Quantifying the Discord: Order Discrepancies in Message Sequence Charts [⋆]

Edith Elkind [1], Blaise Genest [2], Doron Peled[3], and Paola Spoletini[4]

[1] School of Electronics and Computer Science University of Southampton, UK
[2] CNRS/IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
[3]Department of Computer Science, University of Warwick
Coventry CV4 7AL, United Kingdom
and Department of Computer Science, Bar Ilan University,
Ramat Gan 52900, Israel
[4] Dipartimento di Elettronica e Informazione, Politecnico di Milano
via Ponzio 34/5 - 20133, Milano, Italy

Abstract. Message Sequence Charts (MSCs) and High-level Message Sequence Charts (HMSC) are formalisms used to describe scenarios of message passing protocols. We propose using Allen's logic to study the temporal order of the messages. We introduce the concept of *discord* to quantify the order discrepancies between messages in different nodes of an HMSC and study its algorithmic properties. We show that while discord of a pair of messages is hard to compute in general, the problem becomes polynomial-time computable if the number of nodes of the HMSC or the number of processes is constant. Moreover, for a given HMSC, it is always computationally easy to identify a pair of messages that exhibits the worst-case discord, and compute the discord of this pair.

1 Introduction

Message Sequence Charts (MSCs) and High-level Message Sequence Charts (HMSC) are very useful tools for describing executions of communication protocols. They provide an intuitive visual notation, which is widely used in practice and has been formally described in the MSC standard [11]. A related notation was also adopted as part of the UML standard. Intuitively, an MSC is described by a set of *processes* and a set of *messages* between these processes. The notation allows us to specify the order in which each process sends and receives messages. An HMSC is a graph whose nodes are labeled with MSCs. An execution of an HMSC is a concatenation of MSCs that appear on a path in this graph. Using HMSC notation, one can describe alternative behaviors of systems, or even use it as a scenario-based programming formalism [10]. The reader is referred to Section 2 for formal definitions.

Besides being used in practice, MSCs and HMSCs have been extensively studied from theoretical perspective over the past few years. This research has pointed out several difficulties with these formalisms. One such example is the problem of detecting race conditions in MSCs [2], i.e., the possibility that messages arrive out of order due

[⋆] Work partly supported by the ESF project Automatha and the ANR project DOTS.

to lack of synchronization. This problem has also been generalized to HMSCs [14] and sets of MSCs [7]. Another problem is related to global choice [4, 3], where some processes behave according to one MSC scenario and other processes behave according to another MSC scenario, resulting in new behaviors.

Continuing this line of research, in this paper we identify another ambiguity of the MSC notation. Namely, in the definition of an HMSC, a concatenation of MSCs along a path intuitively suggests that messages that appear in an earlier MSC precede in time any message that appears in a later MSC. In fact, in some frameworks such as *live sequence charts* [6] there is a hidden assumption of such synchronous nature. However, according to the MSC semantics, this is is not the case: independence between events happening in different sets of processes may allow messages in later MSCs to overlap or even sometimes appear earlier than messages in previous MSCs. Moreover, it is not clear how to achieve this kind of synchronization without an additional mechanism or extra messages. Clearly, this discrepancy may result in users misinterpreting the notation and, as a result, designing protocols that do not work as intended. This is reminiscent of the concept of race conditions: the straightforward visual interpretation of concatenation is different from the intended semantics. However, unlike for race conditions, this discrepancy has not been studied before.

In this paper, we provide a formal treatment of this issue. We introduce the notion of *discord* of a pair of messages in different nodes of an HMSC. Intuitively, the discord of two messages is the worst possible discrepancy between their order in an execution and their "ideal" order, in which the message in the MSC that appears earlier on the path precedes the message in the MSC later on the path. To formalize this intuition, we need several tools that we introduce below.

We start our study of the message order in MSCs and HMSCs by defining the concept of a *chain*. Informally, a chain is a sequence of events where any adjacent pair of events is ordered either by being a send-receive pair, or by belonging to the same process line. Hence, a chain represents a possible flow of information. Clearly, the order between messages is determined not only by the relevant messages themselves, but also by chains between their endpoints. We characterize the possible message orders by describing the possible communication patterns between their endpoints. We then project each such pattern on a global timeline and classify the resulting scenarios. To do so, we use a subset of *Allen's interval logic* [1]. Allen's logic is a formalism for describing the relative order of time intervals. For example, Allen's logic formula $A\mathbf{d}B$ expresses the fact that A happens during B, i.e., A starts after B starts and ends before B ends. It has been widely studied in the context of artificial intelligence and knowledge representation, and its expressive power and computational properties are well understood [12]. As messages can easily be seen as time intervals, it provides a convenient language for describing the message order. We introduce a natural ordering on Allen's logic primitive predicates and define the discord of a pair of messages in an MSC as the worst possible Allen's logic primitive predicate (according to this ordering) that corresponds to the communication pattern of this pair.

We study the concept of discord from the algorithmic perspective. First, we show that computing the discord of a pair of messages is coNP-complete. Our reduction assumes that both the number of nodes in the HMSC and the number of processes are part of the input. We show that this is inevitable: if either of these numbers is fixed, the discord can be computed in polynomial time. We then focus on characterizing the discord of an HMSC by a single parameter. To this end, we define the discord of an HMSC as the worst possible discord of a pair of messages in this HMSC. Surprisingly, it turns out that this quantity can be computed in time polynomial both in the size of the MSC graph and the number of processes. Intuitively, the reason for that is that it is easy to identify a pair of messages that exhibits the worst-case behavior for a given HMSC and compute the discord of such a pair. The study of discords provides also a generic study of the existence of communication chains, which we believe will be interesting in its own right in studies of layered combination of communication algorithms.

2 Preliminaries

2.1 Message Sequence Charts

Following [11], we formally define message sequence charts (MSCs), MSC concatenation, and high-level message sequence charts (HMSCs).

Definition 1. *A* Message Sequence Chart *(MSC) is a tuple* $C = (\mathcal{P}, E, P, \mathcal{M}, <_{p:p\in\mathcal{P}})$, *where*

- $\mathcal{P}$ *is a finite set of* processes*;*
- E *is a finite set of* events*;*
- $P : E \mapsto \mathcal{P}$ *is a function that maps every event to the process on which it occurs;*
- $\mathcal{M}$ *is a finite set of messages. Each message* $m \in \mathcal{M}$ *consists of a pair of events* (s, r) *for* send *and* receive*;*
- *For each process* $p \in \mathcal{P}$, $<_p$ *is a total order on the events of that process.*

We define a relation $<$ *as* $<= \bigcup_{p\in\mathcal{P}} <_p \cup\{(s, r) \mid (s, r) \in M\}$ *and let* $<^*$ *be the transitive closure of* $<$. *We require* $<^*$ *to be acyclic. We assume that MSCs are FIFO, that is, if two messages* (s_1, r_1) *and* (s_2, r_2) *are between the same processes, i.e.,* $P(s_1) = P(s_2)$ *and* $P(r_1) = P(r_2)$, *then* $s_1 < s_2$ *implies* $r_1 < r_2$.

We will occasionally abuse notation and write $m \in C$ instead of $m \in \mathcal{M}$.

Definition 2. *Let* C_1, C_2 *be two MSCs where* $C_1 = (\mathcal{P}^1, E^1, P^1, \mathcal{M}^1, <^1_{p:p\in\mathcal{P}^1})$, $C_2 = (\mathcal{P}^2, E^2, P^2, \mathcal{M}^2, <^2_{p:p\in\mathcal{P}^2})$ *with* $\mathcal{P}^1 = \mathcal{P}^2 = \mathcal{P}$ *and* $E^1 \cap E^2 = \emptyset$. *Define their* concatenation *as an MSC* $(C_1; C_2) = (\mathcal{P}, E, P, \mathcal{M}, <_{p:p\in\mathcal{P}})$, *where* $E = E^1 \cup E^2$, $\mathcal{M} = \mathcal{M}^1 \cup \mathcal{M}^2$, *the function* P *is given by* $P(e) = P^1(e)$ *if* $e \in E^1$ *and* $P(e) = P^2(e)$ *if* $e \in E^2$, *and for each* $p \in \mathcal{P}$ *we define* $<_p = <^1_p \cup <^2_p \cup\{(e_1, e_2)|P^1(e_1) = P^2(e_2)\}$.

Notice that there are no sends in one MSC that are received in the other. This definition can be naturally extended to sequences $C_1, C_2, \ldots, C_n$ of three or more MSCs by setting $(C_1; C_2; \ldots; C_n)$ = $((\ldots(C_1; C_2); C_3)\ldots)$.

Definition 3. *A High-level Message Sequence Chart (HMSC) is a tuple* $H = (\mathcal{G}, \mathcal{C}, \mathcal{V}_0, \lambda)$, *where* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *is a directed graph with the vertex set* $\mathcal{V} = \{v_1, \ldots, v_n\}$ *and the edge set* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, $\mathcal{C} = \{C_1, \ldots, C_n\}$ *is a collection of MSCs with a* *common set of processes and mutually disjoint sets of events, $\mathcal{V}_0 \subseteq \mathcal{V}$ is a set of* initial *nodes, and $\lambda : \mathcal{V} \mapsto \mathcal{C}$ is a bijective mapping between the nodes of the graph and the MSCs in $\mathcal{C}$. To simplify notation, we assume $\lambda(v_i) = C_i$. Each vertex of $\mathcal{G}$ is reachable from one of the initial nodes. An* execution *of the HMSC is a finite MSC $(C_i; \ldots; C_j)$ obtained by concatenating the MSCs in the nodes of a path $v_i, \ldots, v_j$ of the HMSC that starts with some initial node $v_i \in \mathcal{V}_0$. The* size *$|H|$ of an HMSC H is defined as $|H| = |E_1| + \cdots + |E_n| + |\mathcal{V}| + |\mathcal{E}|$, where E_i is the set of events of the MSC C_i.*

Given a path $L = (v_i, \ldots, v_j)$ in $\mathcal{G}$ of length at least 2, we denote by $\lambda(L)$ the MSC that is obtained by concatenating the MSCs along L, i.e., $(C_i; \ldots; C_j)$. The set of executions of an HMSC is also referred to as the set of MSCs *generated* by that HMSC.

We can define infinite executions in a similar way. This requires defining the concatenation of an infinite sequence of MSCs, which is the limit of the sequence of finite concatenations of prefixes. As this does not add to the results in the paper, we only refer the reader to, e.g., [9].

