Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 68 (2008) 729–751
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Data-stream-based global event monitoring using pairwise interactions☆

Punit Chandra, Ajay D. Kshemkalyani*

Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607, United States

Received 14 November 2006; received in revised form 16 October 2007; accepted 23 January 2008
Available online 4 March 2008

Abstract

The problem of global state observation is fundamental to distributed systems and to the analysis of data streams. Many interactions in distributed systems can be analyzed in terms of the building block formed by the pairwise interactions of intervals at two processes. Considering causality-based pairwise interactions by which two processes may interact with each other, there are 40 orthogonal interaction types. For each pair of processes (P_i, P_j), let interaction type $r_{i,j}$ be of interest. This paper examines the problem: "If a global state of interest to an application is specified in terms of such pairwise interaction types, one per pair of processes, how can such a global state be detected?" A solution identifies a global state in which the interaction type specified for each process pair is satisfied. This paper formulates the specific conditions on the communication structures to determine which of the intervals being examined at any time may never satisfy the stipulated interaction type for that pair of processes, and therefore that interval(s) need no longer be considered as forming a part of any solution. Based on this theory, the paper proposes two on-line distributed algorithms to solve the problem.

Keywords: Causality; Data streams; Data fusion; Distributed system; Global state; Interactions; Intervals; Snapshot; State observation

1. Introduction

A global state is represented by a collection of local states, one from each process. The problem of global state observation in a consistent manner is fundamental to distributed systems, as identified by Chandy's and Lamport's seminal paper on recording global states [14]. Furthermore, the problem of monitoring and analyzing data streams is becoming increasingly important, particularly for crisis management (see [13,38]). Often, data streams have to be analyzed on-line [1,2] under different constraints and modalities. For example, much research focuses on data streams in sensor networks [19,31,32], and in generic middleware roles [30,36].

In this paper, we consider applications that require the consistent observation and continuous processing of data in the data streams. The consistent observation of the global state [14,24] requires observing the causality constraints among the events in the execution [28]. This paper generalizes the problem of global state observation to executions where the "event" at a process is a high-level abstract event that can contain multiple events that span across a time *interval*. The *interval* has the property that a predicate defined on local variables is always true in the interval and is false immediately before and immediately after the interval. The semantics of the interval depend on the predicate which is application-specific [17,18,20,22,25,27,29]; application areas such as sensor networks, distributed debugging, deadlock characterization [26], predicate detection [7,11,12], checkpointing [3–5,33,34], and industrial process control model such intervals. Such high-level abstract events at processes and the corresponding time intervals that they span have been explicitly studied [18,20,21,29].

Fig. 1(a) shows a consistent global state $GS2$ and an inconsistent global state $GS1$ in a distributed execution where events are modeled as atomic send, receive, or internal events,

☆ Some portions of this paper have appeared in [P. Chandra, A.D. Kshemkalyani, Detection of orthogonal interval relations, in: Proceedings 9th International High Performance Computing Conference (HiPC), in: Lecture Notes in Computer Science, vol. 2552, Springer-Verlag, 2002, pp. 323–333; P. Chandra, A.D. Kshemkalyani, Analysis of interval-based global state detection, in: Proceedings 2nd Int. Conference on Distributed Computing and Internet Technology (ICDCIT), in: Lecture Notes in Computer Science, vol. 3816, Springer, 2005, pp. 203–216; P. Chandra, A.D. Kshemkalyani, Global state detection based on peer-to-peer interactions, in: Proceedings IFIP Int. Conference on Embedded and Ubiquitous Computing (EUC), in: Lecture Notes in Computer Science, vol. 3824, Springer, 2005, pp. 560–571].

* Corresponding author.
E-mail address: ajayk@cs.uic.edu (A.D. Kshemkalyani).

0743-7315/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2008.01.006

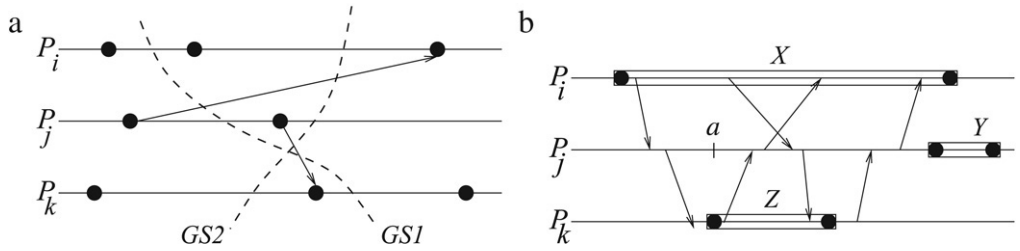

Fig. 1. (a) Event positions determine the consistency of a global state. (b) Interval positions define the interaction types in a global state.

as per the traditional model of a distributed execution [28]. Fig. 1(b) shows high-level abstract events that may contain multiple events and span across a time interval. The intervals are shown as a rectangle. An application-specific predicate defined at each process becomes true at the start of each interval shown and it becomes false at the end of each interval shown. Intervals X, Y, and Z occur at P_i, P_j, and P_k, respectively. Observe that if interval Y were to begin at event a marked by a vertical line, the inherent causality-based *interaction type* of Y with respect to Z as well as with respect to X would be different from what is shown. Thus, the relative placement of one interval with respect to another defines different interaction types between the interval pair.

It has been observed that causality-based interactions in distributed executions can be analyzed in terms of the building block formed by point-to-point pairwise interactions of intervals at two processes [20]. A detailed analysis of the causality-based pairwise interactions by which two processes may interact with each other identified 29 (40) causality-based orthogonal interaction types, denoted as $\mathfrak{R}$, under the dense (and non-dense) time model, respectively [20]. The orthogonality of the interaction types in $\mathfrak{R}$ implies that no interaction type can be expressed in terms of the other interaction types. Each interaction type in $\mathfrak{R}$ between a pair of intervals is essentially a *relationship* between the two intervals. Hence, the interaction types in $\mathfrak{R}$ are also interchangeably termed as relations. For each pair of processes (P_i, P_j), let interaction type $r_{i,j}$ be of interest. This paper examines the state detection problem: "If a global state of interest to an application is specified in terms of such pairwise interaction types, one per pair of processes, how can such a global state be detected?" For any relationship $r \in \mathfrak{R}$, let $r_{i,j}(X_i, Y_j)$ denote that r holds for interval X_i at process P_i and interval Y_j at process P_j. The above state detection problem is formally formulated as the following problem **DOOR** for the Detection of Orthogonal Relations [6,22].

Problem DOOR: Given a relation $r_{i,j}$ from $\mathfrak{R}$ for each pair of processes P_i and P_j, devise a distributed on-line algorithm to identify the intervals, if they exist, one from each process, such that each relation $r_{i,j}$ is satisfied by the (P_i, P_j) process pair.

Devising an efficient on-line algorithm to solve problem **DOOR** is a challenge because of having to track the intervals at different processes and to determine the pairwise orthogonal relations in a search space of an exponential number of global states. In this paper, we first identify the underlying principles that can be used to solve problem **DOOR**. We then propose two distributed algorithms to solve the problem.

Summary of results and contributions:

1. To devise any efficient solution to problem **DOOR**, this paper formulates specific conditions on the structure of the causal communication patterns to determine which of two intervals being examined from processes P_i and P_j may never satisfy $r_{i,j}$, and therefore that interval will never form part of any solution and should no longer be considered. This result is embodied as:
   - a basic principle that we prove in Theorem 1 — the main result, and
   - Lemma 4 — a useful lemma derived from the above theorem, that we will use to efficiently manage the distributed data structures in solving problem **DOOR**.

   Any algorithms to solve **DOOR** can leverage this principle.
2. The paper proposes two distributed on-line algorithms to solve problem **DOOR**, based on the above formulated conditions to determine when an interval no longer needs to be considered as a candidate for a solution. Let n be the number of processes, m_s be the total number of messages sent, m_r be the total number of messages received, and p be the maximum number of intervals at any process. For unicast communication, $m_s = m_r$. The first distributed algorithm uses $O(\min(np, 2m_s+2m_r))$ number of messages with a message size of $O(n^2)$. The second algorithm uses $O(n \cdot \min(np, 2m_s + 2m_r))$ number of messages with a message size of $O(n)$. For both the algorithms, the total space complexity across all the processes is $\min(4n^2p - 2np, 6m_sn + 4m_rn)$, and the total time complexity across all processes is $O(n \cdot \min(np, 2m_s + 2m_r))$. The performance of the algorithms is compared in Table 1.
3. Global state observation and predicate detection are fundamental problems in distributed systems. This paper provides an understanding of interval-based global states in terms of the causal communication patterns induced by the message-passing interactions in an execution [25].
4. The process of devising the principle (Theorem 1) which determines whether at least one interval in any pair of intervals being examined at any time can be identified as never forming a part of any solution (Lemma 4), gives a deeper insight into the nature of reasoning with causality in a distributed execution. Schwarz and Mattern have identified this as an important problem [37].

A centralized algorithm to solve problem **DOOR** (also referred to as problem *Fine_Rel*,) was given in [7]. The algorithm was presented without any formal discussion or analysis of the theoretical basis — embodied here in

Table 1
Summary of space, time, and message complexities

Metric	Algorithm 1	Algorithm 2
Total space complexity	$O(\min(2np(2n-1), 6m_s n + 4m_r n))$	$O(\min(2np(2n-1), 6m_s n + 4m_r n))$
Space per process (worst case)	$O(\min(4np - 2p, 4m_s n + 2m_s))$	$O(\min(4np - 2p, 4m_s n + 2m_s))$
Total time complexity	$O(n \cdot \min(np, 2m_s + 2m_r))$	$O(n \cdot \min(np, 2m_s + 2m_r))$
Time per process (worst case)	$O(n \cdot \min(10p, 20m_s + 4m_r))$	$O(n \cdot \min(10p, 20m_s + 4m_r))$
Total number of messages	$O(\min(np, 2m_s + 2m_r))$	$O(n \cdot \min(np, 2m_s + 2m_r))$
Message size (average case)	$O(n^2)$	$O(n)$
Message size (worst case)	$O(n \cdot \min(2(n-1)p + 2n, 2m_s + 2n))$	$O(\min(2(n-1)p + 2n, 2m_s + 2n))$
Total message space	$O(n^2 \cdot \min(np, 2m_s + 2m_r))$ average	$O(n^2 \cdot \min(4np - 2p, 6m_s + 4m_r))$

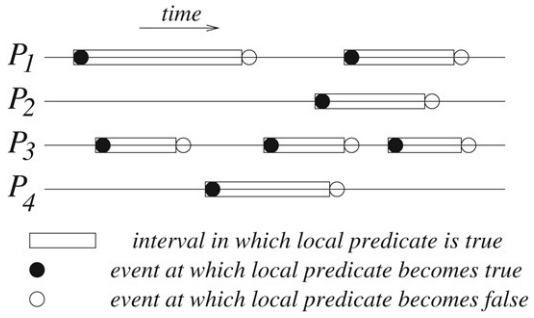

Fig. 2. Intervals at processes. Messages and send and receive events are not shown to simplify the diagram.

Theorems 1–3 and Lemma 4. That paper also showed how to perform predicate detection in the context of traditional modalities on predicates [12]. This paper gives the theory behind the results of [7] and then gives two distributed algorithms for **DOOR**. Distributed algorithms are more elegant than centralized algorithms and do not require a central server. Distributed algorithms result in a better workload and space complexity distribution as compared to a centralized one. The space complexity and the time complexity get distributed linearly with the number of processes. Thus, a distributed algorithm is more scalable than a centralized algorithm. Also, the network traffic gets distributed more uniformly in a distributed algorithm compared to the centralized approach where a traffic bottleneck gets created at the central process. Preliminary versions of this paper appeared in [6,8,9].

Section 2 gives the system model and background. Section 3 gives the theory used to determine which of two given intervals at different processes can never be part of a solution set. Section 4 shows how to track intervals. Sections 5 and 6 present the distributed algorithms to solve Problem **DOOR** based on the results derived in Section 3. Section 7 gives an extended specification of **DOOR**. Section 8 gives concluding remarks.

2. System model and background

System model. We assume an asynchronous distributed system in which n processes communicate by reliable message passing over logical FIFO channels [20,29]. The execution is modeled as $(E, \prec)$, where $\prec$ is an irreflexive partial ordering representing the causality or the "happens before" relation [28] on the event set E.

Definition 1 (*Causality or the "happens before" Relation* $\prec$). Event e happens before event e', denoted $e \prec e'$, if:

1. event e occurs before event e' at the same process, or
2. event e is the send of a message m and event e' is the receive of m, or
3. there is an event e'' such that e happens before e'' and e'' happens before e'.

E is partitioned into local executions at each process. Let E_i denote the linearly ordered set of events executed by process P_i. An event e executed by P_i is denoted e_i and may be of three types — an internal event, a send event, or a receive event. N denotes the set of all processes. A *cut* C is a subset of E such that if $e_i \in C$ then $(\forall e_i')\ e_i' \prec e_i \Longrightarrow e_i' \in C$. A *consistent cut* is a downward-closed subset of E, i.e., if $e \in C$ then $(\forall e')\ e' \prec e \Longrightarrow e' \in C$. A consistent cut denotes an execution prefix. Two special consistent cuts $\downarrow e$ and $e \uparrow$ can be defined for any event e.

Definition 2 (*Past and Future Cuts*). For any event e, cut $\downarrow e$ is the set of events $\{e' | e' \prec e\}$ that happen before e. Cut $e \uparrow$ is the set of events $\{e' | e' \not\succeq e\} \bigcup \{e_i, i = 1, \ldots, n \,|\, e_i \succeq e \bigwedge (\forall e_i' \prec e_i, e_i' \not\succeq e)\}$ up to and including the earliest events at each process for which e happens before the events.

The system state after the events in a cut is a *global state* [14]; if the cut is consistent, the corresponding system state is a consistent global state. We assume that vector clocks are available [16,35].

Intervals. As introduced in Section 1, the *intervals* of interest at each process are the durations during which an application-specific local predicate is true. See Fig. 2. An interval begins when the predicate becomes true, and ends when/just before the predicate becomes false. Each interval defines an abstract event of coarser granularity at a process, as studied by Lamport [29], Helary et al. [18], and Kshemkalyani [27]. Such higher-level abstract events can be used to identify a global state [27]. For example, in the context of checkpointing [4,5,15,33,34], ψ_i can be used to describe the kth checkpoint interval at P_i. To characterize deadlock [26,25], ψ_i denotes the interval from the time of incoming wait-for dependency to the time of an outgoing wait-for dependency, or vice versa; such intervals at different processes capture how dependency chains grow. For conjunctive predicate detection [7,12], ψ_i is the predicate on the local variables that the application wants to detect.

Table 2
Dependent relations for interactions between intervals [20]

Relation r	Expression for $r(X, Y)$	Test for $r(X_i, Y_j)$ using vector timestamps
R1	$\forall x \in X \forall y \in Y, x \prec y$	$V_j^-(Y_j)[i] \geq V_i^+(X_i)[i]$
R2	$\forall x \in X \exists y \in Y, x \prec y$	$V_j^+(Y_j)[i] \geq V_i^+(X_i)[i]$
R3	$\exists x \in X \forall y \in Y, x \prec y$	$V_j^-(Y_j)[i] \geq V_i^-(X_i)[i]$
R4	$\exists x \in X \exists y \in Y, x \prec y$	$V_j^+(Y_j)[i] \geq V_i^-(X_i)[i]$
S1	$\exists x \in X \forall y \in Y, x \not\preceq y \bigwedge y \not\preceq x$	$\exists x^0 \in X_i$: $V_j^-(Y_j)[j] \not< V_i^{x^0}[j] \wedge V_i^{x^0}[i] \not< V_j^+(Y_j)[i]$
S2	$\exists x_1, x_2 \in X \exists y \in Y, x_1 \prec y \prec x_2$	$\exists y^0 \in Y_j$: $V_i^+(X_i)[j] \not< V_j^{y^0}[j] \wedge V_j^{y^0}[i] \not< V_i^-(X_i)[i]$

The second column defines the relations. The third column gives the tests using vector timestamps.

