Characterization of Message Ordering Specifications and Protocols

V. V. Murty and V. K. Garg

TR ECE-PDS-96-002 December 1996

Parallel & Distributed Systems group
Department of Electrical & Computer Engineering
University of Texas at Austin
Austin, Texas 78712

Characterization of Message Ordering Specifications and Protocols

V. V. Murty and V. K. Garg
Department of Electrical & Computer Engineering
University of Texas at Austin
email: {murty, vijay}@pine.ece.utexas.edu

Abstract

We study the problem of determining which message ordering specifications can be implemented in a distributed system. Further, if a specification can be implemented, we give a technique to determine whether it can be implemented by tagging information with user messages or if it requires control messages. To specify the message ordering, we use a novel method called *forbidden predicates*. All existing message ordering guarantees such as FIFO, flush channels, causal ordering, and logically synchronous ordering, (as well as many new message orderings) can be concisely specified using forbidden predicates. We then present an algorithm that determines from the forbidden predicate the type of protocol needed to implement that specification.
Keywords: Message ordering, forbidden predicate, predicate graph, protocols and specifications.

1 Introduction

A distributed computation or a run describes an execution of a distributed program. At an abstract level, a run can be defined as a partially ordered set $(\mathtt{H}, \triangleright)$, where $\mathtt{H}$ is the set of events in the system and $\triangleright$ the "happened before" relation [16] between events. It is often easier to develop distributed programs when the partially ordered set $(\mathtt{H}, \triangleright)$ is guaranteed to satisfy certain message ordering properties. For example, many distributed algorithms work correctly only in the presence of FIFO channels. This guarantee on ordering of messages is either provided explicitly – by communication primitives such as causal ordering [4] and logically synchronous ordering [6, 9]; or is built into the algorithm itself – as with global snapshot and recovery algorithms. In this paper, we propose a general framework in which existing and new message orderings can be specified and studied.

In this paper, a message ordering specification is characterized as the set of acceptable runs, that is, a subset of $\mathbb{X}$, where $\mathbb{X}$ is the set of all runs. For example, a system satisfying causal ordering can be viewed as the set of runs, say $\mathbb{X}_{co}$, such that for all runs in $\mathbb{X}_{co}$, and for all pairs of messages, $(s_1 \triangleright s_2) \Rightarrow \neg(r_2 \triangleright r_1)$, where s_j is the send of a message and r_j is its corresponding receive. In this broad setting, where each message ordering is a subset of $\mathbb{X}$, we first determine whether a given specification can be implemented. We show that a message ordering specification can be implemented if and only if it includes all logically synchronous runs. Further, if it can be implemented then we determine the type of protocol necessary and sufficient to implement it, where the protocols are classified into three types, (1) `general`: those that can tag information and have control messages, (2) `tagged`: those that can tag information, and (3) `tagless`: those that do nothing. For example, we show that a message specification can be implemented by tagging user messages with some additional information if and only if it includes all causally ordered runs. This result implies that there is no protocol for imposing logically synchronous ordering which does not use control messages.

Formally, we define three subsets of $\mathbb{X}$, namely, $\mathbb{X}_{async}$, $\mathbb{X}_{co}$ and $\mathbb{X}_{sync}$. We show that given a specification $\mathbb{Y} \subseteq \mathbb{X}$, it is implementable (there exists a protocol with control messages) if and only if $\mathbb{X}_{sync} \subseteq \mathbb{Y}$. Similarly, there is a protocol without control messages if and only if $\mathbb{X}_{co} \subseteq \mathbb{Y}$. The "do nothing" protocol is sufficient to implement if and only if $\mathbb{X}_{async} \subseteq \mathbb{Y}$. Thus, given a specification, that is the set of acceptable runs, the type of protocol necessary and sufficient can be easily checked by testing the containment of the three limit sets $\mathbb{X}_{async}$, $\mathbb{X}_{co}$ and $\mathbb{X}_{sync}$.

Since $\mathbb{X}$ is an infinite set, we also need a finite representation for its subsets that specify message ordering. We present a method called *forbidden predicates* that can be used to describe a large class of message ordering specifications. All existing message ordering guarantees such as FIFO, flush channels, causal ordering, and logically synchronous ordering as well as others can be concisely specified using forbidden predicates. For example, the specification for causal ordering $\mathbb{X}_{co}$ can be stated as: for all runs in $\mathbb{X}_{co}$, and for all pairs of messages, $\neg\,(\,(s_1 \triangleright s_2) \;\wedge\; (r_2 \triangleright r_1)\,)$. The forbidden predicate for $\mathbb{X}_{co}$ is $\exists\,(s_1, r_1), (s_2, r_2) \;:\; (s_1 \triangleright s_2) \;\wedge\; (r_2 \triangleright r_1)$. In general, a forbidden predicate can be stated as a conjunction of causality relationships between the events (send and receive).

Given a message ordering specification using forbidden predicates, we present an algorithm that determines the type of protocol necessary to implement that specification. The algorithm converts the forbidden predicate into a predicate graph. It is shown that the specification can be implemented if and only if there is a cycle in this graph. Further, to determine the nature of the protocol required for the specification, it is sufficient to examine vertices of the graph. We define the notion of β vertices. If the cycle has two or more β vertices with respect to that cycle, then control message are necessary. If the cycle has one β vertex, then tagging user messages is sufficient. If the cycle has no β vertex, then no action from the protocol is required. Thus, given any message ordering specification using forbidden predicates, the nature of the protocol necessary for implementing it can easily be determined.

We note here that specification using forbidden predicates also permits automatic generation of efficient protocols for a class of message ordering specifications. This is the focus of the companion paper [19].

2 Related Work

A fair amount of research has been done for efficient algorithms to implement different message orderings. Birman and Joseph [4], Raynal, Schiper and Toueg [20], Schiper, Eggli and Sandoz [21], have presented algorithms for the causal ordering of messages. These algorithms tag knowledge of processes about messages sent in the system with the message. For example, process P_i in the algorithm by Raynal, Schiper and Toueg [20] tags a message with the matrix m where $m[j,k]$ is the knowledge of process P_i about the messages sent from P_j to P_k. It is natural to ask whether the message ordering can be further restricted by sending higher-levels of knowledge (for example, by using three dimensional matrices: what P_i knows that P_j knows about messages sent from P_k to P_l). It is an easy consequence of the results of this paper that no additional tagging of information can restrict the message ordering further.

Variants of FIFO ordering have been studied under F-channels [1]. The implementation of F-channels, provides us with some basic synchronization primitives for sending messages: *two-way-*

flush send, *forward-flush* send, *backward-flush* send, and *ordinary* send. Similar flush primitives can be defined for causal ordering [12]. These message orderings can be specified using forbidden predicates. By constructing predicate graphs of these predicates it can be shown that these orderings can be implemented without using any control messages.

Logically synchronous ordering of messages has been studied in [9, 18, 24]. It has also been studied extensively as implementation of the guard statement of Communicating Sequential Process (CSP) [6, 3, 23] and as the binary interaction problem [2, 8]. Thus, these message orderings have either been studied as synchronization primitives, or are embedded in some protocol. Our results show that all these protocols must use additional control messages for implementation.

Many asynchronous consistent-cut protocols [25] such as global snapshot algorithms [7, 11, 17], check-pointing and rollback recovery [10, 15, 14], and deadlock detection [5] require special messages to find consistent-cuts in a computation. These protocols require some form of inhibition of the special messages in order to guarantee correctness. The inhibition of the messages can also be viewed as a restriction on the set $\mathbb{X}$.

In [22], Schmuck presents the necessary and sufficient conditions under which causal/FIFO broadcast instead of atomic broadcast can be used to guarantee correctness of a specification. We on the other hand are concerned with the problem of implementing atomic, causal and FIFO broadcast instead of using these broadcast primitives.

3 Model and Definitions

In this paper, we are interested in characterization of message ordering specifications and protocols that operate by delaying events. Usually an event from a user's view is broken up into two underlying system events: the request of the event and the execution of the event. For example, to implement causal ordering, the receive events are delayed and are usually called receive and delivery of the message. Therefore, we differentiate between the two views by defining a system's view of a run and a user's view of the same run. In this paper, each *user* event h, like sending of a message or receiving of a message, is characterized by two *system* events, that is h^* and h, where h^* represents the request of the event and h its execution.

3.1 System Model

The basic entity in our model is a message. A message x consists of four events. They are *invocation* event $x.s^*$, *send* event $x.s$, *receive* event $x.r^*$ and *delivery* event $x.r$. The set of messages from process i to process j is M_{ij}. The set of all message is denoted by $M = \cup_{i,j} M_{ij}$.

A *run* is a decomposed partially ordered set $\mathcal{H} = (H_1, H_2, \ldots, H_n, \rightarrow)$ [13], where the set H_i is a sequence of events, such that

$$H_i \subseteq \{\, x.s^*, x.s \;:\; x \in M_{ij} \text{ for all } j \,\} \cup \{\, x.r^*, x.r \;:\; x \in M_{ki} \text{ for all } k \,\}.$$

and the order relation $\rightarrow$ is defined as, $h \rightarrow h'$ iff,

1. $\exists\, k \;:\; h, h' \in H_k$ and h is before h' in the sequence H_k, or
2. there exists a message $x \in M$ such that $h = x.s$ and $h' = x.r^*$, or

(a) Run $\mathcal{H}$ (b) $CausalPast_2(\mathcal{H})$

Figure 1: Illustration of Causal past with respect to a process

3. there exists $g \in H = \cup_j H_j$ such that $h \rightarrow g$ and $g \rightarrow h'$.

A *run* satisfies the usual notions of a computation in a distributed system, that is, it is a partial order, it has no spurious messages and it includes the execution of an event only if it has been requested by the user. Formally, these condition can be stated as,

1. the relation $\rightarrow$ on $H = \cup_j H_j$ is a partial order,
2. $x.r^* \in H_i \Rightarrow x.s \in H$, that is, message is received only if it has been sent, and
3. $x.s \in H \Rightarrow x.s^* \rightarrow x.s$ and $x.r \in H \Rightarrow x.r^* \rightarrow x.r$, that is, the events $x.s$ and $x.r$ are always preceded by the events $x.s^*$ and $x.r^*$, respectively.

Given a run $\mathcal{H}$, a prefix of the run $\mathcal{G}$ is also a run, such that G_i is a prefix of H_i for all i. A prefix of interest is the causal past of a run $\mathcal{H}$ with respect to a given process i (denoted by $CausalPast_i(\mathcal{H})$). Figure 1 shows the causal past of the run $\mathcal{H}$ with respect to process 2. Intuitively, the causal past with respect to a process i consists of all the events that are followed by some event in process i. Let $\mathcal{G} = CausalPast_i(\mathcal{H})$, then

1. $G_i = H_i$, and
2. $\forall\, j \neq i \;:\; g \in G_j \;\equiv\; (\exists\, h \in H_i \;:\; g \rightarrow h)$.

