Tagging make local testing of message-passing systems feasible*

Puneet Bhateja and Madhavan Mukund
Chennai Mathematical Institute, Chennai, India

E-mail: {puneet,madhavan}@cmi.ac.in

Abstract

The only practical way to test distributed message-passing systems is to use local testing. In this approach, used in formalisms such as concurrent TTCN-3, some components are replaced by test processes. Local testing consists of monitoring the interactions between these test processes and the rest of the system and comparing these observations with the specification, typically described in terms of message sequence charts. The main difficulty with this approach is that local observations can combine in unexpected ways to define implied scenarios not present in the original specification. Checking for implied scenarios is known to be undecidable for regular specifications, even if observations are made for all but one process at a time. We propose an approach where we append tags to the messages generated by the system under test. Our tags are generated in a uniform manner, without referring to or influencing the internal details of the underlying system. These enriched behaviours are then compared against a tagged version of the specification. Our main result is that detecting implied scenarios becomes decidable in the presence of tagging.

1 Introduction

We consider the problem of testing whether a message-passing system conforms to its specification. Our focus is to check the patterns of communication that arise between the different components of the system. These interaction scenarios are typically specified using Message Sequence Charts.

Message Sequence Charts (MSCs) [9] are an appealing visual formalism that are used in a number of software engineering notational frameworks such as SDL [14] and UML [6]. A collection of MSCs is used to capture the scenarios that a designer might want the system to exhibit (or avoid).

A standard way to generate a set of MSCs is via Hierarchical (or High-level) Message Sequence Charts (HMSCs) [11]. Without losing expressiveness, we consider only a subclass of HMSCs called Message Sequence Graphs (MSGs). An MSG is a finite directed graph in which each node is labeled by an MSC. An MSG defines a collection of MSCs by concatenating the MSCs labeling each path from an initial vertex to a terminal vertex.

A natural way to test a distributed implementation against an MSG specification is to substitute test processes for one or more components and record the interactions between the test process(es) and the rest of the system. We refer to this form of testing distributed message-passing systems as *local testing*. The implementation is said to pass a local test if the observations at the test process(es) are consistent with the MSG specification. Local testing is the methodology adopted in formalisms such as concurrent TTCN-3 [16] that are used to specify test suites for telecommunication applications.

An important impediment to local testing is the possibility of implied scenarios. Let $T = \{P_1, P_2, \ldots, P_k\}$ be a collection of subsets of processes. We say that an MSC M is T-implied by an MSC language $\mathcal{L}$ if the projections of M onto each subset $P_i \in T$ agree with the projections onto P_i of some good MSC $M_{P_i} \in \mathcal{L}$. Implied scenarios have been studied in [3, 4], where the observations are restricted to individual processes rather than arbitrary subsets.

Let T_k denote the set of all subsets of processes of size k. We say that an MSC language $\mathcal{L}$ is *k-testable* if every T_k-implied scenario is already present in $\mathcal{L}$. In other words, if a specification is k-testable, it is possible to accurately test an implementation by performing a collection of local tests with respect to T_k. On the other hand, if $\mathcal{L}$ is not k-testable, even an exhaustive set of local tests with respect to T_k cannot rule out an undesirable implied scenario.

*Partially supported by *Timed-DISCOVERI*, a project under the Indo-French Networking Programme, as well as by a grant from Tata Consultancy Services.

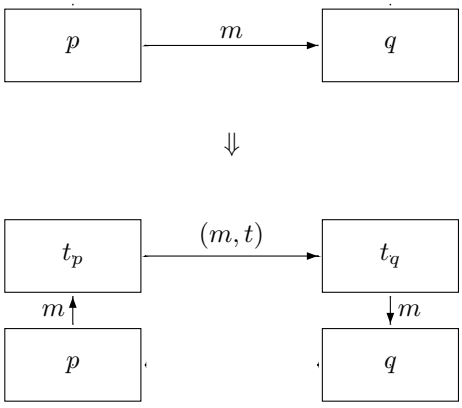

Figure 1. Adding a transport layer to tag messages

It has been shown in [3] that 1-testability is undecidable, even for regular MSG specifications. This result has been extended in [5] to show that for any n, k-testability of an MSG specification with n processes is undecidable, for all $k \in \{1, 2, \ldots, n-1\}$.