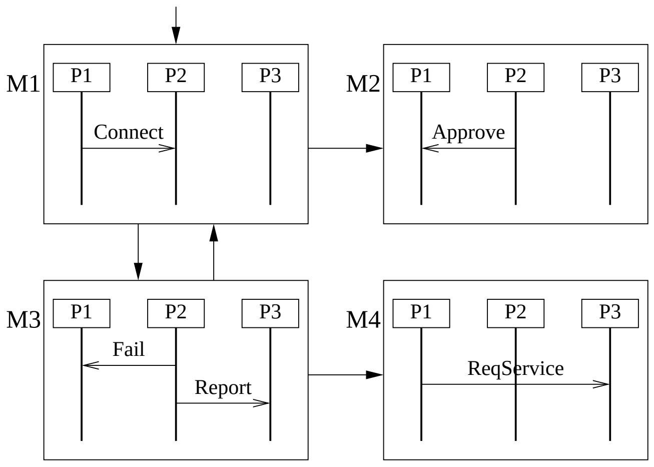

Fig. 1. An HMSC

Figure 1 shows an example of an HMSC. The node in the upper left corner, denoted M1, is the starting node, hence it has an incoming edge that is connected to no other node. Initially, process P1 sends a message to P2, requesting a connection (e.g., to an internet service), according to the node M1. This can result in either an approval message from P2, according to the node M2, or a failure message, according to the node M3. In the latter case, a report message is also sent from P2 to some supervisory process P3. There are two progress choices, corresponding to the two arrows out of the node M3. We can decide to try and connect again, by choosing the arrow from M3 to M1, or to give up and send a service request (from process P1 to process P3), by choosing to progress according to the arrow from M3 to M4. Note how the HMSC description abstracts away from internal process computation, and presents only the communications. Consider the path (M1, M3, M4). According to the HMSC semantics, process P2 does not necessarily have to send its Report message in M3 before process P1 has progressed according to M4 to send its Req_service message. However, process P3 must receive the Report message before the Req_service message.

2.2 Allen's logic

Allen's logic [1] is a formalism that allows one to express temporal relationships between time intervals. It has 13 primitive relations that correspond to possible relationships between two intervals, such as "A precedes B" or "A happens during B". Each primitive relation describes a total order between the endpoints of these intervals. When working with MSCs, we normally assume that no two events can happen at the same time, i.e., no two intervals have a common endpoint. Therefore, to represent relationships between two messages $m_1 = (s_1, r_1)$ and $m_2 = (s_2, r_2)$, we will only use 6 of these primitives, namely:

$\mathbf{p}$ — m_1 precedes m_2 (i.e., $s_1 < r_1 < s_2 < r_2$);
$\mathbf{p}^{-1}$ — m_1 is preceded by m_2 (i.e., $s_2 < r_2 < s_1 < r_1$);
$\mathbf{o}$ — m_1 overlaps m_2 (i.e., $s_1 < s_2 < r_1 < r_2$);
$\mathbf{o}^{-1}$ — m_1 is overlapped by m_2 (i.e., $s_2 < s_1 < r_2 < r_1$);
$\mathbf{d}$ — m_1 is during m_2 (i.e., $s_2 < s_1 < r_1 < r_2$);
$\mathbf{d}^{-1}$ — m_1 contains m_2 (i.e., $s_1 < s_2 < r_2 < r_1$).

Observe that for $\mathbf{t} \in \{\mathbf{p}, \mathbf{o}, \mathbf{d}\}$ the predicate $A\mathbf{t}B$ is equivalent to $B\mathbf{t}^{-1}A$.

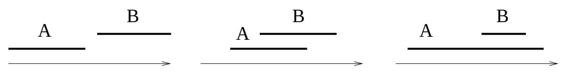

Fig. 2. Allen's logic relationships: $A\mathbf{p}B$, $A\mathbf{o}B$, and $A\mathbf{d}^{-1}B$

An Allen's logic formula consists of concatenation of one or more of these 6 letters, and is interpreted as a disjunction of the corresponding predicates. For example, the formula $A\mathbf{pod}^{-1}B$ says that either A precedes B, or A overlaps B, or B happens during (is included in) A. Given the semantics of the primitive predicates, it is easy to see that this formula says that A starts before B, but may end before ($\mathbf{p}$), during ($\mathbf{o}$), or after ($\mathbf{d}^{-1}$) B. There are several operations that can be performed on Allen's logic formulas, such as composition and intersection. However, in this paper we only use the Allen's logic as a means to describe the relationships between messages. Therefore, we will not formally define these operations.

3 Relationships between Messages

In this section, we will show how to use Allen's logic to reason about the relationship between a given pair of messages.

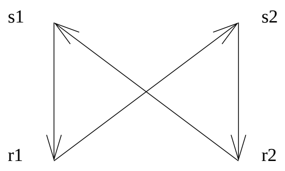

Fig. 3. Impossible relation between messages

Given an MSC C, a *chain* from $x \in E$ to $y \in E$ is a sequence of events $(x = e_{i_1}, e_{i_2}, \ldots, e_{i_{k-1}}, e_{i_k} = y)$ such that $e_{i_j} \in E$ for $j = 1, \ldots, k$, and every adjacent pair $(e_{i_j}, e_{i_{j+1}})$ in the chain is either a send and the corresponding receive, or e_{i_j} appears before (above) $e_{i_{j+1}}$ in the same process line. Clearly, $x <^* y$ if and only if there is a chain of messages from x to y. Now, consider a pair of messages (s_1, r_1) and (s_2, r_2). By definition, there is always a chain from s_1 to r_1 and from s_2 to r_2. Moreover, for any $(a, b) \in \{s_1, r_1\} \times \{s_2, r_2\}$, we have one of the following three cases: (1) there is a chain of messages from a to b; (2) there is a chain of messages from b to a; (3) there is no chain in either direction. As there are four pairs of points, this corresponds to $3^4 = 81$ combinations. However, not all of them are possible, as MSCs do not admit cycles (see Figure 3). In fact, for two messages there are exactly twenty possible combinations of orders between their endpoints. We list them in Figure 4. In these figures, the messages correspond to the vertical arrows, and dashed arrows correspond to relationships derived by transitive closure.

The patterns in Figure 4 correspond to the following Allen's logic relationships: (a) $\mathbf{pp}^{-1}\mathbf{oo}^{-1}\mathbf{dd}^{-1}$; (b) $\mathbf{p}$; (c) $\mathbf{pod}^{-1}$; (d) $\mathbf{po}$; (e) $\mathbf{o}$; (f) $\mathbf{d}^{-1}$; (g) $\mathbf{od}^{-1}$; (h) $\mathbf{poo}^{-1}\mathbf{dd}^{-1}$; (i) $\mathbf{od}$; (j) $\mathbf{opd}$; (k) $\mathbf{oo}^{-1}\mathbf{dd}^{-1}$. Except for cases (a) and (k), which are symmetric, each other case has a symmetric twin that can be obtained by swapping the left and the right message.

To decide between these cases, it suffices to calculate the transitive closure relation $<^*$. While in general transitive closure algorithms run in cubic time [8, 16], it has been observed [2] that in the MSC case one can be more efficient since each event has at most two successors. Formally, we have the following proposition.

Proposition 1. *[2]* Given an MSC M with messages $m_1, \cdots, m_t$, one can decide in time $O(t^2)$ the relation between every m_i, m_j, $1 \leq i, j \leq t$.

We will now derive a corollary that will be useful in bounding the running time of our algorithms.

Corollary 1. *Given an HMSC $H = (\mathcal{G}, \mathcal{C}, \mathcal{V}_0, \lambda)$, $|\mathcal{C}| = n$, one can compute the relation $<^*$ for all MSCs in $\mathcal{C}$ in time $O(|H|^2)$. Moreover, one can compute the relation $<^*$ for all concatenated MSCs of the form $(C_i; C_j)$, where $C_i, C_j \in \mathcal{C}$, in time $O(n|H|^2)$. Finally, given the relation $<^*$ for all concatenated MSCs of the form $(C_i; C_j)$, where $C_i, C_j \in \mathcal{C}$, one can compute the Allen's logic relationship for all pairs of messages m, m', where $m \in C_i$, $m' \in C_j$, $C_i, C_j \in \mathcal{C}$, in time $O(n|H|^2)$.*

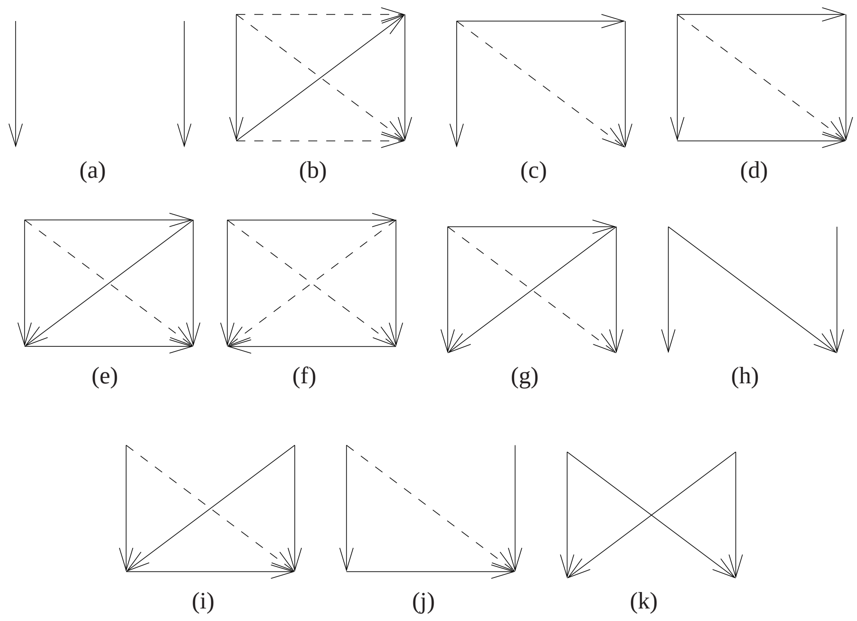

Fig. 4. The possible orders between messages (up to symmetry)

Proof. Let E_i be the set of events of the MSC C_i. By Proposition 1, we can compute $<^*$ for C_i in time $O(|E_i|^2)$. Therefore, computing $<^*$ for all C_i, $i = 1, \ldots, n$, takes time $O(|E_1|^2 + \cdots + |E_n|^2) = O((|E_1| + \cdots + |E_n|)^2) = O(|H|^2)$.

Similarly, computing the relation $<^*$ for $(C_i; C_j)$ can be done in time $O((|E_i| + |E_j|)^2)$. As $(|E_i| + |E_j|)^2 \leq 2|E_i|^2 + 2|E_j|^2$, computing $<^*$ for all MSCs of the form $(C_i; C_j)$, $i, j = 1, \ldots, n$, can be done in time $O(n(|E_1|^2 + \cdots + |E_n|^2)) = O(n|H|^2)$. Now, fix $1 \leq i, j \leq n$. Given the relation $<^*$ for $(C_i; C_j)$, the Allen's logic relationship for any pair (m, m'), $m \in C_i$, $m' \in C_j$, can be computed in constant time. As there are $O(|E_i||E_j|)$ such pairs, computing the Allen's logic relationship for all of them can be done in time $O(|E_i||E_j|) = O((|E_i| + |E_j|)^2)$. Summing over all $i, j = 1, \ldots, n$, we obtain the bound of $O(|H|^2)$, as claimed. $\square$

4 Definition of Discord

Concatenating two MSCs C_1 and C_2 does not necessarily mean that *all* the messages of C_1 precede in time all the messages of C_2: for example, if C_1 consists of a single message from p_1 to p_2, and C_2 consists of a single message from p_3 to p_4, the relation $<$ does not provide any information about the relative order of these two messages. In what follows, we propose an Allen logic-based formalism that allows us to quantify the ordering discrepancies that occur when concatenating MSCs. We start by considering sequences of MSCs, and then extend our analysis to HMSCs.