A distributed system is a tuple $(M, \mathcal{X})$, where $\mathcal{X}$ is a set of all possible runs with respect to the message set M. Given a run $\mathcal{H} \in \mathcal{X}$ we define the following sets;

1. the messages that have not been requested by process i,

$$I_i(\mathcal{H}) = \{\, x.s^* \;:\; (x.s^* \notin H_i) \;\wedge\; (x \in M_{ik}) \,\}\,.$$

2. the messages that have been requested but not yet sent by process i,

$$S_i(\mathcal{H}) = \{\, x.s \;:\; (x.s^* \in H_i) \;\wedge\; (x.s \notin H_i) \,\}\,.$$

3. the messages that have been sent to process i but not yet received by process i,

$$R_i(\mathcal{H}) = \{\, x.r^* \;:\; (x.r^* \notin H_i) \;\wedge\; (\exists\, k \;:\; (x \in M_{ki}) \wedge (x.s \in H_k)) \,\}\,.$$

4. the messages that have been received but not yet delivered by process i,

$$D_i(\mathcal{H}) = \{\, x.r \;:\; (x.r^* \in H_i) \;\wedge\; (x.r \notin H_i) \,\}\,.$$

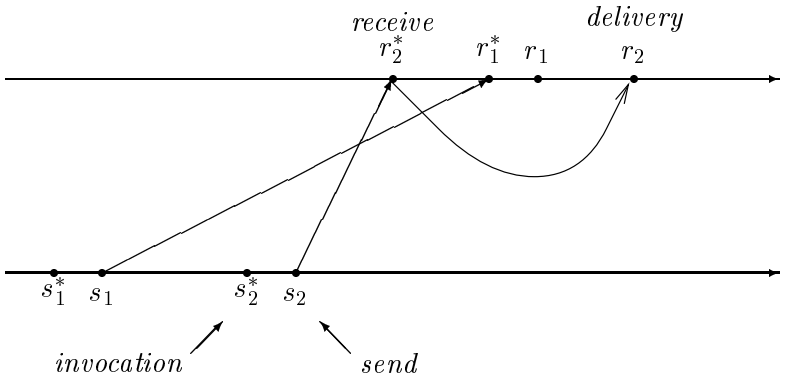


Figure 2: Inhibitory protocol to implement FIFO

3.2 Protocols

In this section we define inhibitory protocols. Informally, an inhibitory protocol specifies that an event may be delayed until the occurrence of prerequisite events. For example, in the implementation of FIFO, a message is delayed until all messages sent earlier have been delivered. In Figure 2, the protocol enables the event r_2 only after the event r_1 has been executed. Thus, a protocol

$$\mathcal{P} = \{ \, (\, P_1(\mathcal{H}), P_2(\mathcal{H}), \ldots, P_n(\mathcal{H}) \,) \; : \; \mathcal{H} \in (\, M, \mathcal{X} \,) \, \},$$

is a vector of enabled event sets for each run, where $P_i(\mathcal{H})$ is the set of enabled events in process i after the execution of the run $\mathcal{H}$.

Now we present some conditions to be satisfied by the vector of enabled event sets. The protocols do not have control over star-events. Clearly, a protocol cannot disable a user from requesting the execution of a message that has not been sent. Thus, we have

$$P_i(\mathcal{H}) \cap I_i(\mathcal{H}) = I_i(\mathcal{H}).$$

Similarly, a protocol cannot disable the receive of a message that has been already sent and is in transit. Thus

$$P_i(\mathcal{H}) \cap R_i(\mathcal{H}) = R_i(\mathcal{H}).$$

A protocol can disable or enable the send event and delivery event of a message if the invocation or the receive has been executed, respectively. Thus,

$$\begin{aligned} P_i(\mathcal{H}) \cap S_i(\mathcal{H}) &\subseteq S_i(\mathcal{H}), \text{ and} \\ P_i(\mathcal{H}) \cap D_i(\mathcal{H}) &\subseteq D_i(\mathcal{H}). \end{aligned}$$

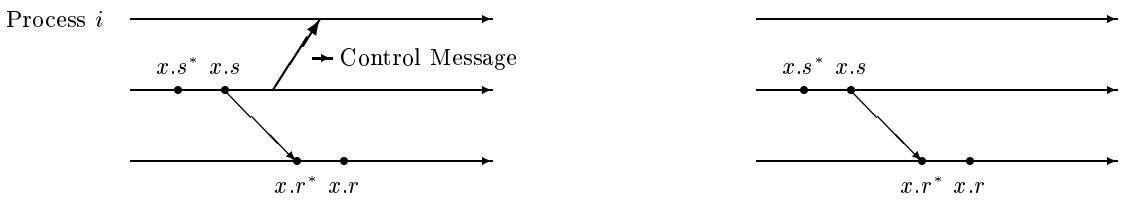
Therefore,

$$I_i(\mathcal{H}) \cup R_i(\mathcal{H}) \subseteq P_i(\mathcal{H}) \subseteq I_i(\mathcal{H}) \cup R_i(\mathcal{H}) \cup C_i(\mathcal{H}),$$

where $C_i(\mathcal{H}) = S_i(\mathcal{H}) \cup D_i(\mathcal{H})$.

Notation: The use of the notation C is to represent the events that are "controllable". We use C_i, with a subscript to identify with a particular process i. When the subscript is absent, say $C(\mathcal{H})$, then we are referring to the union of all $C_i's$, that is, $C(\mathcal{H}) = \bigcup_i C_i(\mathcal{H})$. We follow this convention for the sets $H, P, I, S, D, R,$ and C, that is, $H = \bigcup_i H$, $P(\mathcal{H}) = \bigcup_i P_i(\mathcal{H})$, and so on.

We define the set of *runs*, $\mathcal{X}_P$, possible under the protocol $\mathcal{P}$ in an inductive fashion, based on the events enabled. The base case is a null run $\mathcal{H}_\emptyset$ (where $H = \emptyset$) belongs to the set $\mathcal{X}_P$, since this



(a) With Control Message (b) Without Control Message

Figure 3: Knowledge of concurrent events

run is possible even if the protocol does not enable any event. Let the run $\mathcal{H}$ be possible under the protocol, then if some of the processes simultaneously execute an event enabled in their process, then the resulting run also is possible under the protocol. This can be formally stated as,

1. $\mathcal{H}_\emptyset \in \mathcal{X}_P$.
2. Let $\mathcal{H} \in \mathcal{X}_P$, then $\mathcal{G} \in \mathcal{X}_P$, where
   (a) H_i is a prefix of G_i and they differ by at most one event, and
   (b) $G_i \subseteq H_i \cup P_i(\mathcal{H})$.

In the next lemma, we show that $\mathcal{G}$ is a *run*, that is, it satisfies the three conditions of a run.

Lemma 1 *Let $\mathcal{H}$ be a run, and $(P_1(\mathcal{H}), P_2(\mathcal{H}), \ldots, P_n(\mathcal{H}))$ a vector of enabled event sets, such that for all i*

$$I_i(\mathcal{H}) \cup R_i(\mathcal{H}) \subseteq P_i(\mathcal{H}) \subseteq I_i(\mathcal{H}) \cup R_i(\mathcal{H}) \cup C_i(\mathcal{H}).$$

Then $\mathcal{G}$ is a run*, where*

1. *H_i is a prefix of G_i and they differ by at most one event, and*
2. *$G_i \subseteq H_i \cup P_i(\mathcal{H})$.*

Proof Outline: Clearly, $\mathcal{G}$ satisfies the three conditions of a run, that is,

1. $\mathcal{G}$ is a partial order, since $\mathcal{H}$ is a partial order, and if $g \in G \cap P(\mathcal{H})$ then $\nexists g' \in G : g \rightarrow g'$,
2. $x.r^* \in G \Rightarrow x.s \in G$, since a receive event is added only if H contains the send event, and
3. $x.s \in G \Rightarrow x.s^* \in G$ and $x.r \in G \Rightarrow x.r^* \in G$, by definition of $\mathcal{S}$ and $\mathcal{D}$. □

It is desirable that any protocol allows the system to progress, that is, to satisfy the liveness property. For example, if a user requests a message then it is eventually sent and delivered. In other words, we want the protocol eventually to execute a run $\mathcal{H}$ such that $S(\mathcal{H}) \cup R(\mathcal{H}) \cup D(\mathcal{H}) = \emptyset$, that is, all the messages requested by the user have been sent and delivered and there are no pending events. Thus, the protocol at each stage enables at least one of the pending events if the pending set is not empty. This condition can be formally stated as,

Liveness: $R(\mathcal{H}) \cup C(\mathcal{H}) \neq \emptyset \Rightarrow P(\mathcal{H}) \cap (R(\mathcal{H}) \cup C(\mathcal{H})) \neq \emptyset$.

Consider the case when the above condition is not satisfied. If the user does not request any more messages, the system cannot make any progress and the pending events are never executed.

Next, we classify the set of protocols based on the information exchange possible between the processes. First, consider a protocol that allows processes to exchange information using user messages only, then the processes are limited to the causal past. Intuitively, the class of such protocols can be implemented by tagging information to user messages. Second, if a protocol does not allow any information exchange using either user or control messages, then a process may enable or disable events based only on the local history. Such protocols belong to the class of tagless protocols. Finally, a protocol allows processes to exchange information using both control messages and user messages, then processes are capable of deciding based on events that appear concurrent, when events associated with the control messages are deleted. For example, in Figure 3, process i knows about the events $x.s^*$ and $x.s$ although it appears concurrent when the control message has been deleted. Formally, the types of protocols and the condition satisfied by each type are:

`general` : The class of `general` protocols characterize the environment where an action by a process can be known instantaneously to all processes in the group. Thus, each process enables and disables events based on the knowledge of both causal and concurrent events. This can be formally stated as

$$\mathcal{H} = \mathcal{G} \;\; \Rightarrow \;\; P_i(\mathcal{H}) = P_i(\mathcal{G}).$$

The condition states that a process takes the same action in any two executions if the partial orders are the same. Later, we will show that if there exists a `general` protocol for a given specification then there exists an inhibitory protocol using control and user messages that implements the specification.

`tagged` : The class of `tagged` protocols characterize the environment where an action by a process can be known only in its causal future. Thus, each process enables and disables events, based on the knowledge of causal events. Therefore, if in two different executions, the causal past with respect to a process i, that is $CausalPast_i(\cdot)$, is the same then the action taken by the process i in the two cases is the same. This can be formally stated as

$$CausalPast_i(\mathcal{H}) = CausalPast_i(\mathcal{G}) \;\; \Rightarrow \;\; P_i(\mathcal{H}) = P_i(\mathcal{G}).$$

`tagless` : These protocols cannot tag information to the user messages and cannot use control messages. The condition satisfied by a `tagless` protocol is

$$H_i = G_i \;\; \Rightarrow \;\; P_i(\mathcal{H}) = P_i(\mathcal{G}).$$

The condition states that if the local history is the same then the action taken by the process is the same.

The above conditions are used to capture the three classes of protocols we are interested in studying in this paper. In particular, first case the class of `general` protocols model the behavior of protocols with control messages in the absence of synchronized clocks or a global clock. Therefore, such a protocol cannot differentiate between two runs that have the same partial order relation but may differ in physical global time.

3.2.1 Limitations of Protocols

In this section we explore the limitations of each type of protocol. We answer questions of the form, "if protocol $\mathcal{P}$ is a `tagless` protocol, then does $\mathcal{H}$ necessarily belong to the set $\mathcal{X}_P$?". These questions will provide us with insight into the type of protocol necessary to implement the desired specification. For example, if the run $\mathcal{H}$ is undesirable and $\mathcal{H} \in \mathcal{X}_P$, then $\mathcal{P}$ cannot guarantee safety. We define three subsets of $\mathcal{X}$, that is, $\mathcal{X}_{gn}$, $\mathcal{X}_{td}$ and $\mathcal{X}_{tl}$, such that, they are subsets of $\mathcal{X}_P$, when $\mathcal{P}$ is a `general`, `tagged` or `tagless` protocol, respectively. Formally, these sets are:
A run $\mathcal{H}$ belongs to the set $\mathcal{X}_{tl}$ if and only if

1. for any message x in $\mathcal{H}$, $x.s^*$ immediately precedes $x.s$ and $x.r^*$ immediately precedes $x.r$, and
2. all messages requested have been delivered, that is, $x.s^* \in H \;\Rightarrow\; x.r \in H$.

A run $\mathcal{H}$ belongs to the set $\mathcal{X}_{td}$ if and only if

1. $\mathcal{H} \in X_{tl}$, and
2. the messages are causally ordered, that is, $x.s \rightarrow y.s \Rightarrow \neg\,(y.r^* \rightarrow x.r^*)$.