To get around this negative result, we propose a framework where the system behaviour is augmented using tags. The tagging mechanism is intended to run in parallel with the system under test. This can be implemented as an additional transport layer in the network that encapsulates all communication between the components in the system under test, as shown in Figure 1.

The observed behaviours of the tagged system are compared to an enhanced specification that incorporates the same set of tags. Our main result is that tagging using local time-stamps of the type described in [12] makes 1-testability (and hence k-testability for any k) decidable. Moreover these time-stamps can be generated deterministically and locally on-the-fly, making it easy to construct a test environment into which the system under test can be embedded.

The local time-stamping algorithm proposed in [12] has been applied to construct effective algorithms for message-passing systems in other contexts [1, 8]. However, the important distinction between our use of time-stamps for tagging and these earlier applications is that our tagging mechanism runs independently of the system being tested. In these earlier applications of time-stamping, the time-stamping mechanism is tightly coupled with the original system so that state information is incorporated into the time-stamps and, at the same time, time-stamps are used to constrain the behaviour of the system. In contrast, our tagging protocol is only loosely coupled with the system under test and does not refer to or modify in any way the underlying behaviour.

As mentioned earlier, the problem of implied scenarios was first identified in [3]. Their aim was to look for simple realizations of MSC specifications in terms of communicating finite-state machines, by constructing each component as a projection of the corresponding line in the MSC. The realizability problem corresponds to 1-testability in our framework and was shown to be undecidable for regular MSG specifications in [4]. However, the *safe* realizability problem, which asks for deadlock-free implementations, was shown to be decidable for regular MSG specifications in [4]. Our result in this paper can also be viewed as a solution to the realizability problem that lies in between the naïve approach of [3] and the fully general construction for regular MSC languages described in [8]. In particular, our construction can be viewed as a variant of causal realizability, studied in [1].

The paper is organized as follows. We begin with preliminaries about MSCs, before we formally define k-testability in Section 3. The next section establishes our main result about the decidability of 1-testability in the presence of bounded time-stamping. In Section 5, we describe how to interpret our results in the context of realizability. We conclude with a brief discussion.

2 Preliminaries

2.1 Message sequence charts

Let $\mathcal{P} = \{p, q, r, \ldots\}$ be a finite set of processes (agents) that communicate with each other through messages via reliable FIFO channels using a finite set of message types $\mathcal{M}$. For $p \in \mathcal{P}$, let $\Sigma_p = \{p!q(m), p?q(m) \mid p \neq q \in \mathcal{P}, m \in \mathcal{M}\}$ be the set of communication actions in which p participates. The action $p!q(m)$ is read as *p sends the message m to q* and the action $p?q(m)$ is read as *p receives the message m from q*. We set $\Sigma = \bigcup_{p \in \mathcal{P}} \Sigma_p$. We also denote the set of *channels* by $Ch = \{(p, q) \in \mathcal{P}^2 \mid p \neq q\}$.

Labelled posets A Σ-labelled poset is a structure $M = (E, \leq, \lambda)$ where $(E, \leq)$ is a partially ordered set and $\lambda : E \to \Sigma$ is a labelling function that extends, as usual, to sequences of events. For $e \in E$, let $\downarrow e = \{e' \mid e' \leq e\}$. We refer to $\downarrow e$ as the *causal past* of e. For $p \in \mathcal{P}$ and $a \in \Sigma$, we set $E_p = \{e \mid \lambda(e) \in \Sigma_p\}$ and $E_a = \{e \mid \lambda(e) = a\}$, respectively. For $(p, q) \in Ch$, we

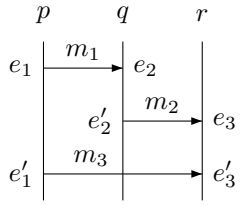

Figure 2. An MSC

define the relation $<_{pq}$:

$$e <_{pq} e' \stackrel{\text{def}}{=} \exists m \in \mathcal{M} \text{ such that } \lambda(e) = p!q(m),\ \lambda(e') = q?p(m) \text{ and } |{\downarrow}e \cap E_{p!q(m)}| = |{\downarrow}e' \cap E_{q?p(m)}|$$

The relation $e <_{pq} e'$ says that channels are FIFO with respect to *each message*—if $e <_{pq} e'$, the message m read by q at e' is the one sent by p at e.