Consider a concatenated MSC $(C_1; C_2)$. For any two messages $m_1 = (s_1, r_1) \in C_1$ and $m_2 = (s_2, r_2) \in C_2$, we know that $s_1 < r_1$ and $s_2 < r_2$. Now, intuitively, the best possible scenario for $(C_1; C_2)$ is when all messages in C_1 precede all messages in C_2. In this case, we also have $r_1 < s_2$, and thus we obtain $s_1 < r_1 < s_2 < r_2$. This corresponds to case (b) in Figure 4. Note that this scenario is only possible when C_1 has a unique maximal event e, C_2 has a unique minimal event e', and e and e' occur on the same process, i.e., $P(e) = P(e')$.

Conversely, the worst possible case is when some message m_2 in C_2 may be completely unordered with respect to a message m_1 in C_1. That is, for some m_1 and m_2 as above, the situation is described by case (a) in Figure 4, or by the Allen's logic formula $m_1\mathbf{pp}^{-1}\mathbf{oo}^{-1}\mathbf{dd}^{-1}m_2$. In this case, at worst, the Allen logic formula allows m_2 to actually precede m_1, since the disjunction permits in particular that $m_1\mathbf{p}^{-1}m_2$. All remaining scenarios lie, as will be formulated below, between these two cases. We will now introduce a measure of discrepancy, which we call the *discord*, which will allow us to order them more precisely,

Given a concatenation of two MSCs $(C_1; C_2)$, two messages $m_1 = (s_1, r_1) \in C_1$ and $m_2 = (s_2, r_2) \in C_2$ are said to be *out of order* if r_1 does not precede s_2, i.e., $\neg m_1\mathbf{p}m_2$. In Figure 4, this happens in cases (a), (c), (d), (h), and (j). Note that in our setting, the cases (e), (f), (g), (i), and (k) are impossible: in each of these cases, there are chains of messages starting from events of m_2 and ending in events of m_1, which cannot happen under concatenation.

We now classify all primitive Allen logic predicates according to how well they order the endpoints of the projected intervals, i.e., represent the order between the events of the two messages m_1 and m_2. Recall that in the ideal case, i.e., when the order between the intervals is described by the Allen logic predicate $\mathbf{p}$, we have $s_1 < r_1 < s_2 < r_2$. In this case, there are zero events in $\{s_2, r_2\}$ that precede those in $\{s_1, r_1\}$. Under the worst case, i.e., if m_2 fully precedes m_1, we count four inversions: namely, $s_2 < s_1$, $r_2 < r_1$, $r_2 < s_1$ and $s_2 < r_1$. We thus order the predicates according to how many of these four relationships are inverted. In case of a tie, we give preference to the relationships that involve s_1 to those that involve r_1.

Definition 4. *The total order $\prec$ is the transitive closure of the partial order $\prec_0$ given by* $\prec_0 = \{(\mathbf{p}, \mathbf{o}), (\mathbf{o}, \mathbf{d}^{-1}), (\mathbf{d}^{-1}, \mathbf{d}), (\mathbf{d}, \mathbf{o}^{-1}), (\mathbf{o}^{-1}, \mathbf{p}^{-1})\}$.

Remark 1. Observe that the number of inversions in $\mathbf{p}^{-1}$ is 4, as explained above, in $\mathbf{o}^{-1}$ it is 3, in $\mathbf{d}$ and $\mathbf{d}^{-1}$ it is 2, in $\mathbf{o}$ it is 1, and in $\mathbf{p}$ it is 0. Therefore, our decision that $\mathbf{d}^{-1} \prec \mathbf{d}$ may appear quite arbitrary. We made this choice for two reasons. First, we do think that the time when the messages are sent is more important than the time when they are received, as the designer has more control over the former, and second, it is convenient to have a total order to work with. However, we believe that many of our ideas and results will apply for different orders, including some that are not total.

Definition 5. *Consider a sequence of MSCs $(C_1, \ldots, C_k)$ and a pair of messages $m_1 \in C_1$, $m_2 \in C_k$ such that in the MSC $C = (C_1; \ldots; C_k)$ we have $m_1\mathbf{R}m_2$, where $\mathbf{R}$ is a (possibly non-primitive) Allen's logic predicate. The* discord *of m_1 and m_2 with respect to C is the* worst *possible primitive predicate (largest according to $\prec$) that appears in*

$\mathbf{R}$, *i.e.*, $\mathrm{discord}_C(m_1, m_2) = \mathbf{t}$, *where* $\mathbf{t} \in \{\mathbf{p}, \mathbf{p}^{-1}, \mathbf{o}, \mathbf{o}^{-1}, \mathbf{d}, \mathbf{d}^{-1}\}$, $\mathbf{t}$ *appears in* $\mathbf{R}$, *and for all* $\mathbf{t}'$ *that appear in* $\mathbf{R}$ *we have* $\mathbf{t}' \preceq \mathbf{t}$.

Let us now apply this definition to the six cases that can occur for a pair of messages in a concatenated MSC, as illustrated in Figure 4. In case (a) the messages are in relationship $\mathbf{pp}^{-1}\mathbf{oo}^{-1}\mathbf{dd}^{-1}$. The worst elementary predicate in this formula is $\mathbf{p}^{-1}$, so we conclude that the discord between the messages is $\mathbf{p}^{-1}$. For case (b), there is only one relation $\mathbf{p}$. Similarly, for case (c) the discord is $\mathbf{d}^{-1}$, for case (d) it is $\mathbf{o}$, for (h) it is $\mathbf{o}^{-1}$, and for (j) it is $\mathbf{d}$. We conclude that the value of $\mathrm{discord}_C(m_1, m_2)$ can be *any* elementary Allen's logic predicate.

We now extend the definition of a discord to messages in HMSCs.

Definition 6. *Given an HMSC* $H = (\mathcal{G}, \mathcal{S}, \mathcal{V}_0, \lambda)$ *and a pair of messages* $m_1 \in \lambda(v)$, $m_2 \in \lambda(v')$, *let* $\mathrm{discord}_H(m_1, m_2) = \max^{\prec}\{\mathrm{discord}_{\lambda(L)}(m_1, m_2) \mid L = (v, \ldots, v')\}$, *where* $\max^{\prec}\mathcal{A}$ *is the maximum element of the set* $\mathcal{A}$ *with respect to* $\prec$.

Consider now the HMSC in Figure 1. For the path (M1, M2), the discord is $\mathbf{p}$, since the maximum event of M1, which is a receive, precedes the minimum event of M2, which is the send of message Approve. On the other hand, for the path (M1, M3, M1), we have that the Report message of M3 corresponds to the Connect message of M1 as in case (h) of Figure 4, which means a discord of $\mathbf{o}^{-1}$. The discord of (M3, M4) is $\mathbf{d}$ due to the relative ordering between Report in M3 and ReqService in M4.

We will now state a simple observation that allows us to compute $\mathrm{discord}_H(m_1, m_2)$.

Claim 1 Consider an HMSC $H = (\mathcal{G}, \mathcal{C}, \mathcal{V}_0, \lambda)$. For any $v, v' \in \mathcal{V}$, $v \neq v'$, and any $m_1 \in \lambda(v)$, $m_2 \in \lambda(v')$, we have $\mathrm{discord}_H(m_1, m_2) = \max^{\prec}\{\mathrm{discord}_{\lambda(L)}(m_1, m_2) \mid L = (v, \ldots, v')$ is a simple path$\}$. Also, for two messages $m_1, m_2 \in \lambda(v)$, we have $\mathrm{discord}_H(m_1, m_2) = \max^{\prec}\{\mathrm{discord}_{\lambda(L)}(m_1, m_2) \mid L = (v, \ldots, v)$ is a simple cycle$\}$.

Intuitively, this is true because removing a loop from a path from v to v' can only increase the discord between m_1 and m_2. Hence, the path that exhibits the worst-case discord is loop-free.

5 Computing the Discord of a Pair of Messages

For a simple path $L = (v = v_{i_1}, \ldots, v_{i_k} = v')$, computing $\mathrm{discord}_{\lambda(L)}(m_1, m_2)$ for $m_1 \in \lambda(v)$, $m_2 \in \lambda(v')$ is easy. Namely, first we run the transitive closure algorithm to determine the causal relationships between the endpoints of m_1 and m_2. We then identify the corresponding scenario of Figure 4 and apply the case analysis presented after Definition 5. The running time of this algorithm is quadratic in the total number of messages in $\lambda(L)$.

For HMSCs, Definition 6 and Claim 1 suggest a straightforward algorithm for computing the discord: given two messages $m_1 \in \lambda(v)$, $m_2 \in \lambda(v')$, we can consider each simple path from v to v' (or each simple cycle, if $v = v'$), compute the discord along this path, and output the maximum discord obtained in this way. This naive algorithm runs in exponential time in the input size. In the next subsection, we show that this is perhaps inevitable: we prove that in general the problem of computing $\mathrm{Discord}_H(m_1, m_2)$

is coNP-hard. However, we will now provide an alternative way of verifying whether $\mathrm{Discord}_H(m_1, m_2) = \mathbf{t}$, where $\mathbf{t} \in \{\mathbf{p}, \mathbf{p}^{-1}, \mathbf{o}, \mathbf{o}^{-1}, \mathbf{d}, \mathbf{d}^{-1}\}$. As we will see later, it can be used to construct an efficient algorithm for computing $\mathrm{Discord}_H(m_1, m_2)$ in the important special case when the number of processes is constant.

We will first define a related problem that will be useful for stating our results.

PATH WITH NO CHAIN: Given an HMSC $H = (\mathcal{G} = (\mathcal{V}, \mathcal{E}), \mathcal{C}, \mathcal{V}_0, \lambda)$, a pair of nodes $v, v' \in \mathcal{V}$, and a pair of events $e \in \lambda(v), e' \in \lambda(v')$, is there a path L from v to v' in $\mathcal{G}$ such that in the MSC $\lambda(L)$ there is no chain of events from e to e'? We will write $\mathrm{PNC}_H(e, e') = 1$ if such path exists and $\mathrm{PNC}_H(e, e') = 0$ otherwise.

