Receiving messages in their correct order: Analyzing broadcast protocols in dynamic epistemic logics (a technical report)

Spandan Das[1] and Sujata Ghosh[2]
[1] *Indian Statistical Institute, Kolkata, India*
[2] *Indian Statistical Institute, Chennai, India*
spandandas94@gmail.com, sujata@isichennai.res.in

Keywords: Broadcast Protocols, Correctness of Protocols, Dynamic Epistemic Logic, Algorithm, Complexity

Abstract: In this paper we analyse certain distributed protocols using logical frameworks. In particular, we provide a dynamic epistemic logic analysis of certain broadcast protocols, viz. Birman-Schiper-Stephenson protocol and Lamport's mutual exclusion protocol. In the process, we provide correctness proofs of these protocols via knowledge-based reasoning. We also provide a detailed algorithmic analysis of the logical modeling of these protocols.

1 INTRODUCTION

Distributed systems occur ubiquitously in todays world of computing and as such, a great deal of effort has been provided towards improving our understanding of these systems. A major challenge is to deal with such systems efficiently and effectively. The distributed nature of control and information in these systems lead us to consider the tasks of the systems in terms of the global behavior of the system, even though the actions that an individual processor performs can depend only on its local information.

Since the design of a distributed system involves the behavior and interaction between individual processors in the system, designers frequently find it useful to reason intuitively about processors' states of knowledge at various points in the execution of a system. Over the years, states of knowledge of groups of processors have become useful concepts for the design and analysis of distributed protocols. Epistemic logic and its variants provide us with a proven methodology to analyze these knowledge states of processors and their interactions. A seminal work in this area is (Halpern and Zuck, 1992) which provides a uniform knowledge-based framework to deal with the *sequence transmission* problem in both synchronous and asynchronous settings.

Over the years, such frameworks have been used to model various authentication protocols (for a survey, see (Ahmadi et al., 2019)), and epistemic and temporal epistemic logics played a major role in the analyses of such protocols. Dynamic epistemic logics have also been used to model such protocols, but as described in (Dechesne and Wang, 2010), there are certain limitations to such approaches in terms of handling equivalence of messages. However, these logics are quite adept in handling higher-order uncertainties and partial information in protocols (e.g., see (van Ditmarsch et al., 2014)).

In dynamic epistemic logics, the actions/events bring in changes in the epistemic states of the agents, but these changes happen immediately. Consequently, these logics are generally not appropriate for modelling communication of information where there is a time gap between the send-event and the receive-event, a natural occurrence in asynchronous systems. Recently, some variants of dynamic epistemic logics have been developed to deal with communication in asynchronous systems, e.g., see (Knight et al., 2019; Balbiani et al., 2020), both of which extends the usual dynamic epistemic language so as to express the time-lag in communication.

This work uses a simple variant of dynamic epistemic logic, where the action updates are generally modeled as soft updates (e.g., see (Baltag and Smets, 2008)), i.e., only the agent relations get updated, and there is no change in the set of possible worlds. The logic is used to model communication of information in form of message passing in asynchronous systems. In such systems, communication speed might differ from channel to channel leading to problems in the arrangement of messages from different processes in their correct temporal order. In case the messages are not executed in their correct temporal order, the system might act erroneously. The goal of distributed protocols such as BSS and Lamport's mutual exclusion is to choose an execution order so that such inconsistencies do not occur.

The focus is on the modelling of broadcast protocols (Singhal and Shivaratri, 1994) in the framework mentioned above together with the correctness proofs of such protocols in terms of knowledge-based reasoning provided by the framework. In addition, implementation details and complexity analyses are also provided. We basically model the effect of following these protocols on certain message passing systems. The novelty of this work is two-fold. On one hand, broadcast protocols are natural candidates for analyses in terms of dynamic epistemic logics, but to the best of the knowledge of the author(s), such logical analyses of these protocols have not been done before. On the other hand, a simple variant of dynamic epistemic logic has been used to deal with these asynchronous systems, and that was made possible due to the usage of the different possible orderings of the messages as possible worlds of the underlying Kripke model used to describe such systems.

We note here that these broadcast protocols have been studied quite extensively over the years and correctness proofs are also there in the literature. Our goal here is to showcase the use of dynamic epistemic logic in handling the knowledge-based reasoning of such systems. We believe that this methodology can be adopted for modelling the underlying reasoning processes in different distributed protocols which would provide a better understanding of such protocols applied on asynchronous systems.

We have assumed some properties of the asynchronous systems for our analyses which might not be valid in general. First of all, we assume that each process has a local clock with respect to which it can arrange local send and receive events in their correct temporal order. Secondly, inter-process communication channels are First In First Out (FIFO) in nature, i.e., two messages from the same process are received in their order of generation by any other process. Thus each process can always arrange two messages from the same process in their correct temporal order. Thirdly, the set of messages generated by all the processes is finite and each process has an estimate of this set, that is, each process has an initial knowledge about the set of messages generated in the system. This estimate must be uniform in nature, i.e., all processes must know the same set of messages, and also the actual set of messages generated in the system must be a subset of the estimated set. Due to overestimation if a message is never generated, we can always assume that it is sent at local time ∞ and received at local time ∞ by other processes. If more than two messages from a process remain unsent (i.e., they have sending times at ∞), we assign an arbitrary generation order to them and assume that they are received in the same order by other processes.

In section 2, we introduce the broadcast protocols that we would analyze in this work. Section 3 contains the logical analyses of the protocols, with worked-out examples being provided in the appendices. Section 4 deals with the correctness proofs, whereas, section 5 deals with relevant algorithms and the run-time complexity of the algorithms. We conclude in section 6.

2 BROADCAST PROTOCOLS

In asynchronous distributed systems (Singhal and Shivaratri, 1994), when the communication of information is taken care of by message-passing, more often than not, the messages cannot be ordered according to their time of generation, the main reason being the fact that the global clock is unavailable. However, there is a reasonable way in which one can say that a message m_1 is generated before another message m_2. If m_1 is received by the sender of m_2 before m_2 is sent then evidently m_1 is generated before m_2. Such an ordering, defined by Lamport, is given as follows:

Definition 1. (Causal ordering). Let a and b be two events. We say a causally precedes b (notation $a \rightarrow b$) if one of the following three cases happen:

1. a and b are events on the same process and a chronologically precedes b by local clock of the process.
2. a is the send event and b is the receive event of the same message.
3. There exist an event c such that $a \rightarrow c$ and $c \rightarrow b$.

If none of these three cases happen then we say a and b are concurrent.

Various protocols have been developed for such distributed systems so that the causal ordering of messages could be enforced. For example, in Birman-Schiper-Stephenson (BSS) protocol, the processes always execute messages in the correct causal order, whereas, in Lamport's mutual exclusion (LME) protocol critical section requests are granted in their correct causal order. Before moving any further, let us describe those protocols.

The BSS protocol helps the processes to execute the messages received in correct causal order. If two messages are concurrent, they are sorted in *first come first serve* basis.

Definition 2. BSS Protocol is defined as follows:

1. When a process broadcasts a message it also broadcasts how many messages it has received from each of the other processes.
2. When a process receives a message from another process it checks whether it has received all previous messages from that same process and also whether it knows about at least as many messages in the system as the sender process does. If both checks are true, the recipient process executes the message, otherwise it puts the message on hold and waits.

