  is the greatest fixed point satisfying  $X = E^c(X \wedge \phi)$

$E^c(\phi)$  means that every process at the (given) cut knows only that  $\phi$  is true in *some* cut that is consistent with its own local state. The concurrent knowledge definitions are weaker than the corresponding knowledge definitions in Definition 3. But for a local, stable fact, and assuming other processes learn the fact via message chains, it can be seen that the two definitions become equivalent [14]. If concurrent common knowledge  $C^c(\phi)$  is attained at a consistent cut, then (informally speaking) each process at its local cut state knows that “in some state consistent with its own local cut state,  $\phi$  is true *and that* all processes know all this same knowledge”. Although concurrent common knowledge  $C^c$  is attainable in asynchronous systems using specific protocols [18], such as those that use protocol messages in a snapshot-like algorithm for each fact, it can be seen that these are not suited for the specific problem being addressed in this paper.

We note that when reasoning about knowledge and time in the theory of knowledge, “a typical assumption . . . [is that of a] completely synchronous system,” as stated by Halpern and Fagin [9, p. 66]. Classical problems such as the “muddy children” problem [10] and the “cheating husbands” problem [9] which are widely used to illustrate the theory of knowledge have all been expounded in the synchronous system model. Observe that the properties dealt with in all such classical problems are monotonic properties, formally defined in the next section. As we proceed with the results in this paper, we

can observe that the classical problems solved by assuming a synchronous system can also be readily mapped to and solved in the framework of our asynchronous system.

### 2.3 Monotonic properties and timestamps

Stable facts, which have received considerable attention in the literature, are facts that, once they become true, continue to be true in that execution [2].

**Definition 7 (Stable fact)** For a given execution “ $a$ ”, a fact  $\phi$  is stable if and only if for every cut  $c$  at which  $(a, c) \models \phi$ , and for every cut  $c'$  such that  $c \subset c'$ , we have  $(a, c') \models \phi$ .

As explained in Sect. 1, for a variety of applications, the facts about a property of interest are also related by a semantic inclusion relation “ $\sqsubseteq$ ” (if  $\phi \sqsubseteq \psi$ , then  $\psi$  semantically includes  $\phi$ ). Such facts are stable. To characterize both (i) the stability of facts, and (ii) the semantic inclusion relation on the facts in an execution, we term such facts as *monotonic facts* about the property of interest. The stable property of interest has the characteristic that multiple (stable) facts about it can potentially be true in the same execution, and the property is called a *monotonic property* of interest.

**Definition 8 (Monotonic fact about property  $\Phi$ )** For a given execution “ $a$ ”, fact  $\phi$  about property  $\Phi$  is monotonic if and only if for every cut  $c$  at which  $(a, c) \models \phi$ , and for every cut  $c'$  such that  $c \subset c'$ , there exists some semantically related fact  $\psi$  about  $\Phi$  such that:

- (i)  $(a, c') \models \psi$ ,
- (ii)  $(a, c') \models (\phi \sqsubseteq \psi)$ , and
- (iii)  $\psi$  is the semantically-greatest fact in  $c'$ .

A monotonic fact is stable. A stable fact is also monotonic as  $\psi$  may always be the same as  $\phi$  in all states after  $\phi$  becomes true. Even if only one fact about the (stable) property of interest can ever be true in an execution, this stable fact can be trivially viewed as being monotonic. Thus, the stable fact “the system is deadlocked since it entered deadlock in state  $s$ ” is (trivially) monotonic because  $\psi$  is always the same as  $\phi$  and there is only one fact  $\phi$  that is true about this deadlock property in this execution. However, the intention of defining monotonic facts is to highlight the semantic inclusion relationship among multiple stable facts about the same property, that can become true in an execution.

From Axiom 1, we can treat the semantically greatest fact in a state as synonymous to that state, and identify the fact by the vector timestamp of that state. As an example for a vector timestamp protocol, for any execution, if fact  $\phi$  is defined to be “the execution has reached at least a state with vector timestamp  $T_\phi$ ”, then fact  $\phi$  is monotonic, i.e., for  $c \subseteq c'$ , if we have that  $(a, c) \models T^1(c)$  then we have (i)  $(a, c') \models T^1(c')$ , (ii)  $(a, c') \models [T^1(c) \sqsubseteq T^1(c')]$ , and (iii)  $T^1(c')$  is the greatest value of the timestamp up to which the execution has reached in cut  $c'$ . Similarly for a matrix timestamp protocol. Observe that the local clock components of both vector and matrix clocks, i.e.,  $T_i^1[i]$  and  $T_i^2[i, i]$ , are also monotonic.

In order to study the power of logical clock abstraction, we use the formalism of knowledge in the system. We first define *k-bounded knowledge*, and then clock protocols that use such knowledge. The power of such protocols in attaining knowledge about monotonic facts is then studied.

### 3 k-bounded knowledge

Based on Axiom 1, the applications mentioned in Sect. 1 represented facts about some monotonic property of interest, which are a function of the global state, by the timestamp of the global state. Vector clocks provide knowledge of a global monotonic fact  $\phi$ , equivalent to knowledge  $E^0(\phi)$ , for the application domain. Matrix clocks which contain vector clock information about other processes’ views of the global state provide knowledge  $E^1(\phi)$  about global monotonic facts  $\phi$  in the application domain.

To abstract the properties of vector and matrix clocks, we examine the feasibility of and mechanisms for achieving levels of knowledge  $E^k(\phi)$  ( $k > 1$ ) using *inhibition-free ambient message-passing*, and satisfying the five observations made about vector and matrix clocks in Sect. 1. In a nutshell,

1. *No protocol messages* can be used. Protocol information may be piggybacked on messages of the underlying execution. The latest knowledge about the (past) execution should be continually *diffused* as much as possible, using only the messages of the underlying execution, whenever sent.
2. *No inhibition* of messages or of events is allowed.

We now define *k-bounded knowledge* to formulate logical clock abstractions.

**Definition 9 (k-bounded knowledge)** At any process  $j$ ,

- *k-bounded knowledge* about a fact  $\phi$  is any knowledge of the form  $K_j(K_{i_1}(K_{i_2} \dots (K_{i_k}(\phi)) \dots))$ , where  $i_1, i_2, \dots, i_k \in N$ .
- *complete k-bounded knowledge* about a fact  $\phi$  is knowledge  $\{K_j(K_{i_1}(K_{i_2} \dots (K_{i_k}(\phi)) \dots))\}$ , for all  $i_1, i_2, \dots, i_k \in N$ .
- *complete k-bounded knowledge* about a property  $\Phi$  is knowledge  $\{K_j(K_{i_1}(K_{i_2} \dots (K_{i_k}(\phi_{i_1, i_2, \dots, i_k})) \dots))\}$ , for all  $i_1, i_2, \dots, i_k \in N$ , where  $\phi_{i_1, i_2, \dots, i_k}$  is a fact about the property  $\Phi$  that process  $j$  knows that process  $i_1$  knows that  $\dots$  process  $i_k$  knows.

We make the following note on the notation  $\phi$  and  $\phi_{i_1, i_2, \dots, i_k}$ . The fact  $\phi$  about which *k-bounded knowledge*  $K_j(K_{i_1}K_{i_2} \dots K_{i_{k-1}}K_{i_k}(\phi))$  exists at any process  $j$  is formally denoted as  $\phi_{i_1, i_2, \dots, i_k}$ . The subscripts may be omitted for simplicity. When there exists complete *k-bounded knowledge* of a fact  $\phi$ , it is not necessary to use any subscripts for  $\phi$  because the *k-bounded knowledge* of  $\phi$  exists for all instantiations of  $i_1, i_2, \dots, i_k$ . When there exists complete *k-bounded knowledge* of a property  $\Phi$ , *k-bounded knowledge* of separate facts  $\phi_{i_1, i_2, \dots, i_k}$  is known for all instantiations of  $i_1, i_2, \dots, i_k$ .

We adopt the convention that when  $\phi$  is a fact local to a process  $i_k$ , then  $K_j(K_{i_1}K_{i_2} \dots K_{i_{k-1}}K_{i_k}(\phi_{i_k}))$  is  $(k - 1)$ -bounded knowledge, and not *k-bounded knowledge*. This is justified because  $\phi_{i_k}$  implies  $K_{i_k}(\phi_{i_k})$  by the Knowledge Generalization Rule for local facts [6].

Observe that *k-bounded knowledge*  $K_j(K_{i_1}K_{i_2} \dots K_{i_{k-1}}K_{i_k}(\phi))$  implicitly includes all lower levels of knowledge when two or more adjacent indices in  $[i_1, i_2, \dots, i_k]$  are instantiated similarly.

To formalize the condition that *k-bounded knowledge* can only be piggybacked on messages of the underlying execution in order to propagate it, we define *TIPP<sub>k</sub>* protocols.

**Definition 10** (*TIPP<sub>k</sub> protocol*) A  $k$ -bounded timestamp information piggybacking protocol (*TIPP<sub>k</sub>*) is an inhibition-free protocol such that on messages of the underlying execution, complete  $k$ -bounded knowledge<sup>1</sup> about the property “vector timestamp of the global state” is piggybacked by the sender and no protocol messages are sent.

The vector clock protocol is a *TIPP<sub>0</sub>* protocol. The matrix clock protocol is a *TIPP<sub>1</sub>* protocol. We formulate a generalization of these clocks such that the clocks contain  $k$ -bounded knowledge of the vector timestamps of global states, and use a *TIPP<sub>k</sub>* protocol.

The vector timestamp of a global state is composed of the  $n$  independent local components of the clock value of each process  $i$ . The local time at a process that captures the progression of local events is an important requirement of all clock definitions and is formalized as follows.

**Definition 11** (*Local component of a clock value*) The local component of a clock value at process  $i$  is the monotonically increasing component of the local clock  $Clk_i$  that is independent of the contents of any message received and that depends only on the occurrence of local events.

- The acronym *LCCV* is used when referring to this component as a property.
- The acronym *L.C.C.V.* is used when referring to a fact about this property, (i.e., when referring to a specific value of this component).

A subscript on these acronyms denotes the process to which this clock component belongs.

For vector clocks, the  $LCCV_i$  is  $Clk[i]$ . For matrix clocks, the  $LCCV_i$  is  $Clk[i, i]$ .

From the monotonicity of  $LCCV_i$ , we have that  $k$ -bounded knowledge of  $L.C.C.V._i$  implies  $k$ -bounded knowledge of all  $L.C.C.V._i^t$  such that  $L.C.C.V._i^t < L.C.C.V._i$ . Assuming  $k$ -bounded knowledge about vector timestamps is piggybacked on all messages, we now examine the relation between the knowledge that can be attained and message chains.

As events in an execution occur,  $k$ -bounded knowledge about the *LCCV*s can change as processes learn more recent information. The following remark, which also follows from Chandy and Misra’s Knowledge Gain Theorem [2], specifies the message chain that must exist in order for the  $k$ -bounded knowledge of any specific local fact, such as the *L.C.C.V.* at an event, to be learnt by another process. The “if” part follows from the use of the *TIPP<sub>k</sub>* protocol.

**Remark 1** In a system using a *TIPP<sub>k</sub>* protocol, process  $i_0$  has knowledge  $K_{i_0}(K_{i_1}K_{i_2} \dots K_{i_k}(\phi_{i_k}))$  after event  $e_{i_0}^x$ , where  $\phi_{i_k}$  is the *L.C.C.V.* <sub>$i_k$</sub> , if and only if there exists a chain of messages  $[m_{i_k}, m_{i_{k-1}}, m_{i_{k-2}}, \dots, m_{i_2}, m_{i_1}]$  such that

1. process  $i_k$  sends  $m_{i_k}$  at the time or after  $LCCV_{i_k}$  is  $\phi_{i_k}$ ,
2.  $m_{i_j}$  is sent by process  $i_j$  to process  $i_{j-1}$ , for  $k \geq j \geq 1$
3.  $receive(m_{i_j}) \prec send(m_{i_{j-1}})$  at process  $i_{j-1}$ , for  $k \geq j > 1$ , and
4.  $receive(m_{i_1}) \preceq e_{i_0}^x$ .

<sup>1</sup> Using this knowledge, the underlying execution can infer monotonic facts about some property of interest to it.

Process  $i_0$  has complete  $(k - 1)$ -bounded knowledge of fact  $\phi_{i_k}$  if message chains such as the above exist for all  $i_1, i_2, \dots, i_{k-1} \in N$ . In this case, we can see from Proposition 1 that process  $i_0$  has knowledge  $E^{k-1}(\phi)$  after event  $e_{i_0}^x$ , i.e.,  $s_{i_0}^x \models K_{i_0}(E^{k-1}(\phi_{i_k}))$ . Proposition 2 is the converse of Proposition 1. The proofs of these propositions are given in the Appendix.

**Proposition 1** A process  $i$  that attains complete  $k$ -bounded knowledge about a fact  $\phi_{i_k}$  that is the *L.C.C.V.* <sub>$i_k$</sub>  also attains knowledge  $K_i(E^k(\phi_{i_k}))$ .

**Proposition 2** A process  $i$  that attains knowledge  $K_i E^k(\phi_{i_k}^w)$ , where  $\phi_{i_k}^w$  is the *L.C.C.V.* <sub>$i_k$</sub> , also attains complete  $k$ -bounded knowledge of that *L.C.C.V.* <sub>$i_k$</sub> .

The following corollary follows from Propositions 1 and 2.

**Corollary 1** A process that attains knowledge  $E^k(\phi)$ , where  $\phi$  is the *L.C.C.V.* <sub>$i_k$</sub> , attains complete  $k$ -bounded knowledge of that *L.C.C.V.* <sub>$i_k$</sub> , and vice versa.

Propositions 1 and 2, and Corollary 1 also hold when fact  $\phi$  is the timestamp of a global state, as opposed to the *L.C.C.V.* <sub>$i_k$</sub> , by observing that  $\phi$  can now be treated as  $n$  independent facts, each of which is the *L.C.C.V.* of a distinct process.

We now define the notion of “greatest knowledge” about monotonic properties.

Given complete  $(k - 1)$ -bounded knowledge about the property “*LCCV<sub>i\_k</sub>*”, then  $\phi = \min_{i_1, i_2, \dots, i_{k-1} \in N}(\phi_{i_1, i_2, \dots, i_k})$  is the latest local timestamp at  $i_k$  about which complete  $(k - 1)$ -bounded knowledge is available to  $i_0$ .