Finally, for each $p \in \mathcal{P}$, we define the relation $\leq_{pp} = (E_p \times E_p) \cap \leq$, with $<_{pp}$ standing for the largest irreflexive subset of $\leq_{pp}$.

Definition 1 *An MSC over $\mathcal{P}$ is a* finite Σ*-labelled poset $M = (E, \leq, \lambda)$ where:*

1. *Each relation $\leq_{pp}$ is a linear (total) order.*
2. *If $p \neq q$ then for each $m \in \mathcal{M}$, $|E_{p!q(m)}| = |E_{q?p(m)}|$.*
3. *If $e <_{pq} e'$, then $|{\downarrow}e \cap \left(\bigcup_{m\in\mathcal{M}} E_{p!q(m)}\right)| = |{\downarrow}e' \cap \left(\bigcup_{m\in\mathcal{M}} E_{q?p(m)}\right)|$.*
4. *The partial order $\leq$ is the reflexive, transitive closure of $\bigcup_{p,q\in\mathcal{P}} <_{pq}$.*

The second condition ensures that every message sent along a channel is received. The third condition says that every channel is FIFO across all messages.

In diagrams, the events of an MSC are presented in *visual order*. The events of each process are arranged in a vertical line and messages are displayed as horizontal or downward-sloping directed edges. Figure 2 shows an example with three processes $\{p, q, r\}$ and six events $\{e_1, e'_1, e_2, e'_2, e_3, e'_3\}$ corresponding to three messages—m_1 from p to q, m_2 from q to r and m_3 from p to r.

For an MSC $M = (E, \leq, \lambda)$, we let $\text{lin}(M) = \{\lambda(\pi) \mid \pi \text{ is a linearization of } (E, \leq)\}$. For instance, $p!q(m_1)\ q?p(m_1)\ q!r(m_2)\ p!r(m_3)\ r?q(m_2)\ r?p(m_3)$ is one linearization of the MSC in Figure 2.

MSC languages An *MSC language* is a set of MSCs. We can also regard an MSC language $\mathcal{L}$ as a word language over Σ given by $\text{lin}(\mathcal{L}) = \bigcup\{\text{lin}(M) \mid M \in \mathcal{L}\}$.

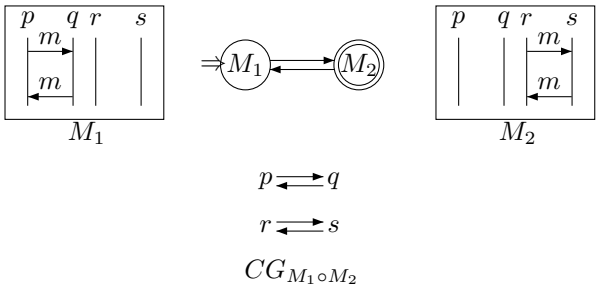

Figure 3. A message sequence graph

Definition 2 *An MSC language $\mathcal{L}$ is said to be a* regular MSC language *if the word language* $\text{lin}(\mathcal{L})$ *is a regular language over* Σ.

Let M be an MSC and $B \in \mathbb{N}$. We say that $w \in \text{lin}(M)$ is B-bounded if for every prefix v of w and for every channel $(p, q) \in Ch$, $\sum_{m\in\mathcal{M}} |\pi_{p!q(m)}(v)| - \sum_{m\in\mathcal{M}} |\pi_{q?p(m)}(v)| \leq B$, where $\pi_\Gamma(v)$ denotes the projection of v on $\Gamma \subseteq \Sigma$. This means that along the execution of M described by w, no channel ever contains more than B messages. We say that M is (universally) B-bounded if every $w \in \text{lin}(M)$ is B-bounded. An MSC language $\mathcal{L}$ is B-bounded if every $M \in \mathcal{L}$ is B-bounded. Finally, $\mathcal{L}$ is bounded if it is B-bounded for some B.