The LME protocol helps the processes to decide the order of accessing the critical section (a set of objects which can be modified by at most one process at a time) in case of multiple requests coming from different processes. If one request causally precedes another, the former process is granted access before the latter. Additionally, the protocol assumes that each process has a unique process ID from a totally ordered set and in case of concurrent requests, it breaks tie using process ID. This means that the protocol gives first access of the critical section to the process with the smallest ID.

Definition 3. LME protocol is defined as follows in terms of the movements around the critical section:

- Requesting the critical section:
  1. When a process wants to access the critical section, it generates a *request* and a timestamp for that *request*. The process keeps this *request* in its own *request* queue sorted according to timestamps and then broadcasts it to all other processes along with the timestamp.

2. When a process receives a timestamped *request*, it keeps the *request* in its own *request* queue sorted according to timestamps. Then it generates a timestamped *reply* corresponding to the *request* and sends the *reply* to the sender of the *request*.

- Accessing the critical section: When a process observes
  1. its *request* is at the top of its own *request* queue, and
  2. it has received at least one message (*request* or *reply* or *release*) with higher timestamp than its *request* from all other processes,

  the process enters the critical section.
- Exiting the critical section:
  1. When a process exits the critical section, it generates a timestamped *release* message and broadcasts it to all other processes. It deletes its own *request* from its *request* queue.
  2. Upon receiving a *release* message, a process removes the *request* of the sender process from its own *request* queue.

In the remainder of this paper we will model these two protocols in a variant of dynamic epistemic logic, so as to exemplify a novel way of formal modelling such protocols towards proving correctness of the protocols using knowledge-based reasoning.

3 MODELLING PROTOCOLS IN DYNAMIC EPISTEMIC LOGIC

Let us first provide a brief introduction to a relevant variant of dynamic epistemic logic before moving on to the modelling of the protocols. For a detailed exposition, see (van Ditmarsch et al., 2007). Then we would move on to describe the protocols in this dynamic epistemic language.

Syntax: Given a finite set of propositions $\mathcal{P}$, a finite set of agents $\mathcal{A}$ and a finite set of actions Act, the language $\mathcal{L}$ is defined as follows:

$$\phi := p \mid \neg\phi \mid \phi \wedge \phi \mid K_a\phi \mid A\phi,$$

where $p \in \mathcal{P}$, $a \in \mathcal{A}$, $A \in \mathsf{Act}$. The formulas $\phi \vee \psi$, $\phi \rightarrow \psi$ are defined as usual. A formula of the form $K_a\phi$ reads as "ϕ is known to agent a" and $A\phi$ reads as "ϕ holds after some action A". The actions are generally considered to be protocol-dependent and changes the model suitably. We use $L_a\phi$ to denote the formula $\neg K_a \neg\phi$.

Semantics: A model M for this language is a tuple (W,R,V) where W denotes a non-empty set of worlds, R denotes a set of agent-relations and V denotes a valuation function. Each agent defines a relation $R_a \in R$ such that $R_a \subseteq W \times W$. In essence, R_a models knowledge of an agent a which can be modified by certain events or actions. A valuation function $V : \mathcal{P} \rightarrow 2^W$ associates propositions to sets of worlds. For this logic, we consider an action A to bring about certain changes in the set of agent relations, which we will explain in details while modelling the protocols. If an action A is applied to M, we call the resulting new model M_A. The truth definition of a formula at a world $w \in W$ in the model M is as follows:

$$\begin{aligned}
&M,w \models p \text{ iff } p \in V(w),\\
&M,w \models \neg\phi \text{ iff } M,w \not\models \phi,\\
&M,w \models \phi \wedge \psi \text{ iff } M,w \models \phi \text{ and } M,w \models \psi,\\
&M,w \models K_a\phi \text{ iff for all } \tilde{w} \in W \text{ with } wR_a\tilde{w},\ M,\tilde{w} \models \phi\\
&M,w \models A\phi \text{ iff } M_A,w \models \phi.
\end{aligned}$$

$M \models \phi$ means that for all $w \in W$, $M,w \models \phi$.

3.1 BSS Protocol

Before we define the language for analyzing BSS protocol, let us first note some important notations and definitions. Let $\mathcal{A}$ denote the set of agents/processes in the distributed system, and let $\mathcal{M}$ denote the set of messages whose causal ordering we are interested in. Each message is denoted by x_i, which can be read as the i^{th} message from agent x.

Definition 4. (Permutation). We say π is a permutation on a finite set S iff $\pi : S \to \{1,2,\ldots,|S|\}$ is a bijection.

Definition 5. (Allowed permutation). A permutation π on $\mathcal{M}$ is an **allowed permutation** iff $\pi(x_i) < \pi(x_j)$ whenever $i < j$. Here $\pi(x_i)$ ($\pi(x_j)$) denotes the position of x_i (x_j) in the permutation π.

Definition 6. (MSN). At a particular local time of agent x, **MSN** (message to send next) of x (denoted $MSN(x)$) is $k \in \mathbb{N}$ if x has already sent $x_{k-1} \in \mathcal{M}$ but has not sent $x_k \in \mathcal{M}$. However if x has already sent all its messages in $\mathcal{M}$ then $MSN(x)$ is ∞.

3.1.1 Language

Now we are ready to define the language that can describe the content of the exchanged messages. Let $\mathcal{P}$ be the set of propositions. We have,

$$\mathcal{P} = \{P^{x,i} \mid x \in \mathcal{A};\ x_i \in \mathcal{M}\} \cup \{Q^{x,j}_{a,i} \mid a,x \in \mathcal{A};\ a_i,x_j \in \mathcal{M};\ a_i \neq x_j\}.$$

Here $P^{x,i}$ reads "agent x has spoken i times" and $Q^{x,j}_{a,i}$ reads "agent a in its i^{th} message notifies that j^{th} message from agent x has reached it". The formulas are given by,

$$\phi := \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid K_a\phi \mid A^a_\phi\phi.$$

Here $K_a\phi$ reads "agent a knows ϕ". We include $A^a_\phi\psi$ in the language only for Boolean formulas ϕ (no restriction on ψ) and not for more complex formulas since they are not needed for modeling the BSS protocol. For a Boolean formula ϕ and a general formula ψ in the language, $A^a_\phi\psi$ reads "ψ is true after ϕ is received by a". Let us now have the following definitions for the Boolean formulas in the language:

Definition 7. (Sender of a formula). The **sender** of a Boolean formula ϕ is defined recursively,

- $sender(\phi) = x$ if $\phi = P^{x,i}$.
- $sender(\phi) = a$ if $\phi = Q^{x,j}_{a,i}$.
- $sender(\neg\phi) = sender(\phi)$.
- $sender(\phi \wedge \psi) = sender(\phi)$ if $sender(\phi) = sender(\psi)$ and undefined otherwise.

Definition 8. (Index of a formula). The **index** of a Boolean formula ϕ is defined recursively,

- $index(\phi) = i$ if $\phi = P^{x,i}$.
- $index(\phi) = i$ if $\phi = Q^{x,j}_{a,i}$.
- $index(\neg\phi) = index(\phi)$.
- $index(\phi \wedge \psi) = index(\phi)$ if $index(\phi) = index(\psi)$ and undefined otherwise.

3.1.2 Semantics

We define the model M as the tuple (W,R,V) where W is the set of worlds, R is the set of indistinguishability relations and $V : \mathcal{P} \to 2^W$ is the valuation function. We have,

$W = \{\pi \mid \pi$ is an allowed permutation on the underlying
message set $\}$

$R = \{R_a \mid a \in \mathcal{A}; R_a$ is a binary relation on $W\}$. Actually R_a models the knowledge of agent a and is changed by the actions.