A run $\mathcal{H}$ belongs to the set $\mathcal{X}_{gn}$ if and only if

1. $\mathcal{H} \in X_{td}$, and
2. the time diagram can be redrawn such that all message arrows are vertical, that is, there exists a numbering scheme N, that assigns a unique number to each event such that,

$$h \rightarrow g \Rightarrow N(h) < N(g) \text{ and } N(x.r) = N(x.r^*) + 1 = N(x.s) + 2 = N(x.s^*) + 3.$$

The main result of this section show the relation of the above sets to a protocol of each type.
Lemma 2 *Let $\mathcal{P}$ be a protocol satisfying the liveness property and $\mathcal{X}_P$ is the set of all runs possible under the protocol.*

1. *If $\mathcal{P}$ is a* `general` *protocol, then $\mathcal{X}_{gn} \subseteq \mathcal{X}_P$.*
2. *If $\mathcal{P}$ is a* `tagged` *protocol, then $\mathcal{X}_{td} \subseteq \mathcal{X}_P$.*
3. *If $\mathcal{P}$ is a* `tagless` *protocol, then $\mathcal{X}_{tl} \subseteq \mathcal{X}_P$.*

Therefore, if $\mathcal{H} \in \mathcal{X}_{gn}$ and it is an undesirable run, then there does not exist a protocol to implement the specification. The lemma is proved in the appendix.

3.3 Specifications

A specification is the set of behavior as desired by the user. For example, a particular run $\mathcal{H}$ may or may not be desirable. In this section, we expand on the concepts of user's view and formally define a specification.

A user is interested in the send and delivery of a message and the order relation among them, rather than the invocation and receive events. For example, causal ordering is stated in terms of the relation between the send and delivery events. Thus, the causality relation between two events

(a) System's View (b) User's View

Figure 4: Illustration for the difference in causality relation

from the user's view can be different the relation from the system's view. Figure 4 illustrates the difference in a system that implements FIFO ordering among the messages. In the system's view the event s_2 happened causally before the event r_1, whereas from user's view s_2 did not happen before the event r_1. Thus, we define a relation from system's view to the user's view of a run which is a projection of the events with the invocation and receive events removed.

1. *UsersView*($\mathcal{H}$), a projection of the *run* $\mathcal{H}$, is a partial order, denoted as ($\mathtt{H}$, $\triangleright$), where

$$\mathtt{H} = \{\, h \;:\; h \in H \;\wedge\; (h \text{ is a } \textit{send} \text{ or a } \textit{delivery} \text{ event}) \,\},$$

   and $\triangleright$ is the order relation on $\mathtt{H}$. For the *projected run* ($\mathtt{H}$, $\triangleright$), $h \triangleright h'$ if and only if
   (a) $\exists\, k$ such that $h, h' \in H_k$ and $h \rightarrow h'$, or
   (b) $\exists\, x \in M$, such that $x.s = h$ and $x.r = h'$, or
   (c) $\exists\, g \in H$ such that $h \triangleright g$ and $g \triangleright h'$.

2. A *complete run* is a *projected run* such that $x.s \in \mathtt{H} \iff x.r \in \mathtt{H}$.

A specification $\mathbb{Y}$ is a set of *complete runs*, where

$$\mathbb{Y} \subseteq \mathbb{X} = \{\, (\mathtt{H}, \triangleright) \;:\; x.s \in \mathtt{H} \iff x.r \in \mathtt{H} \text{ and } \triangleright \text{ is a partial order} \,\}.$$

The reason for considering only complete runs is to satisfy the notion that all messages sent are eventually delivered in a reliable system. In other words, if a send of a message does not invalidate the run, then there should be a possible completion (the message is delivered), that is valid.

A protocol $\mathcal{P}$ is characterized by the set of *complete runs* $\mathbb{X}_P$, where

$$\mathbb{X}_P = \{\, (\mathtt{H}, \triangleright) = \mathit{UsersView}(\mathcal{H}) \;:\; (\mathcal{H} \in \mathcal{X}_P) \;\wedge\; (x.s \in \mathtt{H} \Leftrightarrow x.r \in \mathtt{H}) \,\}.$$

We say, a protocol $\mathcal{P}$ guarantees safety, if the projection of a run $\mathcal{H} \in \mathcal{X}_P$ is valid in the user's view. In other words, if $\mathcal{H} \in \mathcal{X}_P$, where $x.s \in H \Leftrightarrow x.r \in H$, then $(\mathtt{H}, \triangleright) = \mathit{UsersView}(\mathcal{H}) \in \mathbb{Y}$.

If a protocol $\mathcal{P}$ implements a specification $\mathbb{Y}$ then

$$\forall\, \mathcal{H} \in (M, \mathcal{X}), \quad R(\mathcal{H}) \cup C(\mathcal{H}) \neq \emptyset \;\Rightarrow\; P(\mathcal{H}) \cap (R(\mathcal{H}) \cup C(\mathcal{H})) \neq \emptyset, \qquad \textbf{(Liveness)}$$

$$\mathbb{X}_P \subseteq \mathbb{Y}. \qquad \textbf{(Safety)}$$

3.4 Limit Sets

In this section, we consider the problem of finding the type of protocol sufficient and necessary to implement a given specification.

In Section 3.2.1, we investigated the question whether a run necessarily belongs to the $\mathcal{X}_P$, given a protocol $\mathcal{P}$. In this section, we pose the same question but in a different setting; that is, given a projected run $(\mathtt{H}, \triangleright)$ does it necessarily belong to the set $\mathbb{X}_P$. Given a specification $\mathbb{Y}$, this gives us lower bounds on the specification $\mathbb{Y}$ that is *necessary* for the existence of a `general`, a `tagged` or a `tagless` protocol. For example, if a `general` protocol implements the specification $\mathbb{Y}$ then $\mathbb{X}_n \subseteq \mathbb{Y}$, where $\mathbb{X}_n$ is the lower bound for the class of `general` protocols. In this section, we present results in the other direction, that is, does there exists a limit $\mathbb{X}_s$ that is *sufficient* for the existence of a `general` protocol. For example, if $\mathbb{X}_s \subseteq \mathbb{Y}$ then there exists a `general` protocol that implements the specification $\mathbb{Y}$.

We define three subsets of $\mathbb{X}$ (or specifications) similar to ones in Section 3.2.1 that will be used to provide answer to the problem stated in this section. The three subsets of $\mathbb{X}$ are:

Asynchronous ordering (ASYNC) : This is the same as the ground set $\mathbb{X}$. Therefore, it includes all possible runs. There exists a `tagless` algorithm (i.e., enable all pending events) that will guarantee safety and liveness for this specification. Formally, we can state $\mathbb{X}_{async}$, the set of all partial orders as

$$\mathbb{X}_{async} = \{\ (\mathtt{H}, \triangleright)\ :\ (x.s \in \mathtt{H} \Leftrightarrow x.r \in \mathtt{H}) \text{ and } \triangleright \text{ is a partial order}\ \}.$$

Causal Ordering (CO) : Causal ordering can be stated as $s_1 \triangleright s_2 \Rightarrow \neg(r_2 \triangleright r_1)$. There exists a `tagged` algorithm where with each message a matrix of size $n \times n$ is tagged to the message [20, 21]. Formally, we can state $\mathbb{X}_{co}$, the set of partial orders satisfying causal ordering as

$$\mathbb{X}_{co} = \{\ (\mathtt{H}, \triangleright)\ :\ \neg(\ (x.s \triangleright y.s)\ \wedge\ (y.r \triangleright x.r)\ )\ \forall x, y \in M\ \}.$$

Logically Synchronous (SYNC) : A run is logically synchronous if its time diagram can be drawn such that all message arrows are vertical. Formally, we can state $\mathbb{X}_{sync}$, the set of logically synchronous partial orders as

$$\mathbb{X}_{sync} = \{\ (\mathtt{H}, \triangleright)\ :\ \neg(\ (x_1.s \triangleright x_2.r) \wedge (x_2.s \triangleright x_3.r) \cdots (x_k.s \triangleright x_1.r)\ )\ ,\ \forall k > 1,\ \ \forall x_j \in M\ \}.$$

Since the message arrows can be drawn vertically, the messages can be linearly ordered such that $(y.h \triangleright x.f) \Rightarrow (x < y)$ Thus, we can get an alternative definition of a logically synchronous run as,

$$\begin{array}{rcl} \exists \mathrm{T} : M \longrightarrow \mathbb{N} & : & \forall x, y \in M \\ \bigvee_{h,f \in \{s,r\}} x.h \triangleright y.f & \Rightarrow & \mathrm{T}(x) < \mathrm{T}(y). \end{array} \qquad \text{(SYNC)}$$

In [18] it was shown that the two definitions are equivalent. This property can be implemented using control messages, for details refer to [3, 18]. Therefore, there exists a `general` protocol that will guarantee the absence of any partial order in the set $\mathbb{X} - \mathbb{X}_{sync}$.

Figure 5: Construction of $\mathcal{H}$ from (H, $\triangleright$)

It is easy to see that

$$\mathbb{X}_{sync} \subseteq \mathbb{X}_{co} \subseteq \mathbb{X}_{async}.$$

The sets $\mathbb{X}_{async}$, $\mathbb{X}_{co}$, and $\mathbb{X}_{sync}$ exhibit an important property, i.e., they are the limiting specifications, in terms of whether there exists a protocol that can guarantee safety and liveness, for each of the three classes of protocols. For example, there exists a `tagged` protocol (i.e., no control messages) that guarantees safety and liveness for the specification $\mathbb{X}_{co}$. Further, given a specification $\mathbb{Y}$, there exists a `tagged` protocol that guarantees safety and liveness, if and only if $\mathbb{X}_{co} \subseteq \mathbb{Y}$. Thus, given a specification, i.e., the set of acceptable runs, the type of protocol necessary and sufficient can be easily checked by testing the containment of the three limit sets, $\mathbb{X}_{async}$, $\mathbb{X}_{co}$, and $\mathbb{X}_{sync}$.

Theorem 1 *Let $\mathbb{Y}$ be a specification. Then*

1. *A* `general` *protocol can guarantee safety and liveness iff $\mathbb{X}_{sync} \subseteq \mathbb{Y}$.*
2. *A* `tagged` *protocol can guarantee safety and liveness iff $\mathbb{X}_{co} \subseteq \mathbb{Y}$.*
3. *A* `tagless` *protocol can guarantee safety and liveness iff $\mathbb{X}_{async} \subseteq \mathbb{Y}$.*

Proof: It is easy to show the "if part" in each of the cases. We use the fact that if a protocol $\mathcal{P}$ implements the specification $\mathbb{Y}$, then $\mathbb{X}_P \subseteq \mathbb{Y}$.

1. There exists a `general` protocol $\mathcal{P}$ such that $\mathbb{X}_P = \mathbb{X}_{sync}$ [3, 18].
2. There exists a `tagged` protocol $\mathcal{P}$ such that $\mathbb{X}_P = \mathbb{X}_{co}$ [20, 21].
3. There exists a `tagless` protocol $\mathcal{P}$ such that $\mathbb{X}_P = \mathbb{X}_{async}$ (enable all events).

We now proceed to show the "only if part".

1. Let $\mathcal{P}$ be a `general` protocol. From lemma 2, we have $\mathcal{X}_{gn} \subseteq \mathcal{X}_P$. We have to show that if (H, $\triangleright$) $\in \mathbb{X}_{sync}$ then $\exists\, \mathcal{H} \in \mathcal{X}_{gn}$ such that (H, $\triangleright$) $= \mathit{UsersView}(\mathcal{H})$.