Table 3
Definitions of the 40 orthogonal interaction types in $\mathfrak{R}$[20]

Orthogonal interaction type (on intervals X and Y)	Dependent relation $r(X, Y)$						Dependent relation $r(Y, X)$					
	R1	R2	R3	R4	S1	S2	R1	R2	R3	R4	S1	S2
$IA\ (= IQ^{-1})$	1	1	1	1	0	0	0	0	0	0	0	0
$IB\ (= IR^{-1})$	0	1	1	1	0	0	0	0	0	0	0	0
$IC\ (= IV^{-1})$	0	0	1	1	1	0	0	0	0	0	0	0
$ID\ (= IX^{-1})$	0	0	1	1	1	1	0	1	0	1	0	0
$ID'(= IU^{-1})$	0	0	1	1	0	1	0	1	0	1	0	1
$IE\ (= IW^{-1})$	0	0	1	1	1	1	0	0	0	1	0	0
$IE'(= IT^{-1})$	0	0	1	1	0	1	0	0	0	1	0	1
$IF\ (= IS^{-1})$	0	1	1	1	0	1	0	0	0	1	0	1
$IG\ (= IG^{-1})$	0	0	0	0	1	0	0	0	0	0	1	0
$IH\ (= IK^{-1})$	0	0	0	1	1	0	0	0	0	0	1	0
$II\ (= IJ^{-1})$	0	1	0	1	0	0	0	0	0	0	1	0
$IL\ (= IO^{-1})$	0	0	0	1	1	1	0	1	0	1	0	0
$IL'\ (= IP^{-1})$	0	0	0	1	0	1	0	1	0	1	0	1
$IM\ (= IM^{-1})$	0	0	0	1	1	0	0	0	0	1	1	0
$IN\ (= IM'^{-1})$	0	0	0	1	1	1	0	0	0	1	0	0
$IN'\ (= IN'^{-1})$	0	0	0	1	0	1	0	0	0	1	0	1
$ID''\ (= (IUX)^{-1})$	0	0	1	1	0	1	0	1	0	1	0	0
$IE''\ (= (ITW)^{-1})$	0	0	1	1	0	1	0	0	0	1	0	0
$IL''\ (= (IOP)^{-1})$	0	0	0	1	0	1	0	1	0	1	0	0
$IM''\ (= (IMN)^{-1})$	0	0	0	1	0	0	0	0	0	1	1	0
$IN''\ (= (IMN')^{-1})$	0	0	0	1	0	1	0	0	0	1	0	0
$IMN''\ (= (IMN'')^{-1})$	0	0	0	1	0	0	0	0	0	1	0	0

The upper part gives the 29 interaction types for dense time. The lower part gives 11 additional interaction types for non-dense time.

In our discrete event system model, an interval X_i at process P_i is identified by the (totally ordered) subset of adjacent events of E_i, beginning from the event that makes the predicate true up to the event that precedes the event that makes the predicate false. Intervals are denoted by capitals such as X, Y, and Z. The subscripts are omitted when not necessary or when the context is clear. Lower-case alphabet x denotes an individual event in an abstract event X.

Definition 3 (*Interval*). For a predicate ψ_i defined on process P_i, an `interval` $X_i(\psi_i) \subseteq E_i$, satisfies the following.

- $\forall e_i \in E_i$, if $\min(X_i) \prec e_i \prec \max(X_i)$ then $e_i \in X_i$
- ψ_i becomes true at $\min(X_i)$ and becomes false at $next(\max(X_i))$
- $\forall x_i$ such that $\min(X_i) \prec x_i \prec \max(X_i)$, x_i does not falsify ψ_i.

The execution history at P_i is $\langle s_i^0, e_i^1, s_i^1, e_i^2, s_i^2 \ldots e_i^k, s_i^k, \ldots \rangle$, where s_i^k is the state after event e_i^k. So the definition of X_i attempts to capture the time duration in which ψ_i remains true. ψ_i is application-specific and so we henceforth refer to intervals without using this parameter.

Orthogonal interaction types/relationships. There are 29 or 40 possible mutually orthogonal ways in which any two durations can be related to each other, depending on whether the *dense* or the *non-dense* time model is assumed [20]. Informally, with dense time, $\forall z1, z2$ in interval Z, $z1 \prec z2 \Longrightarrow \exists z \in Z \mid z1 \prec z \prec z2$. These orthogonal interaction types were identified by first using the six dependent relations defined in the first two columns of Table 2. Relations R1 (strong precedence), R2 (partially strong precedence), R3 (partially weak precedence), R4 (weak precedence) define *causality conditions*; S1 and S2 define *coupling conditions*. The tests in the third column are explained later in this section.

- (Dense time:) The 29 orthogonal interaction types between a pair of intervals are given in the upper part of Table 3. For any intervals X and Y, the orthogonal interaction types listed in the first column are specified using boolean vectors of length 12 on the dependent relations R1–R4 and S1–S2

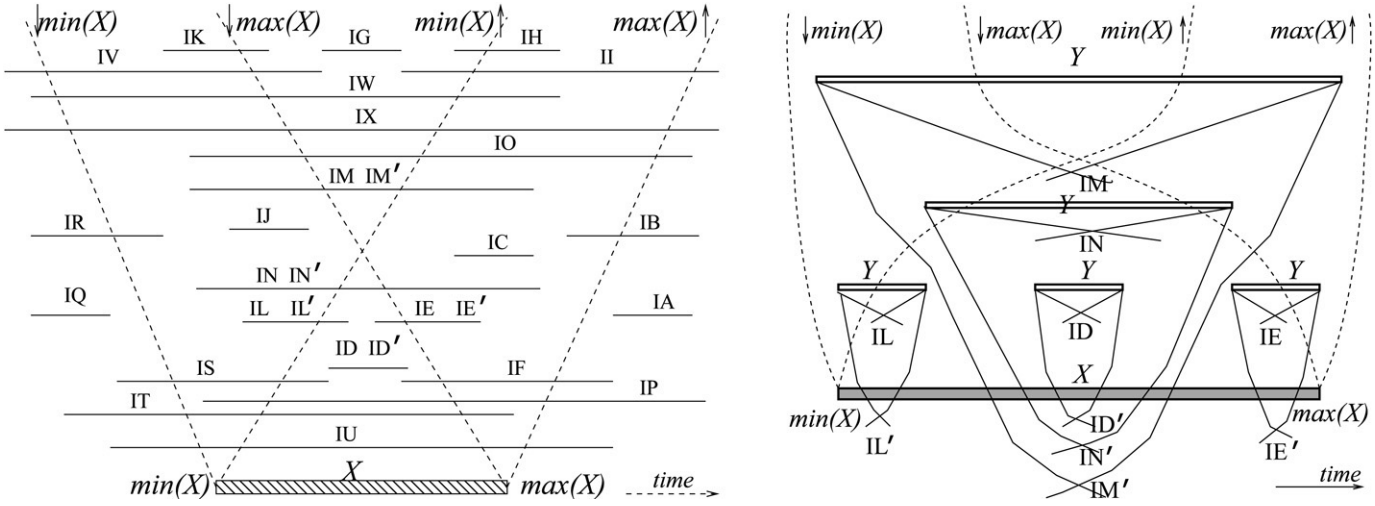

Fig. 3. Timing diagram for orthogonal interaction types between intervals under the dense time model [20].

(six bits for $r(X, Y)$ and six bits for $r(Y, X)$). Of the 29 interaction types, there are 13 pairs of inverses, while three are self-inverses. E.g., *IB* and *IR* are inverses because $IB(X, Y) = IR(Y, X)$. The interaction types are illustrated in Fig. 3, where interval X is shown by a box and interval Y is in different positions relative to X. Each position of Y is labeled by an interaction type. Five positions of Y have two labels each — the differences between these labeled interaction types are depicted to the right.

- (Non-dense time:) The non-dense time model, which captures the reality that event sequences and real clocks are both discrete, permits 11 additional interaction types between a pair of intervals, defined in the lower part of Table 3. Of these, there are five pairs of inverses, while one is its own inverse. For illustrations, refer to [20].

The set of 40 relations is denoted as $\mathfrak{R}$.

Example specification of DOOR: In a system with three processes P_i, P_j, and P_k, detect a global state satisfying $IC(X_i, Y_j)$, $IA(Z_k, Y_j)$, and $IX(Z_k, X_i)$, or alternately, in terms of inverses, $IV(Y_j, X_i)$, $IQ(Y_j, Z_k)$, and $ID(X_i, Z_k)$. Observe using Fig. 3 that the intervals in Fig. 1(b) satisfy this specification on the interval-based global state.

Another example specification of DOOR: Detect a global state satisfying $IE(X_i, Y_j)$, $IP(Z_k, Y_j)$, and $IX(Z_k, X_i)$, or alternately, in terms of inverses, $IW(Y_j, X_i)$, $IL'(Y_j, Z_k)$, and $ID(X_i, Z_k)$. In Fig. 1(b), if interval Y_j were to begin from event a, this specification is satisfied.

Evaluating an orthogonal interaction type. Each of the relations in $\mathfrak{R}$ can be tested for using the bit-patterns for the dependent relations, as given in Table 3. The tests for $R1$, $R2$, $R3$, $R4$, $S1$, and $S2$ using vector timestamps are given in Table 2. V_i^- and V_i^+ denote the vector timestamp at process P_i at the start of an interval and the end of an interval, respectively. V_i^x denotes the vector timestamp of event x_i at process P_i. $R1$–$R4$ have $O(1)$ cost. Observe from the tests for S1 and S2 that it is not sufficient to retain the timestamps of only the start and end of intervals. Unless some intelligent mechanism is used, the timestamp of every single event might be needed.

3. The elimination principle

Each process P_i, $1 \leq i \leq n$, maintains information about the timestamps of the start and end of its local intervals, and certain other local information, in a local queue Q_i. The n processes collectively run the distributed algorithms to process the information in the local queues and solve problem **DOOR**. Specifically, the intervals from the queues are examined pairwise to check if the relation $r_{i,j}$ specified for P_i and P_j holds. Devising an efficient algorithm to solve problem **DOOR** is a challenge because of having to track the intervals at different processes and to search p^n combinations of intervals that might satisfy the problem specification. We formulate the exact condition on when, for any interval pair from P_i and from P_j, the interval from P_i may potentially satisfy $r_{i,j}$ with a future interval from P_j, and therefore the interval at P_i must be tracked. This gives a basic principle –Theorem 1 – that can be used to efficiently manage the distributed data structures. This theorem in the form of Lemma 4 will be used in practice by the proposed algorithms to solve **DOOR**. Specifically, we show that for a pair of intervals from P_i and from P_j being tested for $r_{i,j}$, if the relationship does not hold, then at least one of the intervals can never be part of any solution and its record can be deleted without affecting the correctness of the solution (Lemma 4).

We assume that interval X occurs at P_i and interval Y occurs at P_j. For any two intervals X and X' that occur at the same process, if $R1(X, X')$, then we say that X is a *predecessor* of X' and X' is a *successor* of X.

We next define the prohibition function $\mathcal{H}(r_{i,j})$ and the relation $\rightsquigarrow$ which will be used to formulate and prove the main results. These definitions are modified from the definitions in [7], to give more structure and simplify the technical details.

Intuitively, for each $r_{i,j} \in \mathfrak{R}$, we define a *prohibition function* $\mathcal{H}(r_{i,j})$ as the set of all relations R such that if $R(X, Y)$ is true, then $r_{i,j}(X, Y')$ can never be true for some successor Y' of Y. $\mathcal{H}(r_{i,j})$ is the set of relations that prohibit $r_{i,j}$ from being true in the future.

Table 4
Axioms for the causality relations of Table 2 [20]

Axiom label	$r_1(X, Y) \bigwedge r_2(Y, Z) \Longrightarrow r(X, Z)$
AL1	$R1(X, Y) \bigwedge R2(Y, Z) \Longrightarrow R2(X, Z)$
AL2	$R1(X, Y) \bigwedge R3(Y, Z) \Longrightarrow R1(X, Z)$
AL3	$R1(X, Y) \bigwedge R4(Y, Z) \Longrightarrow R2(X, Z)$
AL4	$R2(X, Y) \bigwedge R1(Y, Z) \Longrightarrow R1(X, Z)$
AL5	$R3(X, Y) \bigwedge R1(Y, Z) \Longrightarrow R3(X, Z)$
AL6	$R4(X, Y) \bigwedge R1(Y, Z) \Longrightarrow R3(X, Z)$
AL7	$R2(X, Y) \bigwedge R3(Y, Z) \Longrightarrow true$
AL8	$R2(X, Y) \bigwedge R4(Y, Z) \Longrightarrow true$
AL9	$R3(X, Y) \bigwedge R2(Y, Z) \Longrightarrow R4(X, Z)$
AL10	$R4(X, Y) \bigwedge R2(Y, Z) \Longrightarrow R4(X, Z)$
AL11	$R3(X, Y) \bigwedge R4(Y, Z) \Longrightarrow R4(X, Z)$
AL12	$R4(X, Y) \bigwedge R3(Y, Z) \Longrightarrow true$
AL13	$R1(X, Y) \Longrightarrow \overline{S1(X, Y)} \bigwedge \overline{S2(X, Y)} \bigwedge \overline{R4(Y, X)} \bigwedge \overline{S1(Y, X)} \bigwedge \overline{S2(Y, X)}$
AL14	$R2(X, Y) \Longrightarrow \overline{S1(X, Y)} \bigwedge \overline{R2(Y, X)}$
AL15	$R3(X, Y) \Longrightarrow \overline{R3(Y, X)} \bigwedge \overline{S1(Y, X)}$
AL16	$R4(X, Y) \Longrightarrow \overline{R1(Y, X)}$
AL17	$S1(X, Y) \Longrightarrow \overline{R2(X, Y)} \bigwedge \overline{R3(Y, X)} \bigwedge \overline{S2(Y, X)}$
AL18	$S2(X, Y) \Longrightarrow \overline{R1(X, Y)} \bigwedge R4(X, Y) \bigwedge \overline{R1(Y, X)} \bigwedge R4(Y, X) \bigwedge \overline{S1(Y, X)}$

$\overline{R}$ stands for "R is *false*".

Definition 4 (*Prohibition Function*). Prohibition function $\mathcal{H} : \mathfrak{R} \rightarrow 2^{\mathfrak{R}}$ is defined to be $\mathcal{H}(r_{i,j}) = \{R \in \mathfrak{R} \mid$ if $R(X, Y)$ is true then $r_{i,j}(X, Y')$ is false for all Y' that succeed $Y\}$.

Two relations R' and R'' in $\mathfrak{R}$ are related by the *allows* relation $\rightsquigarrow$ if the occurrence of $R'(X, Y)$ does not prohibit $R''(X, Y')$ for some successor Y' of Y.

Definition 5 (*Allows Relation*). The "allows" relation $\rightsquigarrow$ is a relation on $\mathfrak{R} \times \mathfrak{R}$ such that $R' \rightsquigarrow R''$ if the following holds: if $R'(X, Y)$ is true then $R''(X, Y')$ can be true for some Y' that succeeds Y.

Lemma 1. *If $R \in \mathcal{H}(r_{i,j})$ then $R \not\rightsquigarrow r_{i,j}$ else if $R \notin \mathcal{H}(r_{i,j})$ then $R \rightsquigarrow r_{i,j}$.*

Proof. If $R \in \mathcal{H}(r_{i,j})$, using Definition 4, it can be inferred that $r_{i,j}$ is false for all Y' that succeed Y. This does not satisfy Definition 5. Hence $R \not\rightsquigarrow r_{i,j}$. If $R \notin \mathcal{H}(r_{i,j})$, it follows that $r_{i,j}$ can be true for some Y' that succeeds Y. This satisfies Definition 5 and hence $R \rightsquigarrow r_{i,j}$. ■

Given that $R'(A, B) \rightsquigarrow R''(A, B')$, where R' and R'' are orthogonal relations from $\mathfrak{R}$, the following Lemma 2 shows some relations between interval pairs A, B and A, B' in terms of the dependent set of causality relations $R1 - R4$. These relations will be useful to show a critical relationship between R'^{-1} and R''^{-1} (Theorem 1) that allows efficient pruning of intervals on the queues in any algorithm to solve Problem **DOOR**.

Lemma 2. *If $R' \rightsquigarrow R''$, $R'(A, B)$ and $R''(A, B')$, where R', $R'' \in \mathfrak{R}$, then the statements in Table 5 are true.*

Proof. As $R' \rightsquigarrow R''$ and $R'(A, B)$ is true, we can safely assume that there can exist an interval B' that succeeds B and such that $R''(A, B')$ is true. Now consider axioms AL2, AL4, AL5 and AL6 given in Table 4. Applying the following transformations gives statements T1 to T4 of Table 5, respectively.

Table 5
Given $R' \rightsquigarrow R''$, $R'(A, B)$ and $R''(A, B')$, for R', $R'' \in \mathfrak{R}$, statements between interval pairs A, B and A, B' using the dependent relations $R1$–$R4$

Statement label	Statements
T1	$R1(A, B) \Longrightarrow R1(A, B')$
T2	$R2(A, B) \Longrightarrow R1(A, B')$
T3	$R3(A, B) \Longrightarrow R3(A, B')$
T4	$R4(A, B) \Longrightarrow R3(A, B')$
T5	$R1(B', A) \Longrightarrow R1(B, A)$
T6	$R2(B', A) \Longrightarrow R2(B, A)$
T7	$R3(B', A) \Longrightarrow R1(B, A)$
T8	$R4(B', A) \Longrightarrow R2(B, A)$

1. Substitute A, B, B' for X, Y, Z, respectively, in Table 4.
2. As B' succeeds B, substitute true for $R1(B, B')$, $R2(B, B')$, $R3(B, B')$, and $R4(B, B')$.

Consider axioms AL1, AL2, AL3, and AL4 given in Table 4. Applying the following transformations gives statements T5 to T8, of Table 5, respectively.