We define $V : \mathcal{P} \to 2^W$ as,

$$V(P^{x,i}) = W \text{ for all } x_i \in \mathcal{M}$$
$$V(Q^{x,j}_{a,i}) = \{\pi \in W \mid \pi(x_j) < \pi(a_i)\} \text{ for all } a_i,x_j \in \mathcal{M}.$$

Here $\pi(x_j)$ ($\pi(a_i)$) denotes the position of message x_j (a_i) in the permutation π and the order between two positions is shown by the relation '$<$'.

Definition 9. (Semantics). Let a model M be given. We inductively define the interpretation of formula $\phi \in \mathcal{L}$ on (M,π) as follows,

$$\begin{aligned}
&M,\pi \models \top, \text{ always },\\
&M,\pi \models p \in \mathcal{P} \text{ iff } \pi \in V(p),\\
&M,\pi \models \neg\phi \text{ iff } M,\pi \not\models \phi,\\
&M,\pi \models \phi \wedge \psi \text{ iff } M,\pi \models \phi \text{ and } M,\pi \models \psi,\\
&M,\pi \models K_a\phi \text{ iff for all } \tilde{\pi} \text{ with } \pi R_a \tilde{\pi},\ M,\tilde{\pi} \models \phi,\\
&M,\pi \models A^a_\phi \psi \text{ iff } M_{\phi,a} \models \psi,
\end{aligned}$$

where $M_{\phi,a}$ is the updated model under the action A^a_ϕ. We provide the details of model updation under actions in the following subsection.

3.1.3 Action Update Model to model BSS Protocol

In BSS protocol the content of each message is a formula in $\mathcal{L}$ either of the form $\phi = P^{x,i}$ or of the form $\phi = P^{x,i} \wedge \bigwedge_{y \neq x} Q^{y,j}_{x,i}$, since the sender also broadcasts the information how many messages from each of the other agents it has received along with the message. We now concentrate on such formulas to model the BSS protocol. The initial model is $M = (W,R,V)$ where W is the set of all allowed permutations on $\mathcal{M}$ and for each agent x, R_x is a universal relation on W, i.e., $R_x = W \times W$. Suppose the formula ϕ is received by agent x. Then the corresponding action will transform the current model M into be the model denoted as $M_{\phi,x} = (W, R^{\phi,x}, V)$ where only the agent relations are modified. The new set of agent relations is $R^{\phi,x} = \{R'_a \mid a \in \mathcal{A}\}$ where,

- For each $y \neq x$, $R'_y = R_y$,
- R'_x is the intersection of R_x and the complete relation on E' where, $E' = \{\pi \in W \mid \pi(sender(\phi)_{index(\phi)}) < \pi(x_{MSN(x)})$ and $M,\pi \models \phi\}$.

When a message having content ϕ is received by an agent x, the corresponding action A^x_ϕ is performed on the current model M. This results into transforming M to $M_{\phi,x}$. We note that that the operator A^x_ϕ acts globally. After all messages are received, the set of permutations can be partitioned into two sets. In one set, each permutation π is isolated with respect to every R_x, i.e., $\pi \not R_x \sigma$ for all $\sigma \in W$ and for all x. The other set is universal with respect to every R_x (i.e., for any π, σ in the set, $\pi R_x \sigma$ for all x) and in each of them the messages appear in correct causal order. Formally, $x_i \rightarrow y_j$ if and only if we have that in the final model M_{fin}, $M_{fin} \models \wedge_{a \in \mathcal{A}}(L_a \top \rightarrow K_a Q^{x,i}_{y,j})$. For a worked out example, see Appendix A.

3.2 Lamport's mutual exclusion protocol

For the analysis of LME Protocol, we will develop the language corresponding to *request* messages only, as the other kinds of messages do not play a role in generating the order of these messages. The ordering of *requesst* messages generated by LME Protocol is an index-based ordering where ties between *request* messages with equal indices are broken based on the IDs of the requesting processes. We will later show that this is actually a causal ordering of the *request* messages. Let $\mathcal{A}$ denote the set of agents/processes in the distributed system, and $\mathcal{M}$ denote the set of *request* messages whose causal ordering we are interested in. Each *request* is of the form x_i, which means that it is the i^{th} request made by agent x.

Definition 10. (Permutation). We say π is a permutation on a finite set S iff $\pi : S \rightarrow \{1,2,\ldots,|S|\}$ is a bijection.

Definition 11. (Allowed permutation). A permutation π on $\mathcal{M}$ is an **allowed permutation** iff $\pi(x_i) < \pi(x_j)$ whenever $i < j$. Here $\pi(x_i)$ $(\pi(x_j))$ denotes the position of x_i (x_j) in the permutation π.

Definition 12. (Index of a message). **Index of a message** generated by a process is 1 if the process has not sent or received any other message before. Otherwise it is $k+1$ where k is the maximum index of all messages received or sent by the process before.

Definition 13. (Timestamp of a message). **Timestamp of a message** is generated by concatenating its index (definition 12) at the front of the ID of the sender process.

3.2.1 Language

We will now define the language for modeling the LME protocol. The set of propositions is given by,

$$\mathcal{P} = \{Q_{x,i}^{y,j} \mid x,y \in \mathcal{A};\ x_i, y_j \in \mathcal{M};\ x_i \neq y_j\}.$$

Here, $Q_{x,i}^{y,j}$ reads "*request* x_i causally precedes *request* y_j". The formulas are given by,

$$\phi := \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid K_a\phi \mid A_s^{x_i,j}\phi \mid A_{r,y}^{x_i,j}\phi,$$

where, $p \in \mathcal{P}$. As earlier, the formula $K_a\phi$ reads as "agent a knows ϕ". $A_s^{x_i,j}\phi$ reads "after *request* x_i with index j is sent, ϕ holds" and $A_{r,y}^{x_i,j}\phi$ reads "after *request* x_i with index j is received by y, ϕ holds".

3.2.2 Semantics

The model M is defined as the tuple (W, R, Ind, V) where,

$W = \{\pi \mid \pi$ is an allowed permutation on $\mathcal{M}\}$.

$R = \{R_a \mid a \in \mathcal{A};\ R_a$ is a binary relation on $W\}$. Actually, R_a models the knowledge of agent a and is modified by the actions.

$Ind = \{Ind_a \mid a \in \mathcal{A}\}$ **where** $Ind_a = \{(a_i, j) \mid a_i \in \mathcal{M}, index(a_i) = j \in \mathbb{N}\}$. Actually Ind_a stores the *requests* observed by agent a and their corresponding indices.

$V : \mathcal{P} \to 2^W$ **defined as** $V(Q_{x,i}^{y,j}) = \{\pi \mid \pi(x_i) < \pi(y_j)\}$. Here $\pi(x_i)$ ($\pi(y_j)$) denotes the position of *request* x_i (y_j) in the permutation π and the order between two positions is shown by the relation '$<$'.