We then have the following result [8].

Theorem 3 *If an MSC language $\mathcal{L}$ is regular then it is bounded.*

2.2 Message sequence graphs

Message sequence graphs (MSGs) are finite directed graphs with designated initial and terminal vertices. Each vertex in an MSG is labelled by an MSC. The edges represent (asynchronous) MSC concatenation, defined as follows.

Let $M_1 = (E^1, \leq^1, \lambda_1)$ and $M_2 = (E^2, \leq^2, \lambda_2)$ be a pair of MSCs such that E^1 and E^2 are disjoint. The *(asynchronous) concatenation* of M_1 and M_2 yields the MSC $M_1 \circ M_2 = (E, \leq, \lambda)$ where $E = E^1 \cup E^2$, $\lambda(e) = \lambda_i(e)$ if $e \in E^i$, $i \in \{1, 2\}$, and $\leq = (\leq^1 \cup \leq^2 \cup \bigcup_{p\in\mathcal{P}} E^1_p \times E^2_p)^*$.

A *Message Sequence Graph* is a structure $\mathcal{G} = (Q, \rightarrow, Q_{in}, F, \Phi)$, where Q is a finite and nonempty set of states, $\rightarrow \subseteq Q \times Q$, $Q_{in} \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states and Φ labels each state with an MSC.

A *path* π through an MSG $\mathcal{G}$ is a sequence $q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow q_n$ such that $(q_{i-1}, q_i) \in \rightarrow$ for $i \in$

$\{1, 2, \ldots, n\}$. The MSC generated by π is $\Phi(\pi) = M_0 \circ M_1 \circ M_2 \circ \cdots \circ M_n$, where $M_i = \Phi(q_i)$. A path $\pi = q_0 \to q_1 \to \cdots \to q_n$ is a *run* if $q_0 \in Q_{in}$ and $q_n \in F$. The language of MSCs accepted by $\mathcal{G}$ is $L(\mathcal{G}) = \{\Phi(\pi) \mid \pi \text{ is a run through } \mathcal{G}\}$. We say that an MSC language $\mathcal{L}$ is *MSG-definable* if there exists an MSG $\mathcal{G}$ such that $\mathcal{L} = L(\mathcal{G})$.

An example of an MSG is depicted in Figure 3. The initial state is marked $\Rightarrow$ and the final state has a double line. The language $\mathcal{L}$ defined by this MSG is *not* regular: $\mathcal{L}$ projected to $\{p!q(m), r!s(m)\}^*$ consists of $\sigma \in \{p!q(m), r!s(m)\}^*$ such that $|\sigma\restriction_{p!q(m)}| = |\sigma\restriction_{r!s(m)}| \geq 1$, which is not a regular string language.

In general, it is undecidable whether an MSG describes a regular MSC language [8]. However, a sufficient condition for the MSC language of an MSG to be regular is that the MSG be *locally synchronized*.

Communication graph For an MSC $M = (E, \leq, \lambda)$, let CG_M, *the communication graph of* M, be the directed graph $(\mathcal{P}, \mapsto)$ where:

- $\mathcal{P}$ is the set of processes of the system.
- $(p, q) \in \mapsto$ iff there exists an $e \in E$ with $\lambda(e) = p!q(m)$.

M is said to be *com-connected* if CG_M consists of one nontrivial strongly connected component and isolated vertices.

Locally synchronized MSGs The MSG $\mathcal{G}$ is *locally synchronized* [13] (or *bounded* [2]) if for every loop $\pi = q \to q_1 \to \cdots \to q_n \to q$, the MSC $\Phi(\pi)$ is com-connected. In Figure 3, $M_1 \circ M_2$ is not com-connected, so the MSG is not locally synchronized. We have the following result for MSGs [2].

Theorem 4 *If $\mathcal{G}$ is locally synchronized, $L(\mathcal{G})$ is a regular MSC language.*

3 Locally testable MSC languages

In local testing, we substitute test process(es) for one or more components and record the interactions between the test process(es) and the rest of the system. The implementation is said to pass a local test if the observations at the test process(es) are consistent with the MSG specification. An important impediment to local testing is the possibility of implied scenarios.