   Given (H, $\triangleright$) $\in \mathbb{X}_{sync}$ we construct $\mathcal{H}$, such that (H, $\triangleright$) $= \mathit{UsersView}(\mathcal{H})$ and $\mathcal{H} \in \mathcal{X}_{sync}$, as shown in Figure 5. For each event $x.s$ add $x.s^*$ such that $x.s^*$ immediately precedes $x.s$. Similarly, for each event $x.r$ add $x.r^*$ such that $x.r^*$ immediately precedes $x.r$. We claim that $\mathcal{H} \in \mathcal{X}_{gn}$, that is, $\mathcal{H}$ satisfies the conditions satisfied by elements of $\mathcal{X}_{gn}$.
   (a) $x.s^*$ immediately precedes $x.s$, and $x.r^*$ immediately precedes $x.r$, by construction of $\mathcal{H}$.
   (b) $\mathcal{H}$ is a complete run, that is, $x.s^* \in H \Rightarrow x.r \in H$, since (H, $\triangleright$) is a complete run.
   (c) Since (H, $\triangleright$) $\in \mathbb{X}_{sync}$, there exists a function T satisfying SYNC. Using the function T we can derive the numbering scheme N, where $h \rightarrow g \Rightarrow N(h) < N(g)$ and $N(x.r) = N(x.r^*) + 1 = N(x.s) + 2 = N(x.s^*) + 3$

2. Let $\mathcal{P}$ be a `tagged` protocol. From lemma 2, we have $\mathcal{X}_{tg} \subseteq \mathcal{X}_P$. We have to show that if $(\mathtt{H}, \triangleright) \in \mathbb{X}_{co}$ then $\exists\, \mathcal{H} \in \mathcal{X}_{tg}$ such that $(\mathtt{H}, \triangleright) = \mathit{UsersView}(\mathcal{H})$.

   The proof is similar to the previous case. We construct $\mathcal{H}$ as above and show that $\mathcal{H} \in \mathcal{X}_{td}$.

3. Let $\mathcal{P}$ be a `tagless` protocol. From lemma 2, we have $\mathcal{X}_{tl} \subseteq \mathcal{X}_P$. We have to show that if $(\mathtt{H}, \triangleright) \in \mathbb{X}_{async}$ then $\exists\, \mathcal{H} \in \mathcal{X}_{tl}$ such that $(\mathtt{H}, \triangleright) = \mathit{UsersView}(\mathcal{H})$.

   The proof is similar to the previous case. We construct $\mathcal{H}$ as above and show that $\mathcal{H} \in \mathcal{X}_{tl}$.

□

Corollary 1 *A specification* $\mathbb{Y}$ *is implementable, that is, there exists a* `tagless`, `tagged`, *or* `general` *protocol, if and only if* $\mathbb{X}_{sync} \subseteq \mathbb{Y}$.

4 Forbidden Predicates

In previous sections, specifications were stated as a subset of $\mathbb{X}$. Since $\mathbb{X}$ is an infinite set, we need a finite representation for its subsets that specify message ordering. We present a method called *forbidden predicates* that can be used to describe a large class of message ordering specifications. All existing message ordering guarantees such as FIFO, flush channels, causal ordering, and logically synchronous ordering as well as others can be concisely specified using forbidden predicates.

In this section we describe *forbidden predicates* and present an algorithm to address the main concerns of this paper: What are the necessary and sufficient conditions for the existence of a protocol of each type?

Definition 4.1:

1. A *forbidden predicate* B is defined as

$$B \equiv \exists\, x_1, x_2, \ldots, x_m \in M \;:\; B(x_1, x_2, \cdots, x_m)$$

   where
$$B(x_1, x_2, \cdots, x_m) = \bigwedge_{(j,k) \in J \times K} (x_j.p \triangleright x_k.q),$$

   and p and q represent s or r. And J, K are subset of $\{1, 2, \ldots, m\}$.

2. Given a forbidden predicate B, the corresponding *specification set* $\mathbb{X}_B \subseteq \mathbb{X}$ is defined as

$$\mathbb{X}_B = \{\, (\mathtt{H}, \triangleright) \;:\; \neg B(x_1, \cdots, x_m),\;\; \forall\, x_0, x_1, \cdots, x_m \in M \,\}.$$

■

Notation: Let $B \equiv \exists\, x, y \in M \;:\; (x.s \triangleright y.s)$. We write the predicate B as $(x.s \triangleright y.s)$ dropping the quantifier $\exists$ for ease of use. $B(a, b)$ implies the evaluation of $(x.s \triangleright y.s)$ for the instances a and b in M. Therefore, $B(a, b)$ is true if and only if $a.s \triangleright b.s$. In case of ambiguity we express the predicate as $B \equiv \exists\, x, y \in M \;:\; B(x, y)$.

Given two forbidden predicates B and B' for the sets $\mathbb{X}_B$ and $\mathbb{X}_{B'}$, respectively, $B' \Rightarrow B$ iff $\mathbb{X}_B \subseteq \mathbb{X}_{B'}$. If a protocol for B guarantees that all the allowable partial orders belong to the set $\mathbb{X}_B$, then the same protocol guarantees that all the allowable partial orders belong to the set $\mathbb{X}_{B'}$.

Consider, the example of causal ordering. The predicate can be stated as $B \equiv (x.s \triangleright y.s) \;\wedge\; (y.r \triangleright x.r)$. For each element $(\mathtt{H}, \triangleright)$ of $\mathbb{X}_{co}$ (the corresponding specification set),

$$\forall\, x, y \in M \;::\; \neg\, ((x.s \triangleright y.s) \;\wedge\; (y.r \triangleright x.r)).$$

Further, we can define three attributes for each message receiving process, and sending process, and color. We can use these attributes to define a range for the variables of the predicate. For example, FIFO can be stated as

$$\forall\, x, y \in M : \text{process}\,(x.s) = \text{process}\,(y.s) \wedge \text{process}\,(x.r) = \text{process}\,(y.r) : \neg\left((x.s \triangleright y.s) \wedge (y.r \triangleright x.r)\right),$$

or we may be interested in runs where messages should not overtake the red marker message, that is

$$\forall\, x, y \in M \;\; : \; \text{color}\,(y) = \mathit{red} \; : \;\; \neg\left((x.s \triangleright y.s) \wedge (y.r \triangleright x.r)\right).$$

In this paper we will be interested in predicates where the variables range over all messages.

4.1 Forbidden Predicates and Limit Sets

In this section we characterize limit sets using forbidden predicates. For example, $\mathbb{X}_{co}$ corresponds to the forbidden predicate $B \equiv (x.s \triangleright y.s) \wedge (y.r \triangleright x.r)$.

Lemma 3

1. *The specification set for each of the following predicates contain* $\mathbb{X}_{sync}$.
   - *a)* $B \equiv ((x_1.s \triangleright x_2.r) \wedge (x_2.s \triangleright x_3.r) \cdots (x_k.s \triangleright x_1.r))$ *for any* $k = 2, 3 \ldots$
2. *The specification set for the following predicates is* $\mathbb{X}_{co}$.
   - *a)* $B_1 \equiv (x.s \triangleright y.r) \wedge (y.r \triangleright x.r)$.
   - *b)* $B_2 \equiv (x.s \triangleright y.s) \wedge (y.r \triangleright x.r)$.
   - *c)* $B_3 \equiv (x.s \triangleright y.s) \wedge (y.s \triangleright x.r)$.
3. *The specification set for the following predicates is* $\mathbb{X}_{async}$.
   - *a)* $B \equiv (x.s \triangleright y.s) \wedge (y.s \triangleright x.s)$.
   - *b)* $B \equiv (x.s \triangleright y.s) \wedge (y.r \triangleright x.s)$.
   - *c)* $B \equiv (x.s \triangleright y.r) \wedge (y.r \triangleright x.s)$.
   - *d)* $B \equiv (x.r \triangleright y.s) \wedge (y.r \triangleright x.s)$.
   - *e)* $B \equiv (x.r \triangleright y.r) \wedge (y.r \triangleright x.s)$.
   - *f)* $B \equiv (x.r \triangleright y.r) \wedge (y.r \triangleright x.r)$.

Proof: In the first part, the intersection of all specification sets is $\mathbb{X}_{sync}$, for details refer to [18]. For the third part, each of the predicates implies the existence of an event $h \in H$ such that $h \triangleright h$. No run in $\mathbb{X}_{async}$ satisfies such a predicate. Therefore, the specification set for the predicates is $\mathbb{X}_{async}$.

In the second part, B_2 corresponds to $\mathbb{X}_{co}$ by definition. We will show $B_1 \Leftrightarrow B_2$; the proof of $B_2 \Leftrightarrow B_3$ is similar. Let the corresponding specification sets be $\mathbb{X}_1$ and $\mathbb{X}_2$, respectively. We have to show that $\mathbb{X}_1 = \mathbb{X}_2$. It is easy to see that $B_2 \Rightarrow B_1$. Since $B_2 \equiv (x.s \triangleright y.s) \wedge (y.r \triangleright x.r)$ and $y.s \triangleright y.r$ is true, $B_2 = (x.s \triangleright y.s) \wedge (y.r \triangleright x.r) \wedge (y.s \triangleright y.r)$. Combining the first and third conjuncts, we get $B_2 \Rightarrow (x.s \triangleright y.r) \wedge (y.r \triangleright x.r) \equiv B_1$. Therefore, $\mathbb{X}_1 \subseteq \mathbb{X}_2$.

We now show that $\overline{\mathbb{X}}_1 \subseteq \overline{\mathbb{X}}_1$ where, $\overline{\mathbb{X}}_1 = \mathbb{X} - \mathbb{X}_1$. Using the definition of $\mathbb{X}_1$, we get the complement of $\mathbb{X}_1$ as,

$$\overline{\mathbb{X}}_1 = \{\, (\mathtt{H}, \triangleright) \;:\; \exists\, x, y \in M \text{ such that } B_1(x, y) \,\}.$$

Let $(\mathtt{H}, \triangleright) \in \overline{\mathbb{X}}_1$. We have to show $(\mathtt{H}, \triangleright) \in \overline{\mathbb{X}}_2$. In the run $(\mathtt{H}, \triangleright)$, we have at least two messages x and y such that, $(x.s \triangleright y.s) \wedge (y.s \triangleright x.r)$.

1. Let $x.s$ and $y.s$ be in different processes.

   Since $(x.s \triangleright y.s)$, and $x.s$ and $y.s$ are in different processes, there exists a message z such that $(x.s \triangleright z.s)$, $(z.s \triangleright z.r)$ and $(z.r \triangleright y.s)$. Since $(y.s \triangleright x.r)$ and $(z.r \triangleright y.s)$, $z.r \triangleright x.r$. Therefore, $x.s \triangleright z.s$ and $z.r \triangleright x.r$, thus $B_2(x, z)$ is true.

2. Assume $x.s$ and $y.s$ are in the same process ($x.r$ and $y.s$ are in different processes).

   Since $(y.s \triangleright x.r)$ and $x.r$ and $y.s$ are in different processes, $y.r \triangleright x.r$ or $\exists\, z \in M$, such that $(y.s \triangleright z.s)$, $(z.s \triangleright z.r)$ and $(z.r \triangleright x.r)$.

   (a) If $y.r \triangleright x.r$, then $(x.s \triangleright y.s)$ and $(y.r \triangleright x.r)$. Thus $B_2(x, y)$ is true.

   (b) If $\exists\, z:\ (y.s \triangleright z.s), (z.s \triangleright z.r)$ and $(z.r \triangleright x.r)$, then $(x.s \triangleright y.s) \wedge (y.s \triangleright z.s) \Rightarrow (x.s \triangleright z.s)$ and $(z.r \triangleright x.r)$, thus $B_2(x, z)$ is true.

Therefore, $\exists\, x, z \in M$ such that $B_2(x, z)$ is true. Thus, $(\mathtt{H}, \triangleright) \in \overline{\mathbb{X}}_2$. □

4.2 Specification Graph

In this section we classify the forbidden predicates to determine the type of algorithm necessary and sufficient to guarantee safety and liveness.