Definition 14. (Semantics). Let a model M be given. The truth definition of the formulas ϕ of the language above at (M, π) is given as follows,

$$\begin{aligned}
&M,\pi \models \top, \text{ always },\\
&M,\pi \models Q_{x,i}^{y,j} \text{ iff } \pi \in V(Q_{x,i}^{y,j})\\
&M,\pi \models \neg\phi \text{ iff } M,\pi \not\models \phi,\\
&M,\pi \models \phi \wedge \psi \text{ iff } M,\pi \models \phi \text{ and } M,\pi \models \psi,\\
&M,\pi \models K_a\phi \text{ iff for all } \tilde{\pi} \text{ with } \pi R_a \tilde{\pi},\ M,\tilde{\pi} \models \phi,\\
&M,\pi \models A_s^{x_i,j}\psi \text{ iff } M',\pi \models \psi,\\
&M,\pi \models A_{r,y}^{x_i,j}\psi \text{ iff } M'',\pi \models \psi.
\end{aligned}$$

Here M' denotes the updated model under the action $A_s^{x_i,j}$ and M'' denotes the updated model under the action $A_{r,y}^{x_i,j}$ (with respect to the current model M). We provide the details of model updation under actions in the following subsection.

3.2.3 Action Update Model to model LEM Protocol

The model M changes when a *request* is generated or received by a process. The corresponding action transforms the model M to the model $M' = (W, R', Ind', V)$ where only the agent relations and Ind sets corresponding to the agents are modified.

Suppose *request* x_i is generated by agent x.

- If $y \neq x$ then $Ind'_y = Ind_y$.
- $Ind'_x = Ind_x \cup \{(x_i, j)\}$ where $j = index(x_i)$.
- If $y \neq x$ then $R'_y = R_y$.

- R'_x is the intersection of R_x and the complete relation on E' where
$$E' = \{\pi \in W \mid \forall (y_j, k) \in Ind_x; \pi(y_j) < \pi(x_i)\}.$$

  Suppose *request* x_i is received by agent y.
- If $z \neq y$ then $Ind'_z = Ind_z$.
- $Ind'_y = Ind_y \cup \{(x_i, j)\}$ where $j = index(x_i)$.
- If $z \neq y$ then $R'_z = R_z$.
- R'_y is the intersection of R_y and the complete relation on E' where,
  $E' = \{\pi \in W \mid \forall (z_j, k) \in Ind_y; \pi(x_i) < \pi(z_j)$ iff
  $index(x_i) < k$ and $\pi(x_i) > \pi(z_j)$ iff $index(x_i) > k\}$.

The initial model in $M = (W, R, Ind, V)$ where W is the set of all allowed permutations on $\mathcal{M}$, for each agent x, R_x is a universal relation on W (i.e., $R_x = W \times W$) and $Ind_a = \emptyset$ for all $a \in \mathcal{A}$. When an agent x sends its i^{th} *request* or when an agent y receives the *request* x_i, the corresponding actions $A_s^{x_i,j}$ or $A_{r,y}^{x_i,j}$ is performed on the current model respectively. The model update happens according to the rules defined above. After all *requests* are received, the set of permutations gets partitioned into two sets, as in the case of BSS Protocol. In one set, each permutation is isolated with respect to every R_x (i.e., $\pi \not R_x \sigma$ for all $\sigma \in W$ and for all x) and the other set is universal with respect to every R_x (i.e., for any π, σ in the set, $\pi R_x \sigma$ for all x) and in each world of it, the *requests* appear in correct causal order. Observe that $index(x_i) < index(y_j)$ iff in the final model $M_{fin} \models \wedge_{a \in \mathcal{A}} (L_a \top \rightarrow K_a Q^{y,j}_{x,i})$, i.e., every agent considers only those permutations where the *requests* are ordered according to their indices. For a worked out example, see Appendix B.

4 CORRECTNESS OF THE PROTOCOLS

In the following, we prove the correctness of the protocols, using knowledge-based reasoning as provided by the formal frameworks we just discussed.

4.1 Correctness of BSS Protocol

It is claimed that in BSS protocol, each agent executes messages in the correct causal order. To prove this we will show that after all the messages are received, each agent knows the causal order between any two messages. Since any agent executes messages only in one of the orders known to it, it is not possible for an agent to violate the causal order. In logical language, we will show that the final model $M_{fin} \models \wedge_{a \in \mathcal{A}} (L_a \top \rightarrow K_a Q^{x,i}_{y,j})$ iff $x_i \rightarrow y_j$. The following theorems prove this.

Theorem 1. (Agreement). After all the messages are received, an agent a is sure that x_i precedes y_j iff every other agent in the system is sure about this. Formally, we have that $M_{fin} \models (L_a \top \rightarrow K_a Q^{x,i}_{y,j}) \leftrightarrow (\wedge_{b \in \mathcal{A}; b \neq a} (L_b \top \rightarrow K_b Q^{x,i}_{y,j}))$.

Proof. ($\Rightarrow$) Let us assume an agent a knows that x_i precedes y_j, i.e., $M_{fin} \models L_a \top \rightarrow K_a Q^{x,i}_{y,j}$. Now agent a can only gather this information via some message passed in the system because initially it cannot distinguish any two permutations. If this information is gathered from a $Q^{x,i}_{y,j}$ formula then every other agent has also received it (since this formula only appears in a broadcast message). Thus every agent of the system is aware of the information. On the other hand, agent a can be y itself and when it received the message $P^{x,i}$ it still has not sent the message y_j. Thus it knows $Q^{x,i}_{y,j}$. But when it will send its next message, which may be y_k ($k \leq j$), it will share the information x_i precedes y_k with all the other agents via the formula $Q^{x,i}_{y,k}$. Thus every other agent will know that x_i precedes y_k, which implies x_i precedes y_j because $k \leq j$.
($\Leftarrow$) Conversely, let us assume a does not know that x_i precedes y_j. We need to prove that every other agent is ignorant about this information. Let us suppose, if possible, some agent b knows this information. Again b can know it only by a message. It cannot be a Q-type message since then a should also have received it (Q-type messages are broadcast to every agent. Since message passing is reliable, every agent should receive the same set

of Q-messages). Thus b must have known it from a P-type message. But then b must have had at least one more message to send when it received the P-type message. So b should have shared this information with a when it broadcast the next message. Hence contradiction. Thus any other agent b cannot know this information. $\square$

Definition 15. (Basic precedence). We say a causal precedence $a_i \rightarrow b_j$ is a basic precedence iff either $a = b$ and a_i is sent before a_j or $a \neq b$ and a_i is received by b before b_j is sent.

It is easy to observe from the definition of causal order that if $a_i \rightarrow b_j$ then either it is a basic precedence or there are $c^{(1)}_{i_1}, \dots, c^{(m)}_{i_m}$ such that $a_i \rightarrow c^{(1)}_{i_1} \rightarrow \cdots \rightarrow c^{(m)}_{i_m} \rightarrow b_j$ where each precedence is a basic precedence. Let us use this observation to show that each agent knows all the causal orders when BSS protocol is used.

Theorem 2. If $x_i \rightarrow y_j$ is a basic precedence then agent y knows x_i precedes y_j. In other words, in the final model $M_{fin} \models L_y\top \rightarrow K_y Q^{x,i}_{y,j}$.