**Definition 12** (*Up to date knowledge*) Up to date knowledge  $E^k(\phi)$  about a monotonic property  $\Phi$  is the knowledge  $E^k$  about the semantically greatest (i.e., most recent possible) fact  $\phi$  about  $\Phi$  about which  $E^k$  knowledge can be attained.

**Corollary 2** Up to date knowledge  $E^k(\phi)$ , where  $\phi$  is a value of (monotonic) property  $\Phi$ , is attained by a process if and only if a process attains complete  $k$ -bounded knowledge about the fact  $\phi$ , where  $\phi$  is the greatest value of (monotonic) property  $\Phi$  about which complete  $k$ -bounded knowledge can be attained by the process.

*Proof.* Observe that Corollary 1 can be generalized to facts about any monotonic property. The proof follows by combining this generalization of Corollary 1 with Definition 12.  $\square$

Let  $E^k(\phi_{i_k}^w)$  be true, where  $\phi_{i_k}^w$  is the *L.C.C.V.* <sub>$i_k$</sub>  for event  $e_{i_k}^w$ . From Corollary 2, up to date knowledge  $E^k(\phi_{i_k}^w)$  implies that  $\phi_{i_k}^w$  is the greatest value of *LCCV<sub>i\_k</sub>* such that complete  $k$ -bounded knowledge of  $\phi_{i_k}^w$  is also attained. From Remark 1, there are message chains  $[m_{i_k}, m_{i_{k-1}}, m_{i_{k-2}}, \dots, m_{i_2}, m_{i_1}]$  from  $e_{i_k}^w$  to process  $i_0$ , for all instantiations of  $i_1, i_2, \dots, i_{k-1}$ .

To study the relationship between the size of logical clocks and levels of knowledge about monotonic facts in an asynchronous distributed message-passing system, we formulate the following problems in the context of a *TIPP* protocol; an overview of the results is given in Table 1 in Sect. 5.

**Problem 1** In a system using a *TIPP* protocol, what are the necessary and sufficient conditions on the clock information required for a process to attain up to date knowledge  $E^k(\phi)$ , where  $\phi$  is the timestamp of the (most recent possible) global state?

**Problem 2** Given a timestamp  $\phi$  of any global state, what is the timestamp of the earliest global state in which  $E^k(\phi)$  is attained by each process in a system using a TIPP protocol?

**Problem 3** Given a timestamp of a global state, what is the timestamp  $\phi$  of the (most recent possible) global state about which up to date knowledge  $E^k(\phi)$  can be attained in the given state in a system using a TIPP protocol?

The homomorphism from  $(\mathcal{T}^1, <)$  to  $(\mathcal{M}, \sqsubseteq)$ , which was captured by Axiom 1, allows us to equate monotonic facts with global states in which they are defined and with timestamps of such states. Hence, solutions to the above problems can be directly used by applications to reason about monotonic facts of interest.

#### 4 Clocks of arbitrary dimensions

Matrix clocks provide more information than vector clocks which in turn provide more information than scalar clocks. This section (i) defines clocks of higher dimensions, (ii) gives a protocol to implement such clocks and proves its correctness, and (iii) studies properties of timestamps assigned using such clocks. The results of this section are used in the next section to analyze the knowledge that can be stored by such clocks.

##### 4.1 Clock definition and notations

**Definition 13 (A generic clock requirement)** An  $\alpha$ -dimensional clock defines the mapping from  $H$  to the latest complete  $(\alpha - 1)$ -bounded knowledge of monotonic  $LCCV_j$  ( $\forall j$ ), known at each event in  $H$ .

As knowledge is known only by a process and the clock is stored at each process, it is reasonable to define  $Clk$  as a function of the event set  $H$  rather than of the set of cuts  $Cuts$ . Timestamps of cuts in  $Cuts$  that are not of the form  $\downarrow e$  will be defined later in Sect. 4.3 using the more elementary timestamps of events. The following definition shows a way to realize the above requirement.

**Definition 14 (Multi-dimensional clock)** An  $\alpha$ -dimensional clock  $Clk^\alpha$  defines the mapping:  $H \mapsto (Z^*)^{\alpha n}$ , i.e.,  $Clk^\alpha$  is an  $\alpha$ -dimensional array of integers, where each dimension is of size  $n$ , satisfying the following properties.

- SP1. Complete  $(\alpha - 1)$ -bounded knowledge of the initial value of  $Clk_j^\alpha[j, \dots, j]$ , the local clock component at process  $j$  ( $LCCV_j$ ), is known to all the processes in the initial state  $H^0$ , i.e.,  $(a, H^0) \models E^\alpha(Cl k_j^\alpha[j, \dots, j])$ .
- SP2.  $Clk_j^\alpha[j, \dots, j]$ , the local clock component at process  $j$  ( $LCCV_j$ ), must be incremented by a positive integer when an event occurs at  $j$ .
- SP3. Any element  $Clk^\alpha(e_j)[i_1, i_2, \dots, i_\alpha]$  is the maximum local clock component  $\phi_{i_\alpha} = Clk_{i_\alpha}^\alpha[i_\alpha, i_\alpha, \dots, i_\alpha]$  at  $i_\alpha$  (i.e., the maximum l.c.c.v.  $i_\alpha$ ) such that  $K_j(K_{i_1}K_{i_2} \dots K_{i_\alpha}(\phi_{i_\alpha}) \dots)$ .

Observe that  $Clk^\alpha(e_j)[i_1, i_2, \dots, i_\alpha]$  is the local clock component of event  $F_{i_\alpha}(\dots \downarrow F_{i_3}(\downarrow F_{i_2}(\downarrow F_{i_1}(\downarrow e_j)))) \dots$ .

The value of  $Clk^\alpha$  assigned as a timestamp is denoted  $T^\alpha$ . We now describe the notations.

$T^\alpha[i]$ , alternately represented as  $T^\alpha[i, \cdot]$ , is a timestamp of dimension  $(\alpha - 1)$  and is derived from  $T^\alpha$  by instantiating the first dimension variable  $i_1$  by  $i$ .  $T^\alpha(e_p)[i, \cdot]$  is the  $(\alpha - 1)$  dimensional timestamp of the most recent event at process  $i$ , as known to process  $p$  after event  $e_p$ . Moreover, this most recent event at process  $i$  has a scalar timestamp  $T^\alpha(e_p)[i, i, \dots, i]$ . In terms of knowledge,  $T^\alpha(e_p)[i, \cdot]$  at process  $p$  represents the knowledge  $s_p^x \models \bigwedge_{i_2, i_3, \dots, i_\alpha \in N} K_p(K_{i_1}K_{i_2}K_{i_3} \dots K_{i_\alpha}(\phi_{i_2, i_3, \dots, i_\alpha}))$ .

The timestamp  $T^\alpha[\underbrace{a, b, c, \dots, f}_{\beta \text{ entries, } \alpha \geq \beta \geq 0}]$ , alternately represented as  $T^\alpha[\underbrace{a, b, c, \dots, f}_{\beta \text{ entries, } \alpha \geq \beta \geq 0}, \cdot]$ , is a timestamp of dimension

$(\alpha - \beta)$ . It is derived from  $T^\alpha$  by instantiating the first  $\beta$  dimension variables  $i_1, \dots, i_\beta$  by  $a, b, c, \dots, f$ , respectively. In terms of knowledge,  $T^\alpha(e_p)[a, b, c, \dots, f, \cdot]$  at process  $p$  represents the knowledge  $s_p^x \models \bigwedge_{i_{\beta+1}, \dots, i_\alpha \in N} K_p(K_aK_bK_c \dots K_fK_{i_{\beta+1}}K_{i_{\beta+2}} \dots K_{i_\alpha}(\phi_{a, b, \dots, f, i_{\beta+1}, \dots, i_\alpha}))$ .

Using  $\alpha$ -dimensional clocks, each event  $e_i$  can be assigned timestamp  $T^\beta$ , where  $0 \leq \beta \leq \alpha$ . The timestamp  $T^\beta(e_i)$  is  $T^\alpha(e_i)[\underbrace{i, i, \dots, i}_{\alpha - \beta \text{ times}}, \cdot]$ , where  $\cdot$  represents a  $\beta$ -dimensional timestamp. Given  $T^\alpha(e_i^x)$ , the l.c.c.v.  $i$  of event  $e_i^x$  is  $T^\alpha(e_i^x)[i, i, \dots, i]$ .  $T^\alpha(e_i^x)[i, i, \dots, i, j] = T^\alpha(e_i^x)[i, j, \dots, j]$  is the l.c.c.v. of the latest event at process  $j$ , as known to process  $i$  in state  $s_i^x$ .

From Definition 14, observe that  $Clk^\alpha$  contains complete  $(\alpha - 1)$ -bounded knowledge of the  $n$  local monotonic properties “ $LCCV_i$ ”, for all  $i$  in  $N$ , as required in a TIPP protocol.  $Clk^\alpha$  equivalently contains  $(\alpha - 1)$ -bounded knowledge of the monotonic property “timestamp of a global state”. Also observe from (SP3) that there is some replication of knowledge in  $Clk_i^\alpha$ . Whenever there is at least one instance of two or more adjacent indices in  $Clk_i^\alpha[i_1, i_2, \dots, i_\alpha]$  being instantiated by the same process, the knowledge corresponds to a message chain of less than  $\alpha$  messages. There will exist “replicated instances of this knowledge” in several positions in  $Clk_i^\alpha$ , each of which corresponds to an equivalent instantiation of the variables  $i_1, \dots, i_\alpha$ . An in-depth analysis is performed in Sect. 5.1.

##### 4.2 Clock protocols

An implementation of the  $\alpha$ -dimensional clock specification of Definition 14 is given by the rules in Fig. 1.

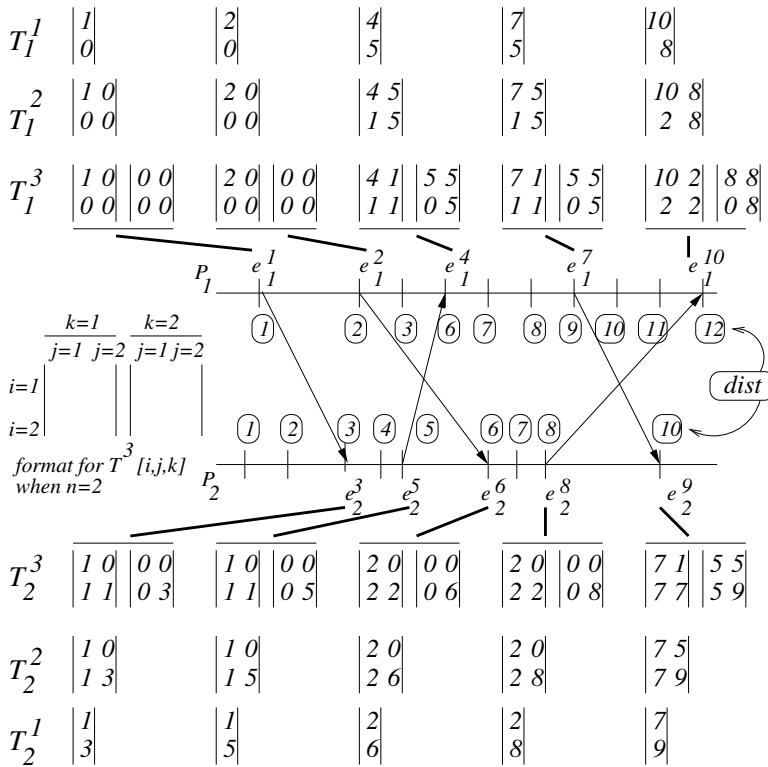
**Example:** We consider the execution in Fig. 2 to illustrate the clock notations and the operation of the clock protocol of Fig. 1. The vector, matrix, and 3-dimensional timestamps of all send and receive events are shown in the figure. The notations used in Sect. 4.1 are illustrated in Fig. 3. These illustrations will be used subsequently as we continue to use this example execution to illustrate further concepts.

**Theorem 1** The protocol in Fig. 1 implements the  $\alpha$ -dimensional clock specification of Definition 14.

*Proof.* In the initial global state,  $Clk_i^\alpha[i, \dots, i] = 0$  and complete  $(\alpha - 1)$ -bounded knowledge of  $Clk_i^\alpha[i, \dots, i]$  at all pro-

- R0. Initialization:  $Clk_i^\alpha = \alpha$ -dimensional 0-vector.
- R1. Before process  $i$  executes an internal event, it does the following.  
 $Clk_i^\alpha[i, i, \dots, i] = Clk_i^\alpha[i, i, \dots, i] + d \quad (d > 0)$ .
- R2. Before process  $i$  executes a send event, it does the following:  $Clk_i^\alpha[i, i, \dots, i] = Clk_i^\alpha[i, i, \dots, i] + d \quad (d > 0)$ .  
 Send message timestamped with  $Clk_i^\alpha$ .
- R3. When process  $j$  receives a message with timestamp  $T^\alpha$  from process  $i$ , it does the following.
1. **for**  $\beta = 1$  **to**  $\alpha - 1$  **do**  
 $\forall q_1 \in N \setminus \{j\}, \forall q_2, q_3, \dots, q_\beta \in N$ ,  
 $Clk_j^\alpha[\underbrace{j, \dots, j}_{\alpha-\beta \text{ times}}, \underbrace{q_1, q_2, \dots, q_\beta}_{\beta \text{ entries}}] = \max(Clk_j^\alpha[\underbrace{j, \dots, j}_{\alpha-\beta \text{ times}}, \underbrace{q_1, q_2, \dots, q_\beta}_{\beta \text{ entries}}, T^\alpha[\underbrace{i, \dots, i}_{\alpha-\beta \text{ times}}, \underbrace{q_1, q_2, \dots, q_\beta}_{\beta \text{ entries}}])$ ;
  2.  $\forall q_1 \in N \setminus \{j\}, \forall q_2, \dots, q_\alpha \in N$ ,  
 $Clk_j^\alpha[q_1, q_2, \dots, q_\alpha] = \max(Clk_j^\alpha[q_1, q_2, \dots, q_\alpha], T^\alpha[q_1, q_2, \dots, q_\alpha])$ ;
  3.  $Clk_j^\alpha[j, j, \dots, j] = Clk_j^\alpha[j, j, \dots, j] + d \quad (d > 0)$ ;
  4. Deliver the message.