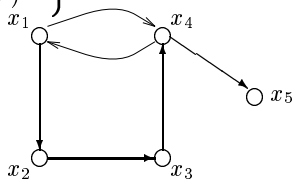
Definition 4.2: Let $B \equiv \exists\, x_1, \ldots, x_m \in M\ :\ B(x_1, \ldots, x_m)$ be a forbidden predicate. A *predicate graph* $G_B(V, E)$ is a multi-graph such that

$$\begin{aligned} V &= \{x_1, \ldots, x_m\} \\ E &= \{(x_j, x_k) \mid (x_j.p \triangleright x_k.q) \text{ is a conjunct of } B \text{ where } p, q \text{ is } s \text{ or } r\ \} \end{aligned}$$

■

Example 1: Let a predicate be

$$B \equiv \left\{ \begin{array}{l} (x_1.r \triangleright x_2.s) \wedge (x_2.s \triangleright x_3.s) \wedge (x_3.r \triangleright x_4.r) \wedge \\ (x_4.s \triangleright x_1.r) \wedge (x_4.s \triangleright x_5.r) \wedge (x_1.s \triangleright x_4.r) \end{array} \right\},$$



then $G_B(V, E)$ is

$$\begin{aligned} V &= \{x_0, x_1, x_2, x_3, x_4, x_5, x_6\} \\ E &= \{(x_1, x_2), (x_2, x_3), (x_3, x_4), (x_4, x_1), (x_4, x_5), (x_1, x_4)\}, \end{aligned}$$

■

Using the graph, we can determine whether the specification is implementable, and if it is, the type of protocol necessary and sufficient to guarantee safety and liveness.

Theorem 2 *A specification $\mathbb{X}_B$ (or forbidden predicate B) is implementable if and only if there exists a cycle in the predicate graph $G_B(V, E)$.*

Proof: We first prove the "only if" part. Let the predicate be $B \equiv \exists\, x_1, \cdots, x_m \in M\ :\ B(x_1, \cdots, x_m)$ such that the predicate graph $G_B(V, E)$ does not have a cycle and let the corresponding specification set be $\mathbb{X}_B$. Consider a run $(\mathtt{H}, \triangleright)$ such that the set of messages is $M = \{x_1, \ldots, x_m\}$. The run is constructed such that if $x_j.p \triangleright x_k.q$ is a conjunct of $B(x_1, \cdots, x_m)$

then $(x_j.p,\ x_k.q) \in (\mathtt{H},\ \triangleright)$. For each message $x \in M$, $(x.s,\ x.r) \in (\mathtt{H},\ \triangleright)$. Now take the transitive closure $(^+)$ to make it a run. Therefore,

$$(\mathtt{H},\ \triangleright) = \{\, \{\, (x_j.p,\ x_k.q) \ :\ (x_j.p \triangleright x_k.q) \text{ is a conjunct in } B(\ldots)\,\} \cup \{\, (x_l.s,\ x_l.r) \ :\ \forall l = 0, \ldots, m \,\}\,\}^+ .$$

It is easy to see that the predicate B is true in the run $(\mathtt{H},\ \triangleright)$, therefore, $(\mathtt{H},\ \triangleright) \notin \mathbb{X}_B$. We claim that $(\mathtt{H},\ \triangleright) \in \mathbb{X}_{sync}$, hence the theorem (only if) follows. Since the predicate graph does not have any cycles, it can be linearly ordered. Using the same ordering we define a function $\mathrm{T} : M \to N$ satisfying the SYNC condition. Therefore, $(\mathtt{H},\ \triangleright) \in \mathbb{X}_{sync}$ and $(\mathtt{H},\ \triangleright) \notin \mathbb{X}_B$. From corollary 1, we have that there exists a protocol only if $\mathbb{X}_{sync} \subseteq \mathbb{X}_B$.

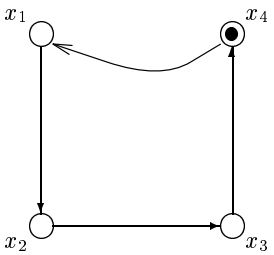
The "if" part follows from theorem 3 which will be proved in Section 5.3. □

4.2.1 β Vertex

We are interested in cycles in the specification graphs. Pick any cycle $G_c(V^c, E^c) \subseteq G(V, E)$ in the specification graph and let the corresponding forbidden predicate be B_c.

Example 2: Consider the forbidden predicate and the graph from example 1. A possible cycle and the corresponding predicate is shown below. It is easy to see that $B \Rightarrow B_c$, since B_c is the same as B with some conjuncts removed.

$$\begin{aligned} V^c &= \{\, x_1, x_2, x_3, x_4 \,\} \\ E^c &= \{\, (x_1, x_2), (x_2, x_3), (x_3, x_4), (x_4, x_1) \,\} \\ P_c &= \left\{ \begin{array}{l} (x_1.r \triangleright x_2.s) \wedge (x_2.s \triangleright x_3.s) \wedge \\ (x_3.r \triangleright x_4.r) \wedge (x_4.s \triangleright x_1.s) \end{array} \right\} \end{aligned}$$



■

The specification graphs can contain a number of cycles. We classify a cycle into different categories based on the number of β vertices (defined next) it contains.

Definition 4.3: Given, a cycle $G_c(V^c, E^c)$ in the graph $G(V, E)$, we say $x \in V^c$ is a β vertex with respect to the cycle $G_c(V^c, E^c)$ if the incoming edge is either $y.s \triangleright x.r$ or $y.r \triangleright x.r$ and the outgoing edge is either $x.s \triangleright z.s$ or $x.s \triangleright z.r$. The *order* of a cycle is equal to the number of β vertices it contains. ■

Example 3: Continuing with the previous example. With respect to the cycle $G_c(V^c, E^c)$, only x_4 is a β vertex, thus the order of the cycle is 1. Consider a non-β vertex, say x_3. Consider the conjuncts that result in the input and output edges of the vertex x_3. They are, $x_2.s \triangleright x_3.s$ and $x_3.r \triangleright x_4.r$. Since $x_3.s \triangleright x_3.r$, combining the three conjuncts we get, $x_2.s \triangleright x_4.r$. We can get a predicate B',

$$B' \equiv (x_1.r \triangleright x_2.s) \ \wedge\ (x_2.s \triangleright x_4.r) \ \wedge\ (x_4.s \triangleright x_1.r),$$

such that $B_c \Rightarrow B'$. Since $B \Rightarrow B_c$ and $B_c \Rightarrow B'$, $B \Rightarrow B'$. If we consider the predicate graph $G_{B'}(V', E')$, it is a cycle of order 1 and the β vertex is x_4, thus maintaining the order and the β vertex of the cycle. ■

Lemma 4 *Let B be a predicate and $G_B(V,E)$ be the corresponding predicate graph with a cycle of order l. Then there exists a predicate B' weaker than B whose predicate graph $G_{B'}(V',E')$ is a cycle of order k such that*

1. *$|V'| = 2$, or*
2. *all the vertices are β vertices.*

Proof: Let $G(V^c,E^c) \subseteq G(V,E)$ be a cycle in the predicate graph with the corresponding predicate as B_c. We know that $B \Rightarrow B_c$.

If the graph $G(V',E') = G(V^c,E^c)$ and predicate $B' = B_c$ satisfy the condition of the lemma, we are done. If not, pick a vertex, say y, that is not a β vertex. Then one of the following is true, with $x \neq z$,

1. $B' \equiv \cdots (x.p \triangleright y.s) \wedge (y.s \triangleright z.q) \wedge \cdots$,
2. $B' \equiv \cdots (x.p \triangleright y.s) \wedge (y.r \triangleright z.q) \wedge \cdots$,
3. $B' \equiv \cdots (x.p \triangleright y.r) \wedge (y.r \triangleright z.q) \wedge \cdots$.

Such a vertex exists since the graph (cycle) has more then two vertices and has at least one non-β vertex. In each case, $B' \Rightarrow B''$, where $B'' \equiv \cdots \wedge (x.p \triangleright z.q) \wedge \cdots$. Since $B \Rightarrow B'$ and $B' \Rightarrow B''$, $B \Rightarrow B''$. Let the graph predicate for B'' be $G(V'',E'')$. The graph $G(V'',E'')$ satisfies the condition $|V''| = |V'| - 1$, and the number of β vertices in $G(V'',E'')$ is k.

If the graph $G(V'',E'')$ and the corresponding predicate B'' satisfy the conditions of the lemma, we are done, otherwise repeat the above process. □

4.3 Impossibility and Lower-Bounds

In this section we prove the necessary and sufficient conditions for a specification to be implementable by a protocol of a given class. This can be summed up by the following table which is a consequence of the next two theorems proved later in this section:

Specification graph has a cycle	$\Longleftrightarrow$	specification is implementable
and, if there exists a cycle with		
– zero or more β vertices	$\Longleftrightarrow$	tagging and control messages are sufficient,
– zero or one β vertex	$\Longleftrightarrow$	tagging is sufficient, and
– zero β vertex	$\Longleftrightarrow$	trivial protocol is sufficient.

The next theorem proves the sufficient condition for a protocol to implement a given specification. Theorem 4 presents the necessary conditions to be satisfied by the specifications to be implementable by a protocol of a given class.

Theorem 3 (Sufficient Conditions) *Let $\mathbb{X}_B$ be a specification with B as the corresponding forbidden predicate. Let the predicate graph be $G_B(V,E)$ with a cycle $G_c(V^c,E^c) \subseteq G_B(V,E)$.*

1. *If there exists a cycle $G_c(V^c,E^c) \subseteq G_B(V,E)$ of order 0, then $\mathbb{X}_{async} \subseteq \mathbb{X}_B$.*
2. *If there exists a cycle $G_c(V^c,E^c) \subseteq G_B(V,E)$ of order 1, then $\mathbb{X}_{co} \subseteq \mathbb{X}_B$.*
3. *If there exists a cycle $G_c(V^c,E^c) \subseteq G_B(V,E)$ of order $k\,(>1)$, then $\mathbb{X}_{sync} \subseteq \mathbb{X}_B$.*

Proof:

1. Let $G_c(V^c, E^c) \subseteq G_B(V, E)$ be a cycle of order 0. Then from Lemma 4, there exists a predicate B', such that, $B \Rightarrow B'$ and the corresponding graph $G_{B'}(V', E')$ is a cycle of order 0 with $|V'| = 2$. Since $B \Rightarrow B'$, $\mathbb{X}_{B'} \subseteq \mathbb{X}_B$.

   Since the graph is a cycle with two vertices (both non-β), the predicate $B' \equiv \exists\, x, y\, :\, B'(x, y)$ can only be one of the predicates in the statement of Lemma 3.3. From Lemma 3 we have that the specification corresponding to the above predicates are equivalent to $\mathbb{X}_{async}$. Therefore, $\mathbb{X}_{async} = \mathbb{X}_{B'}$, and $\mathbb{X}_{async} \subseteq \mathbb{X}_B$.

2. We have to show $\mathbb{X}_{co} \subseteq \mathbb{X}_B$. Let $G_c(V^c, E^c) \subseteq G_B(V, E)$ be a cycle of order 1. Then from Lemma 4, there exists a predicate $B \Rightarrow B'$. The corresponding graph $G_{B'}(V', E')$ is a cycle of order 1, such that $|V'| = 2$. Since $B \Rightarrow B'$, $\mathbb{X}_{B'} \subseteq \mathbb{X}_B$.

   Since the graph is a cycle with two vertices (one β), the predicate $B' \equiv \exists\, x, y\, :\, B'(x, y)$ can only be one of the predicates in the statement of Lemma 3.2. From Lemma 3, the specification corresponding to the above predicates is equivalent to $\mathbb{X}_{co}$. Therefore, $\mathbb{X}_{co} = \mathbb{X}_{B'}$, and $\mathbb{X}_{co} \subseteq \mathbb{X}_B$.