Proposition 2. Given an HMSC $H = (\mathcal{G} = (\mathcal{V}, \mathcal{E}), \mathcal{C}, \mathcal{V}_0, \lambda)$, a pair of nodes $v, v' \in \mathcal{V}$, and a pair of messages $m_1 = (s_1, r_1) \in \lambda(v)$, $m_2 = (s_2, r_2) \in \lambda(v')$, we have

– $\mathrm{discord}_H(m_1, m_2) = \mathbf{p}$ if and only if $\mathrm{PNC}_H(r_1, s_2) = 0$.
– $\mathrm{discord}_H(m_1, m_2) = \mathbf{o}$ if and only if $\mathrm{PNC}_H(r_1, s_2) = 1$, $\mathrm{PNC}_H(s_1, s_2) = 0$, and $\mathrm{PNC}_H(r_1, r_2) = 0$.
– $\mathrm{discord}_H(m_1, m_2) = \mathbf{d}^{-1}$ if and only if $\mathrm{PNC}_H(r_1, r_2) = 1$ and $\mathrm{PNC}_H(s_1, s_2) = 0$.
– $\mathrm{discord}_H(m_1, m_2) = \mathbf{d}$ if and only if $\mathrm{PNC}_H(s_1, s_2) = 1$ and for any path $L = (v, \ldots, v')$ in $\mathcal{G}$, the MSC $\lambda(L)$ contains a chain from s_1 to s_2 or a chain from r_1 to r_2.
– $\mathrm{discord}_H(m_1, m_2) = \mathbf{o}^{-1}$ if and only if there exists a path $L = (v, \ldots, v')$ in $\mathcal{G}$ such that the MSC $\lambda(L)$ contains no chain from s_1 to s_2 and no chain from r_1 to r_2, and $\mathrm{PNC}_H(s_1, r_2) = 0$.
– $\mathrm{discord}_H(m_1, m_2) = \mathbf{p}^{-1}$ if and only if $\mathrm{PNC}_H(s_1, r_2) = 1$.

Proof. The analysis for $\mathbf{p}$, $\mathbf{o}$, and $\mathbf{p}^{-1}$ is straightforward.

If $\mathrm{discord}_H(m_1, m_2) = \mathbf{d}^{-1}$, then there is a path $L = (v, \ldots, v')$ that satisfies $\mathrm{discord}_{\lambda(L)}(m_1, m_2) = \mathbf{d}^{-1}$. Clearly, $\lambda(L)$ contains no chain from r_1 to r_2, so $\mathrm{PNC}_H(r_1, r_2) = 1$. Also, for any path L' from v to v', we have $\mathrm{discord}_{\lambda(L')}(m_1, m_2) \in \{\mathbf{p}, \mathbf{o}, \mathbf{d}^{-1}\}$, so L' contains a chain from s_1 to s_2. Hence, $\mathrm{PNC}_H(s_1, s_2) = 0$. Conversely, if $\mathrm{PNC}_H(r_1, r_2) = 1$, then there is a path L from v to v' with no chain from r_1 to r_2, so it cannot be the case that $\mathrm{discord}_{\lambda(L)}(m_1, m_2) \in \{\mathbf{p}, \mathbf{o}\}$. Hence, $\mathrm{discord}_{\lambda(L)}(m_1, m_2) \succeq \mathbf{d}^{-1}$. On the other hand, $\mathrm{PNC}_H(s_1, s_2) = 0$ means that any path L' from v to v' contains a chain from s_1 to s_2, so we have $\mathrm{discord}_{\lambda(L')}(m_1, m_2) \notin \{\mathbf{d}, \mathbf{o}^{-1}, \mathbf{p}^{-1}\}$. Other cases can be analyzed similarly. □

Note that to check if $\mathrm{discord}_H(m_1, m_2) = \mathbf{t}$ for $\mathbf{t} \in \{\mathbf{p}, \mathbf{p}^{-1}, \mathbf{d}^{-1}, \mathbf{o}\}$, it suffices to make a small number of calls to PNC_H. However, to check if $\mathrm{discord}_H(m_1, m_2) = \mathbf{t}$ for $\mathbf{t} \in \{\mathbf{d}, \mathbf{o}^{-1}\}$, calling PNC_H is not enough. Indeed, to verify, e.g., whether $\mathrm{discord}_H(m_1, m_2) = \mathbf{d}$, we have to check that any path between the corresponding nodes contains *either* one of two chains: a chain from s_1 to s_2 or a chain from r_1 to r_2, and this check cannot be simulated by calls to PNC_H.

5.1 Computational hardness

We will now show that for HMSCs the problem of upper-bounding $\mathrm{discord}_H(m_1, m_2)$ is coNP-complete. Formally, we consider the following problem:

DISCORD$(H, \mathbf{t}, m_1, m_2)$: Given an HMSC H, a predicate $\mathbf{t} \in \{\mathbf{p}, \mathbf{p}^{-1}, \mathbf{o}, \mathbf{o}^{-1}, \mathbf{d}, \mathbf{d}^{-1}\}$, and two messages m_1, m_2 in H, is it the case that $\mathrm{discord}_H(m_1, m_2) \preceq \mathbf{t}$?

Theorem 1. *The problem* DISCORD$(H, \mathbf{t}, m_1, m_2)$ *is coNP-complete.*

Proof. To see that DISCORD$(H, \mathbf{t}, m_1, m_2)$ is in coNP, observe that the complementary problem of checking whether $\mathrm{discord}_H(m_1, m_2) \succ \mathbf{t}$ is in NP: a certificate can be provided by a path L such that $\mathrm{discord}_{\lambda(L)}(m_1, m_2) \succ \mathbf{t}$. In particular, for $\mathbf{t} = \mathbf{p}$ a certificate is a path with no chain from r_1 to s_2, for $\mathbf{t} = \mathbf{o}$ it is a path with no chain from r_1 to r_2, for $\mathbf{t} = \mathbf{d}^{-1}$ it is a path with no chain from s_1 to s_2, for $\mathbf{t} = \mathbf{d}$ it is a path with no chain from s_1 to s_2 and no chain from r_1 to r_2, and for $\mathbf{t} = \mathbf{o}^{-1}$ it is a path with no chain from s_1 to r_2.

The coNP-hardness proof is by reduction from 3SAT. Suppose that we are given a 3CNF formula with a set of variables $x_1, \ldots, x_n$ and a set of clauses $c_1, \ldots, c_m$. Let l_j^1, l_j^2, l_j^3 be the literals that appear in the jth clause, i.e., $c_j = l_j^1 \vee l_j^2 \vee l_j^3$, $l_j^k \in \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$. We construct an HMSC H as follows. Set $\mathcal{P} = \{p_1, p_2, p_3, p_4, p_{x_1}, p_{\bar{x}_1}, \ldots, p_{x_n}, p_{\bar{x}_n}, p_{c_1}, \ldots, p_{c_m}\}$. The HMSC H has the following structure. Its underlying graph $\mathcal{G}$ has a source v_0, a sink v_1, n variable gadgets $X_1, \ldots, X_n$ and m clause gadgets $Y_1, \ldots, Y_m$. The variable gadget X_i consists of four vertices u_i^0, u_i^1, u_i^2, u_i^3 and four edges (u_i^0, u_i^1), (u_i^0, u_i^2), (u_i^1, u_i^3), (u_i^2, u_i^3). The clause gadget Y_i consists of five vertices w_i^0, w_i^1, w_i^2, w_i^3, w_i^4 and six edges (w_i^0, w_i^1), (w_i^0, w_i^2), (w_i^0, w_i^3), (w_i^1, w_i^4), (w_i^2, w_i^4), (w_i^3, w_i^4). The source, the vertex gadgets, the clause gadgets, and the sink are all connected in series as depicted in Figure 5. More precisely, there is an edge from v_0 to the vertex u_1^0, for all $i = 1, \ldots, n-1$ there is an edge from u_i^3 to u_{i+1}^0, there is an edge from u_n^3 to w_1^0, for all $i = 1, \ldots, m-1$ there is an edge from w_i^4 to w_{i+1}^0, and finally there is an edge from w_m^4 to v_1.

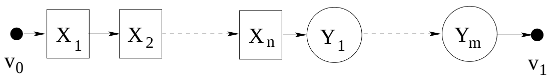

Fig. 5. The high-level structure of the HMSC H used in the proof of Theorem 1.

It remains to define the MSCs that are placed in the vertices of $\mathcal{G}$. The MSC in v_0 consists of a single message (s_1, r_1) from p_1 to p_2. The MSCs in the vertices u_i^0, u_i^3, w_j^0, w_j^4 are empty for all $i = 1, \ldots, n$, $j = 1, \ldots, m$. For $i = 1, \ldots, n$, the MSC in u_i^1 consists of a message from p_2 to p_{x_i}, and the MSC in u_i^2 consists of a message from p_2 to $p_{\bar{x}_i}$. For $j = 1, \ldots, m$, $k = 1, 2, 3$, the MSC in w_j^k contains a message from $p_{l_j^k}$ to p_{c_j}, where l_j^k is the kth literal of c_j. Finally, the MSC in v_1 has $m + 1$ messages: a message from each p_{c_j}, $j = 1, \ldots, m$, to p_3, and a message $m_2 = (s_2, r_2)$ from p_3 to p_4 that is sent after all messages from all p_{c_j} are received.

We claim that the original 3CNF formula is satisfiable if and only if the tuple $(H, \mathbf{p}, m_1, m_2)$ constitutes a "no"-instance of DISCORD$(H, \mathbf{p}, m_1, m_2)$, i.e., there is a path L from v_0 to v_1 such that the MSC $\lambda(L)$ contains no chain from r_1 to s_2.

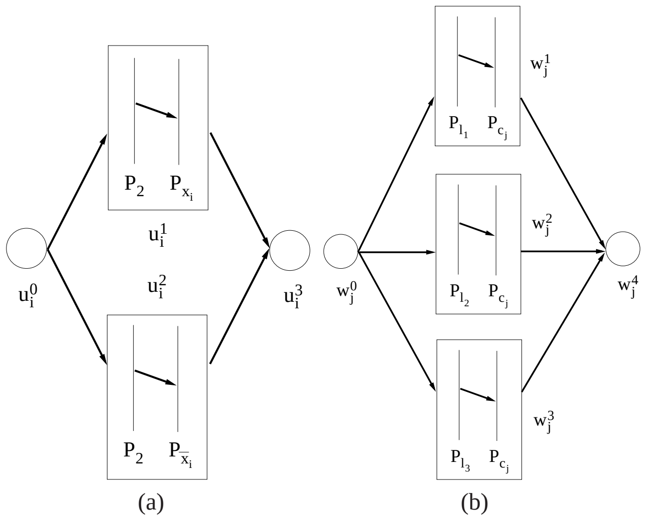

Fig. 6. (a) The gadget X_i; (b) The gadget Y_j

Indeed, suppose that our formula is satisfiable, and let $\mathcal{T} = (t_1, \ldots, t_n)$, $t_i \in \{T, F\}$ be a satisfying assignment for it. Consider a path L that satisfies the following conditions:

– L starts at v_0 and ends at v_1;
– $L \cap X_i = \{u_i^0, u_i^1, u_i^3\}$ if $t_i = F$ and $L \cap X_i = \{u_i^0, u_i^2, u_i^3\}$ if $t_i = T$;
– $L \cap Y_j = \{w_i^0, w_j^k, w_j^4\}$ for some $k \in \{1, 2, 3\}$ such that l_j^k is true under $\mathcal{T}$, i.e., $l_j^k = x_z$ and $t_z = T$ or $l_j^k = \bar{x}_z$ and $t_z = F$. Note that such l_j^k is guaranteed to exist since $\mathcal{T}$ has to satisfy c_j.