**Fig. 1.** Protocol to update  $\alpha$ -dimension clocks



**Fig. 2.** The example execution for  $Clk^3$  (3-dimensional clocks) and  $n = 2$  processes

cesses  $j$  is also 0, as observed from rule (R0). It follows that specification (SP1) is satisfied by rule (R0) of the protocol.

Specification (SP2) is seen to be satisfied by the increment operations in rules (R1), (R2), and (R3.3) of the protocol for internal events, send events, and receive events, respectively.

We prove that specification (SP3) is satisfied by using induction on the “distance” of an event from the set of initialization events. This distance is the same as Lamport’s well-known scalar clock value at an event [15] when the local increment at each event is one. The term “distance” is used to capture the notion of the length of the longest path from any initial event to the event under consideration.

**Definition 15 (Distance of an event)** For any execution, the distance function  $dist(e_i^x)$  on the events is as follows.

if  $x = 0$  then  $dist(e_i^x) = 0$

if  $x > 0$  and  $e_i^x$  is an internal or a send event then  $dist(e_i^x) = dist(e_i^{x-1}) + 1$

if  $x > 0$  and the message sent at  $e_s^k$  is received at  $e_i^x$  then  $dist(e_i^x) = \max(dist(e_i^{x-1}), dist(e_s^k)) + 1$

**Example:** The distances of the events in the example of Fig. 2 are indicated therein by encircled values.

**Induction hypothesis:** For event  $e_i^x$  at distance  $dist(e_i^x)$ ,  $Clk_i^\alpha(e_i^x)[i_1, i_2, \dots, i_\alpha]$  indicates the maximum *l.c.c.v.*  $i_\alpha$   $\phi_{i_\alpha} = Clk_{i_\alpha}^\alpha[i_\alpha, i_\alpha, \dots, i_\alpha]$  such that  $K_i(K_{i_1}K_{i_2}K_{i_3} \dots K_{i_\alpha}(\phi_{i_\alpha}) \dots)$ .

**Case**  $dist(e_i^x) = 0$ : Event  $e_i^x$  is an initial event. As (SP1) is satisfied in the initial state, it follows that  $Clk_i^\alpha[i_1, \dots, i_\alpha]$ , which is 0 in this state, is the maximum *l.c.c.v.*  $i_\alpha = \phi_{i_\alpha}$  such

1.  $T^3(e_2^9)[1, \cdot] = T^3(e_2^9)[1] = \begin{bmatrix} 7 & 5 \\ 1 & 5 \end{bmatrix}$ . This is the matrix time-stamp of event  $e_1^7$  which is  $F_1(\downarrow e_2^9)$ .
2.  $T^3(e_2^9)[2, \cdot] = T^3(e_2^9)[2] = \begin{bmatrix} 7 & 5 \\ 7 & 9 \end{bmatrix}$ . This is the matrix time-stamp of event  $e_2^9$  which is  $F_2(\downarrow e_2^9)$ .
3.  $T^3(e_1^{10})[1, \cdot] = T^3(e_1^{10})[1] = \begin{bmatrix} 10 & 8 \\ 2 & 8 \end{bmatrix}$ . This is the matrix time-stamp of event  $e_1^{10}$  which is  $F_1(\downarrow e_1^{10})$ .
4.  $T^3(e_1^{10})[2, \cdot] = T^3(e_1^{10})[2] = \begin{bmatrix} 2 & 0 \\ 2 & 8 \end{bmatrix}$ . This is the matrix time-stamp of event  $e_2^8$  which is  $F_2(\downarrow e_1^{10})$ .
5.  $T^3(e_2^9)[1, 2, \cdot] = T^3(e_2^9)[1, 2] = \begin{bmatrix} 1 \\ 5 \end{bmatrix}$ . This is the vector time-stamp of event  $e_2^5$  which is  $F_2(\downarrow F_1(\downarrow e_2^9))$ .
6.  $T^3(e_2^9)[2, 1, \cdot] = T^3(e_2^9)[2, 1] = \begin{bmatrix} 7 \\ 5 \end{bmatrix}$ . This is the vector time-stamp of event  $e_1^7$  which is  $F_1(\downarrow F_2(\downarrow e_2^9))$ .

**Fig. 3.** Illustration of notations and definitions used for the clock protocol of Fig. 1, based on the example in Fig. 2

that  $K_i(K_{i_1} \dots K_{i_\alpha}(\phi_{i_\alpha}) \dots)$  and  $\phi_{i_\alpha}$  is also 0 in this state. Hence (SP3) is satisfied.

*Case  $\text{dist}(e_i^x) = 1$ :* Observe that in this case,  $x$  must be 1 and  $e_i^x$  must be a send event or an internal event. At this event, there is no gain in knowledge beyond what is known in  $s_i^{x-1}$  except that  $i$  learns of the (local) internal or send event  $e_i^1$ . Note that  $\text{dist}(e_i^0) = 0$  and therefore  $\text{Clk}^\alpha(e_i^0)$  satisfies the specification (SP3). To reflect the gain in knowledge about the new event  $e_i^1$ , only  $\text{Clk}_i^\alpha[i, i, \dots, i]$  needs to be updated at  $e_i^1$ . This happens by rule (R1) or (R2). Hence (SP3) is satisfied.

*Case  $\text{dist}(e_i^x) = p$ :* Assume that the induction hypothesis holds for  $\text{dist}(e_i^x) = p$ .

*Case  $\text{dist}(e_i^x) = p+1$ :* If  $e_i^x$  is a send event or an internal event, there is no gain in knowledge beyond what is known in  $s_i^{x-1}$  after  $e_i^{x-1}$ , except that  $i$  learns of the (local) internal or send event  $e_i^x$ . Note that  $\text{dist}(e_i^{x-1}) \leq p$  and therefore  $\text{Clk}^\alpha(e_i^{x-1})$  satisfies the specification (SP3). To reflect the gain in knowledge about the new event  $e_i^x$ , only  $\text{Clk}_i^\alpha[i, i, \dots, i]$  needs to be updated at  $e_i^x$ . This happens by rule (R1) or (R2). Hence (SP3) is satisfied when  $e_i^x$  is a send event or an internal event.

If  $e_i^x$  is a receive event of a message sent at  $e_s^k$ , there is no gain in knowledge beyond what is known in  $s_i^{x-1}$  after  $e_i^{x-1}$ , except for (i) what is learnt from the timestamp  $T_s^\alpha(e_s^k)$  of the message, and (ii) the occurrence of the receive event itself.

$T_s^\alpha(e_s^k)$  is the clock value  $\text{Clk}_s^\alpha(e_s^k)$ . Observe that  $\text{dist}(e_i^{x-1}), \text{dist}(e_s^k) \leq p$  and hence both  $\text{Clk}^\alpha(e_i^{x-1})$  and  $\text{Clk}^\alpha(e_s^k)$  satisfy specification (SP3) by the induction hypothesis. ....(1)

To ensure that the knowledge gained by (1(ii)) above is reflected in  $\text{Clk}^\alpha(e_i^x)$  in the protocol, observe that rules (R3.1) and (R3.2) update each entry  $\text{Clk}_i^\alpha[i_1, \dots, i_\alpha]$  representing knowledge  $K_i(K_{i_1}, \dots, K_{i_k}(\phi_{i_k}))$  with the corresponding entry  $T_s^\alpha[i_1, \dots, i_\alpha]$  representing knowledge  $K_s(K_{i_1}, \dots, K_{i_k}(\phi_{i_k}))$ , systematically as follows. In (R3.1), the outer loop has  $(\alpha - 1)$  iterations. In iteration  $\beta$ ,  $1 \leq \beta \leq \alpha - 1$ ,  $(\beta - 1)$ -bounded knowledge in  $\text{Clk}_i^\alpha$ , corresponding to

message chains of size  $\beta$  and less, is updated with the corresponding  $(\beta - 1)$ -bounded knowledge in  $T^\alpha(e_s^k)$ . In (R3.2), the  $(\alpha - 1)$ -bounded knowledge in  $\text{Clk}_i^\alpha$ , corresponding to message chains of size  $\alpha$  and less, is updated with the corresponding  $(\alpha - 1)$ -bounded knowledge in  $T^\alpha(e_s^k)$ . (Note that “replicated instances of the same knowledge” in the clock get updated in different iterations.) .....(3)

To ensure that the knowledge gained by (1(ii)) above is reflected in  $\text{Clk}^\alpha(e_i^x)$  in the protocol, only  $\text{Clk}_i^\alpha[i, i, \dots, i]$  needs to be updated at  $e_i^x$ . This happens by rule (R3.3). .....(4)

Claims (3) and (4) above imply that the knowledge gained by (1)((i) and (ii)) is reflected in  $\text{Clk}^\alpha(e_i^x)$  in the protocol. Combining this with (2) that the knowledge gained by claim (1) is correct, it follows that (SP3) is satisfied for the receive event  $e_i^x$ .  $\square$

The protocol in Fig. 1 which implements the  $\alpha$ -dimensional clock specification of Definition 14 has an asymptotically tight space complexity  $\theta(n^\alpha)$  at each process and an asymptotically tight time complexity  $\theta(n^\alpha)$  for each receive event. As will be shown in Sect. 5.1, even if “replicated instances of knowledge” in the clocks were eliminated, the complexity would still be  $\theta(n^\alpha)$ . Still, this protocol can be simplified as follows. Rule (R3.1) can be simplified by observing that the entire  $\text{Clk}_j^\alpha[j, \dots, j, q_1, \dots, q_1]$  needs to be replaced by  $T^\alpha[i, \dots, i, q_1, \dots, q_1]$  if and only if the timestamp on the incoming message indicates a more recent state of  $q_1$  than does  $\text{Clk}_j^\alpha$ . Similarly for rule (R3.2). The simplified rule (R3), relabeled (R3'), is shown below. The correctness proof is a straightforward modification of that of Theorem 1 and is left to the reader.

**R3'.** When process  $j$  receives a message with timestamp  $T^\alpha$  from process  $i$ , it does the following.

1. **for**  $\beta = 1$  **to**  $\alpha - 1$  **do**  
 $\forall q_1 \in N \setminus \{j\}$ ,  
**if**  $T^\alpha[\underbrace{i, \dots, i}_{\alpha-\beta \text{ times}}, \underbrace{q_1, q_1, \dots, q_1}_{\beta \text{ entries}}] > \text{Clk}_j^\alpha[\underbrace{j, \dots, j}_{\alpha-\beta \text{ times}}, \underbrace{q_1, q_1, \dots, q_1}_{\beta \text{ entries}}]$   
**then**  $\text{Clk}_j^\alpha[\underbrace{j, \dots, j}_{\alpha-\beta \text{ times}}, q_1, \dots, q_1] = T^\alpha[\underbrace{i, \dots, i}_{\alpha-\beta \text{ times}}, q_1, \dots, q_1]$ ;
2.  $\forall q_1 \in N \setminus \{j\}$ ,  
**if**  $T^\alpha[q_1, q_1, \dots, q_1] > \text{Clk}_j^\alpha[q_1, q_1, \dots, q_1]$   
**then**  $\text{Clk}_j^\alpha[q_1, \dots, q_1] = T^\alpha[q_1, \dots, q_1]$ ;
3.  $\text{Clk}_j^\alpha[j, j, \dots, j] = \text{Clk}_j^\alpha[j, j, \dots, j] + d \quad (d > 0)$ ;
4. Deliver the message.

Although the size of each clock and timestamp of dimension  $\alpha$  is  $n^\alpha$  integers, this can be reduced by making certain assumptions on the message communication pattern, the partial order  $(H, <)$ , logical network topology, and on the reliable and ordered message delivery, using schemes similar to those used by Singhal and Kshemkalyani [22] and by Meldal, Sankar, and Vera [17]. Certain optimizations of matrix clocks that do not compute the accurate matrix clock but some approximation of it were described by Krishnakumar and Bernstein [12], Ruget [19], Torres-Rojas and Ahamad [23], and Wu and Bernstein [24]. Similar techniques can be used for  $\alpha$ -dimensional clocks if accuracy can be sacrificed. However, for maintaining accurate clocks without making any simplifying assumptions, the

order of magnitude of the size of clocks and the piggybacked timestamp on messages will be shown to remain  $\theta(n^\alpha)$ .

**Remark 2** Each element of the  $\alpha$ -dimensional clock at any process is monotonically nondecreasing. As a result,  $K_p(Clk_p^\alpha) \implies K_p(T_p^\alpha(e_p))$  for all previous local events  $e_p$ .

Remark 2 implies that all previous clock values at a process continue to be known by that process.

The following remark follows from Chandy and Misra's Knowledge Gain Theorem [2] and the clock protocol of Fig. 1. Its analog for vector clocks is well known [16].

**Remark 3** There exists a message chain from event  $e_i^x$  to event  $e_j^y$  in the execution  $(H, \prec)$  if and only if  $T^\alpha(e_j^y)[i, i, \dots, i] \geq T^\alpha(e_i^x)[i, i, \dots, i]$ .

### 4.3 Timestamps of cuts and their properties

If a cut  $Cut$  equals  $\downarrow e$ , then  $T^\alpha(Cut) = T^\alpha(\downarrow e) = T^\alpha(e)$ . The timestamp of a cut that is not expressible as  $\downarrow e$  needs to be defined explicitly. The  $\alpha$ -dimensional timestamp of a cut is defined using the  $(\alpha - 1)$ -dimensional timestamp of the latest event at each process in that cut.

**Definition 16 (Multi-dimensional timestamp of a cut)** For  $\alpha \geq 1$ , the  $\alpha$ -dimensional timestamp of a cut is defined as  $T^\alpha(Cut) =_{def} (i \in N) T^\alpha(Cut)[i, \cdot] = T^\alpha(F_i(Cut))[i, \cdot]$ .

**Example:** In the example of Fig. 2, let us compute  $T^3(Cut)$ , where  $Cut$  is defined by its vector timestamp  $T^1(F(Cut)) = \begin{bmatrix} 10 \\ 9 \end{bmatrix}$ . We have from Definition 16,

- $T^3(Cut)[1, \cdot] = T^3(F_1(Cut))[1, \cdot] = T^3(e_1^{10})[1, \cdot] = \begin{bmatrix} 10 & 8 \\ 2 & 8 \end{bmatrix}$  (from Fig. 3).
- $T^3(Cut)[2, \cdot] = T^3(F_2(Cut))[2, \cdot] = T^3(e_2^9)[2, \cdot] = \begin{bmatrix} 7 & 5 \\ 7 & 9 \end{bmatrix}$  (from Fig. 3).