Proof. If $x = y$ this claim trivially holds since all permutations are allowed permutations. So we assume $x \neq y$. Then x_i has been received by y before y_j is sent. Hence at the time x_i is received by y, $MSN(y) \leq j$. Thus the updated model at that point of time, say M', will satisfy $L_y\top \rightarrow K_y Q^{x,i}_{y,MSN(y)}$. This automatically implies $M' \models L_y\top \rightarrow K_y Q^{x,i}_{y,j}$ since $MSN(y) \leq j$. Thus M_{fin} will also satisfy this since y will retain this knowledge at all later updated models. $\square$

Using theorem 1 and theorem 2, we can see each basic precedence is known to all agents. Since any causal precedence can be built up from basic precedence only, we get that all causal precedence are known to every agent. Thus the claim that each agent knows all and only the causal orders after message passing, is true. Thus, no agent can violate the causal order while executing messages. We also observe that BSS protocol generates a unique allowed permutation iff for any two messages x_i and y_j, either every agent knows $x_i \rightarrow y_j$ or every agent knows $y_j \rightarrow x_i$. In logical language, BSS protocol generates a unique permutation iff the final model $M_{fin} \models \wedge_{x_i \neq y_j}((\wedge_{a \in \mathcal{A}}(L_a\top \rightarrow K_a Q^{x,i}_{y,j})) \vee (\wedge_{a \in \mathcal{A}}(L_a\top \rightarrow K_a Q^{x,i}_{y,j})))$.

4.2 Correctness of LME Protocol

For LME protocol, we have to show that no two *requests* can access the critical section at the same time. To prove this we will show that all agents agree on a unique permutation of the *requests* and the critical section is accessed in that order only. Thus there cannot be any conflict between the decisions of any two agents. We will also show that the permutation each agent agrees on, is actually a causal order of the *requests*.

Definition 16. (Order of precedence). Let $x_i \rightarrow y_j$. We define order of causal precedence $x_i \rightarrow y_j$ (denoted $ord(x_i \rightarrow y_j)$) to be 1 iff $x_i \rightarrow y_j$ is a basic precedence (definition 15). Otherwise we define $ord(x_i \rightarrow y_j)$ to be $m > 1$, where m is the smallest natural number such that there exist $c^{(1)}_{i_1}, \dots, c^{(m)}_{i_m}$ with $x_i \rightarrow c^{(1)}_{i_1} \rightarrow \cdots \rightarrow c^{(m)}_{i_m} \rightarrow y_j$, where each of the precedences is a basic precedence.

Theorem 3. If two *requests* x_i and y_j are such that $x_i \rightarrow y_j$, then $index(x_i) < index(y_j)$.

Proof. We prove this by induction on $ord(x_i \rightarrow y_j)$
Base case : When $ord(x_i \rightarrow y_j) = 1$ there may be two cases.

1. x_i and y_j are generated by the same process with x_i generated ahead of y_j. Then clearly $index(x_i) < index(y_j)$ by definition of index of a message. Hence our claim holds.
2. A process receives x_i and sends y_j and the point of receipt of x_i lies before the point of sending of y_j. Then also by the definition of index of a message, $index(x_i) < index(y_j)$ and our claim holds.

Inductive hypothesis : Assume the claim to be true for any $x_i \rightarrow y_j$ such that $ord(x_i \rightarrow y_j) = m - 1$.
Inductive step : Now let x_i and y_j be two *requests* and $ord(x_i \rightarrow y_j) = m$. Then there exists z_k such that $ord(x_i \rightarrow z_k) = m - 1$ and $ord(z_k \rightarrow y_j) = 1$. Then by inductive hypothesis $index(x_i) < index(z_k)$ and from base case $index(z_k) < index(y_j)$. Thus $index(x_i) < index(z_k) < index(y_j)$ and our claim holds. $\square$

Theorem 4. (Agreement). Each agent agrees on permutations where *requests* appear in correct causal order and all agents agree on same set of permutations.

Proof. Take any agent $a \in \mathcal{A}$. Let two *requests* x_i and y_j be observed by a. If $x_i \to y_j$, then by previous theorem 3, $index(x_i) < index(y_j)$. By rule of the update model, a then agrees on only those permutations π such that $\pi(x_i) < \pi(y_j)$ (remember that $M_{fin} \models L_a\top \to K_aQ^{y,j}_{x,i}$ since in the final model both x_i and y_j have been observed by a and thus $(x_i, index(x_i)), (y_j, index(y_j)) \in Ind_a$). Since a observes all *requests* made eventually (*requests* are broadcast), a agrees only on permutations where *requests* appear in correct causal order. Since choice of $a \in \mathcal{A}$ is arbitrary, every agent agrees on permutations where *requests* appear in correct causal order.
Since all agents observe the same set of *requests*, they must agree on the same set of permutations. Hence we are done. □

Now if every agent breaks tie between *requests* with same index using process ID (i.e., the process with smallest ID wins in case of a tie in indices), they must agree on a unique permutation since without using process ID they agree on same set of permutations. Thus LME protocol leads to a unique permutation of *requests* which is agreed upon by all agents. This proves the correctness of the protocol. In other words, each agent knows the order in which the *requests* should access the critical section.

5 ON IMPLEMENTATION AND TIME COMPLEXITY

In all our models we have used only allowed permutations as the set of states. Let us first provide an estimate for the number of allowed permutations. Let for each agent $a \in \mathcal{A}$, $\mathcal{M}_a$ be the set of messages sent by agent a. Then $\mathcal{M} = \cup_{a\in\mathcal{A}}\mathcal{M}_a$. Since $\mathcal{M}_a \cap \mathcal{M}_b = \emptyset$ for $a \neq b$, $|\mathcal{M}| = \sum_{a\in\mathcal{A}}|\mathcal{M}_a|$. Let SA be the set of allowed permutations. In each allowed permutation, all messages from a particular agent (say a) must appear in a particular order. In this respect, all messages from one agent are practically indistinguishable. The problem of determining the size of SA is now actually equivalent to the problem of determining the number of permutations of $|\mathcal{M}|$ balls of $|\mathcal{A}|$ distinct types where two balls of the same type are indistinguishable. Thus,

$$|SA| = \frac{|\mathcal{M}|!}{\prod_{a\in\mathcal{A}}|\mathcal{M}_a|!}$$

In case each agent sends only one message, we have $|SA| = |\mathcal{M}|!$.

5.1 Implementation of the model for BSS protocol

In this section we will provide a possible implementation of the action update model for BSS protocol and its time complexity. The implementation has two steps:

1. Given set of agents $\mathcal{A}$ and set of messages $\mathcal{M}$ build the initial model according to BSS protocol.
2. Update the model according to BSS protocol as messages are received by agents until each message is received by all other agents except for the sender.

5.1.1 Time complexity of the action update model for BSS protocol

Let us now discuss the time complexity of the above implementation stepwise.

1. In the initial model for each agent $a \in \mathcal{A}$ the agent relation R_a is universal on SA (set of allowed permutations on $\mathcal{M}$), i.e., $R_a = SA \times SA$. Thus it is enough to store $support(R_a)$ instead of R_a for each a where by $support(R_a)$ we denote the set $\{\pi \mid \exists\pi'$ such that $\pi R_a \pi'$ or $\pi' R_a \pi\}$. Initially $support(R_a) = SA$ for all a. Thus we make $|\mathcal{A}|$ copies of SA and associate to each agent. This will take $O(|\mathcal{A}||\mathcal{M}|!)$ time.
2. Suppose message with formula ϕ is received by agent a. To update the model we only need to modify $support(R_a)$ to $support(R_a) \cap E'$ where $E' = \{\pi \in SA \mid \pi(sender(\phi)_{index(\phi)}) < \pi(a_{MSN(a)})$ and $M, \pi \models \phi\}$ (here M is the current model not yet updated). We can save time by a simple check whether we need to update the model at all. If ϕ is of the form $P^{x,i}$ and $MSN(a) = \infty$ then there is no need to update $support(R_a)$. This check can be performed in $O(1)$ time. In all other cases we need to update $support(R_a)$ but in doing that we fix the order of at least two messages which was not previously fixed. Thus size of $support(R_a)$ reduces by at least half of its current size. To check whether a $\pi \in support(R_a)$ also belongs to E' we need at most $O(|support(R_a)|)$ time. Thus for each agent, total time for model updation is at most $O(|\mathcal{M}|! + |\mathcal{M}|!/2 + |\mathcal{M}|!/4 + \cdots + 1) = O(|\mathcal{M}|!)$. Thus for all the agents total time is $O(|\mathcal{A}||\mathcal{M}|!)$.