1. Substitute B, B', A for X, Y, Z, respectively, in Table 4.
2. As B' succeeds B, substitute true for $R1(B, B')$, $R2(B, B')$, $R3(B, B')$, and $R4(B, B')$. ■

We now show an important result between any two relations in $\mathfrak{R}$ that satisfy the "allows" relation, and the existence of the "allows" relation between their respective inverses. Specifically, if R' allows R'' (and $R' \neq R''$), then Theorem 1 shows that R'^{-1} necessarily does not allow relation R''^{-1}. This theorem is illustrated in Fig. 4. Part (a) shows $R'(X, Y)$ and $R''(X, Y')$, i.e., $R' \rightsquigarrow R''$. By definition, we have $R'^{-1}(Y, X)$ and $R''^{-1}(Y', X)$. The question then is, as posed in part (b), whether $R''^{-1}(Y, X')$ holds. Theorem 1 shows that $R''^{-1}(Y, X')$ cannot hold, and hence R'^{-1} does not allow R''^{-1}. This theorem is used in deriving Lemma 4 which will be practically used in deriving solutions to problem **DOOR**, and to prove the correctness of such solutions.

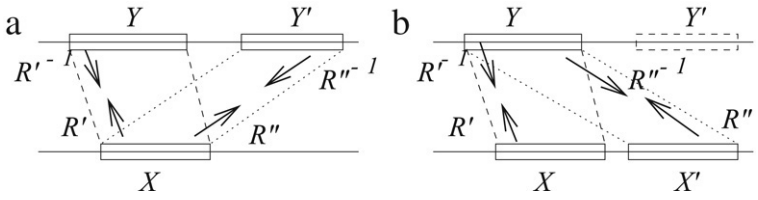

Fig. 4. Illustration of Theorem 1.

Theorem 1. *For $R', R'' \in \mathfrak{R}$ and $R' \neq R''$, if $R' \rightsquigarrow R''$ then $R'^{-1} \not\rightsquigarrow R''^{-1}$.*

Proof. We prove by contradiction. The assumption using which we show a contradiction is the following.

$$R'(X, Y) \texttt{ is true},\ R'(X, Y) \rightsquigarrow R''(X, Y') \quad \texttt{and}$$
$$R'^{-1}(Y, X) \rightsquigarrow R''^{-1}(Y, X'). \tag{1}$$

By Lemma 2, T1 to T8 must hold for both $R'(X, Y) \rightsquigarrow R''(X, Y')$ and $R'^{-1}(Y, X) \rightsquigarrow R''^{-1}(Y, X')$. So we get two sets of constraints for intervals X, X', Y, and Y' in terms of the dependent causality relations $R1$ to $R4$.

Consider $R'(X, Y) \rightsquigarrow R''(X, Y')$. Instantiating A by X, B by Y, and B' by Y' in T1–T8, we have the following set of constraints that need to be satisfied.

C1: $R1(X, Y) \Rightarrow R1(X, Y')$
C2: $R2(X, Y) \Rightarrow R1(X, Y')$
C3: $R3(X, Y) \Rightarrow R3(X, Y')$
C4: $R4(X, Y) \Rightarrow R3(X, Y')$
C5: $R1(Y', X) \Rightarrow R1(Y, X)$
C6: $R2(Y', X) \Rightarrow R2(Y, X)$
C7: $R3(Y', X) \Rightarrow R1(Y, X)$
C8: $R4(Y', X) \Rightarrow R2(Y, X)$.

Now consider $R'^{-1}(Y, X) \rightsquigarrow R''^{-1}(Y, X')$. Instantiating A by Y, B by X, and B' by X' in T1–T8, we have the following set of constraints that need to be satisfied.

C9: $R1(Y, X) \Rightarrow R1(Y, X')$
C10: $R2(Y, X) \Rightarrow R1(Y, X')$
C11: $R3(Y, X) \Rightarrow R3(Y, X')$
C12: $R4(Y, X) \Rightarrow R3(Y, X')$
C13: $R1(X', Y) \Rightarrow R1(X, Y)$
C14: $R2(X', Y) \Rightarrow R2(X, Y)$
C15: $R3(X', Y) \Rightarrow R1(X, Y)$
C16: $R4(X', Y) \Rightarrow R2(X, Y)$.

From Eq. (1), it can be seen that the interval pairs (Y', X) and (Y, X') both are related by the orthogonal relation R''^{-1}. Hence $r(Y', X) \Leftrightarrow r(Y, X')$, where r is any of the six dependent relations given in Table 2. Thus replacing $r(Y, X')$ by $r(Y', X)$ in C9 to C12, we have the following constraints.

C17: $R1(Y, X) \Rightarrow R1(Y', X)$
C18: $R2(Y, X) \Rightarrow R1(Y', X)$
C19: $R3(Y, X) \Rightarrow R3(Y', X)$
C20: $R4(Y, X) \Rightarrow R3(Y', X)$.

From Eq. (1), it can also be seen in a similar way that the interval pairs (X, Y') and (X', Y) both are related by the orthogonal relation R''. Hence $r(X, Y') \Leftrightarrow r(X', Y)$, where r is any of the six dependent relations given in Table 2. Thus replacing $r(X', Y)$ by $r(X, Y')$ in C13 to C16, we have the following constraints.

C21: $R1(X, Y') \Rightarrow R1(X, Y)$
C22: $R2(X, Y') \Rightarrow R2(X, Y)$
C23: $R3(X, Y') \Rightarrow R1(X, Y)$
C24: $R4(X, Y') \Rightarrow R2(X, Y)$.

The two constraint sets (C1)–(C8) and (C17)–(C24) given above can be combined to obtain restrictions on the type of interactions (given in Table 3) that $R'(X, Y)$ can belong to. Combining constraints C1 to C4 with constraints C21 to C24 gives

$$R1(X, Y) \vee R2(X, Y) \vee R3(X, Y) \vee R4(X, Y) \Rightarrow R1(X, Y).$$

Note from the definitions in Table 2 that $R1(X, Y) \Rightarrow R2(X, Y) \wedge R3(X, Y) \wedge R4(X, Y)$. Thus,

$$R1(X, Y) \vee R2(X, Y) \vee R3(X, Y) \vee R4(X, Y)$$
$$\Rightarrow R1(X, Y) \wedge R2(X, Y) \wedge R3(X, Y) \wedge R4(X, Y). \tag{2}$$

The above implication implies that relations $R1(X, Y)$, $R2(X, Y)$, $R3(X, Y)$, and $R4(X, Y)$ are either all true or all false.

Using a similar approach, combining constraints C17 to C20 with constraints C5 to C8 gives

$$R1(Y, X) \vee R2(Y, X) \vee R3(Y, X) \vee R4(Y, X)$$
$$\Rightarrow R1(Y, X) \wedge R2(Y, X) \wedge R3(Y, X) \wedge R4(Y, X). \tag{3}$$

This means relations $R1(Y, X)$, $R2(Y, X)$, $R3(Y, X)$, and $R4(Y, X)$, are either all true or all false.

Implications (2) and (3) restrict the interaction type (given in Table 3) to which $R'(X, Y)$ can belong. We now examine all the restricted cases to which $R'(X, Y)$ can belong, i.e., when $R1(X, Y)$ to $R4(X, Y)$ are all true, and when $R1(X, Y)$ to $R4(X, Y)$ are all false, and show that $R'(X, Y)$ cannot exist; which is a contradiction to Eq. (1). Using Implication (2), there are two broad cases for $R'(X, Y)$.

Case 1. $R1(X, Y)$, $R2(X, Y)$, $R3(X, Y)$, and $R4(X, Y)$ are all true.

From Table 3, $R'(X, Y)$ must be *IA*. Further, from constraints C1 to C4, we get

$$R1(X, Y'), R2(X, Y'), R3(X, Y'), R4(X, Y') \texttt{ are true}. \tag{4}$$

Using axioms AL13 to AL16, we get $R1(Y, X)$, $R2(Y, X)$, $R3(Y, X)$, $R4(Y, X)$, $S1(X, Y)$, $S2(X, Y)$, $S1(Y, X)$, $S2(Y, X)$

are all false. Now substituting X, Y' for X, Y in axioms AL13 to AL16, we get

$$R1(Y', X), R2(Y', X), R3(Y', X), R4(Y', X), S1(X, Y'), S2(X, Y'), S1(Y', X), S2(Y', X) \texttt{ are false}. \quad (5)$$

From Table 3, $R''(X, Y')$ must be IA. Thus, the only combination by which to instantiate R' and R'' so that they satisfy the case assumption and Eqs. (4) and (5) is IA. Thus, we have $R'(X, Y) = R''(X, Y') = IA$. As $R' \neq R''$ by the theorem statement, this case cannot exist.

Case 2. $R1(X, Y)$, $R2(X, Y)$, $R3(X, Y)$ and $R4(X, Y)$ are all false.

From Table 3, $R'(X, Y) \in \{IG, IK, IJ, IQ, IR, IV\}$. There are two subcases considering Implication (3).

1. $R1(Y, X)$, $R2(Y, X)$, $R3(Y, X)$, and $R4(Y, X)$ are all true. Then $R'(X, Y)$ must be IQ. From constraints C17 to C20, we get

   $$R1(Y', X), R2(Y', X), R3(Y', X), R4(Y', X) \texttt{ are true}. \quad (6)$$

   Substituting Y', X for X, Y in axioms AL13 to AL16, we get

   $$R1(X, Y'), R2(X, Y'), R3(X, Y'), R4(X, Y'), S1(X, Y'), S2(X, Y'), S1(Y', X), S2(Y', X) \texttt{ are false}. \quad (7)$$

   From Table 3, $R''(X, Y')$ must be IQ. Thus, the only combination by which to instantiate R' and R'' so that they satisfy the case/subcase assumptions and Eqs. (6) and (7) is IQ. Thus, we have $R'(X, Y) = R''(X, Y') = IQ$. As $R' \neq R''$ by the theorem statement, this case cannot exist.
2. $R1(Y, X)$, $R2(Y, X)$, $R3(Y, X)$, and $R4(Y, X)$ are all false. Then $R'(X, Y)$ must be IG. From constraints C5 to C8, we get

   $$R1(Y', X), R2(Y', X), R3(Y', X), R4(Y', X) \texttt{ are false}. \quad (8)$$

   Now substituting Y', X for X, Y in axioms AL13 to AL16, we get

   $$R1(X, Y'), R2(X, Y'), R3(X, Y'), R4(X, Y') \texttt{ are false}. \quad (9)$$

   From Table 3, $R''(X, Y')$ must be IG. Thus, the only combination by which to instantiate R' and R'' so that they satisfy the case/subcase assumptions and Eqs. (8) and (9) is IG. Thus, we have $R'(X, Y) = R''(X, Y') = IG$. As $R' \neq R''$ by the theorem statement, this case cannot exist.

Hence there cannot exist a case where $R'(X, Y) \rightsquigarrow R''(X, Y')$ and $R'^{-1}(Y, X) \rightsquigarrow R''^{-1}(Y', X)$. This contradicts the assumption in Eq. (1), proving the theorem. ■

Theorem 2. *For any $R', R'' \in \mathfrak{R}$, and for any Y' that is a successor of Y, $R'(X, Y) \in \mathcal{H}(R''(X, Y'))$ if and only if*

1. $R2(Y, X) \bigwedge \overline{R1(Y, X)} \bigwedge \overline{R4(X, Y)} \bigwedge R3(Y', X)$ *or*
2. $R2(Y, X) \bigwedge R4(X, Y) \bigwedge \overline{R3(X, Y')}$ *or*
3. $\overline{R4(X, Y)} \bigwedge \overline{R2(Y, X)} \bigwedge R4(Y', X)$ *or*
4. $(\overline{R2(Y, X)} \bigwedge \overline{R2(X, Y)} \bigwedge R4(X, Y)) \bigwedge (\overline{R3(X, Y')} \bigvee R4(Y', X))$ *or*
5. $R2(X, Y) \bigwedge \overline{R1(X, Y')}$.

Proof. Observe from the definition of *allows* that whether one orthogonal relation $R'(X, Y)$ allows another $R''(X, Y')$ is based on the values of dependent relations $R1$–$R4$ using which the orthogonal relations are defined. $R1$–$R4$ determine the *allows* relation because they fundamentally determine the orthogonal relationship that results by the placement of the start of Y' after the completion of Y. However, the coupling relations $S1$ and $S2$ are not involved in determining the *allows* relation because they deal with interactions between X and Y, or between X and Y', internal to the interval pair.

From Lemma 1, $R' \in \mathcal{H}(R'')$ if and only if $R' \not\rightsquigarrow R''$. We now prove the theorem with the aid of Figs. 4 and 5. Fig. 5 shows that space-time can be partitioned into six regions I–VI with respect to any interval X. Graphically, an event on the line depicting the border of $\min(X)\uparrow$ or of $\max(X)\uparrow$ belongs to the right side region. An event on the line depicting the border of $\downarrow\min(X)$ or of $\downarrow\max(X)$ belongs to the left side region. $R'(X, Y) \in \mathcal{H}(R''(X, Y'))$ if and only if the end of Y does not permit the beginning of Y' to occur after it. This can be true if one of the following five cases holds:

1. Y ends in Region II; Y' begins in Region I
2. Y ends in Region III; Y' begins in Region I or II or IV
3. Y ends in Region IV; Y' begins in Region I or II or III
4. Y ends in Region V; Y' begins in Region I or II or III or IV
5. Y ends in Region VI; Y' begins in Region I or II or III or IV or V.

Table 6 gives the conditions for interval Y to end in a region, and for interval Y' to begin in a region, in terms of the dependent set of relations $R1$–$R4$. For each condition, the number of conjuncts equals the number of borders that constrain the region. For each of the five cases above, the conditions on Y' beginning in the union of regions can be observed from Fig. 5 and Table 2. We show the formal derivations using Table 6.

Case 1. *Y ends in Region* II; *Y' begins in Region* I. Conjuncting the conditions from Table 6, we get:

$$(R2(Y, X) \wedge \overline{R1(Y, X)} \wedge \overline{R4(X, Y)}) \bigwedge R3(Y', X).$$

Case 2. *Y ends in Region* III; *Y' begins in Region* I *or* II *or* IV. The condition for Y' beginning in Region I or II or IV gives the following.

$$R3(Y', X) \bigvee (R4(Y', X) \wedge \overline{R3(Y', X)} \wedge \overline{R3(X, Y')}) \bigvee (\overline{R4(Y', X)} \wedge \overline{R3(X, Y')}). \quad (10)$$

We have $a \vee b \vee c = a \bigvee (\overline{a}(b \vee c))$ where a, b, c are boolean variables. Substituting $R3(Y', X)$ for a, $R4(Y', X) \wedge \overline{R3(Y', X)} \wedge \overline{R3(X, Y')}$ for b, and $\overline{R4(Y', X)} \wedge \overline{R3(X, Y')}$ for c in Eq. (10), we get:

$$\begin{aligned} &R3(Y', X) \bigvee (\overline{R3(Y', X)} \bigwedge ((R4(Y', X) \wedge \overline{R3(Y', X)} \\ &\quad \wedge \overline{R3(X, Y')}) \bigvee (\overline{R4(Y', X)} \wedge \overline{R3(X, Y')}))) \\ &= R3(Y', X) \bigvee (\overline{R3(Y', X)} \wedge R4(Y', X) \wedge \overline{R3(X, Y')}) \end{aligned}$$

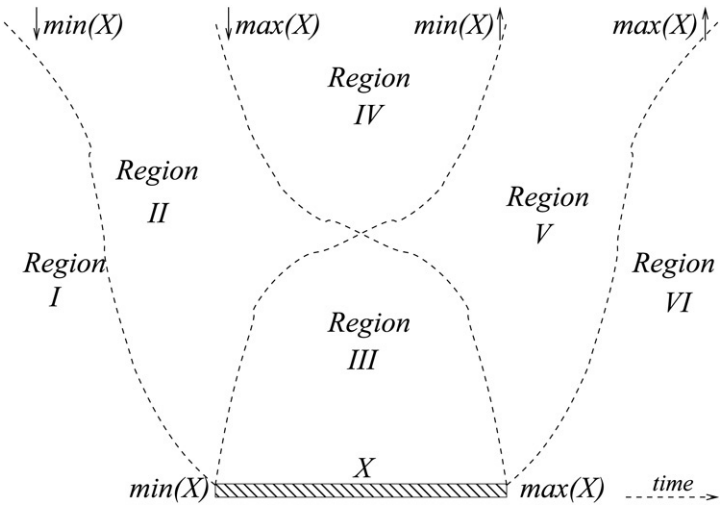

Fig. 5. Six causality-based regions identified by an interval.