Composing  $T^3(Cut)[1, \cdot]$  and  $T^3(Cut)[2, \cdot]$  and adjusting positions to account for the new dimension, we have

$$T^3(Cut) = \begin{bmatrix} 10 & 2 \\ 7 & 7 \end{bmatrix} \begin{bmatrix} 8 & 8 \\ 5 & 9 \end{bmatrix}.$$

The following lemma gives a way to implement the test for Lemma 1. It will be used in Theorem 8 to identify the maximum timestamp  $\phi$  about which knowledge  $E^k(\phi)$  has been attained at a given cut.

**Lemma 2** The timestamp  $T^\alpha(\cap_\Psi X)$  is expressed as a function of the timestamps of the members of  $X$  as follows:  $(i \in N) T^\alpha(\cap_\Psi X)[i, \cdot]$  is the  $(\alpha - 1)$ -dimensional timestamp  $T^\alpha(x'_i)[i, \cdot]$ , where  $T^\alpha(x'_i)[i, i, \dots, i] = \min_{x \in X} (T^\alpha(x) - [i, i, \dots, i])$ .

*Proof.* From Definition 16, we have  $(i \in N) T^\alpha(\cap_\Psi X)[i, \cdot] = T^\alpha(F_i(\cap_\Psi X))[i, \cdot] = T^\alpha(F_i(\cap_{x \in X} \downarrow x))[i, \cdot]$ . As the intersection and projection operations on cuts are commutative, we therefore have that  $T^\alpha(F_i(\cap_{x \in X} \downarrow x))[i, \cdot] = T^\alpha(\min_{x \in X} (F_i(\downarrow x)))[i, \cdot]$ . ... (1)

From the definition of  $\downarrow x$  (Definition 2) and Specification (SP3) of Definition 14, observe that  $T^\alpha(x)[i, i, \dots, i] = T^\alpha(F_i(\downarrow x))[i, i, \dots, i]$ . ... (2)

By combining (2) above with the fact that timestamps of events at a process, including each component of the timestamps, are monotonically nondecreasing (follows from Remark 2), we have as follows:  $\min_{x \in X} (F_i(\downarrow x)) = x'_i$ , where  $T^\alpha(x'_i)[i, i, \dots, i] = \min_{x \in X} (T^\alpha(x)[i, i, \dots, i])$ . ... (3)

By combining (1) and (3) above, we have  $T^\alpha(\cap_\Psi X)[i, \cdot] = T^\alpha(x'_i)[i, \cdot]$ , where  $x'$  is such that  $T^\alpha(x'_i)[i, i, \dots, i] = \min_{x \in X} (T^\alpha(x)[i, i, \dots, i])$ .  $\square$

**Example:** In the example of Fig. 2, let  $X = \{e_1^{10}, e_2^9\}$ .  $T^3(\cap_\Psi X)$  is composed using  $T^3(\cap_\Psi X)[1, \cdot]$  and  $T^3(\cap_\Psi X)[2, \cdot]$ . Based on Lemma 2, these are computed as follows.

- $\min(T^3(e_1^{10})[1, 1, 1], T^3(e_2^9)[1, 1, 1]) = \min(10, 7) = 7$ . As  $T^3(e_1^7)[1, 1, 1] = 7$ , hence,  $T^3(\cap_\Psi X)[1, \cdot] = T^3(e_1^7)[1, \cdot] = \begin{bmatrix} 7 & 5 \\ 1 & 5 \end{bmatrix}$ . This is obtained from Fig. 2, by taking the first row of  $T^3(e_1^7) = \begin{bmatrix} 7 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 5 & 5 \\ 0 & 5 \end{bmatrix}$ , and adjusting positions to account for the reduced dimension.
- $\min(T^3(e_1^{10})[2, 2, 2], T^3(e_2^9)[2, 2, 2]) = \min(8, 9) = 8$ . As  $T^3(e_2^8)[2, 2, 2] = 8$ , hence,  $T^3(\cap_\Psi X)[2, \cdot] = T^3(e_2^8)[2, \cdot] = \begin{bmatrix} 2 & 0 \\ 2 & 8 \end{bmatrix}$ . This is obtained from Fig. 2, by taking the second row of  $T^3(e_2^8) = \begin{bmatrix} 2 & 0 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 8 \end{bmatrix}$ , and adjusting positions to account for the reduced dimension.

$T^3(\cap_\Psi X)[1, \cdot]$  and  $T^3(\cap_\Psi X)[2, \cdot]$  are now composed, and adjusting positions to account for the new dimension, we have  $T^3(\cap_\Psi X) = \begin{bmatrix} 7 & 1 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} 5 & 5 \\ 0 & 8 \end{bmatrix}$ .

The following corollary follows from Lemma 2 when  $\alpha$  is set to 1.

**Corollary 3** The timestamp  $T^1(\cap_\Psi X)$  can be expressed as a function of the timestamps of the members of  $X$  as follows:  $(i \in N) T^1(\cap_\Psi X)[i] = \min_{x \in X} (T^1(x)[i])$ .

**Example:** In the example of Fig. 2, let  $X = \{e_1^{10}, e_2^9\}$ .  $T^1(\cap_\Psi X)$  is composed using  $T^1(\cap_\Psi X)[1]$  and  $T^1(\cap_\Psi X)[2]$ . Based on Corollary 3, these are computed as follows.

- $T^1(\cap_\Psi X)[1] = \min(e_1^{10}[1], e_2^9[1]) = \min(10, 7) = 7$ .
- $T^1(\cap_\Psi X)[2] = \min(e_1^{10}[2], e_2^9[2]) = \min(8, 9) = 8$ .

$$\text{Hence, } T^1(\cap_\Psi X) = \begin{bmatrix} 7 \\ 8 \end{bmatrix}.$$

The following corollary follows from Lemma 2 and Corollary 3 by using the union instead of intersection operators and replacing the min function by the max function. The corollary will be used in the proof of Theorem 7.

**Corollary 4** The timestamp  $T^1(\cup_\Psi X)$  can be expressed as a function of the timestamps of the members of  $X$  as follows:  $(i \in N) T^1(\cup_\Psi X)[i] = \max_{x \in X} (T^1(x)[i])$ .

Recall from Sect. 2.1 that the cut  $\cap_\Psi X$  is the largest cut such that each event in it happens before each event in set  $X$ . The

cut  $\cup_{\downarrow} X$  is the smallest consistent cut that contains all the events in set  $X$ .

As each cut in an execution has a unique timestamp of a given dimension, henceforth, a cut will also be identified by its timestamp.

## 5 Clocks and their knowledge

We now show the main results that establish the conceptual link between clocks and knowledge, and answer Problems 1, 2, and 3. Table 1 gives an overview of the results.

### 5.1 Conditions on the size of clocks

**Lemma 3** *Representation of complete  $k$ -bounded knowledge about the property “vector timestamp of a global state” needs  $\sum_{j=-1}^k (n-1)^{j+1}$  integers.*

*Proof.* At process  $i$ ,  $k$ -bounded knowledge about a fact  $\phi$  is of the form  $K_i(K_{i_1}K_{i_2}K_{i_3}\dots K_{i_k}(\phi)\dots)$ . The vector timestamp  $\phi$  consists of  $n$  independent *l.c.c.v.s*, one for each process. For the complete  $k$ -bounded knowledge  $K_i(K_{i_1}K_{i_2}K_{i_3}\dots K_{i_k}(\phi_{i_{k+1}})\dots)$  at process  $i$  about the *LCCV* of any one process  $i_{k+1}$ , corresponding to a message chain of exactly  $k+1$  messages, we require

- $i_1 \neq i$ ,
- $\forall j \in [2, k-1], i_j \neq i_{j-1}$ , and
- $i_k \neq i_{k-1}$  and  $i_k \neq i_{k+1}$ .

This gives

$$(n-1)^{k-1} \left[ \frac{1}{n} \cdot (n-1) + \frac{(n-1)}{n} \cdot (n-2) \right] \\ = \frac{(n-1)^{k+1}}{n}$$

permutations on the average for a given  $i_{k+1}$ ; each permutation identifies the latest *l.c.c.v.*  $i_{k+1}$  about which  $k$ -bounded knowledge, corresponding to a chain of exactly  $k+1$  messages whose senders are ordered by the permutation, exists at process  $i$ . When  $i_{k+1}$  can be any of the  $n$  processes, the number of permutations becomes  $(n-1)^{k+1}$ . For complete  $k$ -bounded knowledge corresponding to message chains of all sizes from 1 to  $k+1$ , the number of permutations is  $\sum_{j=0}^k (n-1)^{j+1}$ . Additionally, a message chain of size 0 denotes knowledge about *LCCV* at process  $i$ . Assuming each *l.c.c.v.* can be represented by an integer, the total space complexity is  $\sum_{j=0}^k (n-1)^{j+1} + 1 = \sum_{j=-1}^k (n-1)^{j+1}$  integers. This geometric series sums to  $((n-1)^{k+2} - 1)/(n-2)$  if  $n > 2$ , and it sums to  $k+2$  if  $n = 2$ .  $\square$

We analyze the impact of the “replicated instances of knowledge” in a clock  $Clk^{k+1}$  of size  $n^{k+1}$  by first proving a theorem.

### Theorem 2

$$\sum_{j=-1}^k (n-1)^{j+1} \cdot \frac{(k+1)!}{(j+1)!(k-j)!} = n^{k+1}$$

*Proof.* Consider the complete  $k$ -bounded knowledge of *LCCVs* at  $i_0$ . For  $j$ -bounded knowledge of  $\phi_{i_{j+1}}$  at  $i_0$ , where  $j \leq k$ , corresponding to a message chain of exactly  $j+1$  messages, we identify the number of ways in which we can map an array  $B[0, 1, \dots, j, j+1]$  corresponding to knowledge  $K_{i_0}(K_{i_1}, \dots, K_{i_j}, K_{i_{j+1}}(\phi_{i_{j+1}})\dots)$  to an array  $A[0, 1, \dots, k, k+1]$  corresponding to knowledge  $K_{i_0}(K_{i_1}, \dots, K_{i_k}, K_{i_{k+1}}(\phi_{i_{k+1}})\dots)$ . For a specific process chain representing  $j$ -bounded knowledge, the mapping from  $B = [i_0, i_1, \dots, i_j, i_{j+1}]$  to  $A$  has the following constraints.

1. There is a string of  $q_w$  occurrences of  $i_w$  in array  $A$ , for each  $w \in [0, j+1]$ .
2. For each  $w \in [0, j+1]$ ,  $q_w \geq 1$  and  $\sum_{w=0}^{j+1} q_w = k+1$ .
3. For  $w = 0$ , the string of  $i_0$ s is from  $A[0]$  to  $A[q_0 - 1]$ .
4. For each  $w \in (0, j+1]$ , the string of  $i_w$ s is from  $A[\sum_{t=0}^{w-1} q_t]$  to  $A[\sum_{t=0}^w q_t - 1]$ .

For a given instantiation of  $i_0$  through  $i_j$ , the number of ways to map array  $B$  to array  $A$ , subject to these constraints can be observed to be  $\binom{k+1}{j+1}$ . The L.H.S. follows by weighing the number of instantiations of  $i_0$  through  $i_j$  (from the proof of Lemma 3) by the number of possible mappings from  $B$  to  $A$  for each instantiation, and summing for all levels of knowledge up to  $k$ , including *LCCV* $_{i_0}$ . This number must equal the size of  $Clk_0^{k+1}$ , which is  $n^{k+1}$ , as given on the R.H.S.  $\square$

We recognize Theorem 2 as a special case of the Binomial Theorem [1].

From this theorem, it immediately follows that the storage of redundant knowledge in a clock of dimension  $k+1$  consumes  $\sum_{j=-1}^k (n-1)^{j+1} \cdot [\frac{(k+1)!}{(j+1)!(k-j)!} - 1]$  integers out of  $n^{k+1}$ . For low  $k$  relative to  $n$ , this is a small fraction of  $n^{k+1}$ . Data structures other than  $Clk^\alpha$  as defined by Definition 14 can be used to represent complete  $k$ -bounded knowledge. However, one would lose the simplicity of indexing and updating the clocks for receive events would require manipulating these data structures. Figure 4 shows the efficiency of the  $Clk^\alpha$  data structure, measured as  $(\sum_{j=-1}^k (n-1)^{j+1})/n^{k+1}$  for various combinations of  $n$  and  $k$ . For  $n > 2$ , the efficiency is expressed as  $((n-1)^{k+2} - 1)/((n-2)(n^{k+1}))$  and for  $n = 2$ , the efficiency is  $(k+2)/2^{k+1}$ .

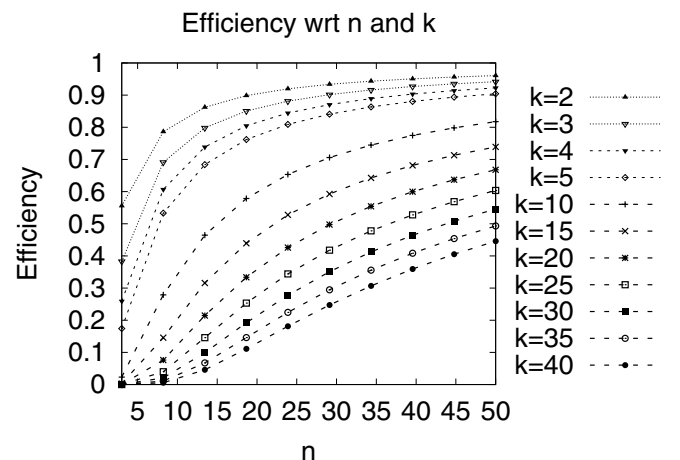


Fig. 4. The efficiency of the  $Clk^\alpha$  data structure

**Table 1.** Overview of results to attain  $E^k(\phi)$ , where  $\phi$  is a timestamp of a global state, in a system using a  $TIPP_k$  protocol. Here  $n$  is the number of processes in the system

	Problem/ Theorem	Space complexity	Algo.	Time complexity
What is the space complexity to attain up to date knowledge $E^k(\phi)$ ?	Problem 1 Lemma 3 Theorems 4, 5, 6	$\sum_{j=-1}^k (n-1)^{j+1}$ integers/process $= \theta(n^{k+1})$ integers/process	n/a	n/a
Given $\phi$ , what is the earliest state where $E^k(\phi)$ is attained?	Problem 2 Theorem 7	vector clocks + (trace of send and receive event timestamps after state timestamped $\phi$ )	Fig. 8	$O(kn^2 + \# \text{ send and receive events in } (H, \prec) \text{ after state timestamped } \phi)$
Given a state, what is the latest $\phi$ such that $E^k(\phi)$ is attained in given state?	Problem 3 Theorem 8	$(k+1)$ dim. clocks $= \theta(n^{k+1})$ integers/process	Fig. 9	$\theta(kn^2)$

**Theorem 3** *Complete  $k$ -bounded knowledge of the property “vector timestamp of a global state” cannot always be represented using the space required by a  $k$ -dimensional clock system (defined in Definition 14), but can be represented by a  $(k+1)$ -dimensional clock system.*

*Proof.* For complete  $k$ -bounded knowledge about the vector timestamp of a global state, Lemma 3 identified the space requirement at each process as  $\sum_{j=-1}^k (n-1)^{j+1}$  integers.