First, note that in the corresponding MSC $\lambda(L)$ there is no chain from r_1 to any event of any of the processes p_{c_j}, $j = 1, \ldots, m$. Indeed, the only message received by p_{c_j} in $\lambda(L)$ is from some $p_{l_j^k}$ such that l_j^k is true under $\mathcal{T}$. Since l_j^k is true under $\mathcal{T}$, in $\lambda(L)$ the process $p_{l_j^k}$ receives no messages whatsoever. As p_3 only receives messages from p_{c_j}, $j = 1, \ldots, m$, we conclude that in $\lambda(L)$ there is no chain from r_1 to s_2.

Conversely, suppose that there is a path L such that in the corresponding MSC $\lambda(L)$ there is no chain from r_1 to s_2. Consider a satisfying assignment $\mathcal{T} = (t_1, \ldots, t_n)$ such that $t_i = F$ if $L \cap X_i = \{u_i^0, u_i^1, u_i^3\}$ and $t_i = T$ if $L \cap X_i = \{u_i^0, u_i^2, u_i^3\}$. Note that for any $j = 1, \ldots, m$, if $L \cap Y_j = \{w_i^0, w_j^k, w_j^4\}$ for some $k = 1, 2, 3$, it must be the case that $p_{l_j^k}$ receives no message from p_2 in $\lambda(L)$, because otherwise there would be a chain of messages from r_1 to s_2. Hence, the literal l_j^k is true under $\mathcal{T}$, i.e., c_j is satisfied. As this holds for any $j = 1, \ldots, m$, we have successfully constructed a satisfying assignment for our instance of 3CNF.

□

Remark 2. Clearly, the proof of Theorem 1 implies that PATH WITH NO CHAIN is NP-hard. Moreover, we can consider a weaker version of DISCORD, in which the Allen logic predicate is not part of the input. Namely, for $\mathbf{t} \in \{\mathbf{p}, \mathbf{p}^{-1}, \mathbf{o}, \mathbf{o}^{-1}, \mathbf{d}, \mathbf{d}^{-1}\}$, let $\textsc{Discord}_{\mathbf{t}}(H, m_1, m_2)$ be the problem of checking whether $\mathrm{discord}_H(m_1, m_2) \preceq \mathbf{t}$. Obviously, for $\mathbf{t} = \mathbf{p}^{-1}$ this problem is trivially in P: the answer is always "yes". Our proof shows that for $\mathbf{t} = \mathbf{p}$ this problem is coNP-hard. To show that it is hard for $\mathbf{t} = \mathbf{o}$, we can modify the reduction by changing the direction of m_1 (i.e., setting $P(s_1) = p_2$, $P(r_1) = p_1$) and adding to the MSC in v_0 a message $m'_1 = (s'_1, r'_1)$ from p_1 to p_4 with $r_1 <_{p_1} s'_1$. Then in any path in H there is a chain from r_1 to r_2, and there is a path with no chain from s_1 to s_2 if and only if the 3CNF formula has a satisfying assignment. Similarly, to show that $\textsc{Discord}_{\mathbf{d}^{-1}}(H, m_1, m_2)$ is coNP-hard, we change the direction of m_1, to show that $\textsc{Discord}_{\mathbf{d}}(H, m_1, m_2)$ is coNP-hard, we change the direction of m_2, and to show that $\textsc{Discord}_{\mathbf{o}^{-1}}(H, m_1, m_2)$ is coNP-hard, we change the direction of both m_1 and m_2. We conclude that all five non-trivial versions of the problem are coNP-hard.

5.2 Polynomial-time algorithms for bounded number of processes

In our hardness result, both the size of the graph $\mathcal{G}$ and the number of processes $\mathcal{P}$ are unbounded. It turns out that this is necessary: if either of these parameters is constant, there is an algorithm whose running time is polynomial in the other parameter.

This is easy to see if the size of the graph is constant. In particular, the naive algorithm described in the beginning of this section will run in polynomial time: in a graph with a constant number of vertices, there is a constant number of simple paths and cycles, and one can compute the discord along a path in polynomial time.

The case when the number of processes is constant is considerably more complicated. Our algorithm for this setting is based on Dijkstra's shortest path algorithm combined with dynamic programming approach. The underlying idea is that given a pair of events $e \in \lambda(v)$, $e' \in \lambda(v')$ and a subset of processes $\mathcal{S}$, we can check if there is a path L from v to v' such that the set of processes reachable from e in $\lambda(L)$ is exactly $\mathcal{S}$. A straightforward generalization of this idea allows us to compute the discord of any pair of messages in an HMSC in polynomial time for any fixed value of $|\mathcal{P}|$. Formally, we prove the following result.

Theorem 2. *It is possible to compute* $\mathrm{discord}_H(m_1, m_2)$ *in time* $O(n^3 2^{4|\mathcal{P}|}|H|^2)$.

We start by describing an algorithm for $\textsc{Path with no Chain}$. Next, we show how to generalize it to compute $\mathrm{discord}_H(m_1, m_2)$. Note that just like in Dijkstra's algorithm, we simultaneously check whether there is a path with no chain from a given event $e \in C = \lambda(v)$ to all other events. Therefore, this algorithm can be easily adapted to compute the discords for all pairs of messages in H in time $O(n^3 2^{4|\mathcal{P}|}|H|^3)$.

Let K be a strict upper bound on the number of events on any process line in any MSC in H. Re-number all events so that $e^k_{i,j}$, $k = 1, \ldots, K-1$, is the kth event on the process line p_j in the MSC C_i. For the purposes of the algorithm, we will introduce two dummy events $e^{\min}_{i,j}$ and $e^{\max}_{i,j}$ on each process line of every MSC in H. The event $e^{\min}_{i,j}$ precedes all events $e^k_{i,j}$, and the event $e^{\max}_{i,j}$ follows all events $e^k_{i,j}$. It is important to note that these are not send or receive events, so they have no effect on the information flow in H. However, we will occasionally talk about chains to and from these events, where a chain is defined in the same way as for regular events. We say that a process p_j is *reachable* from e along a path $v = (v, \dots, v_i)$ if in the MSC $(C; \dots; C_i)$ there is a chain from e to $e_{i,j}^{\max}$.

The outline of the algorithm is presented in Figure 7. First, for each MSC C_i and all $l = 1, \dots, |\mathcal{P}|$, the procedure `ComputeX();` checks whether there is a chain from $e_{i,l}^{\min}$ to all other events in this MSC. More precisely, for $k = 1, \dots, K-1$, `ComputeX();` sets $X(i,j,k,l) = 1$ if in C_i there is a chain from $e_{i,l}^{\min}$ to $e_{i,j}^{k}$ and $X(i,j,k,l) = 0$ otherwise. Also, it sets $X(i,j,K,l) = 1$ if in C_i there is a chain from $e_{i,l}^{\min}$ to $e_{i,j}^{\max}$, and $X(i,j,K,l) = 0$ otherwise. Note that for $j \neq l$ there can be no chain from $e_{i,j}^{\min}$ to $e_{i,l}^{\min}$. By Corollary 1, we can implement `ComputeX();` in time $O(|H|^2)$.

```
PNC_H(e, e^k_{i,j});
1. ComputeX();
2. ComputeY();
3. forall $i'$ such that $(v_{i'}, v_i) \in \mathcal{E}$
4.    forall $\mathcal{S} \subseteq \mathcal{P}$
5.       if Y$[\mathcal{S}, i] = 0$ break;
6.       forall $p_l \in \mathcal{P} \setminus \mathcal{S}$
7.          if $X(i,j,k,l) = 1$ break;
8.       return ‘‘yes’’;
```

Fig. 7. The algorithm for $\mathrm{PNC}_H(e, e_{i,j}^k)$. The implementation of `ComputeY();` is given in Figure 8.

Now, for any $\mathcal{S} \subseteq \mathcal{P}$ let $Y(\mathcal{S}, i)$ be a variable that indicates whether there is a path L in $\mathcal{G}$ from v to v_i such that in $\lambda(L)$ none of the processes in $\mathcal{S}$ is reachable from e. We set $Y(\mathcal{S}, i) = 1$ if such a path exists and $Y(\mathcal{S}, i) = 0$ otherwise. The values of $Y(\S, i)$ are computed by the procedure `ComputeY();` given in Figure 8. We will discuss how to implement `ComputeY();` later on.

Now, assume that we have computed $Y(\mathcal{S}, i)$, $X(i,j,k,l)$, for all $\mathcal{S} \subseteq \mathcal{P}$, $i = 1, \dots, n$, $i, j = 1, \dots, |\mathcal{P}|$, $k = 1, \dots, K$. Then there is a path with no chain from e to $e_{i,j}^k$ if and only if the conditions in the lines 3 — 7 hold, i.e., there is a path L of the form $(v, \dots, v_{i'}, v_i)$ and a set $\mathcal{S} \subseteq \mathcal{P}$ such that for any process p_l that is reachable from e along $L' = (v, \dots, v_{i'})$ (i.e., a process in $\mathcal{P} \setminus \mathcal{S}$), there is no chain from $e_{i,l}^{\min}$ to $e_{i,j}^k$. For a fixed event $e_{i,j}^k$ this condition can be verified in time $n2^{|\mathcal{P}|}|\mathcal{P}|$.

It remains to argue that the procedure `ComputeY`(); in Figure 8 correctly computes the values of $Y(\mathcal{S}, i)$. The procedure starts by initializing the variables Y$[\mathcal{S}, i]$ (lines 1—6). For $i \neq 1$, it sets Y$[\mathcal{S}, i] = 0$ for all $\mathcal{S} \subseteq \mathcal{P}$. For $i = 1$, it computes $Y(\mathcal{S}, 1)$ (recall that $Y(\mathcal{S}, 1) = 1$ if and only if there is no chain from e to $e_{1,j}^{\max}$ for any $p_j \in \mathcal{S}$) and sets Y$[\mathcal{S}, i] = Y(\mathcal{S}, i)$. The algorithm then repeats a Dijkstra-like “relaxation” step n times. During each step, the value of each Y$[\mathcal{S}, i]$ may be changed from 0 to 1.