3. We have to show $\mathbb{X}_{sync} \subseteq \mathbb{X}_B$. Let $G_c(V^c, E^c) \subseteq G_B(V, E)$ be a cycle of order $k\,(> 1)$. Then from Lemma 4, there exists a predicate $B \Rightarrow B'$. The corresponding graph $G_{B'}(V', E')$ is a cycle of order k, such that $|V'| = k$. Since $B \Rightarrow B'$, $\mathbb{X}_{B'} \subseteq \mathbb{X}_B$.

   Since the graph is a cycle with k β vertices, the predicate B' is

$$B' \equiv (x_1.s \triangleright x_2.r) \wedge (x_2.s \triangleright x_3.r) \cdots (x_k.s \triangleright x_1.r).$$

   This implies the predicate in the statement of Lemma 3.1. Therefore, $\mathbb{X}_{sync} \subseteq \mathbb{X}_{B'} \subseteq \mathbb{X}_B$. □

Theorem 4 (Necessary Conditions) *Let $\mathbb{X}_B$ be a specification with B as the corresponding forbidden predicate. Let the predicate graph be $G_B(V, E)$ with a cycle $G_c(V^c, E^c) \subseteq G_B(V, E)$.*

1. *If there does not exist a cycle $G_c(V^c, E^c) \subseteq G_B(V, E)$ of order 0, 1 or n, then $\mathbb{X}_{sync} \not\subseteq \mathbb{X}_B$.*
2. *If there does not exist a cycle $G_c(V^c, E^c) \subseteq G_B(V, E)$ of order 0 or 1, then $\mathbb{X}_{co} \not\subseteq \mathbb{X}_B$.*
3. *If there does not exist a cycle $G_c(V^c, E^c) \subseteq G_B(V, E)$ of order 0, then $\mathbb{X}_{async} \not\subseteq \mathbb{X}_B$.*

Proof:

1. If there does not exist a cycle of order 0, 1, or n, then there does not exist a cycle. From theorem 2 it follows that $\mathbb{X}_{sync} \not\subseteq \mathbb{X}_B$.

2. We have to show that $\mathbb{X}_{co} \not\subseteq \mathbb{X}_B$, given there does not exist a cycle of order 0 or 1 in the predicate graph. We will construct a run $(\mathtt{H}, \triangleright)$ and show that $(\mathtt{H}, \triangleright) \notin \mathbb{X}_B$, but $(\mathtt{H}, \triangleright) \in \mathbb{X}_{co}$.

   Let the forbidden predicate be $B(x_1, \cdots x_n)$. Consider a run $(\mathtt{H}, \triangleright)$ such that the set of messages $M = \{x_1, \cdots x_n\}$. The causality relation is defined as

$$(\mathtt{H}, \triangleright) = \{\{\ (x_i.h, x_j.f)\ :\ (x_i.h \triangleright x_j.f) \text{ is a conjunct of } B\ \} \cup \{\ (x_i.s, x_i.r)\ :\ i = 1, \ldots n\ \}\}^+,$$

   where $^+$ represents the transitive closure. Since for this run $B(x_1, \cdots x_n)$ is true, $(\mathtt{H}, \triangleright) \notin \mathbb{X}_B$. The claim is $(\mathtt{H}, \triangleright) \in \mathbb{X}_{co}$. We will show $(\mathtt{H}, \triangleright) \in \mathbb{X}_{co}$ by contradiction.

Assume $(\mathtt{H}, \triangleright) \notin \mathbb{X}_{co}$. From the definition of $\mathbb{X}_{co}$,

$$\mathbb{X}_{co} = \{\ (\mathtt{H}, \triangleright)\ :\ \neg((x.s \triangleright y.s)\ \wedge\ (y.r \triangleright x.r)), \forall\, x, y \in M\ \}.$$

Therefore, $\exists\, x_i, x_j \in M$ such that $(x_i.s \triangleright x_j.s)\ \wedge\ (x_j.r \triangleright x_i.r)$ is true.

We can rewrite $(x_i.s \triangleright x_j.s)\ \wedge\ (x_j.r \triangleright x_i.r)$ as $(C_1\ \wedge\ C_2\ \wedge \cdots \wedge\ C_p)\ \wedge\ (C'_1\ \wedge\ \cdots\ C'_q)$, where C is either a conjunct of B or of the form $x_k.s \triangleright x_k.r$.

We are interested in forming a graph (cycle) from the Cs. Drop all the Cs which are not a conjunct of B, since they do not contribute to the cycle.

Consider the predicate graph formed by the resulting Cs. Let the predicate be B_c and $G_c(V^c, E^c)$. It is a cycle and $G_c(V^c, E^c) \subseteq G(V, E)$. For each C remaining there is an edge. We have to analyze the vertex formed by two Cs. If C_i is of form $(x.s \triangleright x.r)$ (thus dropped) then $C_{i-1} = (y.h \triangleright x.s)$ and $C_{i+1} = (x.r \triangleright z.f)$, thus the vertex formed by C_{i-1} and C_{i+1} is not a β vertex.

Let us consider the case when C_i and C_{i+1} are parts of the conjunct. Then, $C_i = (y.f \triangleright x.h)$ and $C_{i+1} = (x.h \triangleright z.g)$. Thus the vertex formed by C_i and C_{i+1} is not a β vertex.

Therefore the vertices formed by the Cs in the same group do not result in any β vertex. There are two more vertices to be considered, that is, the vertices formed by the group joining. Note that C_p and C'_1 are conjuncts of B since they cannot be of the form $x.s \triangleright x.r$. Therefore, the vertex formed by joining C_p and C'_1 results in a non-β vertex. The is because C_p is of the form $(y.h \triangleright x_j.s)$ and C'_1 is of the form $(x_j.r \triangleright z.f)$.

Therefore, the number of vertices left to be considered is one (it may or may not be a β vertex). Thus the resulting graph is of order 0 or 1.

3. We have to show that $\mathbb{X}_{async} \not\subseteq \mathbb{X}_B$ given that there does not exist a cycle of order 0 in the predicate graph $G_B(V, E)$.

   Let us assume that the predicate graph does not have a cycle of order 0. We construct a run $(\mathtt{H}, \triangleright)$ and show that $(\mathtt{H}, \triangleright) \notin \mathbb{X}_B$ but $(\mathtt{H}, \triangleright) \in \mathbb{X}_{async}$.

   Let the forbidden predicate be $B(x_1, \cdots x_n)$. Consider a run $(\mathtt{H}, \triangleright)$ such that the set of messages $M = \{x_1, \cdots x_n\}$. The causality relation is defined as

   $$(\mathtt{H}, \triangleright) = \{\{\ (x_i.h, x_j.f)\ :\ (x_i.h \triangleright x_j.f) \text{ is a conjunct of } B\ \} \cup \{\ (x_i.s, x_i.r)\ :\ i = 1, \ldots n\ \}\}^+,$$

   where $^+$ represents the transitive closure. Since for this run $B(x_1, \cdots x_n)$ is true, $(\mathtt{H}, \triangleright) \notin \mathbb{X}_B$. The claim is $(\mathtt{H}, \triangleright) \in \mathbb{X}_{async}$. We show $(\mathtt{H}, \triangleright) \in \mathbb{X}_{async}$ by contradiction.

   Assume $(\mathtt{H}, \triangleright) \notin \mathbb{X}_{async}$. From the definition of $\mathbb{X}_{async}$,

   $$\mathbb{X}_{async} = \{\ (\mathtt{H}, \triangleright)\ :\ \nexists\, h \in \mathtt{H} \text{ such that } h \triangleright h\ \}.$$

   Therefore, $\exists\, h \in \mathtt{H}$ such that $(h \triangleright h)$.

   We can rewrite $(h \triangleright h)$ as $(C_1\ \wedge\ C_2\ \wedge \cdots \wedge\ C_p)$ where C is either a conjunct of B or of the form $x_k.s \triangleright x_k.r$.

We are interested in forming a graph (cycle) from the Cs. Drop all the Cs which are not a conjunct of B, since they do not contribute to the cycle.

Consider the predicate graph formed by the resulting Cs. Let the predicate be B_c and $G_c(V^c, E^c)$. It is a cycle and $G_c(V^c, E^c) \subseteq G(V, E)$. By similar reasoning as in the previous case the only β vertex that can be possible is between C_1 and C_p.

(a) C_1 is of the form $x.s \triangleright x.r$ (thus dropped), then C_p is of the form $y.f \triangleright x.r$, and C_2 is of the form $x.r \triangleright z.g$. The vertex formed by C_p and C_2 is not a β vertex.

(b) C_p is of the form $x.s \triangleright x.r$ (thus dropped), then C_1 is of the form $x.r \triangleright y.f$, and C_{p-1} is of the form $z.g \triangleright x.s$. The vertex formed by C_{p-1} and C_1 is not a β vertex.

(c) C_1 and C_p are not of the form $x.s \triangleright x.r$. Thus C_1 is of the form $x.f \triangleright y.g$ and C_p is of the form $z.h \triangleright x.f$. The vertex formed by C_p and C_1 is not a β vertex.

Therefore, the graph is a cycle of order 0. Thus, $G(V, E)$ has a cycle of order 0. (contradiction). □

5 Discussion

We considered limitation of protocols that operate by delaying events using local knowledge, causal knowledge and concurrent knowledge. The theory developed in this paper provides an algorithm in determining the existence of a protocol, given a specification using forbidden predicates. Further if there does exist a protocol; whether local, causal or concurrent knowledge is sufficient and necessary to guarantee safety and liveness.

The process of determining the existence of a type of protocol can be done easily, if the specification can be expressed using forbidden predicates. For example, in mobile computations, when a mobile unit moves from one region serviced by a fixed station to another region, it has to communicate with the handshake and the handoff. These handshake and handoff messages, have to satisfy properties like

$$\forall\, y, x \;:\; x \text{ is a handoff message } :\; (y.r \triangleright x.r) \vee (y.s \triangleright x.r) \vee (y.r \triangleright x.s) \vee (y.s \triangleright x.s) \\ \Longrightarrow\; \neg\left((x.r \triangleright y.r) \vee (x.s \triangleright y.r) \vee (x.r \triangleright y.s) \vee (x.s \triangleright y.s)\right).$$

Using the results of this paper it can be easily concluded that guaranteeing the condition requires additional control messages.

Similarly, a specification such as receive the second message before the first, may seem implementable, but using the results of this paper it can be seen that it is not. A protocol can gaurantee safety either by knowing the future, that is, the process knows the existence/absence of the second message (in the future) or the protocol violates liveness condition, that is, a message may not be delivered to the destination process even in a reliable system.

We can see that the following specifications can be implemented by merely tagging the user messages:

FIFO : The messages are received in the same order that they are sent between any pair of processes.

$$(\text{process}\,(s_1) = \text{process}\,(s_2)) \wedge (\text{process}\,(r_1) = \text{process}\,(r_2)) \;::\; (s_1 \triangleright s_2) \wedge (r_2 \triangleright r_1)$$

k–Weaker Causal Ordering : The messages can be out of order by at most k messages.

$$(s_1 \triangleright s_2) \wedge (s_2 \triangleright s_3) \wedge \cdots \wedge (r_{k+2} \triangleright r_1).$$

Local Forward–Flush : All messages sent before any *red* message are received before the *red* message between any pair of processes.

$$\text{process}(s_1) = \text{process}(s_2) \wedge \text{process}(r_1) = \text{process}(r_2) \wedge \text{color}(x_2) = red \;\; : \;\; (s_1 \triangleright s_2) \wedge (r_2 \triangleright r_1).$$

Global Forward–Flush : All messages sent before a *red* message are received before the *red* message.

$$\text{color}(x_2) = red \;\; : \;\; (s_1 \triangleright s_2) \wedge (r_2 \triangleright r_1).$$

This paper presented a general characterization of message ordering specifications. We classified the protocols into three major types (1) those that do nothing, (2) those that can tag information, and (3) those that can have control messages and tag information. The limitations of the three classes of protocols was studied. The results in this paper can be extended to incorporate multicast messages.