Thus total time complexity of the entire implementation is $O(|\mathcal{A}||\mathcal{M}|! + |\mathcal{A}||\mathcal{M}|!)$, i.e., $O(|\mathcal{A}||\mathcal{M}|!)$.

5.2 Implementation of the model for LME protocol

In this section we will provide a possible implementation of the action update model for LME protocol and its time complexity. The implementation has two steps:

1. Given a set of agents $\mathcal{A}$ and a set of *requests* $\mathcal{M}$ build the initial model according to LME protocol.
2. Update the model according to LME protocol as messages are sent and received by agents until each message is received by all other agents except the sender.

5.2.1 Time complexity of the action update model for LME protocol

Let us now discuss the time complexity of the above implementation stepwise.

1. We can build the initial model exactly like we did for BSS protocol. This is because for each agent $a \in \mathcal{A}$, R_a is a complete relation on $\mathcal{M}$ and $Ind_a = \emptyset$. Thus time complexity for building the initial model is $O(|\mathcal{A}||\mathcal{M}|!)$.
2. We will discuss the time complexity for two cases, when a *request* is sent by an agent and when a *request* is received by an agent.
   (a) Suppose *request* x_i is sent by agent x. Now we need to include x_i along with its index into Ind_x. This can be done in $O(1)$ time. Now $support(R_x)$ must be modified to $support(R_x) \cap E'$ where $E' = \{\pi \in SA \mid \forall (y_j, k) \in Ind_x; \pi(y_j) < \pi(x_i)\}$. We can save time by performing a simple check whether we need to modify $support(R_x)$ at all. If Ind_x contains only *requests* from x, we need not update R_x since each π is an allowed permutation. Also if for each *request* y_j in Ind_x $(y \neq x)$ index of y_j is same as that of x_i, we need not update R_x. Both these checks can be performed in $O(1)$ time using suitable data structures. For example, we can maintain 3 counters $IndO$, $MaxO$, $MinO$ for each $x \in \mathcal{A}$. $IndO$ stores the number of agents other than x whose *requests* are received by x. $MaxO$ stores the maximum index of such REQUESTs and $MinO$ stores the minimum index of such *requests*. These counters can be updated in $O(1)$ time during model updation. Now for the first check we need to see if $IndO > 0$ and for the second check we need to see if $MaxO = MinO = index(x_i)$. Both checks can be done in $O(1)$ time. In all other cases, $support(R_x)$ must be modified and the size of the updated $support(R_x)$ reduces by at least half. The reason for this is discussed in detail in section 5.1.1. Using the reasoning from section 5.1.1 we conclude that the total time complexity for model updation during sending *requests* for any agent is $O(|\mathcal{M}|!)$.
   (b) For the other case, let us suppose *request* x_i is received by agent y. As before, inclusion of x_i into Ind_y can be done in $O(1)$ time. However, time for updating $support(R_y)$ can be reduced using the following check. Note that $support(R_y)$ must be modified to $support(R_y) \cap E'$ where $E' = \{\pi \in SA \mid \forall (z_j, k) \in Ind_y; \pi(x_i) < \pi(z_j) \text{ iff } i < k \text{ and } \pi(x_i) > \pi(z_j) \text{ iff } i > k\}$. Now if all *requests* in Ind_y has the same index as x_i, we need not modify R_y. This check can be performed in $O(1)$ time using suitable data structures. We maintain 2 counters Max and Min for each $y \in \mathcal{A}$. The first counter maintains the maximum index of any *request* received by y whereas the second counter maintains the minimum index of any *request* received by y. These counters can be updated in $O(1)$ time during model updation. Now we can simply perform the check $Max = Min = index(x_i)$ to see whether all *requests* of Ind_y has the same index as x_i. Clearly this takes $O(1)$ time. For all the other cases $support(R_y)$ must be modified and the size of the updated $support(R_y)$ reduces by at least half. The reason for this is discussed in detail in section 5.1.1. Using the same reasoning, we conclude that the total time complexity for model updation while receiving *requests* for any agent is $O(|\mathcal{M}|!)$.

   Thus total time for model updation is $O(|\mathcal{A}||\mathcal{M}|!)$ considering each agent spends $O(|\mathcal{M}|!)$ in updating the model while sending and receiving *requests*.

Thus total time complexity of the entire implementation is $O(|\mathcal{A}||\mathcal{M}|! + |\mathcal{A}||\mathcal{M}|!)$, i.e, $O(|\mathcal{A}||\mathcal{M}|!)$.

6 CONCLUSION AND FUTURE DIRECTIONS

To summarize, we have shown that simple variants of dynamic epistemic logic can indeed be used to model distributed protocols in asynchronous settings. The knowledge-based framework can be used to model the effect of these protocols, and the underlying reasoning process becomes explicit. In some sense, the action update operators model the effect of receipt of messages, and can be considered as private announcements to the individual processes/agents. However, the logical languages developed here are protocol-specific and quite ad hoc in nature. It would be interesting to see whether a uniform framework could be developed and how that framework would relate to the existing literature on announcement logics. We leave this for the future.

Another direction to work on is to bring other distributed protocols in the purview of the logical framework described here. To this end, one can consider Maekawa's quorum based mutual exclusion algorithm (Singhal and Shivaratri, 1994), where each process consults only a proper subset of all processes before accessing the critical section. It would be interesting to see the variant of dynamic epistemic logic which could be used to model the underlying reasoning process of the protocol. Once again, we leave this for future work.

REFERENCES

Ahmadi, S., Fallah, M. S., and Pourmahdian, M. (2019). On the properties of epistemic and temporal epistemic logics of authentication. *Informatica*, 43(2).

Balbiani, P., van Ditmarsch, H., and González, S. F. (2020). From public announcements to asynchronous announcements. In Giacomo, G. D., Catala, A., Dilkina, B., Milano, M., Barro, S., Bugarn, A., and Lang, J., editors, *Proceedings of the 24th European Conference on Artificial Intelligence*, volume 325 of *Frontiers in Artificial Intelligence and Applications*.

Baltag, A. and Smets, S. (2008). A qualitative theory of dynamic interactive belief revision. In Bonanno, G., van der Hoek, W., and Wooldridge, M., editors, *Logic and the Foundations of Game and Decision Theory (LOFT07)*, volume 3 of *Texts in Logic and Games*, pages 9–58. Amsterdam University Press, Amsterdam.

Dechesne, F. and Wang, Y. (2010). To know or not to know: epistemic approaches to security protocol verification. *Synthese*, 177(1):51–76.

Halpern, J. Y. and Zuck, L. D. (1992). A little knowledge goes a long way: knowledge-based derivations and correctness proofs for a family of protocols. *Journal of the ACM (JACM)*, 39(3):449–478.