Table 6
Given $R'(X, Y)$ and $R''(X, Y')$, for $R', R'' \in \mathfrak{R}$, conditions for Y to end in a region and for Y' to begin in a region with respect to interval X, using the dependent relations $R1 - R4$

Region	Y ends in region	Y' begins in region
I	$R1(Y, X)$	$R3(Y', X)$
II	$R2(Y, X) \wedge \overline{R1(Y, X)} \wedge \overline{R4(X, Y)}$	$R4(Y', X) \wedge \overline{R3(Y', X)} \wedge \overline{R3(X, Y')}$
III	$R2(Y, X) \wedge R4(X, Y)$	$\overline{R3(X, Y')} \wedge R4(Y', X)$
IV	$\overline{R4(X, Y)} \wedge \overline{R2(Y, X)}$	$\overline{R4(Y', X)} \wedge \overline{R3(X, Y')}$
V	$\overline{R2(Y, X)} \wedge \overline{R2(X, Y)} \wedge R4(X, Y)$	$R3(X, Y') \wedge \overline{R1(X, Y')} \wedge \overline{R4(Y', X)}$
VI	$R2(X, Y)$	$R1(X, Y')$

$$\begin{aligned} &\bigvee(\overline{R3(Y', X)} \wedge \overline{R4(Y', X)} \wedge \overline{R3(X, Y')}) \\ &= R3(Y', X) \bigvee((\overline{R3(Y', X)} \wedge \overline{R3(X, Y')}) \\ &\quad \bigwedge(R4(Y', X) \vee \overline{R4(Y', X)})) \\ &= R3(Y', X) \bigvee(\overline{R3(Y', X)} \wedge \overline{R3(X, Y')}). \end{aligned} \tag{11}$$

From axiom AL15, we get:

$$R3(Y', X) \Longrightarrow \overline{R3(X, Y')}. \tag{12}$$

Replacing $R3(Y', X)$ in Eq. (11) by $R3(Y', X) \wedge \overline{R3(X, Y')}$, we get:

$$\begin{aligned} &(R3(Y', X) \wedge \overline{R3(X, Y')}) \bigvee(\overline{R3(Y', X)} \wedge \overline{R3(X, Y')}) \\ &= \overline{R3(X, Y')} \bigwedge(R3(Y', X) \vee \overline{R3(Y', X)}) \\ &= \overline{R3(X, Y')}. \end{aligned} \tag{13}$$

Conjuncting this with the condition from Table 6 that Y ends in Region III gives $R2(Y, X) \wedge R4(X, Y) \wedge \overline{R3(X, Y')}$.

Case 3. *Y ends in Region* IV; *Y′ begins in Region* I *or* II *or* III. The condition for Y' beginning in Region I or II or III gives the following.

$$\begin{aligned} &R3(Y', X) \bigvee(R4(Y', X) \wedge \overline{R3(Y', X)} \wedge \overline{R3(X, Y')}) \\ &\bigvee(R4(Y', X) \wedge R3(X, Y')). \end{aligned} \tag{14}$$

By using Eq. (12) and replacing $R3(X, Y')$ in Eq. (14) by $R3(X, Y') \wedge \overline{R3(Y', X)}$, we get:

$$\begin{aligned} &R3(Y', X) \bigvee(R4(Y', X) \wedge \overline{R3(Y', X)} \wedge \overline{R3(X, Y')}) \\ &\quad \bigvee(R4(Y', X) \wedge R3(X, Y') \wedge \overline{R3(Y', X)}) \\ &= R3(Y', X) \bigvee(R4(Y', X) \wedge \overline{R3(Y', X)} \\ &\quad \bigwedge(\overline{R3(X, Y')} \vee R3(X, Y'))) \\ &= R3(Y', X) \bigvee(R4(Y', X) \wedge \overline{R3(Y', X)}). \end{aligned} \tag{15}$$

We have $R3(Y', X) \Longrightarrow R4(Y', X)$. Replacing $R3(Y', X)$ by $R3(Y', X) \wedge R4(Y', X)$ in Eq. (15), we get:

$$\begin{aligned} &(R3(Y', X) \wedge R4(Y', X)) \bigvee(R4(Y', X) \wedge \overline{R3(Y', X)}) \\ &= R4(Y', X) \bigwedge(R3(Y', X) \vee \overline{R3(Y', X)}) \\ &= R4(Y', X). \end{aligned} \tag{16}$$

Conjuncting this with the condition from Table 6 that Y ends in Region IV gives $\overline{R4(X, Y)} \wedge \overline{R2(Y, X)} \wedge R4(Y', X)$.

Case 4. *Y ends in Region* V; *Y′ begins in Region* I *or* II *or* III *or* IV. The condition for Y' beginning in Region I or II or III or IV can be expressed as the union of the conditions for Y' beginning in regions I,II, or IV, and of Y' beginning in regions I,II, or III. This is a disjunct of Eqs. (13) and (16), viz.,

$$R4(Y', X) \vee \overline{R3(X, Y')}. \tag{17}$$

Table 7
$\mathcal{H}(r_{i,j})$ for the 40 independent interaction types in $\mathfrak{R}$

Interaction type $r_{i,j} \in \mathfrak{R}$	$\mathcal{H}(r_{i,j})$	$\mathcal{H}(r_{j,i})$
$IA\ (= IQ^{-1})$	ϕ	$\mathfrak{R} \setminus \{IQ\}$
$IB (= IR^{-1})$	$\{IA, IB, IF, II, IP, IO, IU, IX, IUX, IOP\}$	$\mathfrak{R} \setminus \{IQ\}$
$IC\ (= IV^{-1})$	$\{IA, IB, IF, II, IP, IO, IU, IX, IUX, IOP\}$	$\mathfrak{R} \setminus \{IQ\}$
$ID\ (= IX^{-1})$	$\mathfrak{R} \setminus \{IQ, IS, IR, IJ, IL, IL', IL'', ID, ID', ID''\}$	$\mathfrak{R} \setminus \{IQ\}$
$ID'\ (= IU^{-1})$	$\mathfrak{R} \setminus \{IQ, IS, IR, IJ, IL, IL', IL'', ID, ID', ID''\}$	$\mathfrak{R} \setminus \{IQ\}$
$IE\ (= IW^{-1})$	$\mathfrak{R} \setminus \{IQ, IS, IR, IJ, IL, IL', IL'', ID, ID', ID''\}$	$\mathfrak{R} \setminus \{IQ\}$
$IE'\ (= IT^{-1})$	$\mathfrak{R} \setminus \{IQ, IS, IR, IJ, IL, IL', IL'', ID, ID', ID''\}$	$\mathfrak{R} \setminus \{IQ\}$
$IF\ (= IS^{-1})$	$\mathfrak{R} \setminus \{IQ, IS, IR, IJ, IL, IL', IL'', ID, ID', ID''\}$	$\mathfrak{R} \setminus \{IQ\}$
$IG\ (= IG^{-1})$	$\mathfrak{R} \setminus \{IQ, IR, IJ, IV, IK, IG\}$	$\mathfrak{R} \setminus \{IQ, IR, IJ, IV, IK, IG\}$
$IH\ (= IK^{-1})$	$\mathfrak{R} \setminus \{IQ, IR, IJ, IV, IK, IG\}$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$
$II\ (= IJ^{-1})$	$\mathfrak{R} \setminus \{IQ, IR, IJ, IV, IK, IG\}$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$
$IL\ (= IO^{-1})$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$
$IL'\ (= IP^{-1})$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$
$IM\ (= IM^{-1})$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$
$IN\ (= IM'^{-1})$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$
$IN'\ (= IN'^{-1})$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$
$ID''\ (= (IUX)^{-1})$	$\mathfrak{R} \setminus \{IQ, IS, IR, IJ, IL, IL', IL'', ID, ID', ID''\}$	$\mathfrak{R} \setminus \{IQ\}$
$IE''\ (= (ITW)^{-1})$	$\mathfrak{R} \setminus \{IQ, IS, IR, IJ, IL, IL', IL'', ID, ID', ID''\}$	$\mathfrak{R} \setminus \{IQ\}$
$IL''\ (= (IOP)^{-1})$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$
$IM''\ (= (IMN)^{-1})$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$
$IN''\ (= (IMN')^{-1})$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$
$IMN''\ (= (IMN'')^{-1})$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$	$\mathfrak{R} \setminus \{IQ, IR, IJ\}$

The upper part gives $\mathcal{H}$ for dense time. The lower part gives $\mathcal{H}$ for the 11 additional interaction types for non-dense time.

Conjuncting this with the condition from Table 6 that Y ends in Region V gives $(\overline{R2(Y,X)} \wedge \overline{R2(X,Y)} \wedge R4(X,Y)) \bigwedge (R4(Y',X) \vee \overline{R3(X,Y')})$.

Case 5. *Y ends in Region* VI; *Y' begins in Region* I *or* II *or* III *or* IV *or* V. Using Eq. (17) and Table 6, the condition for Y' beginning in Region I or II or III or IV or V gives the following.

$$R4(Y',X) \bigvee \overline{R3(X,Y')} \bigvee (R3(X,Y') \wedge \overline{R1(X,Y')} \wedge \overline{R4(Y',X)}). \tag{18}$$

As $R1(X,Y) \implies R3(X,Y)$, it means $\overline{R3(X,Y)} \implies \overline{R1(X,Y)}$. Substituting $\overline{R3(X,Y')} \wedge \overline{R1(X,Y')}$ for $\overline{R3(X,Y')}$ in Eq. (18), we get:

$$\begin{aligned}
&R4(Y',X) \bigvee (\overline{R3(X,Y')} \wedge \overline{R1(X,Y')}) \bigvee (R3(X,Y') \\
&\quad \wedge \overline{R1(X,Y')} \wedge \overline{R4(Y',X)}) \\
&= (\overline{R3(X,Y')} \wedge \overline{R1(X,Y')}) \\
&\quad \bigvee \left((R4(Y',X) \vee (R3(X,Y') \wedge \overline{R1(X,Y')})) \bigwedge true \right) \\
&= (\overline{R3(X,Y')} \wedge \overline{R1(X,Y')}) \bigvee R4(Y',X) \\
&\quad \bigvee (R3(X,Y') \wedge \overline{R1(X,Y')}) \\
&= R4(Y',X) \vee (\overline{R1(X,Y')} \wedge true) \\
&= R4(Y',X) \vee \overline{R1(X,Y')}.
\end{aligned} \tag{19}$$

From axiom $AL16$, we get $R4(Y',X) \implies \overline{R1(X,Y')}$. Now substituting $R4(Y',X) \wedge \overline{R1(X,Y')}$ for $R4(Y',X)$ in Eq. (19), we get:

$$\begin{aligned}
&(R4(Y',X) \wedge \overline{R1(X,Y')}) \bigvee \overline{R1(X,Y')} \\
&= \overline{R1(X,Y')} \bigwedge (true \vee R4(Y',X)) \\
&= \overline{R1(X,Y')}.
\end{aligned} \tag{20}$$

Conjuncting this with the condition from Table 6 that Y ends in Region VI, we get $R2(X,Y) \vee \overline{R1(X,Y')}$. This completes the theorem. ■

Theorem 3. *Table 7 gives $\mathcal{H}(r_{i,j})$ for each of the 40 interaction types in $\mathfrak{R}$.*

Proof. The table is constructed by implementing the tests (1)–(5) in Theorem 2. To determine $\mathcal{H}(R''(X,Y'))$, we first determine which of the five tests (1)–(5) are falsified by R''. For the tests that are satisfied by R'', we determine which of the relations $R \in \mathfrak{R}$ satisfy any of these tests on X and Y — these relations belong to $\mathcal{H}(R''(X,Y'))$. For this determination, we derive the following intermediary sets γ_1 to γ_5 from Table 3 and Theorem 2, corresponding to Y ending in regions II to VI of Table 6.

γ_1: $R2(Y,X) \bigwedge \overline{R1(Y,X)} \bigwedge \overline{R4(X,Y)}$ is true for $\gamma_1 = \{IJ, IR\}$

γ_2: $R2(Y,X) \bigwedge R4(X,Y)$ is true for $\gamma_2 = \{ID, ID', ID'', IL, IL', IL'', IS\}$

γ_3: $\overline{R4(X,Y)} \bigwedge \overline{R2(Y,X)}$ is true for $\gamma_3 = \{IG, IV, IK\}$

γ_4: $\overline{R2(Y,X)} \bigwedge \overline{R2(X,Y)} \bigwedge R4(X,Y)$ is true for $\gamma_4 = \{IC, IE, IE', IE'', IH, IM, IM', IM'', IN, IN', IN'', IMN, IMN', IMN'', IT, IW, ITW\}$

γ_5: $R2(X,Y)$ is true for $\gamma_5 = \{IA, IB, IF, II, IP, IO, IU, IX, IOP, IUX\}$.

Now consider the following relations $R'' \in \mathfrak{R}$.

1. $R'' \in \{IA\}$. Let $R''(X,Y')$ hold.
Then $R3(Y',X) = 0$ falsifying (1), $R3(X,Y') = 1$ falsifying (2), $R4(Y',X) = 0$ falsifying (3), $R3(X,Y') = 1$

and $R4(Y', X) = 0$ falsifying (4), and $R1(X, Y') = 1$ falsifying (5). Hence there cannot exist any relationship in $\mathfrak{R}$ satisfying Theorem 2 and hence $\mathcal{H}(R'') = \emptyset$.

2. $R'' \in \{IB, IC\}$. Let $R''(X, Y')$ hold.
   Then $R3(Y', X) = 0$ falsifying (1), $R3(X, Y') = 1$ falsifying (2), $R4(Y', X) = 0$ falsifying (3), $R3(X, Y') = 1$ and $R4(Y', X) = 0$ falsifying (4), and $R1(X, Y') = 0$, satisfying (5).
   From γ_5, condition (5) is satisfied by all relations in $\mathcal{H}(R'') = \gamma_5 = \{IA, IB, IF, II, IP, IO, IU, IX, IUX, IOP\}$.
3. $R'' \in \{ID, ID', ID'', IE, IE', IE'', IF\}$.
   Let $R''(X, Y')$ hold.
   Then $R3(Y', X) = 0$ falsifying (1), $R3(X, Y') = 1$ falsifying (2), $R4(Y', X) = 1$ satisfying (3), $R3(X, Y') = 1$ and $R4(Y', X) = 1$ satisfying (4), and $R1(X, Y') = 0$, satisfying (5).
   At least one of conditions (3), (4), (5) is satisfied by all relations in $\mathcal{H}(R'') = \gamma_3 \cup \gamma_4 \cup \gamma_5$, i.e., by all except those in $\{IQ, IS, IR, IJ, IL, IL', IL'', ID, ID', ID''\}$.
4. $R'' \in \{IG, IH, II\}$. Let $R''(X, Y')$ hold.
   Then $R3(Y', X) = 0$ falsifying (1), $R3(X, Y') = 0$ satisfying (2), $R4(Y', X) = 0$ falsifying (3), $R3(X, Y') = 0$ and $R4(Y', X) = 0$ satisfying (4), and $R1(X, Y') = 0$, satisfying (5).
   At least one of conditions (2), (4), (5) is satisfied by all relations in $\mathcal{H}(R'') = \gamma_2 \cup \gamma_4 \cup \gamma_5$, i.e., by all except those in $\{IG, IJ, IQ, IR, IV, IK\}$.
5. $R'' \in \{IL, IL', IL'', IM, IM', IM'', IN, IN', IN'', IMN, IMN', IMN'', IO, IP, IOP, IJ, IK\}$.
   Let $R''(X, Y')$ hold.
   Then $R3(Y', X) = 0$ falsifying (1), $R3(X, Y') = 0$ satisfying (2), $R4(Y', X) = 1$ satisfying (3), $R3(X, Y') = 0$ and $R4(Y', X) = 1$ satisfying (4), and $R1(X, Y') = 0$, satisfying (5).
   At least one of conditions (2), (3), (4), (5) is satisfied by all relations in $\mathcal{H}(R'') = \gamma_2 \cup \gamma_3 \cup \gamma_4 \cup \gamma_5$, i.e., by all except those in $\{IJ, IQ, IR\}$.
6. $R'' \in \{IQ, IR, IV, IX, IU, IW, IT, IS, IUX, ITW\}$. Let $R''(X, Y')$ hold.
   Then $R3(Y', X) = 1$ satisfying (1), $R3(X, Y') = 0$ satisfying (2), $R4(Y', X) = 1$ satisfying (3), $R3(X, Y') = 0$ and $R4(Y', X) = 1$ satisfying (4), and $R1(X, Y') = 0$, satisfying (5).
   At least one of conditions (1), (2), (3), (4), (5) is satisfied by all relations in $\mathcal{H}(R'') = \gamma_1 \cup \gamma_2 \cup \gamma_3 \cup \gamma_4 \cup \gamma_5$, i.e., by all except those in $\{IQ\}$.

This completes the theorem. ■

From Lemma 1, $R' \rightsquigarrow R''$ if and only if $R' \notin \mathcal{H}(R'')$. The *allows* relation among the interaction types in $\mathfrak{R}$ is depicted using Fig. 6. The figure is drawn using the complements of the entries of Table 7. Observe that only interaction types in $\{IA, IC, ID, ID', ID'', IG, IJ, IQ\}$ *allow* themselves, i.e., they do not belong to their own prohibition set. These are exactly those interaction types for which Y lies entirely within a single region in Fig. 5. Also observe that only IA, IG, IQ satisfy $r^{-1} \rightsquigarrow r^{-1}$. The *allows* relation is the transitive closure over the edges shown using solid lines and the dashed lines. The dashed lines represent the *allows* relation between interaction types, that lead to loops in the graph. $\mathfrak{R}$ is partitioned into 6 equivalence classes that correspond to the 6 cases to which R'' can belong to in Theorem 3. In each class, the interaction types have the same prohibition function, and hence the same set of interaction types allows the interaction types in that class. We have the following result from Definition 5 and Fig. 6.