References

[1] M. Ahuja. An implementation of F-channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(6):658–667, June 1993.

[2] R. Bagrodia. Process synchronization: Design and performance evaluation of distributed algorithms. *IEEE Transactions on Software Engineering*, 15(9):1053–1065, September 1989.

[3] R. Bagrodia. Synchronization of asynchronous processes in CSP. *ACM Transactions Programming Language Systems*, 11(4):585–597, October 1989.

[4] K. P. Birman and T. A. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, January 1987.

[5] G. Bracha and S. Toeug. Distributed deadlock detection. *Distributed Computing*, 2(3):127–138, January 1987.

[6] G. Buckley and A. Silbershatz. An effective implementation of the generalized input-output construct of CSP. *ACM Transactions Programming Language Systems*, 2(2):223–235, April 1980.

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

[8] K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation.* Addison-Wesley, 1988.

[9] B. Charron-Bost, F. Mattern, and G. Tel. Synchronous and asynchronous communication in distributed computations. Technical Report TR91.55, LITP, University Paris 7, September 1991.

[10] O. P. Damani and V. K. Garg. How to recover efficiently and asynchronously when optimism fails. Technical Report TR-PDS-1995-017, Parallel and Distributed Systems Laboratory, The University of Texas at Austin, 1995.

[11] E. W. Dijkstra. The distributed snapshot of K.M Chandy and L. Lamport. In M. Broy, editor, *Control Flow and Data Flow: Concepts of Distributed Programming*. Springer-Verlag, 1985.

[12] A. Gahlot and M. Ahuja. Global flush communication primitive for inter-process communication. In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 111–120. ACM, 1994.

[13] V. K. Garg. Some optimal algorithms for decomposed partially ordered sets. *Inf. Process. Lett.*, 44:39–43, November 1992.

[14] D. B. Johnson and W. Zwaenepeol. Recovery in distributed systems using optimistic message logging and checkpointing. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing*, pages 171–181. ACM, 1988.

[15] R. Koo and S. Toueg. Checkpointing and rollback-recovery for distributed systems. *IEEE Transactions on Software Engineering*, 13(1):23–31, January 1987.

[16] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications fo the ACM*, 21(7):95–114, July 1978.

[17] F. Mattern. Efficient distributed snapshots and global virtual time algorithms for non-FIFO systems. Draft Version, March 1990.

[18] V. V. Murty and V. K. Garg. An algorithm to gaurantee synchronous ordering of messages. In *Proceedings of Second International Symposium on Autonomous Decentralized Systems*, pages 208–214. IEEE Computer Society Press, 1995.

[19] V. V. Murty and V. K. Garg. Message ordering based on colorful forbidden predicates. Under preparation, 1996.

[20] M. Raynal, A. Schiper, and S. Toueg. The causal ordering abstraction and a simple way to implement it. *Inf. Process. Lett.*, 39(6):343–350, July 1991.

[21] A. Schiper, J. Eggli, and A. Sandoz. A new algorithm to implement causal ordering. In *Proceedings of the Third International Workshop on Distributed Algorithms*, pages 219–232. Springer-Verlay, 1989.

[22] F. Schmuck. Efficient broadcast primitives in asynchronous distributed systems. In K. P. Birman and R. VanRenesse, editors, *Reliable Distributed Computing with the Isis Toolkit: Collected Readings*, pages 263–283. IEEE Press, 1995.

[23] A. P. Sistla. Distributed algorithms for ensuring fair interprocess communication. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, pages 266–277. ACM, 1984.

[24] T. Soneoka and T. Ibaraki. Logically instantaneous message passing in asynchronous distributed systems. *IEEE Transactions on Computers*, 43(5):513–527, May 1994.

[25] K. Taylor. The role of inhibition in asynchronous consistent-cut protocols. In J.-C. Bermond and M. Raynal, editors, *Proc. of the 3rd International Workshop on Distributed Algorithms*, pages 280–291. Springer-Verlag, 1989.

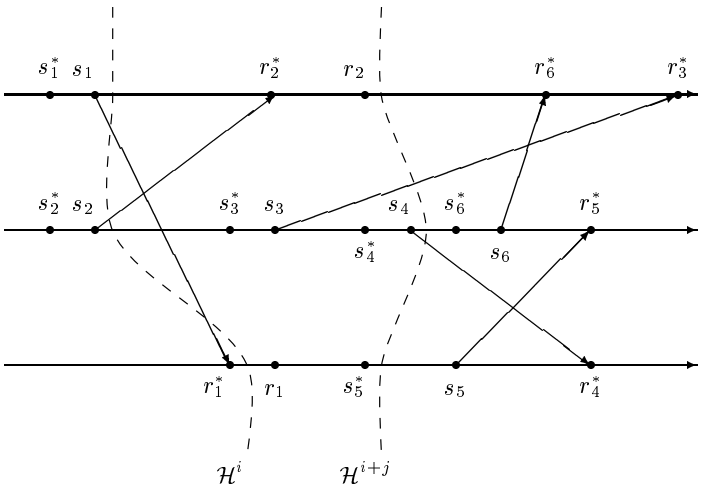


Figure 6: Prefixes of $\mathcal{H}$.

Appendix

A Proof of Lemma 2

Given a protocol $\mathcal{P}$, the set of possible runs under the protocol $\mathcal{X}_P$ is defined inductively. Thus, the proof of the inclusion of a run $\mathcal{H}$ in the set $\mathcal{X}_P$ is also done similarly. Given a run $\mathcal{H}$ we construct a series of prefixes $\mathcal{H}^0, \mathcal{H}^1, \ldots, \mathcal{H}^i, \ldots$, such that $\mathcal{H}^i$ is a prefix of $\mathcal{H}$ and $H^i \subset H^{i+1} \subseteq H$, and $\mathcal{H}^0$ is an empty run, that is $H^0 = \emptyset$. We show that $\mathcal{H} \in \mathcal{X}_P$ inductively. Clearly, the base case follows, that is, $\mathcal{H}^0 \in \mathcal{X}_P$, since $\mathcal{H}^0$ is an empty run. Further, we show that if $\mathcal{H}^i \in \mathcal{X}_P$ then $\mathcal{H}^{i+1} \in \mathcal{X}_P$. Given $\mathcal{H}^i \in \mathcal{X}_P$, the conditions for $\mathcal{H}^{i+1} \in \mathcal{X}_P$ are, for all j,

C1 : H_j^i is a prefix of H_j^{i+1} and they differ by at most one event, and

C2 : $H_j^{i+1} \subseteq H_j^i \cup P_j(\mathcal{H}^i)$.

To prove that $\mathcal{H} \in \mathcal{X}_P$, we will use the following properties of a protocol $\mathcal{P}$,

P1 : $I_i(\mathcal{H}) \cup R_i(\mathcal{H}) \subseteq P_i(\mathcal{H}) \subseteq I_i(\mathcal{H}) \cup R_i(\mathcal{H}) \cup C_i(\mathcal{H})$, and

P2 : $R(\mathcal{H}) \cup C(\mathcal{H}) \neq \emptyset \Rightarrow P(\mathcal{H}) \cap (R(\mathcal{H}) \cup C(\mathcal{H})) \neq \emptyset$.

In our proofs, if we can show that $R(\mathcal{H}) \cup C(\mathcal{H})$ is a singleton or an empty set, than we can conclude $C(\mathcal{H}) \subseteq P(\mathcal{H})$ using the property **P2**. If $C(\mathcal{H}) \subseteq P(\mathcal{H})$, then from **P1** its clear that $P_i(\mathcal{H}) = I_i(\mathcal{H}) \cup R_i(\mathcal{H}) \cup C_i(\mathcal{H})$.

A.1 Proof of Lemma 2, part 1.

Let $\mathcal{H} \in \mathcal{X}_{gn}$. By definition of $\mathcal{X}_{gn}$, there exists a numbering scheme N that assigns a unique number to each event, such that

$$N(x.r) = N(x.r^*) + 1 = N(x.s) + 2 = N(x.s^*) + 3$$

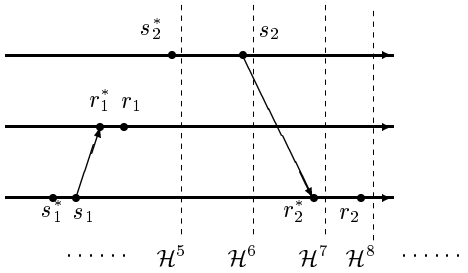


Figure 7: Numbering Scheme for an element $\mathcal{H} \in \mathcal{X}_{gn}$.

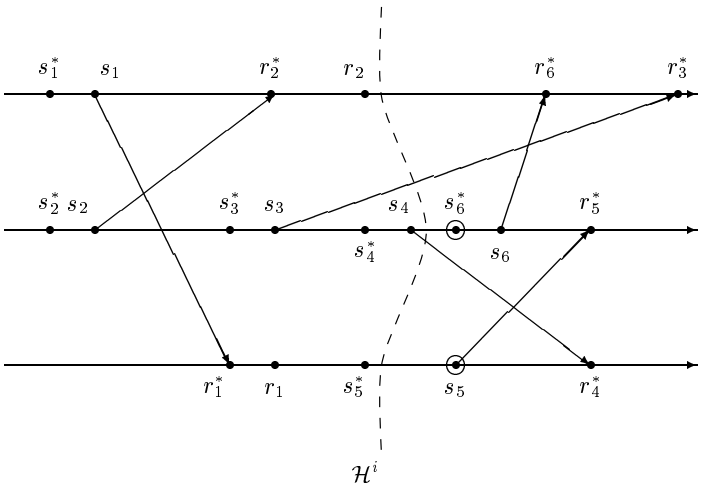


Figure 8: Constructing the next prefix given $\mathcal{H}^i$.

and $(g \rightarrow h) \Rightarrow (N(g) < N(h))$. Using this numbering, we can define a total order in the messages and construct the required prefixes, that is, $\mathcal{H}^0, \mathcal{H}^1, \ldots$, as shown in Figure 7. Since, $\mathcal{H}^0 \in \mathcal{X}_P$ it is sufficient to show that $\mathcal{H}^{i+1} \in \mathcal{X}_P$, given $\mathcal{H}^i \in \mathcal{X}_P$. Clearly, H^{i+1} and H^i differ at most by one event. Therefore, for each j, H_j^i is a prefix of H_j^{i+1} and they differ by at most one event. Thus, **C1** is satisfied for all j.

We have to show that **C2** is satisfied, that is, $\left(H^{i+1} - H^i\right) \subseteq P(\mathcal{H}^i)$. There are four possible cases:

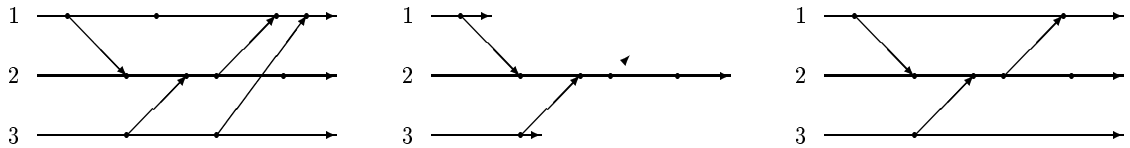
Let $\underline{i = 4m}$. Then $H^{i+1} - H^i = \{\, s_{m+1}^* \,\}$ and $s_{m+1}^* \in I(\mathcal{H}^i)$, since only up to m messages have been executed. Due to property **P1**, $s_{m+1}^* \in I(\mathcal{H}^i)$ implies $s_{m+1}^* \in P(\mathcal{H}^i)$.