Knight, S., Maubert, B., and Schwarzentruber, F. (2019). Reasoning about knowledge and messages in asynchronous multi-agent systems. *Mathematical Structures in Computer Science*, 29(1):127–168.

Singhal, M. and Shivaratri, N. (1994). *Advanced Concepts in Operating Systems: Distributed, Database, and Multiprocessor Operating Systems*. Computer Science Series. McGraw-Hill.

van Ditmarsch, H., Ghosh, S., Verbrugge, R., and Wang, Y. (2014). Hidden protocols: Modifying our expectations in an evolving world. *Artificial Intelligence*, 208:18–40.

van Ditmarsch, H., van der Hoek, W., and Kooi, B. (2007). *Dynamic Epistemic Logic*, volume 337 of *Synthese Library*. Springer Verlag, Berlin.

Appendix A

Let us assume a message-passing system with 3 agents, namely a, b, c. Each agent broadcasts one message, so $\mathcal{M} = \{a_1, b_1, c_1\}$. The set of allowed permutations is $\{a_1b_1c_1, a_1c_1b_1, b_1a_1c_1, b_1c_1a_1, c_1a_1b_1, c_1b_1a_1\}$. The initial model is M^0. We have, $M^0 \models \wedge_{a\in\mathcal{A}} L_a\top$. As messages are received, knowledge of each agent changes. The time points where messages are received are numbered in the space-time diagram. We will discuss about each of these points sequentially. For reference, see Figure 1.

- The first message received is a_1 with formula $P^{a,1}$. The knowledge of agent b will be updated according to the action update model. Since the permutations $\{b_1c_1a_1, b_1a_1c_1, c_1b_1a_1\}$ do not satisfy the condition $\pi(a_1) < \pi(b_{MSN(b)})$, they are not related to any world by b. Observe that in the new model M^1, $M^1 \models L_b\top \to K_bQ^{a,1}_{b,1}$, or in other words $M^0 \models A^b_{P^{a,1}}(L_b\top \to K_bQ^{a,1}_{b,1})$.
- The next message received is b_1 with formula $P^{b,1} \wedge Q^{a,1}_{b,1}$. The knowledge of agent c is updated. Since the permutations $\{b_1c_1a_1, b_1a_1c_1, c_1b_1a_1\}$ do not satisfy $Q^{a,1}_{b,1}$, they are not related to any world by c. Observe that in the new model M^2, $M^2 \models L_c\top \to K_cQ^{a,1}_{b,1}$, or in other words $M^1 \models A^c_{P^{b,1}\wedge Q^{a,1}_{b,1}}(L_c\top \to K_cQ^{a,1}_{b,1})$.

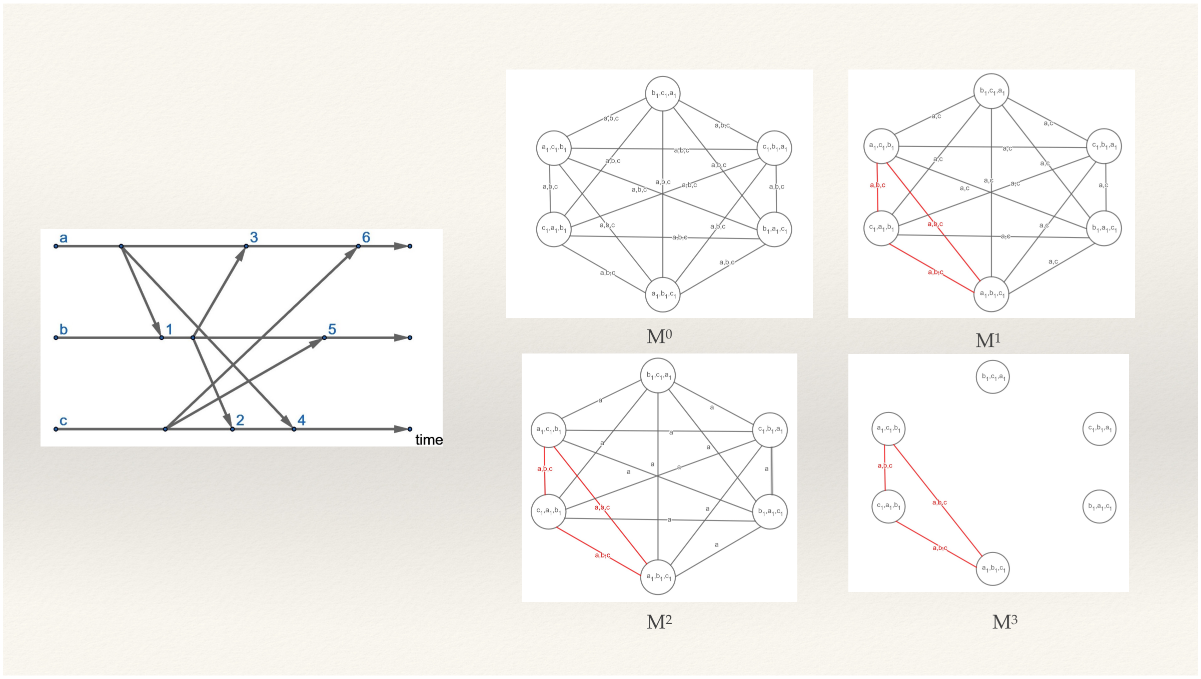

Figure 1: An exemplification of BSS Protocol

- The next message received is b_1 (by a), and the knowledge of a is updated similarly as in the previous step.
- The message received is a_1 with corresponding formula $P^{a,1}$. The recipient is c. But according to the action update model, no further change occurs to the model M^3 because a does not have any message left to send. The rest of the cases can be handled similarly.

Here, M^3 is the final model. Observe that $M^3 \models \wedge_{a\in\mathcal{A}}(L_a\top \rightarrow K_a Q^{a,1}_{b,1})$, since $a_1 \rightarrow b_1$. Also observe that since c_1 is concurrent with both a_1 and b_1, none of the agents know the actual position of c_1. Notice that for a_1 and c_1, neither a_1 was received by c before sending c_1 nor c_1 was received by a before sending a_1. Thus we cannot eliminate relation between any two worlds by the action update model. The same thing is true for b_1 and c_1 as well. Accordingly, each of a,b,c has complete relation on a set of permutations ($\{a_1b_1c_1, a_1c_1b_1, c_1a_1b_1\}$ is the set of permutations in this case) such that in each of these permutations, the messages appear in correct causal order.

Appendix B

The initial assumptions are the same as the previous example with 3 agents and 3 *request* messages, given by the set $\mathcal{M} = \{a_1, b_1, c_1\}$ with allowed permutations as earlier. The initial model is given by M^0. Observe that, initially index of all the *requests* are unknown and hence their orderings are unknown as well. Thus, for any $\pi \in W$ and for any $a \in \mathcal{A}$, $M,\pi \models \neg K_a Q^{y,j}_{x,i}$ where $x_i, y_j \in \mathcal{M}$ and $x \neq y$. The points of sends and receives are numbered in the space-time diagram. For reference, see Figure 2. We will discuss about each of these points sequentially.