Corollary 1. *The allows relation is transitive.*

The following two lemmas are necessary to show the correctness of any algorithm to solve problem **DOOR**.

Lemma 3. *If $R(X_i, Y_j)$ holds, and $R \in \mathcal{H}(r_{i,j})$, where $r_{i,j} \neq R$, then interval X_i can be removed from the queue Q_i.*

Proof. From the definition of $\mathcal{H}(r_{i,j})$, we get that $r_{i,j}(X_i, Y'_j)$ cannot exist, where Y'_j is any successor interval of Y_j. Further, as $r_{i,j} \neq R$, we have that interval X_i can never be a part of the solution and its record can be deleted from the queue. ■

The following final result, although simple in form, is based on the crucial Theorem 1 and shows that both $R \notin \mathcal{H}(r_{i,j})$ and $R^{-1} \notin \mathcal{H}(r_{j,i})$ cannot hold when $R \neq r_{i,j}$. Hence, by Lemma 3, if $R(X_i, Y_j) \neq r_{i,j}$ then the record of at least one of the intervals X_i and Y_j being tested must be deleted.

Lemma 4. *If $R(X_i, Y_j)$ holds and $R \neq r_{i,j}$, then interval X_i or interval Y_j is removed from its queue Q_i or Q_j, respectively.*

Proof. We use contradiction. From Lemma 3, the only time neither X_i nor Y_j will be deleted is when $R \notin \mathcal{H}(r_{i,j})$ and $R^{-1} \notin \mathcal{H}(r_{j,i})$. From Lemma 1, it can be inferred that $R \rightsquigarrow r_{i,j}$ and $R^{-1} \rightsquigarrow r_{j,i}$. As $r_{i,j}^{-1} = r_{j,i}$, we get $R \rightsquigarrow r_{i,j}$ and $R^{-1} \rightsquigarrow r_{i,j}^{-1}$. This is a contradiction as by Theorem 1, $R \rightsquigarrow r_{i,j} \Rightarrow R^{-1} \not\rightsquigarrow r_{i,j}^{-1}$. Hence $R \in \mathcal{H}(r_{i,j})$ or $R^{-1} \in \mathcal{H}(r_{j,i})$, and thus at least one of the intervals will be deleted. ■

Observe from Table 7 or Fig. 3 that it is possible that both intervals being examined should be deleted.

Example. If $r_{i,j} = IC$ and $R(X_i, Y_j) = IF$, then X_i can never satisfy $IC(X_i, Y'_j)$ for any successor Y'_j of Y_j. Indeed, $IF \in \mathcal{H}(IC)$. So X_i must be deleted. Also, $R(Y_j, X_i) = IF^{-1} = IS$ and $r_{j,i} = IV$. Y_j can never satisfy $IV(Y_j, X'_i)$ with any successor X'_i of X_i. Indeed, $IS \in \mathcal{H}(IV)$. So Y_j must also be deleted.

Significance of Theorem 1 and Lemma 4: Lemma 4 embodies a principle (Theorem 1) that can be used to solve Problem **DOOR** efficiently. Essentially, the solutions need to examine the intervals in the queues, a pair of intervals from the queues of different processes, at a time. Lemma 4 guarantees that in each such test, at least one or both intervals being examined are deleted, unless $r_{i,j}(X_i, Y_j)$ is satisfied by that pair of intervals X_i and Y_j. The algorithms differ in how they construct the queues, and in how they process the intervals and the queues.

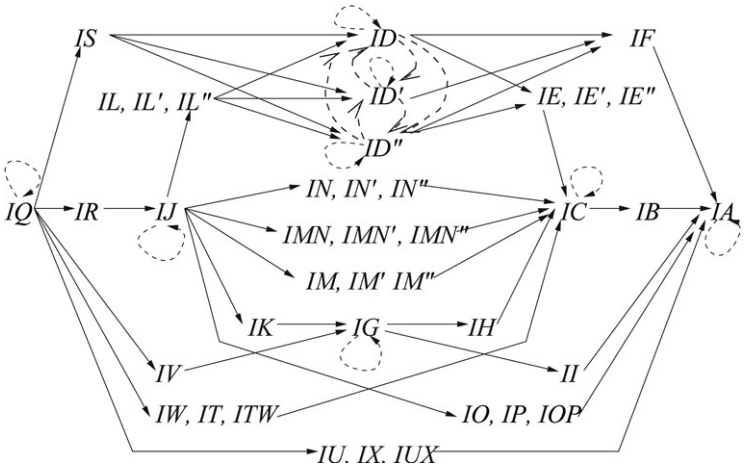

Fig. 6. The *allows* relation is given by the transitive closure of the arrows in solid lines.

V_i, I_i: array $[1 \ldots n]$ of integer;

1. When an internal event or send event occurs at P_i:
   $V_i[i] = V_i[i] + 1$.
2. Send event:
   Every message contains V_i and I_i of its send event.
3. When P_i receives a message *msg*:
   $\forall\ j$ do,

   if $(j == i)$ **then** $V_i[i] = V_i[i] + 1$,
   else $V_i[j] = \max(V_i[j], msg.V[j])$.

4. When an interval starts at P_i, i.e., ϕ_i turns true:
   $I_i[i] = V_i[i]$.
5. When P_i receives a message *msg*:
   $\forall\ j$ do,

   $I_i[j] = \max(I_i[j], msg.I[j])$

Fig. 7. The protocol for vector clock and *Interval Clock* at P_i, $1 \le i \le n$ [7].

4. Data structures

The data structures required by Algorithms 1 and 2 (see Sections 5 and 6) to solve Problems **DOOR** are given in this section. The data structures were used by the centralized algorithm [7] to detect *Possibly* and *Definitely* modalities on predicates.

Each process P_i, $(1 \le i \le n)$ keeps two arrays.

1. V_i: array[1..n] of integer. This is the *Vector Clock* [16,35].
2. I_i: array[1..n] of integer. This is a *Interval Clock* which tracks the latest intervals at processes. $I_i[j]$ is the timestamp $V_j[j]$ when the predicate ϕ_j of interest at P_j last became true, as known to P_i.

Fig. 7 shows how to update the vector clock and *Interval Clock*. Each process P_i, $(1 \le i \le n)$ also keeps a queue Q_i of Log_i.

type *Event_Interval* = **record**
  interval_id : integer;
  local_event: integer;
end
type *Process_Log* = **record**
  event_interval_queue: queue of *Event_Interval*;
end
type *Log* = **record**
  start: array[1..n] of integer;
  end: array[1..n] of integer;
  p_log: array[1..n] of *Process_Log*;
end
Log_i: *Log*;
Q_i: queue of *Log*;

Start of an interval:
$Log_i.start = V_i^-$.
On receiving a message during an interval:
if (change in I_i) **then**
  for each k such that $I_i[k]$ was changed
    insert $(I_i[k], V_i[i])$ in the queue
    $Log_i.p\_log[k].event\_interval\_queue$
End of interval:
$Log_i.end = V_i^+$
if (a receive or send occurs between start of previous interval and end of present interval) **then**
  Enqueue Log_i on to the local queue Q_i.

Fig. 8. Data structures and operations to construct Log_i and enqueue it in Q_i at P_i $(1 \le i \le n)$ [7].

3. Log_i: contains the information for an interval, needed to compare it with other intervals.

The protocol to create a Log_i for a local interval and enqueue it in Q_i is given in Fig. 8. If two or more successive intervals on the same process have the same relationship with all other intervals at all other processes, then the *Log* corresponding to only one of them needs to be stored on the queue. Two

Test for $S2(X, Y)$:

1a) **for** each *event_interval* in the queue $Log_j.p\_log[i].event\_interval\_queue$
1b) **if** $event\_interval.interval\_id < Log_i.start[i]$ **then**
1c) remove *event_interval*
1d) **else** break()

2a) $temp = \infty$
2b) **if** $Log_j.start[i] \geq Log_i.start[i]$ **then**
2c) $temp = Log_j.start[j]$
2d) **else**
2e) $event\_interval = dequeue(Log_j.p\_log[i].event\_interval\_queue)$
2f) $temp = \min(temp, event\_interval.local\_event)$

3) **if** $(Log_i.end[j] \geq temp)$ **then** $S2(X, Y)$ is true.

Test for $S1(Y, X)$:

1) Same as step 1 used for $S2(X, Y)$.
2) Same as step 2 used for $S2(X, Y)$.
3a) **if** $(Log_i.end[j] < temp)$ & $(temp > Log_j.start[j])$ **then**
// for non-dense time, use $Log_i.end[j] < temp - 1$
// instead of $Log_i.end[j] < temp$
3b) $S1(Y, X)$ is true.

Fig. 9. The tests for $S2(X_i, Y_j)$ and $S1(Y_j, X_i)$ [7].

successive intervals Y and Y' on process P_j will have the same relationship if no message is sent or received by P_j between the start of Y and the end of Y'. The *Log* is used to determine the orthogonal relationship between two intervals, based on the tests in Table 2 for the dependent relations. The tests for S1 and S2, in terms of the *Log*, are given in Fig. 9.

5. Distributed algorithm 1

5.1. The algorithm

We reformulate Problem **DOOR** as follows to design a solution.

Problem DOOR: Given a relation $r_{i,j}$ from $\mathfrak{R}$ for each pair of processes P_i and P_j, devise a distributed on-line algorithm to identify the set of intervals $\mathcal{I}$, if they exist, one interval I_i from each process P_i, such that the relation $r_{i,j}(I_i, I_j)$ is satisfied by each (P_i, P_j) process pair.

A token-based algorithm is given in Fig. 10. If no set of intervals satisfying the above conditions exists, the algorithm does not return any interval set. The algorithm uses a token T that has two vectors. $T.Log[i]$ contains the *Log* corresponding to the interval at the head of queue Q_i. $T.C[i] = true$ if and only if the interval at the head of queue Q_i may be a part of the final solution and the corresponding log Log_i is stored in the token $T.Log[i]$. If $T.C[i] = false$ then it is known that the interval at the head of queue Q_i cannot be a part of the solution, its corresponding log is not contained in the token, and the interval can be deleted from Q_i.

A process P_i receives a token only if $T.C[i] = false$, which means the interval at the head of queue Q_i is not a part of the solution, and hence the interval is deleted. The orthogonal relationship $R(X, Y)$ between the next interval X on the queue Q_i and each other interval Y whose log Log_j is contained in $T.Log[j]$ and $T.C[j] = true$ is determined (lines 3d–3i). According to Lemma 4, there are now three cases. (1) $r_{i,j} = R(X, Y)$. (2) $r_{i,j} \neq R(X, Y)$ and interval X can be removed from the queue Q_i. (3) $r_{i,j} \neq R(X, Y)$ and interval Y can be removed from the queue Q_j. In the third case, the log Log_j in $T.Log[j]$ corresponding to interval Y is deleted and $T.C[j]$ is set to false (lines 3m–3n). In the second case, $T.C[i]$ is set to false (line 3k) so that in the next iteration of the **while** loop, the interval X is deleted (lines 3a–3b). Both cases (2) and (3) may occur as a result of a comparison. The above process is repeated until the interval at the head of queue Q_i satisfies the required relationships with each of the interval *Log*s remaining in the token T. The process P_i then adds the log Log_i corresponding to the interval at the head of queue Q_i to the token $T.Log[i]$; $T.C[i]$ is already equal to true. A solution is detected when $T.C[k]$ is true for all indices k (lines 3p–3q), and is given by all the n log entries of all the processes, $T.Log[1 \ldots n]$. If the above condition (line 3p) is not satisfied, then the token is sent to some process P_j whose log Log_j is not contained in the token $T.Log[j]$ (in which case $T.C[j] = false$, lines 3s–3v).

Note that the wait in line 3d can be made non-blocking by restructuring the code using an interrupt-based approach. Algorithm 2 in Section 6 employs this interrupt-based approach.

5.2. Correctness proof

Lemma 5. *After P_i executes loop (3f–3n), if $T.C[i] = true$ then the relationship $r_{i,j}$ is satisfied for interval X_i at the head of queue Q_i and each interval Y_j at the head of queue Q_j such that $T.C[j] = true$.*

Proof. The body of the loop (lines 3j–3m) implements Lemma 3 by testing for $R(X_i, Y_j) \in \mathcal{H}(r_{i,j})$ and $R^{-1}(Y_j, X_i) \in \mathcal{H}(r_{j,i})$. If $r_{i,j}$ is not satisfied between interval X_i and interval Y_j, then by Lemma 4, X_i or Y_j is deleted, i.e., (lines 3j–3k) or (lines 3l–3m) are executed and hence $T.C[i]$ or $T.C[j]$ is set to false. This implies that if both $T.C[i]$ and $T.C[j]$ are true then the relationship $r_{i,j}(X_i, Y_j)$ is true.

It remains to show that Y_j which is $T.Log[j]$ is the same as $head(Q_j)$. This follows by observing that (i) $T.Log[j]$ was the same as $head(Q_j)$ when the token last visited and left P_j, and (ii) $head(Q_j)$ is deleted only when $T.C[j]$ is false and hence the token visits P_j. ■

Theorem 4. *When a solution $\mathcal{I}$ is detected by the algorithm in Fig. 10, the solution is correct, i.e., for each $i, j \in N$ and $I_i, I_j \in \mathcal{I}$, the intervals $I_i = head(Q_i)$ and $I_j = head(Q_j)$ are such that $r_{i,j}(I_i, I_j)$.*

```
type T = token
     Log: array [1..n] of Log;
     C: array [1..n] of boolean;
end

1) Initial state for process P_i
1a)  Q_i has a dummy interval

2) Initial state for the token
2a)  ∀i : T.C[i] = false
2b)  ∀i : T.Log[i] is empty
2c)  A randomly elected process P_i holds the token

3) On receiving token T at P_i
3a)  while (T.C[i] = false)
3b)      Delete head of the queue Q_i
3c)      if (Q_i is empty) then wait until Q_i is non-empty
3d)      T.C[i] = true
3e)      X = head of Q_i
3f)      for j = 1 to n (and j ≠ i)
3g)          if (T.C[j] = true) then
3h)              Y = T.Log[j]
3i)              Determine orthogonal relation R(X, Y)
                 ∈ ℜ using the tests in Tables 2, 3,
                 and Fig. 9
3j)              if (R(X, Y) ∈ H(r_{i,j}) and R ≠ r_{i,j}) then
3k)                  T.C[i] = false
3l)              if (R^{-1}(Y, X) ∈ H(r_{j,i}) and R^{-1} ≠ r_{j,i}) then
3m)                  T.C[j] = false
3n)                  T.Log[j] = ⊥
3o)  T.Log[i] = head of Q_i, i.e., X
3p)  if (∀k : T.C[k] = true) then
3q)      solution found. T has the solution Logs.
3r)  else
3s)      k = i + 1
3t)      while (T.C[k] ≠ false)
3u)          k = (k + 1) mod n
3v)      Send T to P_k
```

Fig. 10. Algorithm 1: Distributed token-based algorithm to solve Problem **DOOR**.

Proof. It is sufficient to prove that for the solution detected, which happens at the time $T.C[k] = true$ for all k (lines 3p–3q), (i) $r_{i,j}(I_i, I_j)$ is satisfied for all pairs (i, j), and (ii) $I_k = head(Q_k)$ for all k. To prove (i) and (ii), note that at this time, the token must have visited each process at least once because only the token-holder P_i can set $T.C[i]$ to true. Consider the latest time t_i when process P_i was last visited by the token (and $T.C[i]$ was set to true and $T.Log[i]$ was set to $head(Q_i)$). Since t_i until the solution is detected, $T.C[i]$ remains true and $head(Q_i)$ is not deleted, otherwise the token would have to revisit P_i again (lines 3s–3v) — leading to a contradiction. Linearly order the process indices in array $Visit[1 \ldots n]$ according to the increasing order of the times of the last visit of the token. Then for k from 2 to n, we have that when the token was at $P_{Visit[k]}$, the intervals corresponding to $T.Log[Visit[k]]$ and $T.Log[Visit[m]]$, for all $1 \leq m < k$, were tested successfully for $r_{Visit[k],Visit[m]}$ and $T.C[Visit[k]]$ and $T.C[Visit[m]]$ were true after this test. This shows that the intervals from every pair of processes got tested, and by Lemma 5, that $R(X_{Visit[k]}, Y_{Visit[m]}) = r_{Visit[k],Visit[m]}$ was satisfied for $X_{Visit}[k] = head(Q_{Visit[k]})$ and $Y_{Visit[m]} = head(Q_{Visit[m]})$ at the time of comparison.