Let $\underline{i = 4m+1}$. Then $H^{i+1} - H^i = \{\, s_{m+1} \,\}$ and $S(\mathcal{H}^i) = \{\, s_{m+1}\}$, $R(\mathcal{H}^i) = \emptyset$ and $D(\mathcal{H}^i) = \emptyset$. Due to property **P2**, singleton set $C(\mathcal{H}^i) \cup R(\mathcal{H}^i)$ implies $S(\mathcal{H}^i) \subseteq P(\mathcal{H}^i)$. Therefore, $s_{m+1} \in P(\mathcal{H}^i)$.

Let $\underline{i = 4m+2}$. Then $H^{i+1} - H^i = \{\, r_{m+1}^* \,\}$ and $S(\mathcal{H}^i) = \emptyset$, $R(\mathcal{H}^i) = \{r_{m+1}^*\}$ and $D(\mathcal{H}^i) = \emptyset$. Due to property **P1**, $r_{m+1}^* \in R(\mathcal{H}^i)$ implies $r_{m+1}^* \in P(\mathcal{H}^i)$.

Let $\underline{i = 4m+3}$. Then $H^{i+1} - H^i = \{\, r_{m+1} \,\}$ and $S(\mathcal{H}^i) = \emptyset$, $R(\mathcal{H}^i) = \emptyset$ and $D(\mathcal{H}^i) = \{r_{m+1}\}$. Due to property **P2**, singleton set $C(\mathcal{H}^i) \cup R(\mathcal{H}^i)$ implies $R(\mathcal{H}^i) \subseteq P(\mathcal{H}^i)$. Therefore, $r_{m+1} \in P(\mathcal{H}^i)$.

Therefore, in each case we have $H^{i+1} = H^i \cup P(\mathcal{H}^i)$.



(a) Run $\mathcal{H}^i$ (b) $CausalPast_2(\mathcal{H}^i)$ (c) Run $\mathcal{G}$

Figure 9: Construction of $\mathcal{G}$ given $\mathcal{H}^i$ for process j.

A.2 Proof of Lemma 2, part 2.

To prove that a run $\mathcal{H} \in \mathcal{X}_P$, we have to construct a sequence of runs $\mathcal{H}^0, \mathcal{H}^1, \ldots,$ that are prefixes of $\mathcal{H}$. We construct the sequence such that if the longest path from start of the computation to an event h is k, then $h \notin H^{k-1}$ and $h \in H^k$.

Let $\mathcal{H}^i$ be a prefix of $\mathcal{H}$, then $\mathcal{H}^{i+1}$ contains the bottom elements among the events that do not belong to $\mathcal{H}^i$. For example, in Figure 8, $H^{i+1} = H^i \cup \{ s_5, s_6^* \}$. Formally,

$$H_j^{i+1} = H_j^i \cup B_j(\mathcal{H}, \mathcal{H}^i),$$

where, $B_j(\mathcal{H}, \mathcal{H}^i) = \left\{ h \in H_j - H_j^i \ : \ (g \rightarrow h) \Rightarrow g \in H^i \right\}$.
Let $\mathcal{P}$ be a `tagged` protocol. By definition of `tagged` protocols,

$$\textbf{P3} \ : \ CausalPast_j(\mathcal{H}) = CausalPast_j(\mathcal{G}) \Rightarrow P_j(\mathcal{H}) = P_j(\mathcal{G}).$$

We have to show that if $\mathcal{H} \in \mathcal{X}_{td}$, then $\mathcal{H} \in \mathcal{X}_P$, by induction. Construct the prefixes of $\mathcal{H}$ as defined above. Clearly, $\mathcal{H}^0 \in \mathcal{X}_P$, since it is the empty run.

Let $\mathcal{H}^i \in \mathcal{X}_P$, we have to show that $\mathcal{H}^{i+1} \in \mathcal{X}_P$. In other words for any j, $\mathcal{H}_j^i$ and $\mathcal{H}_j^{i+1}$ satisfies **C1** and **C2**.

Construct a run $\mathcal{G}$ as shown in Figure 9. Pick $CausalPast_j(\mathcal{H}^i)$ and extend all messages (with destination process not being j) in transit. Therefore, we have

$$CausalPast_j(\mathcal{H}^i) = CausalPast_j(\mathcal{G}).$$

We make the following claims:

1. $P_j(\mathcal{H}^i) = P_j(\mathcal{G})$, by the property **P3**.
2. $C_j(\mathcal{G}) = C_j(\mathcal{H}^i)$, since $H_j^i = G_j$.
3. $R_k(\mathcal{G}) = \emptyset$, where $k \neq j$, by construction of $\mathcal{G}$.
4. $R_j(\mathcal{G}) = \emptyset$. (Proof by contradiction)
   Let $x.r^* \in R_j(\mathcal{G})$, therefore $\exists\, k \ : \ (x.r^* \notin G_j) \wedge (x.s \in G_k) \wedge (x \in M_{kj})$. Since, $(x.s \in G_k)$ and $(x \in M_{kj})$, we have by the definition of $CausalPast$ $\exists\, h \ : \ (x.s \rightarrow h) \wedge (h \in G_j)$.

   Since, $(x.s \rightarrow h)$ and $x.s, h$ in different processes, we have either
   (a) $\exists\, y \in M \ : \ (x.s \rightarrow y.s) \wedge ((y.r^* \rightarrow h) \vee (y, r^* \equiv h))$, or
   (b) $(x.s \rightarrow x.r^*) \wedge (x.r^* \rightarrow h)$.

   But $x.r^* \notin G$, therefore $\neg(x.r^* \rightarrow h)$. Since $h, x.r^* \in H_j$, either $(x.r^* \rightarrow h)$ or $(h \rightarrow x.r^*)$. Therefore,

$$\exists\, x, y \in M \ : \ (x.s \rightarrow y.s) \wedge (y.r^* \rightarrow x.r^*) \Rightarrow \mathcal{H} \notin \mathcal{X}_{td}.$$

Figure 10: Construction of $\mathcal{G}$ given $\mathcal{H}^i$ for process j

5. $C_k(\mathcal{G}) = \emptyset$, where $k \neq j$, since $x.s^*$ immediately precedes $x.s$, $x.r^*$ immediately precedes $x.r$ and construction of $\mathcal{G}$.
6. $C_j(\mathcal{G})$ is a singleton or an empty set. Since, $x.s^*$ immediately precedes $x.s$ and $x.r^*$ immediately precedes $x.r$.
7. $B_j(\mathcal{H}, \mathcal{H}^i)$ is a singleton or an empty set. Let $g, h \in H_j \rightarrow H_j^i$. Since $g, h \in H_j \Rightarrow (g \rightarrow h) \vee (h \rightarrow g)$, we have either $g \in B_j(\mathcal{H}, \mathcal{H}^i)$ or $h \in B_j(\mathcal{H}, \mathcal{H}^i)$.
8. $B_j(\mathcal{H}, \mathcal{H}^i) \subseteq I_j(\mathcal{H}^i) \cup R_j(\mathcal{H}^i) \cup C_j(\mathcal{H}^i)$. The statement is trivially true if $B_j(\mathcal{H}, \mathcal{H}^i)$ is an empty set. Let $B_j(\mathcal{H}, \mathcal{H}^i) = \{h\}$. Then the event h can be either an invocation, a send, a delivery or a receive event.
   (a) Let $h \equiv x.s^*$. By definition $x.s^* \in H_j - H_j^i$, therefore
   $\Rightarrow (x.s^* \in H_j) \wedge (x.s^* \notin H_j^i) \Rightarrow (\exists k : x \in M_{jk}) \wedge (x.s^* \notin H_j^i) \Rightarrow x.s^* \in I_j(\mathcal{H}^i)$.
   (b) Let $h \equiv x.s$. Therefore, $(x.s \in H_j - H_j^i) \wedge (x.s^* \rightarrow x.s)$
   $\Rightarrow (x.s \notin H_j^i) \wedge (x.s \in H_j^i) \Rightarrow x.s \in S_j(\mathcal{H}^i)$.

   Similarly, we can show that $h \equiv x.r^* \Rightarrow x.r^* \in R_j(\mathcal{H}^i)$ and $h \equiv x.r \Rightarrow x.r \in D_j(\mathcal{H}^i)$. Hence, $B_j(\mathcal{H}, \mathcal{H}^i) \subseteq I_j(\mathcal{H}^i) \cup R_j(\mathcal{H}^i) \cup C_j(\mathcal{H}^i)$.

From (3), (4), (5) and (6), we have $R(\mathcal{G}) \cup C(\mathcal{G})$ is a singleton or empty set. Using property **P2** and $R(\mathcal{G}) \cup C(\mathcal{G})$ being either a singleton or empty set, we get $C_j(\mathcal{G}) \subseteq P_j(\mathcal{G})$. Substituting for $C_j(\mathcal{G})$ and $P_j(\mathcal{G})$ from (1) and (2), we get $C_j(\mathcal{H}^i) \subseteq P_j(\mathcal{H}^i)$.
Since, $C_j(\mathcal{H}^i) \subseteq P_j(\mathcal{H}^i)$, we get (using property **P1**),

$$P_j(\mathcal{H}^i) = I_j(\mathcal{H}^i) \cup R_j(\mathcal{H}^i) \cup C_j(\mathcal{H}^i).$$

Therefore, from (8)

$$B_j(\mathcal{H}, \mathcal{H}^i) \subseteq P_j(\mathcal{H}^i).$$

$\mathcal{H}^{i+1} \in \mathcal{X}_P$, since for any j,

1. $H_j^{i+1} = H_j^i \cup B_j(\mathcal{H}, \mathcal{H}^i) \subseteq H_j^i \cup P_j(\mathcal{H}^i) \Rightarrow$ **C1** and
2. $B_j(\mathcal{H}, \mathcal{H}^i)$ is a singleton or an empty set $\Rightarrow$ **C2**.

A.3 Proof of Lemma 2, part 3

Let $\mathcal{P}$ be a `tagless` protocol. Therefore,

$$\mathbf{P3} \ : \ H_j = G_j \Rightarrow P_j(\mathcal{H}) = P_j(\mathcal{G}).$$

We have to show that if $\mathcal{H} \in \mathcal{X}_{tl}$, then $\mathcal{H} \in \mathcal{X}_P$, by induction. Construct the prefixes of $\mathcal{H}$ as defined in the last section.
Let $\mathcal{H}^i \in \mathcal{X}_P$, we have to show that $\mathcal{H}^{i+1} \in \mathcal{X}_P$. Construct a run $\mathcal{G}$ by removing and adding events such that (Figure 10 shows the construction of $\mathcal{G}$ given $\mathcal{H}^i$ with respect to process j.) (a) $H^i_j = G_j$, (b) messages that do not effect (a) are deleted, and (c) the messages sent with destination $\neq j$ are delivered and received (no pending events).
The claims are very similar to the previous proof.

1. $P_j(\mathcal{H}^i) = P_j(\mathcal{G})$, by the property **P3**.
2. $C_j(\mathcal{G}) = C_j(\mathcal{H}^i)$, since $H^i_j = G_j$ by construction of $\mathcal{G}$.
3. $C_k(\mathcal{G}) = \emptyset$, where $k \neq j$, since there are no pending events in the process $k \neq j$.
4. $R_k(\mathcal{G}) = \emptyset$, there are no messages in transit.
5. $C_j(\mathcal{G})$ is a singleton or an empty set. Since $x.s^*$ immediately precede $x.s$ and $x.r^*$ immediately precede $x.r$.
6. $B_j(\mathcal{H}, \mathcal{H}^i)$ is a singleton or an empty set.
7. $B_j(\mathcal{H}, \mathcal{H}^i) \subseteq I_j(\mathcal{H}^i) \cup R_j(\mathcal{H}^i) \cup C_j(\mathcal{H}^i)$.

The rest of the proof is identical to the previous one.