- The *request* a_1 is sent with corresponding index 1. Since $Ind_a = \emptyset$, R'_a is complete relation on W. $Ind'_a = \{(a_1, 1)\}$. Thus $R'_a = R_a$. R'_b and R'_c do not change since b and c are neither sender nor receiver. Thus the initial model M^0 does not change.
- The *request* c_1 is sent with corresponding index 1. $Ind'_c = \{(c_1, 1)\}$. The initial model does not change for similar reason as above.
- The *request* a_1 is received by b. So $Ind'_b = \{(a_1, 1)\}$. The initial model does not change.

- The *request* b_1 is sent with corresponding index 3 (index is 3 since b has sent a *reply* to a before this). Now $Ind'_b = \{(a_1,1),(b_1,3)\}$. Since b_1 is generated by b after a_1 is received, it must be placed after a_1 by condition of the relation update model. Thus R'_b is a complete relation on $\{a_1b_1c_1, c_1a_1b_1, a_1c_1b_1\}$. R'_a and R'_c remain unchanged. In the new model M^1, R'_b is marked in red. Observe that $M^1 \models L_b\top \rightarrow K_bQ^{b,1}_{a,1}$ (alternatively, $M^0 \models A^{b_1,3}_{s}(L_b\top \rightarrow K_bQ^{b,1}_{a,1})$).
- The *request* b_1 (index 3) is received by c. Now $Ind'_c = \{(c_1,1),(b_1,3)\}$. Since index of b_1 is higher than that of c_1, R'_c is the complete relation on $\{c_1b_1a_1, a_1c_1b_1, c_1a_1b_1\}$. R'_a and R'_b do not change. In the new model M^2, R'_c is marked in red. Observe that $M^2 \models L_c\top \rightarrow K_cQ^{b,1}_{c,1}$ (alternatively, $M^1 \models A^{b_1,3}_{r,c}(L_c\top \rightarrow K_cQ^{b,1}_{c,1})$).
- The *request* b_1 (index 3) is received by a. Now $Ind'_a = \{(a_1,1),(b_1,3)\}$. According to the update model, given that index of b_1 is higher than a_1, R'_a is complete relation on $\{a_1c_1b_1, a_1b_1c_1, c_1a_1b_1\}$. R'_b and R'_c do not change. In the new model M^3, R'_a is marked in red. Observe that $M^3 \models L_a\top \rightarrow K_aQ^{b,1}_{a,1}$ (alternatively $M^2 \models A^{b_1,3}_{r,a}(L_a\top \rightarrow K_aQ^{b,1}_{a,1})$).
- The *request* a_1 (index 1) is received by c. Now $Ind'_c = \{(c_1,1),(b_1,3),(a_1,1)\}$. According to the update model, since index of b_1 is higher than both c_1 and a_1, R'_c is complete relation on $\{a_1c_1b_1, c_1a_1b_1\}$. R'_a and R'_b remain as they were. In the new model M^4, R'_c is marked in red. Observe that $M^4 \models L_c\top \rightarrow K_c(Q^{b,1}_{a,1} \wedge Q^{b,1}_{c,1})$.
- The*request* c_1 (index 1) is received by b. Now $Ind'_b = \{(a_1,1),(b_1,3),(c_1,1)\}$. For similar reasons as above, R'_b is complete relation on $\{a_1c_1b_1, c_1a_1b_1\}$. No other change occurs. In the new model M^5, R'_b is marked in red. Observe that $M^5 \models L_b\top \rightarrow K_b(Q^{b,1}_{a,1} \wedge Q^{b,1}_{c,1})$.
- The *request* c_1 (index 1) is received by a. Now $Ind'_a = \{(a_1,1),(b_1,3),(c_1,1)\}$. R'_a is now a complete relation on $\{a_1c_1b_1, c_1a_1b_1\}$. No other change occurs. In the new model M^6, R'_a is marked in red. Observe that $M^6 \models L_a\top \rightarrow K_a(Q^{b,1}_{a,1} \wedge Q^{b,1}_{c,1})$.

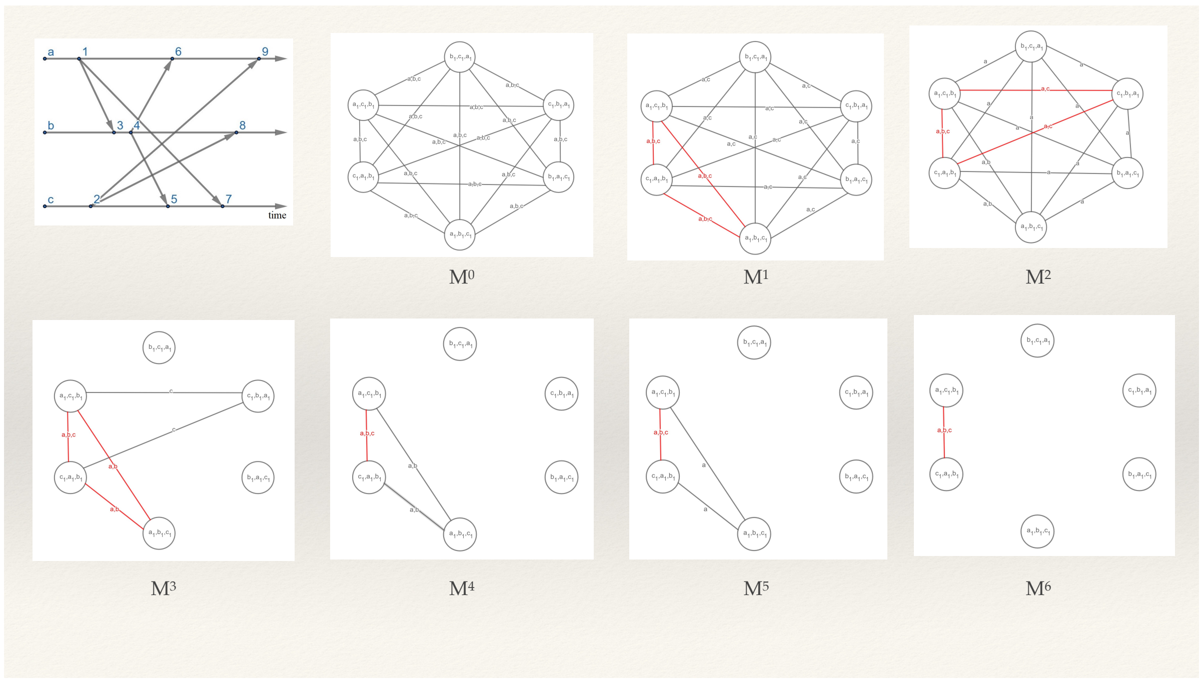

Figure 2: An exemplification of LME Protocol

Thus after all *requests* are received, each of a, b, c has complete relation on the set of permutations $\{a_1c_1b_1, c_1a_1b_1\}$. According to the space-time diagram, $a_1 \rightarrow b_1$ and c_1 is concurrent with both a_1 and b_1. Observe that in each of these permutations, a_1 appears ahead of b_1. Thus these permutations preserve causal ordering of the *requests*. Note that although c_1 is concurrent with both a_1 and b_1, we have some constraints on the position of c_1 in the permutations. Since LME protocol orders *requests* according to their indices and index of c_1 is less than that of b_1, c_1 must always be placed ahead of b_1. Hence we end up with 2 permutations instead of 3. Observe that the final model $M^6 \models \wedge_{x\in\mathcal{A}}(L_x\top \rightarrow K_x(Q^{b,1}_{a,1} \wedge Q^{b,1}_{c,1}))$ which shows that each agent only knows those permutations where *requests* are ordered with respect to their indices.