Hence, all the interval pairs (I_i, I_j) of the solution got tested and satisfied $r_{i,j}$. Further, since the time of each latest test (say, at process a), the solution interval $I_a = head(Q_a)$ did not get deleted because $T.C[a]$ remains true after t_a. ■

Let $\mathcal{I}(h)$ denote the set of intervals, one at the head of each queue, during hop h of the token. Each $\mathcal{I}(h)$ identifies a system

state (not necessarily consistent). Also observe that for any $\mathcal{I}(h)$ and $\mathcal{I}(h+1)$ and any process P_i, interval $I_i(h+1)$ in $\mathcal{I}(h+1)$ is a successor of or equal to interval $I_i(h)$ in $\mathcal{I}(h)$. We thus say that all the $\mathcal{I}$ are linearly ordered, and $\mathcal{I}(h)$ *precedes* $\mathcal{I}(h')$, for all $h' > h$. Let $\mathcal{I}(S)$ denote the set of intervals that form the *first* solution. Let $\mathcal{I}(init)$ denote the initial empty set of intervals. Clearly, $\mathcal{I}(init)$ precedes $\mathcal{I}(S)$.

Lemma 6. *In any hop h of the token, no interval $X_i \in \mathcal{I}(S)$ gets deleted from Q_i.*

Proof. We show by contradiction. Let X_i be the first interval belonging to the solution, that gets deleted. Once X_i appears at $head(Q_i)$ in line 3e, there are two cases by which it could get deleted.

1. At P_i, the test in line 3j evaluates to true, causing X_i to be deleted in line 3b of the next iteration. Now $R(X_i, Y_j) \in \mathcal{H}(r_{i,j})$ must be true from the test. However, Y_j is a predecessor of some interval Y'_j such that $r_{i,j}(X_i, Y'_j)$ holds because of our assumption that X_i is the first solution interval to be deleted. Hence Y'_j could not have been deleted from Q_j yet. Therefore, $r_{i,j}(X_i, Y'_j)$ can be true and $R \rightsquigarrow r_{i,j}$, i.e., $R \notin \mathcal{H}(r_{i,j})$, contradicting the successful test.
2. The token passes to some other process P_j. P_j evaluates the test of line 3l to true, using its own interval $Y_j = head(Q_j)$ and X_i in $T.Log[i]$. As a result, $T.C[i]$ is set to false and the token will later visit P_i at which time X_i gets deleted. Flipping the roles of i and j from P_j's perspective at P_j, the test of line 3l is: "$R(X_i, Y_j) \in \mathcal{H}(r_{i,j})$ and $R \neq r_{i,j}$". Using the same logic as in the previous case, we can see that $R(X_i, Y_j) \rightsquigarrow r_{i,j}(X_i, Y'_j)$ and hence $R \notin \mathcal{H}(r_{i,j})$, contradicting the successful test. ■

Corollary 2. *When an interval $X_i \in \mathcal{I}(S)$ appears at $head(Q_i)$, $T.C[i]$ remains true and $T.Log[i] = X_i$ henceforth.*

Proof. When X_i appears at $head(Q_i)$, $T.C[i]$ is set to true (line 3d) and $Log[i]$ is assigned X_i (line 3o). By Lemma 6, X_i is never deleted, which can happen only if $T.C[i]$ remains true and $T.Log[i]$ remains X_i. ■

Lemma 7. *In any hop h of the token, at least one interval gets deleted.*

Proof. Line 3b deletes the interval at $head(Q_i)$ when the token arrives at process P_i. ■

Theorem 5. *If a solution $\mathcal{I}(S)$ exists, i.e., for each $i, j \in N$, the intervals I_i, I_j belonging to $\mathcal{I}(S)$ are such that $r_{i,j}(I_i, I_j)$, then the solution is detected by the algorithm in Fig. 10.*

Proof. The token keeps circulating until $T.C[i]$ is true for all i. By Theorem 4, this will not happen before $\mathcal{I}(S)$.

From state $\mathcal{I}(init)$ in which $T.Log$ was initialized to contain no log, we apply Lemmas 6 and 7. Lemma 6 guarantees that no interval belonging to the solution set $\mathcal{I}(S)$ gets deleted from its queue. Lemma 7 guarantees progress, i.e., in each iteration of the while loop of line 3a for each hop of the token, the interval at the head of the queue of the token-holder process must get replaced by the immediate successor interval at that process. Specifically, at each hop, some interval I' not belonging to $\mathcal{I}(S)$ gets deleted and is replaced by its successor interval I'' after determining that no I between I' and I'' can be a part of the solution. Thus, the progression of the states $\mathcal{I}(h)$ does not skip any interval.

So, in a finite number of hops, the system must reach the state $\mathcal{I}(S)$, where all the solution intervals are at the heads of their queues. Corollary 2 guarantees that from the time each such solution interval X_i appeared at the head of Q_i until the present, $T.C[i]$ remains true and $T.Log[i]$ is not deleted. At this time, $T.C[i]$ is true for all i, hence a solution is detected, and is given by $T.Log$. ■

Theorems 4 and 5 show that the algorithm detects a solution if and only if it exists.

5.3. Complexity analysis

The complexity analysis is done in terms of the parameters, m_s, m_r, and p, defined in Section 1.

5.3.1. Space complexity at P_1 to P_n

This analysis is similar to that of the centralized algorithm [7] because the local data structures are the same.

1. In terms of m_s and m_r: The *Log* corresponding to an interval is stored on the queue only if the relationship between the interval and all other intervals at other processes is different from the relationship which its predecessor interval had with all the other intervals at other processes (see Fig. 8). Two successive intervals Y and Y' will have different relationships if a receive or a send occurs between the start of Y and the end of Y'. So four intervals are stored for every message — two for the send event and two for the receive event. As there are m_s number of send and m_r number of receive events in the entire execution ($m_s = m_r$ in the case of unicast), a total of $2m_s + 2m_r$ interval *Log*s are stored across all the queues, though not necessarily at the same time.
   - The total space overhead across all processes is $2.m_s n + (2m_s + 2m_r).2n = 6m_s n + 4m_r n$. For each of the m_s messages sent, each other process eventually (due to transitive propagation of *Interval Clock*) may need to insert an *Event_Interval* tuple (size 2) in its *Log*. This can generate $2m_s.n$ overhead in *Log*s across the n processes. The term $(2m_s + 2m_r).2n$ arises because the vector timestamp at the start and at the end of each interval is also stored in each *Log*. Thus, the average number of *Log*s per process is $(2m_s + 2m_r)/n$, the average space overhead per process is $6m_s + 4m_r$, and the average size of *Log* is $O(n)$.
   - For a process, the worst case occurs when it sends and receives the maximum number of messages to/from all the other $n-1$ processes. The maximum number of messages a single process can send and receive is m_s because the number of messages it receives (say m') is less than m_s and the corresponding m' send events cannot happen at this process. In this case, the number of *Log*s stored on the

process queue is $2m_s$, two *Log*s for each send and receive event. The reason being, for each send and each receive, the maximum number of consecutive intervals which can have different relations with respect to intervals at all other processes is two. Hence two intervals need to be stored per send event and per receive event. The total space required at the process is $2m_s.2n + 2m_s = O(m_s n)$. The term $2m_s.2n$ arises because each of the $2m_s$ *Log*s contains two vector timestamps. The term $2m_s$ arises because for each of the m_s messages sent, a process eventually (due to the transitive propagation of *Interval Clock*) inserts an *Event_Interval* tuple of size 2 in its *Log*.

The worst case just discussed is for a single process; the total space overhead always remains $O((m_s + m_r)n)$ and on an average, the space complexity for each process is $O(m_s + m_r)$. The worst case *Log* size is $2m_s + 2n$.

2. In terms of p: The total number of *Log*s stored at each process is p because in the worst case, the *Log* for each interval may need to be stored. The total number of *Log*s stored at all the processes is np. Consider the cumulative space requirement for *Log* over all the intervals at a process.
   - Each *Log* stores the start (V^-) and the end (V^+) of an interval, which requires a maximum of $2np$ integers over all *Log*s per process.
   - Additionally, an *Event_Interval* is added to the *Log* for every component of *Interval Clock* which is modified due to the receive of a message. Since a change in a component of *Interval Clock* implies the start of a new interval on another process, the total number of times the component of *Interval Clock* can change is equal to the number of intervals on all the processes. Thus the total number of *Event_Interval* which can be added to the *Log* of a single process is $(n-1)p$. This gives a total of $2(n-1)p$ integers (corresponding to *Event_Interval*s) per process.

   The total space required for *Log*s corresponding to all p intervals on a single process is $2(n-1)p+2np = 4np-2p$. Hence the total space complexity is $4n^2p - 2np = O(n^2p)$ and the average size of *Log* is $4n - 2$. The worst case *Log* size is $2(n-1)p + 2n$.

Thus, the total number of *Log*s stored on all the processes is $\min(np, 2m_s + 2m_r)$, the total space overhead for all the processes is $\min(4n^2p - 2np, (6m_s + 4m_r)n)$, the worst case space overhead per process is $\min(4np - 2p, 4m_s n + 2m_s)$, and average size of *Log* is $O(n)$. The worst case size of *Log* is $\min(2(n-1)p + 2n, 2m_s + 2n)$.

5.3.2. Time complexity

1. In terms of p: The time complexity is the product of the number of steps required to determine the orthogonal relationships between a pair of processes, and $n(n-1)/2$, the number of process pairs.

   Consider the total number of comparisons between any pair of processes. For processes P_i and P_j, intervals X_i and Y_j are compared from the head of their queues Q_i and Q_j, respectively. From Lemma 4, if $R(X, Y) \neq r_{i,j}$, then interval X or interval Y is deleted from its queue. Thus X and Y get compared only once in this case. Now consider the case when $R(X, Y) = r_{i,j}$. Both $T.C[i]$ and $T.C[j]$ equal *true* (lines 3j–3m in Fig. 10). As the **while** loop on line 3a gets executed only when $T.C[i]$ is false, there will be no comparison between intervals at P_i and P_j until either $T.C[i]$ or $T.C[j]$ becomes false. This will only happen if either of X or Y is compared to another interval Z_j at head of queue Q_k and X or Y gets deleted. So even if $R(X, Y) = r_{i,j}$, X and Y get compared only once. This implies that the intervals at the head of Q_i and Q_j are compared only once. As there can be a total of $2p$ unique intervals at the head of Q_i and Q_j, a maximum of $2p$ interval comparisons can occur between P_i and P_j.

   Now consider the time complexity for determining all relationships between a pair of processes P_i and P_j. To determine $R1(X, Y)$ to $R4(X, Y)$ and $R1(Y, X)$ to $R4(Y, X)$ for one interval pair requires eight tests; hence, $16p$ tests are needed to determine them for the pair of processes. To determine the number of comparisons required by $S1$ and $S2$ (see Fig. 9), consider the maximum number of *Event_Interval*s stored in $Log_j.p\_log[i]$ that are queued on Q_j over the execution lifetime as part of the *Log*s, i.e., the maximum number of *Event_Interval*s corresponding to P_i stored in Q_j over P_j's execution lifetime. An *Event_Interval* is added to $Log_j.p\_log[i]$ only when there is a change in the ith component of *Interval Clock* at the receive of a message. The ith component of *Interval Clock* changes only when a new interval starts at P_i, which happens at most p times. As there are a total of $2p$ comparisons between P_i and P_j, the test for (each of) $S1$ and $S2$ is executed $2p$ times. From Fig. 9 (lines 1a, 1d), it can be observed that a comparison in line 1b will result in either deletion of an *Event_Interval* (line 1c) or an exit from the **for** loop (line 1d). The maximum number of times the **for** loop can iterate is equal to the maximum number of deletes that can occur. This is equal to the number of *Event_Interval*s stored in $Log_j.p\_log[i]$, which is equal to p. As $p < 2p$, the rest of the time, the **for** loop exits at line 1d. This bounds the total running time of the **for** loop to $2p$ for determining $S1$ between all interval pairs between P_i and P_j. Thus, to determine $S1$ and $S2$ between all interval pairs between P_i and P_j, the number of comparisons is $2 * 2p = 4p$. This gives a total of $16p+4p = O(p)$ comparisons between a pair of processes. As there are $2p$ intervals pairs between two processes, the average number of comparisons required to determine an orthogonal relationship is $O(1)$.

   As the number of process pairs is $(n(n-1))/2$, the total time complexity of the algorithm is $O(n^2p)$.

2. In terms of m_s and m_r: Consider a process P_j. The number of *Log*s L_j enqueued on Q_j is equal to $2(r_j + s_j)$, where r_j and s_j are the number of receives and sends on P_j. (see Section 5.3.1). Also as seen above, the total number of comparisons between intervals on a pair of processes is equal to the total number of intervals (or the corresponding *Log*s) on the two processes. This is equal to $L_i + L_j$.

   Now consider the time complexity for determining all relationships between a pair of processes P_i and P_j. A total of $8(L_i + L_j)$ comparisons are necessary to determine

$R1(X, Y)$ to $R4(X, Y)$ and $R1(Y, X)$ to $R4(Y, X)$ for the pair of processes. To determine the number of comparisons required by $S1$ and $S2$ (see Fig. 9), consider the maximum number of *Event_Interval*s stored in $Log_j.p\_log[i]$ that are queued on Q_j over the execution lifetime as part of the *Log*s. An *Event_Interval* is added to $Log_j.p\_log[i]$ only when there is a change in the ith component of *Interval Clock* at the receive of a message. The change in the ith component of *Interval Clock* will only reach P_j if there is a send from P_i that reaches P_j either directly or through the transitive propagation of *Interval Clock*. This happens at most s_i times. As there are a total of $L_i + L_j$ comparisons between P_i and P_j, the test for $S1$ and $S2$ is executed $L_i + L_j$ times. From Fig. 9, observe that the maximum number of times the **for** loop (line 1a) can iterate is equal to the maximum number of deletes (line 1c) that can occur. This is equal to the number of *Event_Interval*s stored in $Log_j.p\_log[i]$, which is equal to s_i. As $s_i < L_i + L_j$, the rest of the time, the **for** loop exits at line 1d. This bounds the total running time of the **for** loop to $L_i + L_j$ for determining $S1$ between all interval pairs between P_i and P_j. Thus, to determine $S1$ and $S2$ between all interval pairs between P_i and P_j, the number of comparisons is $2(L_i + L_j)$. This gives a total of $10(L_i + L_j)$ comparisons between a pair of processes. Thus for all pairs of processes, the total time complexity is equal to $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 10(L_i + L_j) = 10(n-1) \sum_{k=1}^{n} L_k = 10(n-1)(2m_s + 2m_r)$ (refer Section 5.3.1).

Hence the total time complexity of the algorithm is $O((n-1) \cdot \min(20np, 20(m_s + m_r)))$, viz., $O(n \cdot \min(np, 2m_s + 2m_r))$.

We now derive the worst case time complexity per process.

1. In terms of p: Processing at P_i occurs only when it has the token. When P_i receives the token, $T.C[i]$ is false and the existing $head(Q_i)$ is deleted because it cannot be part of the solution. Thus the token is received by P_i at most once for an interval. On each visit to P_i, the new $head(Q_i)$ gets compared once to one interval from each other process. Thus the total number of interval pair comparisons at P_i is $(n-1)p$. Using reasoning similar to that for the total time complexity, it can be easily shown that for all comparisons on P_i, $R1$–$R4$ require $8(n-1)p$ tests and $S1$, $S2$ require $2(n-1)p$ tests; leading to worst case $10(n-1)p$ steps which is $O(np)$.
2. In terms of m_s and m_r: The total number of interval pair comparisons at P_i is $(n-1)$ times the number of *Log*s on P_i. The maximum number of *Log*s on P_i is $2m_s$ (Section 5.3.1). So the total number of interval pair comparisons on P_i is $2(n-1)m_s$. Again, it can be easily shown that for all comparisons on P_i, $R1$–$R4$ require $16(n-1)m_s$ steps and $S1$, $S2$ require $(n-1)[4m_s + 2(\sum_k L_k) - 2L_i] = 4(n-1)(m_s + m_r)$ steps, leading to a worst case $(n-1)(20m_s + 4m_r)$ steps.

Hence, the worst case time complexity per process is $O(n \cdot \min(10p, 20m_s + 4m_r))$.

5.3.3. Message complexity

The token is sent to P_j whenever $C[j]$ is false. $C[j]$ is false if the interval at the head of the queue Q_j has to be deleted. Thus, the maximum number of times the token is sent is equal to the total number of intervals across all the queues, which is equal to $\min(np, 2m_s + 2m_r)$. Hence, the total number of messages sent is $\min(np, 2m_s + 2m_r)$. The maximum number of *Log*s stored on a token is $n-1$ and the size of each *Log* on the average is $O(n)$ (see Section 5.3.1). Thus, the average message size is $O(n^2)$ and the average total message space overhead is $O(n^2 \cdot \min(np, 2m_s + 2m_r))$. The worst case message size is $O(n \cdot \min(2(n-1)p + 2n, 2m_s + 2n))$ (see Section 5.3.1 for worst case *Log* size).