Definition 5 *Let $M = (E, \leq, \lambda)$ be an MSC and $P \subseteq \mathcal{P}$ a set of processes. The* P-observation of M, *$M\restriction_P$, is the collection of local observations $\{(E_p, \leq_{pp})\}_{p \in P}$ — recall that $\leq_{pp} = \leq \cap (E_p \times E_p)$. The collection $\{(E_p, \leq_{pp})\}_{p \in P}$ can also be viewed as a labelled partial order $(E_P, \leq_P)$ where $E_P = \bigcup_{p \in P} E_p$ and $\leq_P = \left(\bigcup_{p,q \in P} <_{pq}\right)^*$.*

Let $T \subseteq 2^{\mathcal{P}}$ be a family of subsets of processes. An MSC M is said to be T-implied *by an MSC-language $\mathcal{L}$ if for every subset $P \in T$ there is an MSC $M_P \in \mathcal{L}$ such that $M_P\restriction_P = M\restriction_P$.*

We denote by T_k the set $\{P \subseteq \mathcal{P} \mid |P| = k\}$ of all subsets of $\mathcal{P}$ of size k and we say that an MSC is k-implied if it is T_k-implied.

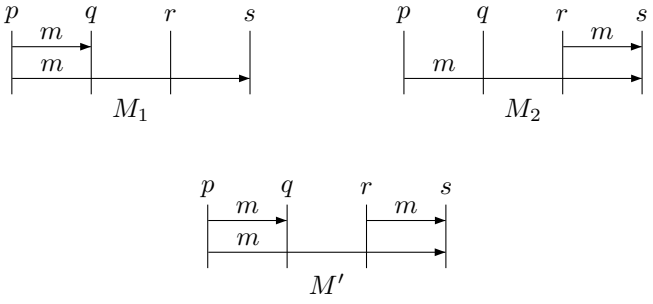

Figure 4. An example of implied scenarios

Figure 4 illustrates the idea of implied scenarios. The MSC M' is 1-implied by $\{M_1, M_2\}$. However, M' is not 2-implied by $\{M_1, M_2\}$ because the $\{p, s\}$-observation of M' does not match either M_1 or M_2.

We are interested in checking the global behaviour of a distributed implementation by testing it locally against an MSG specification. For this to be meaningful, the MSG should be closed with respect to implied scenarios generated by the test observations. This leads to the following definition.

Definition 6 *Let $|\mathcal{P}| = n$. For $k < n$, an MSG $\mathcal{G}$ is said to be k-testable if every scenario M that is k-implied by $L(\mathcal{G})$ is already a member of $L(\mathcal{G})$.*

We have the following negative result from [4, 5].

Theorem 7 *Let $\mathcal{G}$ be a locally-synchronized MSG, so that $L(\mathcal{G})$ is a regular MSC language over n processes. For any $k < n$, it is undecidable whether $L(\mathcal{G})$ is k-testable.*

The root cause of this undecidability is the fact that even when a MSC language $\mathcal{L}$ is regular, and hence B-bounded for some B, the set of scenarios k-implied by $\mathcal{L}$ may not be bounded. An example when k is 1 is shown in Figure 5—all messages are labelled m and labels are omitted.

Since M_1 and M_2 are both com-connected, the language $(M_1 + M_2)^*$ is a regular MSG-definable language.

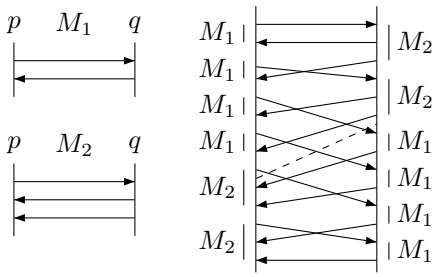

Figure 5.