The correctness of the algorithm follows from two simple claims.

```
ComputeY();
1. forall $i = 2, \ldots, n$
2.    forall $\mathcal{S} \subseteq \mathcal{P}$
3.       set Y$[\mathcal{S}, i] = 0$;
4. Set $\mathcal{S}_0 = \{\mathbf{p}_j \mid$ there is no chain from $e$ to $e^{\max}_{1,j}\}$;
5. forall $\mathcal{S} \subseteq \mathcal{P}$
6.    if $\mathcal{S} \subseteq \mathcal{S}_0$ then set Y$[\mathcal{S}, 1] = 1$ else set Y$[\mathcal{S}, 1] = 0$;
7. Repeat $n$ times
8.    forall $i = 1, \ldots, n$
9.       forall $\mathcal{S} \subseteq \mathcal{P}$
10.         if Y$[\mathcal{S}, i] = 1$ break;
11.         forall $i'$ such that $(v_{i'}, v_i) \in \mathcal{E}$
12.            forall $\mathcal{S}'$ such that $\mathcal{S} \subseteq \mathcal{S}'$ and Y$[\mathcal{S}', i'] = 1$
13.               forall $p_j \in \mathcal{S}$
14.                  forall $p_l \in \mathcal{P} \setminus \mathcal{S}'$
15.                     if X(i, j, K, l)=1 break;
16.               Set Y$[\mathcal{S}, i] = 1$;
17. return;
```

Fig. 8. The implementation of `ComputeY();`

Claim 2 *If $Y(\mathcal{S}, i) = 0$, that at any point in the execution of* ComputeY(); *we have* Y$[\mathcal{S}, i] = 0$.

Proof. The proof is by induction on the execution of the algorithm. The claim is clearly true after the initialization step. Now, suppose that at some point we change the value of Y$[\mathcal{S}, i]$ from 0 to 1 for some $\mathcal{S}, i$. This means that we have discovered some $i', \mathcal{S}'$ such that $(v_{i'}, v_i) \in \mathcal{E}$, Y$[\mathcal{S}', i'] = 1$. By inductive assumption, this means that there exists a path L from v to $v_{i'}$ such that in the MSC $(C; \ldots; C_{i'})$ there is no chain from e to any of the processes in $\mathcal{S}'$. Moreover, we also have $X(i, j, K, l) = 0$ for any $p_l \in \mathcal{P} \setminus \mathcal{S}'$ and any $p_j \in \mathcal{S}$, i.e., in the MSC $(C; \ldots; C_{i'}; C_i)$ there is no chain from $e^{\min}_{i,l}$ to $e^{\max}_{i,j}$. Now, suppose that in $(C; \ldots; C_{i'}; C_i)$ there is a chain from e to some $p_j \in \mathcal{S}$. As there are no events that are sent from one MSC and are received in another MSC, this chain would have to go through some $e^{\max}_{i',l}$, $e^{\min}_{i,l}$, $l = 1, \ldots, |\mathcal{P}|$. If $p_l \in \mathcal{S}'$, this means that there is a chain in $(C; \ldots; C_{i'})$ from e to p_l, a contradiction. On the other hand, if $p_l \in \mathcal{P} \setminus \mathcal{S}'$, there is a chain from $e^{\min}_{i,l}$ to $e^{\max}_{i,j}$, a contradiction again. We conclude that $Y(\mathcal{S}, i) = 1$. □

Claim 3 *If for some $\mathcal{S}, i$, there exists a path $(v, \ldots, v_{i'}, v_i)$ of length l such that in the MSC $(C; \ldots; C_{i'}; C_i)$ there is no chain from e to any of the processes in $\mathcal{S}$, then after l steps* `ComputeY();` *sets* Y$[\mathcal{S}, i] = 1$.

Proof. The proof is by induction on l. The claim is obviously true for $l = 1$. Let $\mathcal{S}'$ be the set of all processes that are not reachable from e along $(C_1, \ldots, C_{i'})$. By inductive assumption, after $l-1$ steps we have Y$[\mathcal{S}, i'] = 1$. Also, by construction, in $(C; \ldots; C_{i'})$ there is a chain from e to $e^{\max}_{i,l}$ for any $l \in \mathcal{P} \setminus \mathcal{S}'$. Hence, we have $X(i, j, K, l) = 0$

for any $p_l \in \mathcal{P} \setminus \mathcal{S}'$, $p_j \in \mathcal{S}$. Therefore, during the lth step, our algorithm will set $\mathtt{Y}[\mathcal{S}, i] = 1$. □

It is not hard to verify that the running time of `ComputeY`() is $O(n|\mathcal{E}|2^{2|\mathcal{P}|}|\mathcal{P}|^2)$. Indeed, the running time of this procedure is dominated by the cycle in lines 8–16, which is repeated n times. During each such cycle, we consider each edge of $\mathcal{E}$ exactly once (in lines 8 and 11), for each such edge we consider two subsets of $\mathcal{P}$, and for each choice of these subsets we consider a pair of processes and do a constant-time check for this pair.

The overall running time of our algorithm can then be expressed as $O(n|\mathcal{E}|2^{2|\mathcal{P}|}|\mathcal{P}|^2 + |H|^2) = O(n^3 2^{2|\mathcal{P}|}|H|^2)$.

Now, suppose that we are given a pair of messages $m_1 = (s_1, r_1) \in \lambda(v)$, $m_2 = (s_2, r_2) \in \lambda(v')$. By Proposition 2, we can check whether $\mathrm{discord}_H(m_1, m_2) = \mathbf{t}$ for $\mathbf{t} \in \{\mathbf{p}, \mathbf{o}, \mathbf{d}^{-1}, \mathbf{p}^{-1}\}$ by making at most three calls to $\mathrm{PNC}_H()$;. However, to decide between $\mathrm{discord}_H(m_1, m_2) = \mathbf{d}$ and $\mathrm{discord}_H(m_1, m_2) = \mathbf{o}^{-1}$, we need additional tools. Fortunately, it turns out that one can modify $\mathrm{PNC}_H()$; to solve this problem.

To verify whether $\mathrm{discord}_H(m_1, m_2) = \mathbf{d}$, we first compute $\mathrm{PNC}_H(s_1, s_2)$. If we have $\mathrm{PNC}_H(s_1, s_2) = 0$, then $\mathrm{discord}_H(m_1, m_2) \prec \mathbf{d}$, so the answer is negative. Otherwise, $\mathrm{discord}_H(m_1, m_2) \neq \mathbf{d}$ if and only if $\mathcal{G}$ contains a path L from v to v' such that in $\lambda(L)$ there is no chain from s_1 to s_2 and no chain from r_1 to r_2. To find such a path, we first compute $X(i, j, k, l)$ using `ComputeX();`. We then define $Y'(\mathcal{S}, \mathcal{S}', i)$ as follows: for any $\mathcal{S}, \mathcal{S}' \subseteq \mathcal{P}$ and any $i = 1, \dots, n$, set $Y'(\mathcal{S}, \mathcal{S}', i) = 1$ if there is a path L from v to v_i such that in $\lambda(L)$ none of the processes in $\mathcal{S}$ is reachable from s_1 and none of the processes in $\mathcal{S}'$ is reachable from r_1. It is straightforward to modify `ComputeY();` so that it computes $Y'(\mathcal{S}, \mathcal{S}', i)$ instead of $Y(\mathcal{S}, i)$. The running time of the modified version is $O(n|\mathcal{E}|2^{4|\mathcal{P}|}|\mathcal{P}|^2)$, as we have to consider all possible *pairs* of subsets of $\mathcal{P}$ in adjacent nodes.

Now, suppose that we have computed $Y'(\mathcal{S}, \mathcal{S}', i)$ for all $\mathcal{S}, \mathcal{S}' \subseteq \mathcal{P}$, $i = 1, \dots, n$. Assume that $v' = v_{i^*}$ and $s_2 = e^k_{i^*,j}$, $r_2 = e^{k'}_{i^*,j'}$ We have $\mathrm{discord}_H(m_1, m_2) \succ \mathbf{d}$ if and only if there exists a triple $\mathcal{S}, \mathcal{S}', i'$ such that

(1) $(v_{i'}, v_{i^*}) \in \mathcal{E}$;
(2) $Y'(\mathcal{S}, \mathcal{S}', i') = 1$;
(3) for any $p_l \in \mathcal{P} \setminus \mathcal{S}$ we have $X(i^*, j, k, l) = 0$;
(4) for any $p_l \in \mathcal{P} \setminus \mathcal{S}'$ we have $X(i^*, j', k', l) = 0$.

These conditions can be verified in time $O(n 2^{2|\mathcal{P}|}|\mathcal{P}|)$. Hence, the overall running time of our algorithm is $O(n^3 2^{4|\mathcal{P}|}|H|^2)$, which proves Theorem 2.

6 From Pairs of Messages to HMSCs

In some situations, it is convenient to characterize the discord of an HMSC with a single parameter rather than list the discords for all pairs of messages in this HMSC. To this end, we extend the definition of discord from pairs of messages to entire HMSCs by defining the discord of an HMSC H to be the worst discord over all pairs of messages in H. Formally, we set $\mathrm{Discord}(H) = \max^{\prec}\{\mathrm{discord}_H(m_1, m_2) \mid m_1 \in \lambda(v), m_2 \in$

$\lambda(v'), (v, v') \in \mathcal{E}^*\}$, where $\mathcal{E}^*$ is the transitive closure of the edge set $\mathcal{E}$, and $\max^{\prec}\mathcal{A}$ is the maximal element of the set $\mathcal{A}$ with respect to $\prec$.

According to this definition, one can compute $\mathrm{Discord}(H)$ by computing the discords for all pairs of messages in H. However, in general, computing $\mathrm{discord}_H(m_1, m_2)$ is coNP-hard, so this approach is not efficient. Quite surprisingly, it turns out that there exists a different approach that allows us to compute $\mathrm{Discord}(H)$ in polynomial time. It is based on the fact that while it may be hard to check whether there exists a chain between two events, it is easy to prove that there is no chain between two *extremal* events, for a suitable definition of extremality.

In the rest of the section, we describe polynomial-time algorithms for checking that $\mathrm{Discord}(H) = \mathbf{t}$ for $\mathbf{t} \in \{\mathbf{p}, \mathbf{p}^{-1}, \mathbf{o}, \mathbf{o}^{-1}, \mathbf{d}^{-1}\}$. To check whether $\mathrm{Discord}(H) = \mathbf{d}$, we can simply run all these algorithms and return "yes" if all of them return "no". We analyze the efficiency of these algorithms in terms of $n = |\mathcal{V}|$, $|\mathcal{P}|$ and $|H|$; observe that we can assume $n = O(|H|)$, $|\mathcal{P}| = O(|H|)$.