The following inequalities hold, the former from observation and the latter from Theorem 2.

$$n^k < \sum_{j=-1}^k (n-1)^{j+1} < n^{k+1}$$

Depending on the values of  $n$  and  $k$ , clocks of dimension  $k$  may not be sufficient to store all levels of knowledge of the “vector timestamp of a global state” up to level  $k$ , but clocks of dimension  $k+1$  are always sufficient. Although clocks of dimension  $k+1$  contain some replicated knowledge,  $k+1$  is the least dimension of a clock for which  $k$ -bounded knowledge about timestamps of global states can always be represented, in addition to providing a data structure that is easy to index.  $\square$

Recall from Sect. 3 that given complete  $(k-1)$ -bounded knowledge about the property “ $LCCV_{i_k}$ ”, then  $\phi = \min_{i_1, i_2, \dots, i_{k-1} \in N} (\phi_{i_1, i_2, \dots, i_k})$  is the latest  $l.c.c.v.$  about which complete  $(k-1)$ -bounded knowledge is available to  $i_0$ . Theorem 4 shows that processes must maintain the latest complete  $(k-1)$ -bounded knowledge about the property “ $LCCV_{i_k}$ ” in order that any process can attain complete  $(k-1)$ -bounded knowledge about the greatest  $l.c.c.v.$  about which such knowledge can be attained, and hence (by Corollary 2) to attain up to date knowledge  $E^{k-1}$  about the property “ $LCCV_{i_k}$ ”.

**Theorem 4** *In a system using a  $TIPP_k$  protocol, unless all processes maintain the latest complete  $k$ -bounded knowledge about the property “ $LCCV_{i_k}$ ”, they cannot attain up to date  $E^k$  knowledge about that property “ $LCCV_{i_k}$ ”.*

*Proof.* Let  $\phi_{i_k}^w$  be the  $l.c.c.v.$  of event  $e_{i_k}^w$ . From Corollary 2, attainment of up to date knowledge  $E^{k-1}(\phi_{i_k}^w)$  at  $i_0$  is equivalent to “ $\phi_{i_k}^w$  is the greatest value of property  $LCCV_{i_k}$  about which complete  $(k-1)$ -bounded knowledge is attainable, and

this knowledge is attained by  $i_0$ ”. From Remark 1, we infer the existence of message chains  $[m_{i_k}, m_{i_{k-1}}, m_{i_{k-2}}, \dots, m_{i_2}, m_{i_1}]$  from  $e_{i_k}^w$  to process  $i_0$ , for all instantiations of  $i_1, i_2, \dots, i_{k-1}$ . As a  $TIPP_{k-1}$  protocol is used, complete  $(k-1)$ -bounded knowledge about the latest possible  $l.c.c.v.s$  of events at  $i_k$  is conveyed by all the message chains. Furthermore, we also have that  $\phi_{i_k}^w = \min_{i_1, i_2, \dots, i_{k-1} \in N} (\phi_{i_1, i_2, \dots, i_k})$ , where  $\phi_{i_1, i_2, \dots, i_k}$ , for all  $i_1, i_2, \dots, i_{k-1} \in N$ , is the latest  $l.c.c.v.$  conveyed by the corresponding message chain. As this information is provided by the  $TIPP_{k-1}$  protocol and no other assumptions can be made, the min function is the only way to determine this  $\phi_{i_k}^w$ . Observe that  $\phi_{i_k}^w$  corresponds to some message chain of exactly  $k$  messages, and hence must be the value of one of  $(n-1)^k/n$   $l.c.c.v.$ ’s (refer to proof of Lemma 3). However, in order that other processes attain knowledge corresponding to message chains of exactly  $k$  messages,  $i_0$  needs to maintain complete  $(k-1)$ -bounded knowledge of  $LCCV_{i_k}$ .<sup>2</sup> As all  $\sum_{j=-1}^{k-1} (n-1)^{j+1}/n$   $l.c.c.v.$ ’s are independent, they must be explicitly tracked. We need to show that *there exists* an execution where, if knowledge  $K_{i_0} K_{i_1} \dots K_{i_{k-1}} K_{i_k}(\phi_{i_k})$  corresponding to the process chain  $[i_0, i_1, \dots, i_k]$  for some instantiation of  $i_1, \dots, i_{k-1}$ , is not explicitly tracked, then up to date knowledge  $E^{k-1}(\phi_{i_k})$  cannot be attained by  $i_0$  because it is determined by the process chain  $[i_0, i_1, \dots, i_k]$ .

We prove the existence of such an execution by construction. For any given execution prefix, let  $\gamma = Clock_{i_k}[i_k, \dots, i_k]$ . After the given execution prefix, no process sends any message until all messages in transit are delivered. Now, the events as described by  $Sync\_Execution([i_0, i_1, \dots, i_k], k)$  in Fig. 5 occur. The execution of  $Sync\_Execution$  can be thought of as occurring in  $(k-1)$  synchronous rounds, each with two phases. A phase of a round begins when all the messages sent in the previous phase (i.e., either the first phase of that round, or the second phase of the previous round) have been delivered. Note that this synchronous scheduling of the events can occur even in the asynchronous system of our model. Round  $r$  is initiated by process  $i_{k-r+1}$ . The message flows in a round are pictorially shown in Fig. 6a. We now show that the process chain  $[i_0, i_1, \dots, i_k]$  identified by  $k$  messages  $[m_k, \dots, m_1]$  does not

<sup>2</sup> This requirement is also met because processes follow the  $TIPP_{k-1}$  protocol and need to track the  $(k-1)$ -bounded knowledge.

**sequence of int**  $I = [i_0, i_1, i_2, i_3, \dots, i_{k-1}, i_k]$  such that  $i_w \neq i_{w+1}$  for  $0 \leq w \leq k-1$ ;

(1) *Sync.Execution*(**sequence of int**  $I$ ; **int**  $k$ )

//There are  $k-1$  rounds of two phases each.

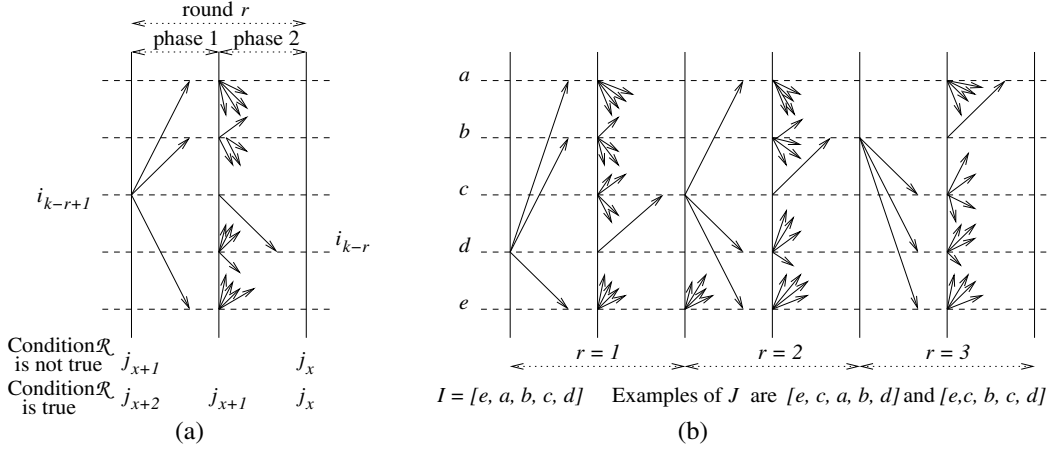
(2) **for**  $r = 1$  **to**  $k-1$  **do**

(3) **Phase I:** process  $i_{k+1-r}$  sends a message to each process in  $N - \{i_{k-r}\}$ ;

(4) **Phase II:** each process in  $N - \{i_{k+1-r}\}$  sends a message to each process in  $N$ ;

(5) process  $i_{k+1-r}$  sends a message to process  $i_{k-r}$ .

**Fig. 5.** Execution *Sync.Execution* with the property that process chain  $I = [i_0, i_1, i_2, i_3, \dots, i_{k-1}, i_k]$  does not exist but all other process chains  $J = [i_0, j_1, j_2, j_3, \dots, j_{k-1}, i_k]$ ,  $J \neq I$  exist



**Fig. 6a,b.** The execution *Sync.Execution* illustrated by a timing diagram. **a** A generic round  $r$  in the execution. The  $j_x$  variables are used in function *add.length*. While examining the execution backwards in time, the process chain can be extended from  $j_x$  to either  $j_{x+2}$  or to  $j_{x+1}$ , depending on whether or not condition  $\mathcal{R}$  holds, respectively. **b** An example execution for process chain  $I = [e, a, b, c, d] = [i_0, i_1, i_2, i_3, i_4]$  that cannot exist whereas all other process chains  $J$  exist. This execution has three rounds

(global variable) **sequence of int**  $I = [i_0, i_1, i_2, i_3, \dots, i_{k-1}, i_k]$  such that  $i_w \neq i_{w+1}$  for  $0 \leq w \leq k-1$ ;

(global variable) **sequence of int**  $J = [j_0 = i_0, j_1, j_2, j_3, \dots, j_{k-1}, j_k = i_k]$  such that  $J \neq I$  and w.l.o.g.,  $j_w \neq j_{w+1}$  for  $0 \leq w \leq k-1$ ;

(1) *function chain.length*(**int**  $k$ ) **returns int**  $x$

//The function detects whether the process chain specified by  $J$ , based on the  $k$  messages sent by  $i_k, j_{k-1}, \dots, j_1$ ,

//exists in the execution *Sync.Execution* which contains only  $(k-1)$  rounds.

//Here  $x$  is the number of messages in the corresponding message chain in *Sync.Execution*.

(2)  $x = 0$ ;

(3) **for**  $r = k-1$  **down to**  $1$  **do**

(4)  $x = x + \text{add.length}(x, r)$ ;

(5) **if**  $x \geq k$  **then return**( $x$ );

(6) **return**( $x$ ).

(1) *function add.length*(**int**  $x$ ; **int**  $r$ ) **returns int**

(2) **if**  $x = k-1$  **then return**(1);

(3) **else if**  $(j_{x+2} = i_{k-r+1})$  and  $(j_{x+1} \neq i_{k-r})$  **then return**(2); //condition  $\mathcal{R}$

(4) **else return**(1).

**Fig. 7.** Checking for message chains of length  $k$  in  $(k-1)$  rounds of *Sync.Execution*

exist in *Sync.Execution*, whereas all other process chains of the same length from  $i_0$  to  $i_k$  exist in *Sync.Execution*.

Let  $I$  be the process chain  $[i_0, i_1, i_2, i_3, \dots, i_{k-1}, i_k]$ , where  $i_w \neq i_{w+1}$  for  $0 \leq w \leq k-1$ , initiated by  $i_k$  at or after its local clock value becomes  $\gamma$ . We examine each successive round of *Sync.Execution* from round  $r = 1$  to  $k-1$  to determine that process chain  $I$  cannot ex-

ist<sup>3</sup>. By the construction of *Sync.Execution*, observe that initially before round 1, the process chain initiated by  $i_k$  is  $[i_k]$ , and after each round  $r$ , only one process  $i_{k-r}$  gets prepended to the process chain  $[i_{k-r+1}, \dots, i_{k-1}, i_k]$  to yield  $[i_{k-r}, \dots, i_{k-1}, i_k]$ . Hence, after  $k-1$  rounds, we have  $K_{i_1} K_{i_2} \dots K_{i_k}(\phi_{i_k})$  at  $i_1$  and we do not have

<sup>3</sup> Recall the relationship between a process chain and the corresponding message chain.

$K_{i_0}(K_{i_1}K_{i_2}\dots K_{i_k}(\phi_{i_k}))$  at  $i_0$ . Hence the process chain  $I$  does not exist. In the example of Fig. 6b, the process chain  $I = [e, a, b, c, d]$  does not exist.

Let  $J$  be any process chain  $[j_0 = i_0, j_1, j_2, j_3, \dots, j_{k-1}, j_k = i_k]$  such that  $J \neq I$ , i.e.,  $j_w \neq i_w$  for some  $w$  between 1 and  $k-1$ . Without loss of generality, we assume<sup>4</sup> that  $j_w \neq j_{w+1}$  for  $0 \leq w \leq k-1$ . We examine the execution *Sync.Execution* backwards in time from round  $r = k-1$  to 1 to show that the process chain  $J$  exists. Specifically, we use the iterative function *chain.length* in Fig. 7 to determine the length of (i.e., number of processes in) the prefix of the chain  $J$  in each round of *Sync.Execution* from last to first, i.e., for  $r = k-1$  to 1. Each iteration of *chain.length* invokes function *add.length* that suffixes the process chain extension in round  $r$  to the process chain identified in previously examined rounds:  $r+1$  to  $k-1$ .

Observe that any process chain can be extended by at most 2 processes in any round because a round contains two phases, each of which can potentially contribute one message to extend the process chain. When function *add.length* is invoked for round  $r$ , the process chain  $[i_0, j_1, j_2, \dots, j_x]$  has already been identified in rounds  $r+1$  through  $k-1$ . Function *add.length* now determines, based on the condition labeled  $\mathcal{R}$  in line (2), if process chain  $[i_0, j_1, j_2, \dots, j_x]$  can be extended by 1 or by 2. Refer to Fig. 6a. Observe that the process chain can be extended by 2 in round  $r$  if and only if all the following conditions, that form part of  $\mathcal{R}$ , hold.