6. Distributed algorithm 2

6.1. The algorithm

This algorithm is also token-based like Algorithm 1 but differs in that the token-holder does not do any direct processing but distributes the work that is performed in the loop of (3f–3n) of Fig. 10. The algorithm is given in Fig. 11. Besides the token (T), *REQUEST* (*REQ*) and *REPLY* (*REP*) message types are used. Each procedure is executed atomically. Only the token-holder can send *REQ*s and receive *REP*s. The token-holder process (P_i) broadcasts a *REQ* to all other processes (line 3b). The *Log* corresponding to the interval at the head of the queue Q_i is piggybacked on the *REQ* (line 3a). When a *REQ* from P_i arrives at P_j, process P_j determines the orthogonal relationship $R(X, Y)$ between the piggybacked interval X and the interval Y at the head of its queue Q_j (line 4d). If $r_{i,j} \neq R(X, Y)$, P_j determines whether X or Y or both can be dequeued (lines 4e, 4g), and if Y, it dequeues Y from Q_j (line 4i). According to Lemma 4, $r_{i,j} = R(X, Y)$ or else there are two alternate cases. If $r_{i,j} \neq R(X, Y)$ and Y can be dequeued from Q_j, Y is dequeued and process index j is stored in *REP.updated* (lines 4h, 4i). If $r_{i,j} \neq R(X, Y)$ and X can be dequeued from Q_i, the process index i is stored in *REP.updated* (line 4f). Both X and Y may get dequeued. P_j then sends a REP message carrying the indices of the queues that got updated/are to be updated to P_i. When P_i receives a REP from all other processes, it stores the indices of all the updated queues in $T.updatedQs$. Note that the token T is currently at P_i. Process P_i then checks if its index i is contained in *updatedQs*. If so, it deletes the interval at the head of Q_i (line 8f). A solution is detected when *updatedQs* becomes empty. If *updatedQs* is non-empty, the token is sent to a randomly selected process from *updatedQs* (line 8g).

Observe that index i gets deleted from $T.updatedQs$ only when P_i receives the token T. Index i may get added back to $T.updatedQs$ when (i) P_i which holds the token receives a REP such that $i \in REP.updated$, or (ii) P_i receives a REQ from the token-holder P_j and while executing lines 4g–4i, $R(X_i, Y_j) \in \mathcal{H}(r_{i,j})$; hence, $i \in REP.updated$ on the REP sent by P_i to P_j and P_j inserts i in $T.updatedQs$. Index i can get alternately added and deleted multiple times from $T.updatedQs$. Also observe that unlike Algorithm 1, a pair of intervals may get compared twice.

type REQ = **message**
  log : Log; **end** //Contains the Log of the interval at the queue head at the sender process
type REP = **message**
  $updated$: set of integer; **end** //Contains the indices of queues updated after a test
type T = **token**
  $updatedQs$: set of integer; **end** //Contains the index of all the updated queues

1) Initial state for process P_i
1a) Q_i is empty
1b) Token holder executes *RcvToken*

(2) Initial state of the token
2a) $T.updatedQs = \{1, 2, \ldots n\}$
2b) A randomly elected process P_i holds the token.

3) *SendReq* : Procedure called by process P_i to send REQ message
a) $REQ.log = Log_i$ at the head of the local queue
b) Broadcast request REQ to all processes

4) *SendReply* : Procedure called by process P_j to send a REP message to P_i
4a) $REP.updated = \phi$
4b) Y = head of local queue Q_j
4c) $X = REQ.log$
4d) Determine orthogonal relation $R(X, Y) \in \Re$ using the tests in Tables 2 and 3, and Figure 9
4e) **if** $(R(X, Y) \in \mathcal{H}(r_{i,j})$ and $r_{i,j} \neq R)$ **then**
4f) $REP.updated = REP.updated \cup \{i\}$
4g) **if** $(R^{-1}(Y, X) \in \mathcal{H}(r_{j,i})$ and $r_{j,i} \neq R^{-1})$ **then**
4h) $REP.updated = REP.updated \cup \{j\}$
4i) Dequeue head of local queue Q_j, i.e., Y
4j) Send reply REP to P_i

5) *RcvToken* : On receiving a token T at P_i
5a) Remove index i from $T.updatedQs$
5b) **if** (Q_i is nonempty) **then** *SendReq*

6) *IntQue* : When an interval gets queued on Q_i at P_i
6a) **if** (number of elements in queue Q_i is 1) and (P_i has the token) **then** *SendReq*
6b) **else**
6c) **if** (number of elements in queue Q_i is 1) and (P_i has a pending request) **then**
6d) REQ is not pending
6e) *SendReply*

7) *RcvReq* : On receiving a request REQ at P_i
7a) **if** (Q_i is nonempty) **then** *SendReply*
7b) **else** REQ is pending

8) *RcvReply* : On receiving a reply REP from P_i
8a) $T.updatedQs = T.updatedQs \cup REP.updated$
8b) **if** (reply received from all processes) **then**
8c) **if** ($T.updatedQs$ is empty) **then**
8d) Solution detected. Heads of the queues identify the intervals.
8e) **else**
8f) **if** ($i \in T.updatedQs$) **then** dequeue the head from Q_i
8g) Send token to P_k, where k is randomly selected from $T.updatedQs$

Fig. 11. Algorithm 2: Distributed algorithm to solve Problem **DOOR**.

6.2. Correctness proof

Lemma 8. *If the relationship between a pair of intervals X_i and Y_j is not equal to $r_{i,j}$, then either i or j or both are inserted into $T.updatedQs$.*

Proof. From Lemma 4, if $r_{i,j}$ is not satisfied then either X_i or Y_j, or both should get deleted. In the algorithm of Fig. 11, the test in line 4e or line 4g evaluate(s) to true. Hence, either i or j or both are inserted in $REP.updated$ which is later included in $T.updatedQs$ (line 8a). ■

Lemma 9. *An interval is deleted from queue Q_i if and only if the index i is inserted into $T.updatedQs$.*

Proof. When comparing two intervals X_i and Y_j at P_j, Y_j is deleted (line 4i) if and only if j is inserted into $REP.updated$ (line 4h) because lines 4h and 4i are a part of the same **then** block. $REP.updated$ is then set-unioned into $T.updatedQs$ by P_i (line 8a). Similarly X_i is deleted at P_i (line 8f) if and only if P_j inserts i in $REP.updated$ (line 4f) and this gets set-unioned into $T.updatedQs$ by P_i (line 8a), thus leading to $i \in T.updatedQs$ (line 8a,8f). ■

Theorem 6. *When a solution $\mathcal{I}$ is detected by the algorithm in Fig. 11, the solution is correct, i.e., for each $i, j \in N$ and $I_i, I_j \in \mathcal{I}$, the intervals $I_i = head(Q_i)$ and $I_j = head(Q_j)$ are such that $r_{i,j}(I_i, I_j)$.*

Proof. It is sufficient to prove that for the solution detected, which happens at the time $T.updatedQs$ is empty (line 8c), (i) $r_{i,j}(I_i, I_j)$ is satisfied for all pairs (i, j), and (ii) $I_k = head(Q_k)$ for all k. To prove (i) and (ii), note that at this time, the token must have visited each process at least once because only the token-holder can delete its index from $T.updatedQs$.

Consider the latest time t_i when process P_i last sent out the token (and $i \notin T.updatedQs$). Since t_i until the solution is detected, $i \notin T.updatedQs$ otherwise the token would have to revisit P_i again (line 8g) — leading to a contradiction. Therefore, applying Lemma 9, $head(Q_i)$ is not deleted from t_i until the solution is detected and hence $head(Q_i) = I_i \in \mathcal{I}$. Linearly order the process indices in array $Visit[1 \ldots n]$ according to the increasing order of the times of the last visit of the token. Then for k from 2 to n, we have that: (i) when the token was at $P_{Visit[k]}$, the intervals corresponding to $X_{Visit[k]} = head(Q_{Visit[k]})$ and $Y_{Visit[m]} = head(Q_{Visit[m]})$, for all $1 \leq m < k$, were tested successfully, i.e., $R(X_{Visit[k]}, Y_{Visit[m]}) = r_{Visit[k],Visit[m]}$, by $P_{Visit[m]}$ via the REQ and REP messages, and (ii) $k, m \notin T.updatedQs$ when the token leaves $P_{Visit[k]}$. If the test was unsuccessful, then i or j would be inserted in $T.updatedQs$ as per Lemma 8 — a contradiction.

This shows that the intervals from every pair of processes got tested successfully. For each k $(1 \leq k \leq n)$, the interval $head(Q_{Visit[k]})$ that is tested successfully with all $head(Q_{Visit[m]})$, $(1 \leq m < k)$, is not deleted until the solution is detected. Hence, the same interval $head(Q_{Visit[k]})$ is used in the later successful tests with $head(Q_{Visit[a]})$, for $k < a \leq n$, that are initiated by $P_{Visit[a]}$. So $head(Q_{Visit[k]})$ is the solution interval I_k. ■

Lemma 10. *In any hop h of the token, no interval $X_i \in \mathcal{I}(S)$ gets deleted.*

Proof. We show by contradiction. Let X_i be the first interval among those in $\mathcal{I}(S)$, that gets deleted. X_i appears at $head(Q_i)$ when its predecessor is deleted. There are two cases by which X_i could get deleted.

1. P_i receives the token and executes $SendReq$, in which $REQ.log$ is set to X_i and REQ is broadcast to other processes. P_j determines in line 4e that the test $R(X_i, Y_j) \in \mathcal{H}(r_{i,j})$ succeeds, where $Y_j = head(Q_j)$. P_j inserts i in $REP.updated$, and sends REP to P_i. On receiving REP, P_i deletes X_i from $head(Q_i)$, see step (8). Note, however, Y_j is a predecessor of some interval Y'_j such that $r_{i,j}(X_i, Y'_j)$ holds, because of our assumption that X_i is the first solution interval to be deleted. Hence Y'_j could not have been deleted from Q_j yet. Therefore, $r_{i,j}(X_i, Y'_j)$ can be true and $R \rightsquigarrow r_{i,j}$, i.e., $R \notin \mathcal{H}(r_{i,j})$, contradicting the successful test.
2. The token reaches some other process P_j. P_j broadcasts REQ, with $REQ.log = Y_j = head(Q_j)$. P_i receives REQ and executes $SendReply$. When P_i executes the test in line 4g, with the roles of X_i and Y_j flipped from P_i's perspective, the test becomes: "$R(X_i, Y_j) \in \mathcal{H}(r_{i,j})$ and $R \neq r_{i,j}$" and evaluates to true. It adds i to $REP.updated$ and dequeues $X_i = head(Q_i)$ (lines 4h–4i). Using the same logic as in the previous case, we can see that $R(X_i, Y_j) \rightsquigarrow r_{i,j}(X_i, Y'_j)$ and hence $R \notin \mathcal{H}(r_{i,j})$, contradicting the successful test. ■

Corollary 3. *When an interval $X_i \in \mathcal{I}(S)$ appears at $head(Q_i)$, the token visits P_i once and after that visit, $i \notin T.updatedQs$ henceforth.*

Proof. X_i appears at $head(Q_i)$ when its predecessor interval is deleted. When the predecessor is deleted, i was inserted in $T.updatedQs$ as per Lemma 9. By line 8g, the token will eventually be sent to P_i, at which time i is deleted from $T.updatedQs$ (line 5a). By Lemma 6, X_i is never deleted from $head(Q_i)$. Hence, by Lemma 9, i is never inserted back in $T.updatedQs$. ■

Lemma 11. *The token makes a hop to P_i if and only if $head(Q_i)$ has been deleted during or after the last visit of the token to P_i.*

Proof. From line 8g, observe that the token is sent to P_i if and only if $i \in T.updatedQs$. From Lemma 9, we have that i is set-unioned into $T.updatedQs$ (during or after the last visit to P_i) if and only if at least one interval is deleted from Q_i (during or after the last visit to P_i). To make this appliable to the initial state in which $T.updatedQs$ is $\{1, 2, \ldots n\}$, we assume that a dummy initial interval has been initially deleted from each Q_i after a prior visit of the token. The result follows. ■

Theorem 7. *If a solution $\mathcal{I}(S)$ exists, i.e., for each $i, j \in N$, the intervals I_i, I_j belonging to $\mathcal{I}$ are such that $r_{i,j}(I_i, I_j)$, then the solution is detected by the algorithm in Fig. 11.*

Proof. The token keeps circulating until $T.updatedQs = \emptyset$. By Theorem 6, $T.updatedQs = \emptyset$ will not happen before $\mathcal{I}(S)$ (otherwise a false solution would be detected).

From state $\mathcal{I}(init)$ in which $T.updatedQs$ was initialized to N, we apply Lemmas 10 and 11. As the token keeps making hops, Lemma 11 guarantees progress because each hop made to P_i is due to at least one interval from Q_i having been deleted (during or after the previous hop to P_i). Lemma 10 guarantees that no interval in the solution set $\mathcal{I}(S)$ gets deleted. Thus, for every hop, some interval not belonging to $\mathcal{I}(S)$ has been replaced by its immediate successor interval and the progression of the states $\mathcal{I}(h)$ does not skip any interval.

So, in a finite number of hops, the system must reach state $\mathcal{I}(S)$, where all the solution intervals are at the heads of their queues. Corollary 3 guarantees that from the time $X_i \in \mathcal{I}(S)$ appears at the head of Q_i, the token visits P_i exactly once, after which $i \notin T.updatedQs$. Once the intervals at the head of each queue belong to $\mathcal{I}(S)$, the token makes $|T.updatedQs| \leq n$ hops after which $T.updatedQs$ becomes $\emptyset$ and the solution is detected (lines 8c, 8d). This follows because whenever no interval gets deleted during a token visit, $|T.updatedQs|$ must decrement by one (line 5a and Lemma 9). ■

Theorems 6 and 7 show that the algorithm detects a solution if and only if it exists.

6.3. Complexity analysis

Space complexity: The space complexity analysis is the same as that for Algorithm 1 because the log operations are common to both the algorithms. The worst case space overhead per process is $\min(4np-2p, 4m_sn+2m_s)$. The total space overhead across all processes is $\min(4n^2p - 2np, (6m_s + 4m_r)n)$.

Time complexity: As for Algorithm 1, consider the number of comparisons between a pair of processes P_i and P_j. Observe that the only time the intervals at head of Q_i and Q_j are compared is when the token is either with P_i or P_j. So the maximum number of comparisons between the heads of Q_i and Q_j is equal to the number of visits of the token to P_i and P_j. By Lemma 11, the token will come back to P_i when at least one interval at the head of Q_i has been deleted. Thus the maximum number of token visits to P_i and P_j, and hence the number of comparisons between P_i and P_j, is equal to the sum of the number of intervals on P_i and P_j. This is equal to $\min(2p, L_i + L_j)$, as in Algorithm 1. Using reasoning similar to that in Algorithm 1, it can easily be shown that the total and per process time complexities are also the same as that for Algorithm 1, i.e., $O(n \cdot \min(np, 2m_s + 2m_r))$ and $O(n \cdot \min(10p, 20m_s + 4m_r))$, respectively.

Message complexity: From Lemma 11, T makes a hop to P_i iff at least one *Log* from Q_i has been deleted since the last visit. Hence, for each *Log*, at most one token T, $n-1$ *REQ*s, $n-1$ *REP*s and are sent. As the total number of *Log*s over all the queues is $\min(np, 2m_s + 2m_r)$, the total number of messages sent by all the processes is $O(n \cdot \min(np, 2m_s + 2m_r))$. Each *REQ* carries a *Log* which has average size $O(n)$ and worst case size $O(\min(2(n-1)p + 2n, 2m_s + 2n))$ (see Section 5.3.1), while the size of each *REP* and each T is $O(1)$ and $O(n)$, respectively. The average message size is $O(n)$ and worst case message size is the worst case *Log* size.

Total T space: $O(n) \cdot O(\min(np, 2m_s + 2m_r))$

Total *REQ* space: Each *Log* is sent once on $n-1$ *REQ*s, hence
$(n-1) \cdot O(\min(4n^2p - 2np, 6m_sn + 4m_rn))$
$= O(n^2 \cdot \min(4np - 2p, 6m_s + 4m_r))$.

Total *REP* space: $(n-1) \cdot O(1) \cdot O(\min(np, m_s + 2m_r))$
$= O(n \cdot \min(np, 2m_s + 2m_r))$.

Hence the total message space overhead over all the messages is equal to $O(n^2 \cdot \min(4np - 2p, 6m_s + 4m_r))$. Note that in the case of broadcast media, the number of *REQ*s sent for each *Log* is one because *REQ*s are broadcast by sending one message (line 3b). The message space complexity reduces to $O(n \cdot \min(4np - 2p, 6m_s + 2m_r))$, although the total number of messages sent stays at $O(n \cdot \min(np, 2m_s + 2m_r))$.

7. An extended specification of DOOR

We now consider an extension to problem **DOOR**, given in [7].

Problem DOOR′: Given a set of relations $r^*_{i,j} \subseteq \mathfrak{R}$ for each pair of processes P_i and P_j, determine on-line the intervals, if they exist, one from each process, such that any one of the relations in $r^*_{i,j}$ is satisfied (by the intervals) for each (P_i, P_j) pair. If a solution exists, identify the interaction type from $\mathfrak{R}$ for each pair of processes in the first solution.