For the cases $\mathbf{t} \in \{\mathbf{p}, \mathbf{o}, \mathbf{d}^{-1}\}$, we will make use of a set $\mathcal{E}^\star \subset \mathcal{V} \times \mathcal{V}$, constructed as follows: $(v, v') \in \mathcal{E}^\star$ if and only if $(v, v') \in \mathcal{E}$ or there exists a path $(v = v_{i_1}, v_{i_2}, \ldots, v_{i_{k-1}}, v_{i_k} = v')$ such that for $j = 2, \ldots, k-1$ the MSC $\lambda(v_{i_j})$ has an empty message set. Note that $\mathcal{E}^\star$ is a subset of the transitive closure of $\mathcal{E}$, i.e., $(v, v') \in \mathcal{E}^\star$ implies that in $\mathcal{G}$ there is a path from v to v'.

To construct $\mathcal{E}^*$, we can run the depth-first search from each node of $\mathcal{V}$, backtracking as soon as we discovered a node whose MSC has a non-empty message set. Clearly, this algorithm finds a path from v to v' if and only if $(v, v') \in \mathcal{E}^\star$. Moreover, as depth-first search runs in time $O(|\mathcal{V}| + |\mathcal{E}|) = O(|H|)$, the total running time of this algorithm is $O(n|H|)$.

$\mathbf{Discord}(\boldsymbol{H}) = \mathbf{p}$. We will show that $\mathrm{Discord}(H) = \mathbf{p}$ if and only if for any $(v, v') \in \mathcal{E}^\star$, and any $m_1 \in \lambda(v)$, $m_2 \in \lambda(v')$ we have $\mathrm{discord}_{(\lambda(v);\lambda(v'))}(m_1, m_2) = \mathbf{p}$.

Indeed, if for some such m_1, m_2 we have $\mathrm{discord}_{(\lambda(v);\lambda(v'))}(m_1, m_2) \neq \mathbf{p}$, then obviously $\mathrm{Discord}(H) \neq \mathbf{p}$. Conversely, consider any pair of messages $m_1 = (s_1, r_1) \in \lambda(v)$, $m_2 = (s_2, r_2) \in \lambda(v')$ and any path $L = (v = v_{i_1}, \ldots, v_{i_k} = v')$. We show by induction on k that if our condition holds then $\mathrm{discord}_{\lambda(L)}(m_1, m_2) = \mathbf{p}$. The proof is based on the fact that for any three time intervals A, B, C, we have $A\mathbf{p}B \wedge B\mathbf{p}C \implies A\mathbf{p}C$. For $k = 2$, the statement is obvious. Now, suppose $k > 2$. If for each $j = 2, \ldots, k-1$, the MSC $\lambda(v_{i_j})$ has an empty message set, then we have $\lambda(L) = (\lambda(v); \lambda(v'))$, so $\mathrm{discord}_{\lambda(L)}(m_1, m_2) = \mathbf{p}$. Now suppose that for some $j \in \{2, \ldots, k-1\}$ the MSC $\lambda(v_{i_j})$ has non-empty message set and consider some $m = (s, r) \in \lambda(v_{i_j})$. Set $L' = (v_{i_1}, \ldots, v_{i_j})$, $L'' = (v_{i_j}, \ldots, v_{i_k})$. By induction hypothesis, $\mathrm{discord}_{\lambda(L')}(m_1, m) = \mathbf{p}$, $\mathrm{discord}_{\lambda(L')}(m, m_2) = \mathbf{p}$, so in $\lambda(L')$ there is a chain from r_1 to s, and in $\lambda(L'')$ there is a chain from r to s_2. We conclude that in $\lambda(L)$ there is a chain from r_1 to s_2, i.e., $\mathrm{discord}_{\lambda(L)}(m_1, m_2) = \mathbf{p}$.

This algorithm can be implemented in time $O(n|H|^2)$ as follows: we first construct $\mathcal{E}^\star$ (as shown above, this can be done in time $O(n|H|)$), and then for each $(v, v') \in \mathcal{E}^\star$ we compute the relation $<^*$ for the concatenated MSC $(\lambda(v); \lambda(v'))$ (this can be done in time $O(n|H|^2)$ for all $(v, v') \in \mathcal{E}^\star$ by Corollary 1) and use it to check the discord of all pairs $m_1 \in \lambda(v)$, $m_2 \in \lambda(v')$ (again, by Corollary 1 this takes time $O(n|H|^2)$).

$\mathbf{Discord}(\boldsymbol{H}) = \mathbf{o}.$ The algorithm and the analysis are similar to the previous case. Namely, $\mathrm{Discord}(H) = \mathbf{o}$ if and only if $\mathrm{Discord}(H) \neq \mathbf{p}$ (which can be verified in polynomial time, as described above) and for any $(v, v') \in \mathcal{E}^\star$ and any $m_1 = (s_1, r_1) \in \lambda(v)$, $m_2 = (s_2, r_2) \in \lambda(v')$ we have $\mathrm{discord}_{(\lambda(v);\lambda(v'))}(m_1, m_2) \in \{\mathbf{p}, \mathbf{o}\}$. The running time of this algorithm is $O(n|H|^2)$.

The proof is based on the fact that for any path $L = (v = v_{i_1}, \dots, v_{i_k} = v')$, any (L', L'') such that $L' = (v_{i_1}, \dots, v_{i_j})$, $L'' = (v_{i_j}, \dots, v_{i_k})$ and any $m = (s, r) \in \lambda(v_{i_j})$, if $\mathrm{discord}_{\lambda(L')}(m_1, m) \in \{\mathbf{p}, \mathbf{o}\}$ and $\mathrm{discord}_{\lambda(L'')}(m, m_2) \in \{\mathbf{p}, \mathbf{o}\}$ then $\mathrm{discord}_{\lambda(L)}(m_1, m_2) \in \{\mathbf{p}, \mathbf{o}\}$. To see this, note that $\mathrm{discord}_{\lambda(L')}(m_1, m) \in \{\mathbf{p}, \mathbf{o}\}$ implies that $\lambda(L')$ has chains from s_1 to s and from r_1 to r, and $\mathrm{discord}_{\lambda(L'')}(m, m_2) \in \{\mathbf{p}, \mathbf{o}\}$ implies that $\lambda(L'')$ has chains from s to s_2 and from r to r_2. Hence, in $\lambda(L)$ there are chains from s_1 to s_2 and from r_1 to r_2, i.e., $\mathrm{discord}_{\lambda(L)}(m_1, m_2) \in \{\mathbf{p}, \mathbf{o}\}$.

$\mathbf{Discord}(\boldsymbol{H}) = \mathbf{d}^{-1}.$ The algorithm and the analysis are similar to the previous two cases. Namely, $\mathrm{Discord}(H) = \mathbf{d}^{-1}$ if and only if $\mathrm{Discord}(H) \neq \mathbf{p}, \mathbf{o}$ (which can be verified in polynomial time, as described above) and for any $(v, v') \in \mathcal{E}^\star$ and any $m_1 = (s_1, r_1) \in \lambda(v)$, $m_2 = (s_2, r_2) \in \lambda(v')$ we have $\mathrm{discord}_{(\lambda(v);\lambda(v'))}(m_1, m_2) \in \{\mathbf{p}, \mathbf{o}, \mathbf{d}^{-1}\}$. The running time of this algorithm is $O(n|H|^2)$.

The proof uses the fact that for any path $L = (v = v_{i_1}, \dots, v_{i_k} = v')$, any (L', L'') such that $L' = (v_{i_1}, \dots, v_{i_j})$, $L'' = (v_{i_j}, \dots, v_{i_k})$ and any $m = (s, r) \in \lambda(v_{i_j})$, if $\mathrm{discord}_{\lambda(L')}(m_1, m) \in \{\mathbf{p}, \mathbf{o}, \mathbf{d}^{-1}\}$ and $\mathrm{discord}_{\lambda(L')}(m, m_2) \in \{\mathbf{p}, \mathbf{o}, \mathbf{d}^{-1}\}$ then $\mathrm{discord}_{\lambda(L)}(m_1, m_2) \in \{\mathbf{p}, \mathbf{o}, \mathbf{d}^{-1}\}$. Indeed, $\mathrm{discord}_{\lambda(L')}(m_1, m) \in \{\mathbf{p}, \mathbf{o}, \mathbf{d}^{-1}\}$ implies that $\lambda(L')$ contains a chain from s_1 to s, and $\mathrm{discord}_{\lambda(L'')}(m, m_2) \in \{\mathbf{p}, \mathbf{o}, \mathbf{d}^{-1}\}$ implies that $\lambda(L'')$ contains a chain from s to s_2. Hence, in $\lambda(L)$ there is a chain from s_1 to s_2, i.e., $\mathrm{discord}_{\lambda(L)}(m_1, m_2) \in \{\mathbf{p}, \mathbf{o}, \mathbf{d}^{-1}\}$.

$\mathbf{Discord}(\boldsymbol{H}) = \mathbf{p}^{-1}.$ If $\mathrm{Discord}(H) = \mathbf{p}^{-1}$, there exists a pair of nodes $v, v' \in \mathcal{V}$, a pair of messages $m_1 = (s_1, r_1) \in \lambda(v)$, $m_2 = (s_2, r_2) \in \lambda(v')$ and a path $L = (v = v_{i_1}, \dots, v_{i_k} = v')$ such that $\mathrm{discord}_{\lambda(L)}(m_1, m_2) = \mathbf{p}^{-1}$, i.e., in $\lambda(L)$ there is no chain from s_1 to r_2. Let $C = \lambda(v)$, $C' = \lambda(v')$, and $\bar{C} = \lambda(v_{i_2}, \dots, v_{i_{k-1}})$.

Let s be a maximal send event in $(C; \bar{C})$ such there is a chain from s_1 to s, and let r be the corresponding receive. Set $p = P(s)$, $q = P(r)$. It is easy to see that in L there is no chain from s to r_2, or, equivalently, $(s, r)\mathbf{p}^{-1}m_2$. Therefore, without loss of generality we can assume $m_1 = (s, r)$, i.e., s_1 is a maximal send event in $(C; \bar{C})$. This implies that in $(C; \bar{C})$ there are no send events on p that happen after s_1, and there are no send events on q that happen after r_1 (for any such event, there would be a chain from s_1 to this event). Moreover, in C' there is no chain from any event of p or q to r_2.

This suggests the following algorithm. For each pair $v, v' \in \mathcal{V}$, and each pair of messages $m_1 = (s_1, r_1) \in \lambda(v)$, $m_2 = (s_2, r_2) \in \lambda(v')$ do the following. Set $p = P(s_1)$, $q = P(r_1)$. Let $H(v, v', p, q)$ be the HMSC obtained by deleting from H all nodes other than v, v' that have send events on p or q. Output "yes" if all of the following four conditions hold:

(1) in $\lambda(v)$ there are no send events on p after s_1;
(2) in $\lambda(v)$ there are no send events on q after r_1;

(3) in $\lambda(v')$ there is no chain from any event of p or q to r_2 (in particular, $P(r_2) \neq p, q$);
(4) the HMSC $H(v, v', p, q)$ contains a path from v to v'.