1.  $j_{x+1} \neq i_{k-r+1}$  unless  $j_x = i_{k-r}$ . In phase 2, the process chain can be extended from  $j_x$  to  $j_{x+1}$  (by a message sent from  $j_{x+1}$  to  $j_x$ ).
2.  $j_{x+1} \neq i_{k-r}$ , otherwise, no message can be sent in phase 1 to  $i_{k-r}$ .
3.  $j_{x+2} = i_{k-r+1}$  so that in phase 1, the process chain can be extended from  $j_{x+1}$  to  $j_{x+2}$  (by a message sent from  $j_{x+2}$  to  $j_{x+1}$ ).

Observe that the first condition is implied by the third condition because of the assumption that  $j_w \neq j_{w+1}$ , and hence is omitted in  $\mathcal{R}$ . If condition  $\mathcal{R}$  is true, then the process chain that has been identified in rounds  $r+1$  to  $k-1$  can be extended from  $j_x$  to  $j_{x+1}$  to  $j_{x+2}$ , otherwise, it can be extended from  $j_x$  to  $j_{x+1}$  only. Hence, the function *add.length* returns 2 or 1 accordingly as the amount by which the chain can be extended.

We now need to prove that there is at least one round among the  $k-1$  rounds such that *add.length* returns 2, and hence the process chain  $J$  exists. Let  $[j_p, \dots, j_q]$  be the longest first substring in  $J$  that differs from the corresponding elements  $[i_p, \dots, i_q]$  of  $I$ .  $J = [i_0, j_1, j_2, \dots, j_{p-1}, j_p, \dots, j_q, j_{q+1}, \dots, j_{k-1}, j_k]$  thus satisfies

1.  $j_1 = i_1, j_2 = i_2, \dots, j_{p-1} = i_{p-1}$ ,
2.  $j_p \neq i_p, j_{p+1} \neq i_{p+1}, \dots, j_q \neq i_q$ , and
3.  $j_{q+1} = i_{q+1}$ .

<sup>4</sup> If  $j_w = j_{w+1}$ , then we can delete one of them to represent the same process chain, which has at most  $k-1$  messages. It is straightforward to observe that all process chains initiated by  $i_k$  sending a message and consisting of at most  $k-1$  messages exist in *Sync.Execution*.

Function *chain.length* invokes *add.length*( $x, r$ ) as follows.

- The first invocation is *add.length*( $x = 0, r = k-1$ ). Condition  $\mathcal{R}$ , viz., ( $j_2 = i_2$  and  $j_1 \neq i_1$ ), is false. Hence, the invocation returns 1.
- The  $m$ th invocation ( $1 < m < q$ ) is *add.length*( $x = m-1, r = k-m$ ). Condition  $\mathcal{R}$ , viz., ( $j_{m+1} = i_{m+1}$  and  $j_m \neq i_m$ ), is false. Hence, the invocation returns 1.
- Invocation  $q$  is *add.length*( $x = q-1, r = k-q$ ). Condition  $\mathcal{R}$ , viz., ( $j_{q+1} = i_{q+1}$  and  $j_q \neq i_q$ ), is true. Hence, the invocation returns 2.
- Each subsequent invocation to *add.length* may return a value of 1 or 2.

In the example of Fig. 6b, if  $J = [e, c, a, b, d]$ , the third invocation to *add.length* returns 2. If  $J = [e, c, b, c, d]$ , the first invocation to *add.length* returns 2. Hence, both process chains exist.

The latest  $l.c.c.v.i_k$  about which complete  $(k-1)$ -bounded knowledge is available to process  $i_0$ , is given by  $\min_{j_1, j_2, \dots, j_{k-1} \in N} (\phi_{j_1, j_2, \dots, j_k})$  which we denote as  $\mathcal{Y}_{i_1, i_2, \dots, i_k}$  for short-hand. Note that  $\mathcal{Y}_{i_1, i_2, \dots, i_k} < \gamma$ ; the inequality follows from the fact that all the process chains  $J$  exist whereas process chain  $I$  does not exist. If process  $i_0$  does not maintain knowledge  $K_{i_0}(K_{i_1}K_{i_2}\dots K_{i_k}(\phi_{i_k}))$  and does not make any other assumptions, then  $\mathcal{Y}_{i_1, i_2, \dots, i_k}$  is not available to it, it cannot compute the min function, and hence cannot attain up to date knowledge  $E^{k-1}$  about  $LCCV_{i_k}$ . The theorem now follows.  $\square$

**Theorem 5** *In a system using the TIPP protocol, in order for the processes to attain up to date knowledge  $E^k$  about the LCCV of all the processes, it is necessary and sufficient for the processes to store the latest complete  $k$ -bounded knowledge about the LCCV of all the processes.*

*Proof.* Theorem 4 showed the necessity. The sufficiency proof is as follows. Given the latest complete  $k$ -bounded knowledge about  $LCCV_{i_{k+1}}$ , the greatest timestamp  $\mathcal{Y}_{i_{k+1}}$  such that for all  $i_1, i_2, \dots, i_k \in N$ ,  $K_i K_{i_1} K_{i_2} \dots K_{i_k} K_{i_{k+1}}(\mathcal{Y}_{i_{k+1}})$  is true is simply  $\min_{i_1, i_2, \dots, i_k \in N} (\phi_{i_1, i_2, \dots, i_k, i_{k+1}})$ , where for each instantiation,  $K_i K_{i_1} K_{i_2} \dots K_{i_k} K_{i_{k+1}}(\phi_{i_1, i_2, \dots, i_k, i_{k+1}})$  is true. This is the greatest timestamp about which complete  $k$ -bounded knowledge is attainable and is attained, and by Corollary 2, up to date  $E^k(\phi_{i_{k+1}})$  knowledge about  $LCCV_{i_{k+1}}$  is also attained.

From Lemma 3, tracking the latest complete  $k$ -bounded knowledge about the LCCV of all the processes takes  $\sum_{j=-1}^k (n-1)^{j+1}$  integers per process.  $\square$

Lemma 3, along with Theorems 4 and 5, answered Problem 1 in terms of the space requirement and the nature of information to attain up to date knowledge  $E^k(\phi)$ , where  $\phi$  is a timestamp of a global state, in a system using the TIPP protocol. Theorem 6 answers Problem 1 by giving the conditions on the clock dimension.

**Theorem 6** *In a system using the TIPP protocol, (i) a  $(k+1)$ -dimensional clock system meeting the requirement of Definition 13 is necessary and sufficient, and furthermore, (ii) the*

space required by a  $k$ -dimensional clock system based on Definition 14 is not (always) sufficient, to attain up to date knowledge  $E^k(\phi)$  at a process, where  $\phi$  is the vector timestamp of the greatest possible global state.

*Proof.* Note that the local components of the clock values of different processes are independent of each other. Likewise,  $E^k$  knowledge about the local components of the clock values of different processes are also independent of each other. The result now follows from Theorems 3 and 5, and the property that the  $\alpha$ -dimensional clock gives the mapping to the latest complete  $(\alpha - 1)$ -bounded knowledge.  $\square$

## 5.2 Knowledge in the future

Theorem 7 addresses Problem 2 which seeks to identify in a system using the TIPP protocol, the timestamp of the earliest global state in which each process attains  $E^k(\phi)$ , where  $\phi$  is a timestamp of a given global state. By Theorem 4, this requires the latest complete  $k$ -bounded knowledge which can be represented by a  $(k + 1)$ -dimensional clock system (Theorem 6). However, the earliest state in which  $E^k(\phi)$  is attained by each process can be specified by a 1-dimensional vector clock. In the proof of Theorem 7, we give an algorithm that specifies such a state using vector clocks.

**Theorem 7** *Given  $\phi$ , the vector timestamp of a global state, the timestamp of the earliest global state in which each process attains  $E^k(\phi)$  in a system using the TIPP<sub>k</sub> protocol is given by the algorithm in Fig. 8.*

*Proof.* We denote the given vector timestamp  $\phi$  by  $T_\phi^1$ . Figure 8 gives an iterative algorithm to determine the timestamp of the earliest global state at which knowledge  $E^k(T_\phi^1)$  can be attained by each process. Function *Compute\_State* takes two inputs: (i)  $T_\phi^1$ , and (ii)  $k$ , the level of knowledge  $E^k(T_\phi^1)$  to be attained. The output is  $TS^1$ , the vector timestamp of the earliest global state in which  $E^k(T_\phi^1)$  is attained by each process. We prove the correctness of the algorithm by showing that the invariants on lines (6), (10), and (13) hold after each iteration.

The given cut timestamped  $T_\phi^1$  can be viewed as the earliest cut where  $E^0(T_\phi^1)$  is supported.<sup>5</sup> For the first iteration, the cut timestamped  $TS^1$  is initialized to  $T_\phi^1$  in line (4). We claim that at the start of each iteration, the invariant on line (6),  $(a, TS^1) \models E^{lvl-1}(T_\phi^1)$ , is valid. This is evident for  $lvl = 1$ , and for  $lvl > 1$ , this follows from the invariant on line (13) of the previous iteration.

Each iteration first identifies the earliest event  $e_p$  at each process  $p$  such that  $T^1(e_p) \geq TS^1$  (lines (7)-(9)). From the properties of vector clocks, it follows that each such event  $e_p$  is the earliest event following which  $K_p(TS^1)$ , i.e.,  $s_p \models K_p(TS^1)$ . From Remark 2, for all later states  $s_p'', s_p''' \models K_p(TS^1)$ . As  $T^1(e_p) \geq TS^1$ , we have from the property of vector timestamps that for any event  $e_q$  satisfying  $T^1(e_q)[q] = TS^1[q]$ ,  $e_q \prec e_p$  and hence there is a message chain from  $e_q$

to  $e_p$ . The cut timestamped  $T'^1$  such that  $F_p(T'^1) = e_p$  is the earliest cut that satisfies the following after line (9) of iteration  $lvl$ .

- $(a, T'^1) \models E(TS^1)$ , i.e.,  $(a, T'^1) \models \bigwedge_i K_i(TS^1)$ .
- As per the invariant on line (6),  $(a, TS^1) \models E^{lvl-1}(T_\phi^1)$  at the start of each iteration. Assuming adequate timestamp knowledge (using  $(k + 1)$ -dimensional clocks) is piggybacked on the messages of the message chain from events  $e_q$  identified above to events  $e_p$ , we have  $(a, T'^1) \models \bigwedge_i K_i(E^{lvl-1}(T_\phi^1))$ , i.e.,  $(a, T'^1) \models E^{lvl}(T_\phi^1)$ . This leads to the invariant on line (10).

Each event  $e_p$  identified in line (8) identifies a prefix of a local history, but the union of the events in such local prefixes, which is the cut timestamped  $T'^1$ , may not be consistent. Hence the invariant on line (10) does not hold for a consistent cut. However, the union of the  $\prec$ -closed cuts  $\downarrow e_p$ , for all  $p$ , gives the smallest consistent cut containing these events (Corollary 4) and the timestamp of this consistent cut, computed in lines (11)-(12) as  $TS^1$  using  $T'^1$ , is as per this corollary. Along this consistent cut, we assert  $(a, TS^1) \models \bigwedge_i K_i(E^{lvl-1}(T_\phi^1))$ . This leads to the invariant on line (13).  $\square$

For each process to attain  $E^k(T_\phi^1)$ , observe that  $k + 1$  iterations are required.

**Example:** For the example of Fig. 2, let the problem inputs be:  $T_\phi^1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$  and  $k = 1$ . There will be two iterations of *Compute\_State*.

Iteration  $lvl = 1$ : line 6 invariant is  $(a, \begin{bmatrix} 1 \\ 2 \end{bmatrix}) \models E^0 \left( \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right)$ ,

line 10 invariant is  $(a, \begin{bmatrix} 4 \\ 3 \end{bmatrix}) \models E^1 \left( \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right)$ ,

line 13 invariant is  $(a, \begin{bmatrix} 4 \\ 5 \end{bmatrix}) \models E^1 \left( \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right)$ . Hence in the consistent global state after events  $e_4^1$  and  $e_2^2$ , the system attains  $E^1(T_\phi^1)$  but for each process to attain  $E^1(T_\phi^1)$ , another iteration is needed.

Iteration  $lvl = 2$ : line 6 invariant is  $(a, \begin{bmatrix} 4 \\ 5 \end{bmatrix}) \models E^1 \left( \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right)$ ,

line 10 invariant is  $(a, \begin{bmatrix} 4 \\ 9 \end{bmatrix}) \models E^2 \left( \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right)$ ,

line 13 invariant is  $(a, \begin{bmatrix} 7 \\ 9 \end{bmatrix}) \models E^2 \left( \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right)$ . Hence the earliest global state in which both  $P_1$  and  $P_2$  know that  $E^1(T_\phi^1)$  is true is the state after events  $e_1^7$  and  $e_2^9$ .

**Complexity:** The time complexity of the algorithm in Fig. 8 is  $O(kn^2) + O(\text{number of send and receive events between the input state timestamped } \phi \text{ and the solution state timestamped } TS^1)$ . This is given by the asymptotic upper bound:  $O(kn^2 + \text{number of send and receive events in } (H, \prec) \text{ after the state timestamped } \phi)$ . The space complexity is that of a vector clock system, and also requires each process to store a trace of the timestamps of its send and receive events beyond the state timestamped  $\phi$ . The algorithm overhead can be reduced by using some knowledge of the patterns of the causal message chains. Such optimizations would complicate the presentation of the algorithm, and are not central to answering Problem 2.

<sup>5</sup> Recall that we defined  $E^0(\phi)$  as simply  $\phi$  to simplify this proof structure. We can now present the proof for the first iteration uniformly with the proof for the other iterations.