To solve **DOOR′**, given an *arbitrary* $r^*_{i,j}$, a solution based on an algorithm to solve **DOOR** will not work because in the tests of lines 3j, 3l (Algorithm 1) and 4e, 4g (Algorithm 2), neither interval may be removable, and yet none of the relations from $r^*_{i,j}$ might hold between the two intervals. In more detail, let $r1, r2 \in r^*_{i,j}$ and let $R(X, Y)$ hold, where $R \notin r^*_{i,j}$. Now let $R \notin \mathcal{H}(r1)$, $R^{-1} \notin \mathcal{H}(r2^{-1})$. Interval Y cannot be deleted because $R^{-1} \notin \mathcal{H}(r2^{-1})$, implying that $r2^{-1}(Y, X')$ may be true for a successor X' of X. Interval X cannot be deleted because $R \notin \mathcal{H}(r1)$, implying that $r1(X, Y')$ may be true for a successor Y' of Y. This leads to a combinatorial explosion of global states to consider.

Example. Let $r^*_{i,j} = \{IF, IQ\}$ and let $R(X, Y) = IL$. Since $IL \notin \mathcal{H}(IF)$, there can exist a successor Y' of Y such that $R(X, Y') = IF$. Hence X cannot be deleted. Similarly, since $IL^{-1}(= IO) \notin \mathcal{H}(IQ^{-1} = (IA))$, there can exist a successor X' of X such that $R^{-1}(Y, X') = IQ$ and hence Y cannot be deleted.

A special property, termed $\mathcal{CONVEXITY}$, on $r^*_{i,j}$ such that the deadlock is prevented was identified in [7]. Informally, this property says that there is no relation R outside $r^*_{i,j}$ such that for any $r1, r2 \in r^*_{i,j}$, $R \rightsquigarrow r1$ and $R^{-1} \rightsquigarrow r2^{-1}$. This property guarantees that when intervals X and Y are compared for $r^*_{i,j}$ and $R(X, Y)$ holds, either X or Y or both get deleted, and hence there is progress.

Definition 6 (*Convexity*).

$$\mathcal{CONVEXITY}: \forall R \notin r^*_{i,j}: \Big(\forall r_{i,j} \in r^*_{i,j}, R \in \mathcal{H}(r_{i,j}) \bigvee \forall r_{j,i} \in r^*_{j,i}, R^{-1} \in \mathcal{H}(r_{j,i})\Big).$$

Lemma 12. *If $R(X_i, Y_j)$ holds and $R \in \bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j})$, then interval X_i can be removed from Q_i.*

Proof. From the definition of $\mathcal{H}(r_{i,j})$, we infer that no relation $r_{i,j}(X_i, Y'_j)$, where $r_{i,j} \in r^*_{i,j}$ and Y'_j is any successor interval of Y_j, can be true. Hence interval X_i can never be a part of the solution and can be deleted from Q_i. ■

Lemma 13. *If $R(X_i, Y_j)$ holds and $R \notin r^*_{i,j}$, where $r^*_{i,j}$ satisfies property $\mathcal{CONVEXITY}$, then either interval X_i or interval Y_j is removed from its queue.*

Proof. We use contradiction. From Lemma 12, the only time neither X_i nor Y_j will be deleted is when both $R \notin \bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j})$, and $R^{-1} \notin \bigcap_{r_{j,i} \in r^*_{j,i}} \mathcal{H}(r_{j,i})$. However, as $r^*_{i,j}$ satisfies property $\mathcal{CONVEXITY}$, we have that $R \in \bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j})$ or $R^{-1} \in \bigcap_{r_{j,i} \in r^*_{j,i}} \mathcal{H}(r_{j,i})$ must be true. Thus at least one of the intervals can be deleted by an application of Lemma 12. ■

The proof of the following theorem is similar to the proofs used for Algorithms 1 and 2. Lemmas 12 and 13 should be used instead of Lemmas 3 and 4, respectively, and reason with $r^*_{i,j}$ instead of with $r_{i,j}$.

Theorem 8. *If the set $r^*_{i,j}$ satisfies property $\mathcal{CONVEXITY}$, then Problem* **DOOR**′ *is solved by the following changes to Algorithms* 1 *and* 2.

- *Replace the test in line* 3j *of Algorithm* 1 *(Fig.* 10*) and line* 4e *of Algorithm* 2 *(Fig.* 11*) by:*

  if $(R(X, Y) \in \bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j}) \quad and \quad R \notin r^*_{i,j})$ **then**

- *Replace the test in line* 3l *of Algorithm* 1 *(Fig.* 10*) and line* 4g *of Algorithm* 2 *(Fig.* 11*) by:*

  if $(R^{-1}(Y, X) \in \bigcap_{r_{j,i} \in r^*_{j,i}} \mathcal{H}(r_{j,i}) \quad and \quad R^{-1} \notin r^*_{j,i})$ **then**

Corollary 4. *The time, space, and message complexities of the algorithms to solve* **DOOR**′ *are the same as those of the algorithms to solve* **DOOR**.

Proof. The only changes to Algorithm 1 for **DOOR** are in lines (3j) and (3l). The only changes to Algorithm 2 for **DOOR** are in lines (4e) and (4g). In both cases, instead of checking $R(X_i, Y_j)$ for membership in $\mathcal{H}(r_{i,j})$, in line (3j) or (4e), $R(X_i, Y_j)$ is checked for membership in $\bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j})$. Both $\mathcal{H}(r_{i,j})$ and $\bigcap_{r_{i,j} \in r^*_{i,j}} \mathcal{H}(r_{i,j})$ are sets of size between 0 and $|\mathfrak{R}| = 40$. Hence, the time, space, and message complexities to solve **DOOR**′ are the same as those for **DOOR**. ■

8. Discussion and conclusions

Causality-based pairwise temporal interactions between intervals in a distributed execution provide a valuable way to specify and model synchronization conditions and information interchange. This paper examined the underlying theory to solve the problem **DOOR** of how to devise algorithms to identify a set of intervals, one from each process, such that a given set of pairwise temporal interactions, one for each process pair, holds for the set of intervals identified. **DOOR** represents a fundamental global state detection problem, in terms of intervals and causality-based pairwise interactions. Devising an efficient on-line algorithm to solve problem **DOOR** is a challenge because of the overhead of having to track the intervals at different processes. For any two intervals being examined from processes P_i and P_j, this paper formulated and proved the underlying principle which identifies which (or both) of the intervals' records can be safely deleted if the intervals do not satisfy $r_{i,j}$. This principle can be used by any algorithm, to efficiently manage the local interval queues.

The paper also proposed two fully distributed algorithms to solve problem **DOOR**. These algorithms leverage the above principle to efficiently manage and prune the local interval queues. The performance of the algorithms is summarized in Table 1. The results in [7] showed how to use the centralized solution to **DOOR** to detect conjunctive predicates under traditional modalities in a centralized manner at the same cost but with a lot more information. The same theory can be used to adapt the distributed algorithms presented here to detect the traditional modalities at the same cost but with a lot more information. Algorithm 2 has also been adapted to detect strong conjunctive predicates efficiently [10].

Problem **DOOR** is important because it generalizes the global state observation and the predicate detection problems; further, solutions to problem **DOOR** which provide a much richer palette of information about the causality structure in the application execution (see [7]), cost about the same as the solutions to traditional forms of global predicate detection.

The process of formulating the underlying principle of determining which intervals can be discarded as never forming a part of a solution that satisfies a specification of **DOOR**, also gave a deeper insight into the structure of causality in a distributed execution, and the global state observation and predicate detection problems [37].

A centralized algorithm that fuses event streams reported from a sensor network to detect temporal predicates was given in [23]. This work assumed tightly synchronized physical clocks, and the predicates were defined using global time. This centralized algorithm can be converted to a distributed algorithm along the lines that the centralized algorithm of [7] has been converted to the distributed algorithms here.

One example application area of **DOOR** is distributed debugging. Let the intervals of interest at a process denote durations in which some local variable state error is flagged. By detecting a global state in which the local intervals satisfy specific causality relationships specified by each $r_{i,j}$, the interdependence (as defined by $R1 - R4, S1, S2$) of the

program states across processes can be precisely identified. This can help to determine the extent, nature, and cause of the error manifestation across the processes.

As another example, consider the problem of detecting crisis situations, as motivated in [13,38]. A crisis situation can be specified as a function of the global state. The local predicate is true when there is a local crisis condition. What is required is the ability to take instantaneous snapshots of the distributed execution. Although this is not possible in the absence of perfectly synchronized physical clocks, we can specify snapshot states representing crisis situations in terms of the causality relation, viz., in terms of $R1$–$R4$, $S1$, $S2$. With such a specification of the global crisis condition, our presented distributed algorithms can detect the global crisis state. Observe that if the full power of $\mathfrak{R}$ is not required, a coarser granularity of specification, provided by the *Possibly* and *Definitely* modalities [12], can be used. As shown by the centralized algorithm of [7], the complexity of detecting the global snapshot under the fine-grained modalities of $\mathfrak{R}$ is the *same as* the complexity of detecting the global snapshot under the coarser-grained modalities of *Possibly* and *Definitely*. The distributed algorithms to solve **DOOR**$'$ presented here can be directly used to detect global states under the coarser-grained modalities, as shown in [7], while providing the extra information of the fine-grained modality $r_{i,j}$ for each (i, j) pair.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream queries, in: Proceedings 21st ACM Symposium on Principles of Database Systems, 2002, p. 1-16.

[2] S. Babu, J. Widom, Continuous queries over data streams, ACM SIGMOD Record 30 (3) (2001) 109–120.

[3] R. Baldoni, J.M. Helary, M. Raynal, Consistent records in asynchronous computations, Acta Informatica 35 (6) (1998) 441–455.

[4] R. Baldoni, F. Quaglia, P. Fornara, An index-based checkpointing algorithm for autonomous distributed systems, IEEE Transactions on Parallel and Distributed Systems 10 (2) (1999) 181–192.

[5] G. Cao, M. Singhal, On coordinated checkpointing in distributed systems, IEEE Transactions on Parallel and Distributed Systems 9 (12) (1998) 1213–1225.

[6] P. Chandra, A.D. Kshemkalyani, Detection of orthogonal interval relations, in: Proceedings 9th International High Performance Computing Conference (HiPC), in: Lecture Notes in Computer Science, vol. 2552, Springer-Verlag, 2002, pp. 323–333.

[7] P. Chandra, A.D. Kshemkalyani, Causality-based predicate detection across space and time, IEEE Transactions on Computers 54 (11) (2005) 1438–1553.

[8] P. Chandra, A.D. Kshemkalyani, Analysis of interval-based global state detection, in: Proceedings 2nd Int. Conference on Distributed Computing and Internet Technology (ICDCIT), in: Lecture Notes in Computer Science, vol. 3816, Springer, 2005, pp. 203–216.

[9] P. Chandra, A.D. Kshemkalyani, Global state detection based on peer-to-peer interactions, in: Proceedings IFIP Int. Conference on Embedded and Ubiquitous Computing (EUC), in: Lecture Notes in Computer Science, vol. 3824, Springer, 2005, pp. 560–571.

[10] P. Chandra, A.D. Kshemkalyani, Distributed algorithm to detect strong conjunctive predicates, Information Processing Letters 87 (5) (2003) 243–249.

[11] B. Charron-Bost, C. Delporte-Gallet, H. Fauconnier, Local and temporal predicates in distributed systems, ACM Transactions on Programming Languages and Systems 17 (1) (1995) 157–179.

[12] R. Cooper, K. Marzullo, Consistent detection of global predicates, in: Proceedings ACM/ONR Workshop on Parallel and Distributed Debugging, May 1991, p. 163–173.

[13] K.M. Chandy, B. Aydemir, E. Karpilovsky, D.M. Zimmerman, Event-driven architectures for distributed crisis management, in: 15th IASTED International Conference on Parallel and Distributed Computing and Systems, Nov. 2003.

[14] K.M. Chandy, L. Lamport, Distributed snapshots: Determining global states of distributed systems, ACM Transactions on Computer Systems 3 (1) (1985) 63–75.

[15] E.N. Elnozahy, L. Alvisi, Y.-M. Wang, D.B. Johnson, A survey of rollback-recovery protocols in message-passing systems, ACM Computing Surveys 34 (3) (2002) 375–408.

[16] C.J. Fidge, Timestamps in message-passing systems that preserve partial ordering, Australian Computer Science Communications 10 (1) (1988) 56–66.

[17] J.M. Helary, A. Mostefaoui, M. Raynal, Communication-induced determination of consistent snapshots, IEEE Transactions on Parallel and Distributed Systems 10 (9) (1999) 865–877.

[18] J.M. Helary, A. Mostefaoui, M. Raynal, Interval consistency of asynchronous distributed computations, Journal of Computer and System Sciences 64 (2) (2002) 329–349.

[19] B. Krishnamachari, D. Estrin, S. Wicker, The impact of data aggregation in wireless sensor networks, in: ICDCS Workshops, 2002, pp. 575–578.

[20] A.D. Kshemkalyani, Temporal interactions of intervals in distributed systems, Journal of Computer and System Sciences 52 (2) (1996) 287–298.

[21] A.D. Kshemkalyani, Causality and atomicity in distributed computations, Distributed Computing 11 (4) (1998) 149–169.

[22] A.D. Kshemkalyani, A fine-grained modality classification for global predicates, IEEE Transactions on Parallel and Distributed Systems 14 (8) (2003) 807–816.

[23] A.D. Kshemkalyani, Temporal predicate detection using synchronized clocks, IEEE Transactions on Computers 56 (11) (2007) 1578–1584.

[24] A.D. Kshemkalyani, M. Raynal, M. Singhal, An introduction to snapshot algorithms in distributed computing, Distributed Systems Engineering Journal 2 (4) (1995) 224–233.

[25] A.D. Kshemkalyani, M. Singhal, Communication patterns in distributed computations, Journal of Parallel and Distributed Computing 62 (6) (2002) 1104–1119.

[26] A.D. Kshemkalyani, M. Singhal, On characterization and correctness of distributed deadlock detection, Journal of Parallel and Distributed Computing 22 (1) (1994) 44–59.

[27] A.D. Kshemkalyani, A framework for viewing atomic events in distributed computations, Theoretical Computer Science 196 (1–2) (1998) 45–70.

[28] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Communications of the ACM 21 (7) (1978) 558–565.

[29] L. Lamport, On interprocess communication, Part I: Basic formalism; Part II: Algorithms, Distributed Computing 1 (1986) 77–85. 1:86-101.

[30] S. Li, Y. Lin, S. Son, J. Stankovic, Y. Wei, Event detection services using data service middleware in distributed sensor networks, Telecommunication Systems 26 (2–4) (2004) 351–368.

[31] S. Madden, M. Franklin, J. Hellerstein, W. Hong, TinyDB: An acquisitional query processing system for sensor networks, ACM Transactions on Database Systems 30 (1) (2005) 122–173.

[32] S. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, TAG: A tiny aggregation service for ad-hoc sensor networks, OSDI (2002).

[33] D. Manivannan, M. Singhal, Quasi-synchronous checkpointing: Models, characterization, and classification, IEEE Transactions on Parallel and Distributed Systems 10 (7) (1999) 703–713.

[34] D. Manivannan, R.H. Netzer, M. Singhal, Finding consistent distributed checkpoints in a distributed computation, IEEE Transactions on Parallel and Distributed Systems 8 (6) (1997) 623–627.

[35] F. Mattern, Virtual time and global states of distributed systems, in: Parallel and Distributed Algorithms, North-Holland, 1989, pp. 215–226.

[36] P. Pietzuch, B. Shand, J. Bacon, Composite event detection as a generic middleware extension, IEEE Network 18 (1) (2004) 44–55.

[37] R. Schwarz, F. Mattern, Detecting causal relationships in distributed computations: In search of the holy grail, Distributed Computing 7 (1994) 149–174.

[38] D. Zimmerman, K.M. Chandy, A parallel algorithm for correlating event streams, in: Proceedings IEEE International Parallel and Distributed Processing Symposium, 2005.

Punit Chandra received the B.Tech. degree in Electrical Engineering from the Indian Institute of Technology, Kanpur, in 1999, and M.S. and Ph.D. degrees in Computer Science from the University of Illinois at Chicago in 2001 and 2005, respectively. Currently, he is working for Siemens. His research interests are in distributed computing, computer networks, and operating systems.

Ajay D. Kshemkalyani received a Ph.D. in Computer and Information Science from The Ohio State University in 1991, and a B.Tech. in Computer Science and Engineering from the Indian Institute of Technology, Bombay, in 1987. His research interests are in computer networks, distributed computing, algorithms, and concurrent systems. He has been an Associate Professor at the University of Illinois at Chicago since 2000, before which he spent several years at IBM Research Triangle Park working on various aspects of computer networks. He is a member of the ACM and a senior member of the IEEE. In 1999, he received the National Science Foundation's CAREER Award. He is currently on the Editorial Board of the Elsevier journal, *Computer Networks*.