If (1) — (4) are all true, then the pair (m_1, m_2) provides a witness that $\mathrm{Discord}(H) = \mathbf{p}^{-1}$. Conversely, by the reasoning above, if $\mathrm{Discord}(H) = \mathbf{p}^{-1}$, then there is a pair (m_1, m_2) that satisfies (1) — (4).

The running time of this algorithm can be bounded by $O(|H|^3)$. To see this, note that there are $O(|H|^2)$ pairs of messages $m_1 \in \lambda(v)$, $m_2 \in \lambda(v')$. For each such pair, conditions (1) — (3) can be verified in time $O(|H|)$ assuming that the relation $<^*$ for $\lambda(v')$ has been precomputed (by Corollary 1, we can precompute $<^*$ for all MSCs that appear in H in time $O(|H|^2)$). Condition (4) corresponds to solving a single instance of reachability problem, so it can be checked in time $O(|H|)$ as well.

We can change the order of operations so that the algorithm runs in time $O(|\mathcal{P}|^2|H|^2)$. This is more efficient if $|\mathcal{P}|^2 < |H|$, which is likely to be the case in practice. First, we compute the transitive closure of each MSC in H; by Corollary 1, this can be done in time $O(|H|^2)$. Then for each $v \in \mathcal{V}$, each event e in $\lambda(v)$, and each $p \in \mathcal{P}$, we use the information about the transitive closure to check whether in $\lambda(v)$ there is a chain from any event of p to e. There are $O(|H|)$ events, $|\mathcal{P}|$ processes, and for each pair (p, e), $e \in E_i$, this computation takes $O(|H|)$ steps, so this can be done in time $O(|\mathcal{P}||H|^2)$.

Next, fix a pair $p, q \in \mathcal{P}$ and set $V^0(p, q) = \{v \in \mathcal{V} \mid \lambda(v)$ has no send events on $p, q\}$. Consider a modified version of depth-first search on $\mathcal{G}$ that backtracks as soon as it reaches a node in $\mathcal{V} \setminus V^0(p, q)$. This algorithm discovers a path from v to v' if and only if the HMSC $H(v, v', p, q)$ contains a path from v to v'. From any given v, it runs in time $O(|H|)$. For each $v_i \in \mathcal{V}$ we will do the following. Find the last send event on p, identify the corresponding receive and check whether it is on q and there are no send events on q after it. This can be done in time $O(|H|)$. Then we run from v_i the modified version of the depth-first search described above. For any v_j discovered during this search and for each receive event of $C_j = \lambda(v_j)$, we check if it is not reachable from any event of p or q using the precomputed information.

For each triple (p, q, v), we traverse each edge of $\mathcal{E}$ at most twice, and do a constant-time computation for each event of H. Hence, the computation that has to be done for each triple (p, q, v) takes $O(|H|)$ steps, and the total running time of our algorithm is $O(|\mathcal{P}||H|^2 + |\mathcal{P}|^2 n|H|) = O(|\mathcal{P}|^2|H|^2)$, as claimed.

$\mathbf{Discord(\boldsymbol{H}) = o^{-1}.}$ Suppose $\mathrm{Discord}(H) = \mathbf{o}^{-1}$. Then there exists a pair of nodes $v, v' \in \mathcal{V}$, a pair of messages $m_1 = (s_1, r_1) \in \lambda(v)$, $m_2 = (s_2, r_2) \in \lambda(v')$ and a path $L = (v = v_{i_1}, \ldots, v_{i_k} = v')$ such that $\mathrm{discord}_{\lambda(L)}(m_1, m_2) = \mathbf{o}^{-1}$, i.e., in $\lambda(L)$ there is a chain from s_1 to r_2, but no chain from s_1 to s_2 and no chain from r_1 to r_2. Let $C = \lambda(v)$, $C' = \lambda(v')$, and $\bar{C} = \lambda(v_{i_2}, \ldots, v_{i_{k-1}})$.

Observe that in $(C; \bar{C})$ there is no chain from r_1 to any send event s. Indeed, suppose such a chain exists, and let r be the receive that corresponds to this send. If in $\lambda(L)$ there is no chain from s to r_2, we would have $(s, r)\mathbf{p}^{-1}(s_1, r_2)$, a contradiction. On the other hand, a chain from r_1 to s together with a chain from s to r_2 gives a chain from r_1 to r_2 in $\lambda(L)$, a contradiction again. By a similar argument, in $(\bar{C}; C')$ there is no chain from any receive event r to s_2.

Set $p = P(r_1)$, $q = P(s_2)$. It follows that in C there are no send events on p after r_1, in C' there are no receive events on q before s_2, and in $\bar{C}$ there are no sends on p and no receives on q. Obviously, in C there is no chain from s_1 to any event of q, and in C' there is no chain from any event of p to r_2. Moreover, it cannot be the case that $p = q$, $q = P(s_1)$ or $p = P(r_2)$.

Consequently, we have the following algorithm for checking whether $\mathrm{Discord}(H) = \mathbf{o}^{-1}$. First check that $\mathrm{Discord}(H) \neq p^{-1}$. Then for each pair $v, v' \in \mathcal{V}$, and each pair of messages $m_1 = (s_1, r_1) \in \lambda(v)$, $m_2 = (s_2, r_2) \in \lambda(v')$ do the following. Set $p = P(r_1)$, $q = P(s_2)$. Let $H(v, v', p, q)$ be the HMSC obtained by deleting from H all nodes other than v and v' that have send events on p or receive events on q. Output "yes" if the following six conditions hold:

(1) we have $p \neq q$, $q \neq P(s_1)$, $p \neq P(r_2)$;
(2) in C there are no send events on p after r_1;
(3) in C' there are no receive events on q before s_2;
(4) in C there is no chain from s_1 to any event of q;
(5) in C' there is no chain from any event of p to r_2;
(6) the HMSC $H(v, v', p, q)$ contains a path from v to v'.

Suppose that for some $v, v' \in \mathcal{V}$, $m_1 \in \lambda(v)$, $m_2 \in \lambda(v')$ (1) — (6) are all true. By (6), there exists a path $L = (v = v_{i_1}, \ldots, v_{i_k} = v')$ in $H(v, v', p, q)$. Set $\lambda(v) = C$, $\lambda(v') = C'$, $\bar{C} = \lambda(v_{i_2}, \ldots, v_{i_{k-1}})$. Suppose that $\lambda(L)$ contains a chain from s_1 to s_2. As $q \neq P(s_1), p$, this chain must contain a receive event on q. By (3), there is no such event in C', and by construction of $H(v, v', p, q)$, there can be no such event in $\bar{C}$. Finally, by (4) there is no such event in C. Hence, in $\lambda(L)$ there is no chain from s_1 to s_2. Similarly, a chain from r_1 to r_2 must contain a send event on p, and there is no such event in C (by (2)), C' (by (5)), or $\bar{C}$ (by construction of $H(v, v', p, q)$). Hence, the pair (m_1, m_2) provides a witness that $\mathrm{Discord}(H) = \mathbf{o}^{-1}$ Conversely, by the reasoning above, if for some pair (m_1, m_2) we have $\mathrm{discord}_H(m_1, m_2) = \mathbf{o}^{-1}$, then our algorithm will succeed. As in the previous case, this algorithm can be implemented in time $O(|H|^3)$ or, by changing the order of operations, in $O(|\mathcal{P}|^2|H|^2)$.

7 Conclusions

We proposed using Allen's logic for detecting and measuring message order discrepancy in HMSCs. We believe that Allen's logic can be a versatile tool for other message order-related problems in MSCs and HMSCs, such as, e.g., race conditions and message overtake. Allen's logic is very well studied from algorithmic perspective [12]; while in this paper we did not use these results, they may be very useful for other applications of Allen's logic for message order analysis.

We introduced the notion of discord, which measures the difference between the message order in an HMSC and the "ideal" message order for that HMSC. We have shown a coNP-hardness result for computing the discord of a pair of messages in an HMSC, as well as polynomial-time algorithms for restricted versions of this problem. In contrast, we showed how to find the worst-case discord of an HMSC in polynomial time. We believe that the concept of discord will be useful in avoiding design errors in HMSCs. In particular, it can be applied when one wants to partition a large HMSC into smaller components: one should prefer partitions with small discord. Finally, consider an MSC-based programming approach such as the “play-in, play-out” framework of [10], which practically assumes synchronous MSC concatenation. Calculating discords allows one to quantify the potential for relaxing the synchronization assumption and check for possible hazards. This may increase concurrency and efficiency of the implementation and thus can be useful in protocol design.

References

1. James F. Allen, Maintaining Knowledge about Temporal Intervals. *Communications of ACM*, vol. 26:11, pp. 832–843, 1983.
2. Rajeev Alur, Gerard Holzmann, Doron Peled, An Analyzer for Message Sequence Charts. *Software — Concepts and Tools* 17, pp. 70–77, 1996.
3. R. Alur, K. Etessami, M. Yannakakis, Realizability and Verification of MSC Graphs. *Theoretical Computer Science* 331(1): pp. 97–114, 2005.
4. H. Ben-Abdallah, S. Leue, Syntactic Detection of Process Divergence and Non-local Choice in Message Sequence Charts. In *TACAS'97*, LNCS 1217, pp. 259–274, 1997
5. D. Brand and P. Zafiropulo, On Communicating Finite-State Machines. *Journal of the ACM*, 30(2), pp. 323–342, 1983.
6. W. Damm, D. Harel, LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design* 19(1), pp. 45–80, 2001.
7. E. Elkind, B. Genest, D. Peled, Detecting Races in Ensembles of Message Sequence Charts, In *TACAS'07*, LNCS 4424, 2007.
8. R. W. Floyd, Algorithm 97 (Shortest Path), Communications of the ACM 1962, 356.
9. B. Genest, M. Minea, A. Muscholl and D. Peled. Specifying and Verifying Partial Order Properties using Template MSCs. In *FOSSACS'04*, LNCS 2987, pp. 195–210, 2004.
10. D. Harel, R. Marelly, Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Springer Verlag, 2003.
11. ITU Z120 standard recommendation, 1996.
12. A. Krokhin, P. Jeavons, P. Jonsson, Reasoning about Temporal Relations: The Tractable Subalgebras of Allen's Interval Algebra. J. ACM 50(5), pp. 591–640, 2003.
13. M. Lohrey and A. Muscholl. Bounded MSC communication. *Information and Computation* 189, pp. 160–181, 2004.
14. A. Muscholl, D. Peled. Message Sequence Graphs and Decision Problems on Mazurkiewicz Traces. In *MFCS'99*, pp. 81–91, 1999.
15. D. Peled. Specification and Verification of Message Sequence Charts. In *FORTE'00*, IFIP CP 183, pp. 139-154, 2000.
16. S. Warshall, A Theorem on Boolean Matrices. *Journal of the ACM* 9(1), pp. 11–12, 1962.