(1) **Problem Inputs:**  
 (1a) **array of int**  $T_\phi^1$ ; //vector timestamp  $\phi$  of earliest state in which predicate of interest to application is true  
 (1b) **int**  $k$ ; //level of knowledge  $E^k(\phi)$  to be attained  
 (2) **Problem Output:**  
 (2a) **array of int**  $TS^1 = \text{Compute\_State}(T_\phi^1, k)$ .  
 (2b) //vector timestamp of earliest state in which  $E^k(\phi)$  is attained by each process  
 (3) *function*  $\text{Compute\_State}(\text{array of int } T_\phi^1; \text{int } k)$  **returns**  $TS^1$   
 (4)  $TS^1 = T_\phi^1$ ;  
 (5) **for**  $lvl = 1$  **to**  $k + 1$  **do**  
 (6) // **invariant**  $(a, TS^1) \models E^{lvl-1}(T_\phi^1)$   
 (7)  $\forall p \in N$  **do**  
 (8) identify earliest event  $e_p \mid T^1(e_p) \geq TS^1$ ;  
 (9) set timestamp  $T'^1$  such that  $T'^1[p] = T^1(e_p)[p]$ ;  
 (10) // **invariant**  $(a, T'^1) \models E^{lvl}(T_\phi^1) \wedge \nexists T''^1 \mid (T''^1 < T'^1 \wedge (a, T''^1) \models E^{lvl}(T_\phi^1))$   
 (11)  $\forall p \in N$  **do**  
 (12)  $TS^1[p] = \max(T^1(e_1)[p], T^1(e_2)[p], \dots, T^1(e_n)[p])$ ;  
 (13) // **invariant**  $(a, TS^1) \models E^{lvl}(T_\phi^1) \wedge \nexists TS'^1 \mid (TS'^1 < TS^1 \wedge (a, TS'^1) \models E^{lvl}(T_\phi^1))$  and  $TS'^1$  is consistent  
 (14) **return**( $TS^1$ ).

**Fig. 8.** Given a vector timestamp  $\phi$  of a global state, algorithm to compute the timestamp of the earliest global state in which  $E^k(\phi)$  is attained by each process

### 5.3 Knowledge about the past

Problem 3 can be restated as follows. “Given a timestamp  $T^{\beta+1}$  of a global state, what is the maximum timestamp  $\phi$  such that  $(a, T^{\beta+1}) \models E^\beta(\phi)$ ?” To determine  $\phi$  requires the latest complete  $\beta$ -bounded knowledge (Theorem 4) which can be represented by a  $(\beta + 1)$ -dimensional clock system (Theorem 6). We can apply the function min (the min of vectors is the vector of the component-wise minimums) to the  $n^\beta$  1-dimensional timestamps of size  $n$  in the given  $T^{\beta+1}$ . This naive approach requires  $n \cdot n^\beta$  comparisons – which is exponential in  $\beta$ .

Theorem 8 gives a  $\theta(\beta n^2)$  time complexity algorithm in Fig. 9 to address Problem 3.

**Theorem 8** *Given timestamp  $T^\beta$  of a global state, the timestamp  $\phi$  of the (most recent possible) global state about which up to date knowledge  $E^k(\phi)$ , where  $k \leq \beta - 1$ , can be attained at the given state timestamped  $T^\beta$  in a system using the  $TIPP_{\beta-1}$  protocol is computed by the algorithm in Fig. 9.*

*Proof.* The proof is by construction. Figure 9 gives an algorithm to compute the timestamp  $\phi$  of the maximum cut about which knowledge  $E^k(\phi)$  is attained in the given state of observation timestamped  $Ob.T^\beta$ . From Theorem 6, note that  $\beta > k$ .

$\text{Compute\_Phi}$  has inputs (i)  $T^\alpha$ , the (variable dimension) timestamp of the maximum cut about which up to date knowledge  $E^{atn}$  is attained in  $Ob.T^\beta$ , (ii)  $m$ , the level of knowledge that is yet to be attained, and (iii)  $atn$ , the level of up to date knowledge already attained. The output is the timestamp  $\phi$  of the maximal cut about which up to date knowledge  $E^k$  is attained in the given state  $Ob.T^\beta$ .

$\text{Compute\_Phi}$  is invoked as  $\text{Compute\_Phi}(Ob.T^\beta, k, 0)$  and is tail-recursive.  $T^\alpha$  is progressively decreased at each recursion level to add another level of knowledge to what is known of  $T^\alpha$  at cut  $Ob.T^\beta$ . So at each additional recursion level,  $T^\alpha$  therein converges towards  $\phi$ . Each recursion level behaves as follows.

- Given  $T^\alpha(\text{Cut})$ , the loop in lines (5)-(6) computes the  $(\alpha - 1)$ -dimensional timestamp of each  $F_p(\text{Cut})$ , the latest event of the cut  $\text{Cut}$  at process  $p$  ( $p \in N$ ).  $T^{(\alpha-1)}(F_p(\text{Cut}))$  is simply  $T^\alpha[p, \cdot]$ .
- Let  $X$  denote set  $F(\text{Cut})$  identified in line (6). The loop in lines (7)-(9) applies Lemma 2 to  $X$  to compute the timestamp of  $\cap_{\downarrow} X$ . To do so, it identifies the timestamps  $T^{(\alpha-2)}(F_p(\cap_{\downarrow} X))$  for each process  $p$ . Then  $T^{(\alpha-1)}(\cap_{\downarrow} X)$  is simply the aggregation of the  $n$  timestamps  $T^{(\alpha-2)}(F_p(\cap_{\downarrow} X))$ , as shown in line (10). By Lemma 1,  $T^{(\alpha-1)}(\cap_{\downarrow} X)$  is the timestamp of the maximum prefix about which all the processes have knowledge in the state identified by  $X = F(\text{Cut})$ . Thus, up to date knowledge  $E(T^{(\alpha-1)})$  is attained in the state with timestamp  $T^\alpha$  in this recursion level and we assert the invariant on line (11).
- The above steps also add a level of up to date knowledge at the given initial state  $Obs.T^\beta$ , and we assert this in the invariant on line (13). If this is the desired level of knowledge, then we have the terminating case for the recursion and the value of  $T^{(\alpha-1)}$  is returned (lines (14)-(16)), otherwise  $\text{Compute\_Phi}$  is recursively invoked to determine the greatest  $\phi$  that is known at  $T^{(\alpha-1)}$  for the remaining  $m$  levels of knowledge to be attained (lines (17)-(18)).

The invariants on lines (11) and (13), justified above, are the only two invariants encountered for the terminating case of recursion. Observe that the dimension of the timestamp  $T^{(\alpha-1)}$  which is returned as  $\phi$  is  $\beta - k$ . Once the tail-recursive call for values of  $m \geq 2$  returns the value of  $\phi$ , which is the same as that for the  $m = 1$  call, the invariants on lines (19)-(21) can be asserted by the following reasoning.

For the call with  $m = 2$ , the invariant on line (19) is the same as the invariant on line (11) for the terminating case call, where  $m = 1$ . The invariant on line (20) follows from the invariants on lines (19) and (11). The invariant on line (21) follows from the invariants on line (13) and (19).

**(1) Problem Inputs:**

- (1a)  $\beta$ -dim. array of int  $Ob.T^\beta$ ; // timestamp of observation state  
 (1b) int  $k$ , where  $\beta > k \geq 1$ ; // level of knowledge to be attained

**(2) Problem Output:**

- (2a)  $(\beta - k)$  dim. array of int  $\phi = \text{Compute\_Phi}(Ob.T^\beta, k, 0)$ .  
 (2b)  $\phi$  is the timestamp of the maximum possible state such that  $(a, Ob.T^\beta) \models E^k(\phi)$

(3) function  $\text{Compute\_Phi}(\text{var dim. array of int } T^\alpha; \text{int } m, \text{atn})$  returns  $\phi$

(4a)  $T^\alpha$  is the timestamp of the maximum possible state such that  $(a, Ob.T^\beta) \models E^{\text{atn}}(T^\alpha)$

(4b)  $m$  is the level of knowledge yet to be attained

(4c)  $\text{atn}$  is the level of knowledge already attained.  $\text{atn} = k - m$ .

(5)  $\forall p \in N$  do

(6)  $T_p^{\alpha-1} = T^\alpha[p, \cdot];$

(7)  $\forall p \in N$  do

(8) let  $r$  be (any) process such that  $T_r^{\alpha-1}[p, p, \dots, p] = \min_{q \in N} (T_q^{\alpha-1}[p, p, \dots, p]);$

(9)  $T_p^{\alpha-2} = T_r^{\alpha-1}[p, \cdot];$

(10)  $T^{\alpha-1}$  is such that  $(\forall p), T^{\alpha-1}[p, \cdot] = T_p^{\alpha-2};$

(11) // invariant  $(a, T^\alpha) \models E^1(T^{\alpha-1}) \wedge \neg T^{\alpha-1} \mid (T^{\alpha-1} > T^{\alpha-1} \wedge (a, T^\alpha) \models E^1(T^{\alpha-1}))$

(12)  $\text{atn} = \text{atn} + 1; m = m - 1;$

(13) // invariant  $(a, Ob.T^\beta) \models E^{\text{atn}}(T^{\alpha-1}) \wedge \neg T^{\alpha-1} \mid (T^{\alpha-1} > T^{\alpha-1} \wedge (a, Ob.T^\beta) \models E^{\text{atn}}(T^{\alpha-1}))$

(14) if  $m = 0$  then

(15)  $\phi = T^{\alpha-1};$

(16) return( $\phi$ );

(17) else

(18)  $\phi = \text{Compute\_Phi}(T^{\alpha-1}, m, \text{atn});$

(19) // invariant  $(a, T^{\alpha-1}) \models E^m(\phi) \wedge \neg \phi' \mid (\phi' > \phi \wedge (a, T^{\alpha-1}) \models E^m(\phi'))$

(20) // invariant  $(a, T^\alpha) \models E^{m+1}(\phi) \wedge \neg \phi' \mid (\phi' > \phi \wedge (a, T^\alpha) \models E^{m+1}(\phi'))$

(21) // invariant  $(a, Ob.T^\beta) \models E^{\text{atn}+m}(\phi) \wedge \neg \phi' \mid (\phi' > \phi \wedge (a, Ob.T^\beta) \models E^{\text{atn}+m}(\phi'))$

(22) return( $\phi$ ).

**Fig. 9.** Algorithm to compute the latest timestamp  $\phi$  for which up to date knowledge  $E^k(\phi)$  holds in a state with given timestamp  $T^\beta$ , where  $\beta > k$

For calls with  $m > 2$ , the invariant on line (19) is the same as the invariant on line (20) for the call with  $m - 1$ . The reasoning for the invariants on lines (20) and (21) is the same as that for  $m = 2$ .

The invariant on line (21) implies that  $\phi$  is the timestamp of the maximum cut such that  $(a, Obs.T^\beta) \models E^k(\phi)$ .  $\square$

**Example:** For the example of Fig. 2, let the problem inputs be:

$Ob.T^3 = \begin{bmatrix} 10 & 2 \\ 7 & 7 \end{bmatrix} \mid \begin{bmatrix} 8 & 8 \\ 5 & 9 \end{bmatrix}$  and  $k = 2$ . There will be two invocations of  $\text{Compute\_Phi}$  recursively.

Iteration  $\text{atn} = 0$ : This is the first invocation, with  $T^\alpha = Ob.T^3, m = 2, \text{atn} = 0$ .

lines 5,6:  $T_1^2 = \begin{bmatrix} 10 & 8 \\ 2 & 8 \end{bmatrix}$  and  $T_2^2 = \begin{bmatrix} 7 & 5 \\ 7 & 9 \end{bmatrix}$ .

lines 7,8,9: for  $p = 1, r$  is 2. Hence  $T_1^1 = T_2^2[1, \cdot] = \begin{bmatrix} 7 \\ 5 \end{bmatrix}$ .

For  $p = 2, r$  is 1. Hence  $T_2^1 = T_1^2[2, \cdot] = \begin{bmatrix} 2 \\ 8 \end{bmatrix}$ .

line 10:  $T^2[1, \cdot] = T_1^1 = \begin{bmatrix} 7 \\ 5 \end{bmatrix}$ .  $T^2[2, \cdot] = T_2^1 = \begin{bmatrix} 2 \\ 8 \end{bmatrix}$ . Hence,

$T^2 = \begin{bmatrix} 7 & 5 \\ 2 & 8 \end{bmatrix}$ .

line 13: Invariant  $\left(a, \begin{bmatrix} 10 & 2 \\ 7 & 7 \end{bmatrix} \mid \begin{bmatrix} 8 & 8 \\ 5 & 9 \end{bmatrix}\right) \models E^1\left(\begin{bmatrix} 7 & 5 \\ 2 & 8 \end{bmatrix}\right)$ .

Invocation  $\text{atn} = 1$ : This is the second invocation, with  $T^\alpha = \begin{bmatrix} 7 & 5 \\ 2 & 8 \end{bmatrix}, m = 1, \text{atn} = 1$ .

lines 5,6:  $T_1^1 = \begin{bmatrix} 7 \\ 5 \end{bmatrix}$  and  $T_2^1 = \begin{bmatrix} 2 \\ 8 \end{bmatrix}$ .

lines 7,8,9: For  $p = 1, r$  is 2. Hence  $T_1^0 = T_2^1[1] = 2$ .

For  $p = 2, r$  is 1. Hence  $T_2^0 = T_1^1[2] = 5$ .

line 10:  $T^1[1] = T_1^0 = 2$ .  $T^1[2] = T_2^0 = 5$ . Hence,  $T^1 = \begin{bmatrix} 2 \\ 5 \end{bmatrix}$ .

line 13: Invariant  $\left(a, \begin{bmatrix} 10 & 2 \\ 7 & 7 \end{bmatrix} \mid \begin{bmatrix} 8 & 8 \\ 5 & 9 \end{bmatrix}\right) \models E^2\left(\begin{bmatrix} 2 \\ 5 \end{bmatrix}\right)$ .

**Complexity:** The time complexity of the algorithm in Fig. 9 is  $\theta(k \cdot n^2)$ . The analysis is as follows. For each invocation of  $\text{Compute\_Phi}$ , line (8) has  $n$  comparisons, and line (9) has one pointer assignment, leading to  $n + 1$  operations. The loop (7)-(9) is executed  $n$  times, and  $\text{Compute\_Phi}$  is invoked  $k$  times, leading to a total time complexity of  $\theta(k \cdot n^2)$ . The space complexity is that of the logical clocks, which is  $\theta(n^\beta)$  integers and meets the asymptotic lower bound  $\Omega(n^\beta)$  shown in Theorem 6. Note that the time complexity is less than the space complexity because information is selectively accessed on a dynamic basis.

## 6 Concluding remarks

Vector and matrix clocks are widely used in asynchronous distributed systems. This paper answered the question: “*how does the clock abstraction generalize?*” The paper first formalized protocols that use an *ambient* form of information exchange,

embodied in the vector clock and matrix clock protocols, that is being used in a wide range of applications. The type of property about which information is exchanged in such protocols was formally characterized as a *monotonic* property; a wide range of applications deal with monotonic properties. In any execution, the set of vector timestamps of global states is homomorphic to the set of greatest (monotonic) facts about a monotonic property in the global states. Timestamp Information Piggybacking Protocols (TIPP) were then introduced to characterize protocols that exchange timestamp information in an ambient manner. The paper then defined logical clocks of arbitrary dimensions using a TIPP protocol to capture causality relationships in distributed executions. The main results performed a knowledge-theoretic analysis of the representational power of such clocks.

Although the size of logical clocks was shown to grow exponentially with the level of knowledge to be attained using a TIPP protocol, nevertheless, the results shed light on the tight relation between the dimension of logical clocks and the level of knowledge attainable. Well-known encoding techniques such as Godel encoding can be used to express the size of clocks. The size of the logical clocks can be reduced by modifying the system model and making simplifying assumptions, such as those made previously for vector clocks [17, 22]. The size of the clocks can also be reduced by making approximations to the accurate clock value, along the lines of earlier approximations [12, 19, 23, 24] made previously for vector and matrix clocks. These options provide directions for future research.

In summary, this paper made the following contributions. (1) It motivated and proposed logical clocks of arbitrary dimensions, and formalized the TIPP protocol for knowledge transfer used by such clocks. (2) It showed that there exists a tight relation between the dimension of logical clocks and the level of knowledge attainable, and established some complexity bounds. Here it identified and explored an important conceptual link. (3) It proposed algorithms to determine the timestamp of the latest global state about which a specified level of knowledge is attainable in a given global state, and to compute the timestamp of the earliest global state in which a specified level of knowledge about a given global state is attainable. The results can be useful to applications that deal with monotonic properties.

In addition to the main contributions, the paper made three other contributions. First, it gave a better insight into the problem of causality-tracking mechanisms, which has been identified by Schwarz-Mattern [21] as an open problem. Second, the paper gave a new and useful definition of levels of knowledge, that is tailored for asynchronous message-passing systems. Third, the paper gave a direct understanding of how classical problems in knowledge theory, such as the “muddy children” and “cheating husbands” problems, that had hitherto been explained in the synchronous system model, can be solved in our system model using TIPP protocols for knowledge transfer.

**Acknowledgements.** We thank the anonymous referees for their comments which greatly helped to increase the clarity of the presentation and also resulted in the addition of various examples. We also thank Punit Chandra for his help in plotting Fig. 4.

## References

1. Bogart KP: Introductory combinatorics. Pitman Publishing, 1983
2. Chandy KM, Misra J: How processes learn. *Distributed Computing* 1(1): 40–52 (1986)
3. Chandy KM, Lamport L: Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems* 3(1): 63–75 (1985)
4. Charron-Bost B: Concerning the size of clocks in distributed systems. *Information Processing Letters* 39(1): 11–16 (1991)
5. Critchlow C, Taylor K: The inhibition spectrum and the achievement of causal consistency. *Distributed Computing* 10(1): 11–27 (1996)
6. Fagin R, Halpern J, Moses Y, Vardi M: Reasoning about knowledge. MIT Press, 1995
7. Fidge CJ: Timestamps in message-passing systems that preserve partial ordering. *Australian Computer Science Communications* 10(1): 56–66 (1988)
8. Fidge CJ: Logical time in distributed computing systems. *IEEE Computer* 24(8): 28–33 (1991)
9. Halpern J, Fagin R: Modeling knowledge and action in distributed systems. *Distributed Computing* 3(4): 159–179 (1989)
10. Halpern J, Moses Y: Knowledge and common knowledge in a distributed environment. *Journal of the ACM* 37(3): 549–587 (1990)
11. Holliday J, Agrawal D, El-Abbadi A: Disconnection modes for mobile databases. *Wireless Networks* 8(4): 391–402 (2002)
12. Krishnakumar A, Bernstein A: Bounded ignorance: A technique for increasing concurrency in a replicated system. *ACM Transactions on Database Systems* 19(4): 586–625 (1994)
13. Kshemkalyani AD: On continuously attaining levels of concurrent knowledge without control messages. Technical Report UIC-EECS-98-6, University of Illinois at Chicago, 1998
14. Kshemkalyani AD: Concurrent knowledge and logical clock abstractions. In: Kapoor S, Prasad S (eds) *Proc. 20th Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science 1974*, Springer, Berlin Heidelberg New York 2000, pp 489–502
15. Lamport L: Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM* 21(7): 558–565 (1978)
16. Mattern F: Virtual time and global states of distributed systems. In: Cosnard M et al. (eds) *Proc. Workshop on Parallel and Distributed Algorithms*. North-Holland, Elsevier, 1989, pp 215–226
17. Meldal S, Sankar S, Vera J: Exploiting locality in maintaining potential causality. *Proc. 10th ACM Symposium on Principles of Distributed Computing*, 1991, pp 231–239
18. Panangaden P, Taylor K: Concurrent common knowledge: Defining agreement for asynchronous systems. *Distributed Computing* 6(2): 73–94 (1992)
19. Ruget F: Cheaper matrix clocks. In: Tel G, Vitanyi P (eds) *Proc. 8th Workshop on Distributed Algorithms, Lecture Notes in Computer Science 857*, Springer, Berlin Heidelberg New York 1994, pp 355–369
20. Sarin S, Lynch N: Discarding obsolete information in a distributed database system. *IEEE Transactions on Software Engineering* 13(1): 39–46 (1987)
21. Schwarz R, Mattern F: Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing* 7(3): 149–174 (1994)
22. Singhal M, Kshemkalyani A: Efficient implementation of vector clocks. *Information Processing Letters* 43(1): 47–52 (1992)

23. Torres-Rojas FJ, Ahamad M: Plausible clocks: Constant size logical clocks for distributed systems. *Distributed Computing* 12(4): 179–195 (1999)
24. Wu G, Bernstein A: Efficient solutions to the replicated log and dictionary problems. *Proc. 3rd ACM Symposium on Principles of Distributed Computing*, 1984, pp 232–242

## Appendix

**Proposition 1** *A process  $i$  that attains complete  $k$ -bounded knowledge about a fact  $\phi_{i_k}$  that is the l.c.c.v. $_{i_k}$  also attains knowledge  $K_i(E^k(\phi_{i_k}))$ .*

*Proof.* The proof is by induction, using the hypothesis that complete  $k$ -bounded knowledge  $K_i(K_{i_1}K_{i_2}\dots K_{i_k}K_{i_{k+1}}(\phi_{i_{k+1}}))$  of a fact  $\phi_{i_{k+1}}$  at a process  $i$  in state  $s_i^w$  implies  $s_i^w \models K_i(E^k(\phi_{i_{k+1}}))$ .

*Case  $k = 0$ :*  $K_i(K_{i_1}(\phi_{i_1}))$  is tautological to  $K_iE^0(\phi_{i_1})$ .

*Case  $k = 1$ :* We need to show that complete 1-bounded knowledge  $K_i(K_{i_1}K_{i_2}(\phi_{i_2}))$  of a fact  $\phi_{i_2}$  at a process  $i$  implies  $K_i(E^1(\phi_{i_2}))$ . From Remark 1, there are message chains from  $i_2$  to  $i_1$  to  $i$ , for all  $i_1 \in N$ . We substitute the variable  $p$  for  $i_1$ . For all  $p$  in  $N$ , denote by  $e_p^{z_p}$  the send event at process  $p$  which sends the message to process  $i$  in the corresponding message chain. Let  $c$  be the cut such that  $F(c) = \{e_p^{z_p-1}, \forall p \in N\}$ . Then  $(a, c) \models \bigwedge_{p \in N} K_p(\phi_{i_2})$  and hence  $(a, c) \models E(\phi_{i_2})$ . Note that there exists a message sent by each process  $p$  at event  $e_p^{z_p}$  to process  $i$  and received by  $i$  before  $s_i^w$ . Hence,  $i$  can safely infer this group knowledge  $E^1(\phi_{i_2})$  from its complete 1-bounded knowledge, resulting in  $s_i^w \models K_iE(\phi_{i_2})$ .

*Case  $k = x$ :* We assume the induction hypothesis for  $k = x$ .

*Case  $k = x + 1$ :* We need to show that complete  $(x + 1)$ -bounded knowledge  $K_i(K_{i_1}K_{i_2}\dots K_{i_{x+1}}K_{i_{x+2}}(\phi_{i_{x+2}}))$  of a fact  $\phi_{i_{x+2}}$  at a process  $i$  in state  $s_i^w$  implies  $s_i^w \models K_i(E^{x+1}(\phi_{i_{x+2}}))$ .

We define two data structures: *Fastest.Chains*, a set of message chains, and *Earliest.Rcv.Events<sub>a</sub>*, a set of receive events at process  $a$ . From Remark 1, there is at least one message chain from  $i_{x+2}$  to  $i_{x+1}$  and so on to  $i_1$  to  $i$ , for all  $i_{x+1}, i_x, \dots, i_1 \in N$ . For each instantiation of each of  $i_{x+1}, i_x, i_{x-1}, \dots, i_1$ , (if there are multiple chains, consider any one that terminates at  $i$  with the earliest message and) add the message chain to set *Fastest.Chains*. For each chain in *Fastest.Chains*, let the event at which process  $i_1$  receives the message from  $i_2$  be added to set *Earliest.Rcv.Events<sub>i\_1</sub>*. For each  $i_1 \in N$ , observe that there exists a message sent from  $i_1$  after  $\max(\text{Earliest.Rcv.Events}_{i_1})$  and received by  $i$  before  $s_i^w$ . We substitute the variable  $p$  for  $i_1$ . Let  $e_p^{z_p}$  be the send event at process  $p \in N$  which sends this message to process  $i$ . Note that after  $\max(\text{Earliest.Rcv.Events}_p)$ , process  $p$  has attained complete  $x$ -bounded knowledge of  $\phi_{i_{x+2}}$ . By the induction hypothesis, for each process  $p \in N$ ,  $s_p^{z_p-1} \models K_p(E^x(\phi_{i_{x+2}}))$ . Let  $c$  be the cut such that  $F(c) = \{e_p^{z_p-1}, \forall p \in N\}$ . Then  $(a, c) \models \bigwedge_{p \in N} K_p(E^x(\phi_{i_{x+2}}))$  and hence  $(a, c) \models E^{x+1}(\phi_{i_{x+2}})$ .

Note that there exists a message sent by each process  $p$  at event  $e_p^{z_p}$  to process  $i$  and received by  $i$  before  $s_i^w$ . Hence,  $i$  can safely infer this group knowledge  $E^{x+1}(\phi_{i_{x+2}})$  from its complete  $x + 1$ -bounded knowledge, resulting in  $s_i^w \models K_iE^{x+1}(\phi_{i_{x+2}})$ .  $\square$

The following Proposition 2 is the converse of Proposition 1.

**Proposition 2** *A process  $i$  that attains knowledge  $K_iE^k(\phi_{i_k}^w)$ , where  $\phi_{i_k}^w$  is the l.c.c.v. $_{i_k}$ , also attains complete  $k$ -bounded knowledge of that l.c.c.v. $_{i_k}$ .*

*Proof.* The proof is by induction using the hypothesis that  $K_iE^x(\phi_{i_{x+1}}^w)$  implies the complete  $k$ -bounded knowledge  $K_iK_{i_1}K_{i_2}\dots K_{i_x}K_{i_{x+1}}(\phi_{i_{x+1}}^w)$ .

*Case  $k = 0$ :*  $K_iE^0(\phi_{i_1}^w)$  is tautological to  $K_i(K_{i_1}(\phi_{i_1}^w))$ .

*Case  $k = 1$ :* We need to show that  $s_i^q \models K_iE(\phi_{i_2}^w)$  implies that for all  $i_1$  in  $N$ ,  $s_i^q \models K_iK_{i_1}K_{i_2}(\phi_{i_2}^w)$ .  $K_iE(\phi_{i_2}^w)$  implies the existence of a cut  $c$  such that  $(a, c) \models \bigwedge_{p \in N} K_pK_{i_2}(\phi_{i_2}^w)$ . Furthermore, process  $i$  knows that for every process  $p$ ,  $K_pK_{i_2}(\phi_{i_2}^w)$ . Hence process  $i$  attains the complete 1-bounded knowledge of  $\phi_{i_2}^w$ .

*Case  $k = x$ :* We assume the induction hypothesis for  $k = x$ .

*Case  $k = x + 1$ :* We need to show that  $s_i^q \models K_iE^{x+1}(\phi_{i_{x+2}}^w)$  implies that for all  $i_1, i_2, \dots, i_{x+1}$  in  $N$ ,  $s_i^q \models K_iK_{i_1}\dots K_{i_{x+1}}K_{i_{x+2}}(\phi_{i_{x+2}}^w)$ .

$K_iE(E^x(\phi_{i_{x+2}}^w))$  implies the existence of a cut  $c$  such that  $(a, c) \models \bigwedge_{p \in N} K_p(E^x(\phi_{i_{x+2}}^w))$ . Furthermore, process  $i$  knows that for every process  $p$  in  $N$ ,  $K_p(E^x(\phi_{i_{x+2}}^w))$  in some state  $s_p^{z_p}$ , where  $e_p^{z_p}$  is the latest event at process  $p$  in cut  $c$ . From the induction hypothesis for  $k = x$ , each process  $p$  in state  $s_p^{z_p}$  has attained complete  $x$ -bounded knowledge of  $\phi_{i_{x+2}}^w$ . Therefore process  $i$  knows that every process  $p$  has attained complete  $x$ -bounded knowledge of  $\phi_{i_{x+2}}^w$ . Hence, process  $i$  in state  $s_i^q$  attains complete  $(x + 1)$ -bounded knowledge of  $\phi_{i_k}^w$ .  $\square$

**Ajay Kshemkalyani** is an Associate Professor of Computer Science at the University of Illinois at Chicago since 2000. He previously spent several years at IBM Research Triangle Park working on various aspects of computer networks. Ajay Kshemkalyani received a Ph.D. and an M.S. in Computer and Information Science from The Ohio State University in 1991 and 1988, respectively, and a B.Tech. in Computer Science and Engineering from the Indian Institute of Technology, Bombay, in 1987. His current research interests include computer networks, distributed computing, algorithms, and concurrent systems. He is a recipient of the National Science Foundation's CAREER Award. He is a member of the ACM and a senior member of the IEEE.