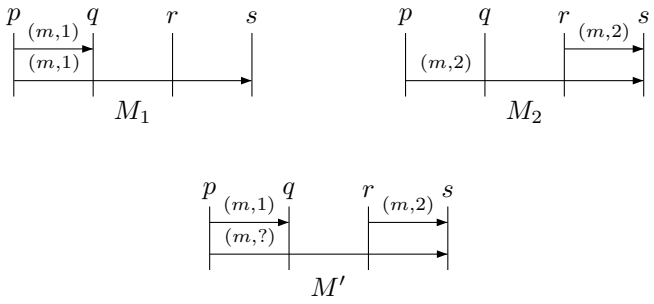

Figure 6. A simple tagging scheme

On the other hand, for each $k \in \mathbb{N}$, the MSC in which the p-observation matches $M_1^{2k}M_2^k$ and the q-observation matches $M_2^kM_1^{2k}$ has a global cut where the channel (p, q) has capacity $k+1$. The figure shows the case $k = 2$. The dotted line marks the global cut where the channel (p, q) has maximum capacity.

4 Local testing with tagging

Our proposal for getting around the negative results from [4, 5] is to tag the messages generated by the MSG specification. Tagging can be done in many different ways. The simplest is to append a label to each message identifying the node in the MSG to which it belongs. For instance, the implied scenario in the example shown in Figure 4 disappears with such a tagging scheme, as shown in Figure 6. In the tagged setting, we cannot assign a tag to the message sent from p to s in M' that is consistent with both M_1 and M_2.

This kind of tagging does not completely eliminate implied scenarios, as shown in Figure 7. In this example, an implied scenario arises because processes p and q traverse the left loop while r and s independently traverse the right loop.

A serious shortcoming of the simple tagging scheme is that it cannot be easily implemented to run alongside the system under test. To generate node based tags on the fly, we have to implicitly simulate the MSG while running the implementation and nondeterministically

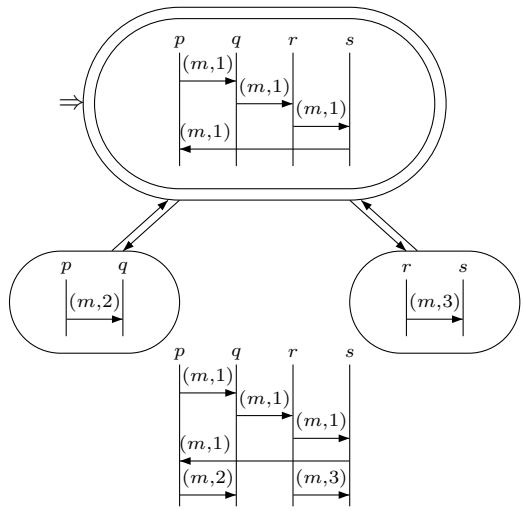

Figure 7. Implied scenario with simple tags

guess when to switch labels. As we shall see, it is important that the tagging scheme used during testing is local and deterministic.

Instead, we propose a more elaborate tagging scheme that uses a (bounded) time-stamping mechanism—for example, the algorithm described in [12]. Let $\mathcal{G}$ be an MSG and $M \in L(\mathcal{G})$. We append to each message m that is sent in M a time-stamp t generated according to the bounded time-stamping protocol described in [12]. This time-stamp assigns a new label to the current send event and records a finite amount of information about its causal past. Using these time-stamps, we can disambiguate copies of m sent in different contexts which would otherwise lead to implied scenarios. Though, again, this does not completely eliminate the possibility of implied scenarios, it does make the problem of detecting implied scenarios decidable. Also, using time-stamping, we can implement an effective procedure for local testing of an implementation against the MSG specification. We begin by recalling the salient features of the bounded time-stamping protocol described in [12].

4.1 Bounded time-stamping

Let $M = (E, \leq, \lambda)$ be an MSC and e an event in E. We say that a send event e' is *unacknowledged* at e if $e' \in \downarrow e$ is a send event for some message m but the matching receive event for m lies outside $\downarrow e$. We also say that m is an *unacknowledged message* at e. Note that this definition does not require m to be sent by the same process that executes e.

Proposition 8 *An MSC $M = (E, \leq, \lambda)$ is B-bounded*

if and only if there can never be more than B unacknowledged messages from p to q in $\downarrow e$, for any $p, q \in \mathcal{P}$ and $e \in E$.

Proof: Suppose the channel (p, q) can contain B+1-messages. We can identify a prefix of M in which these B+1 messages have been sent but not received. In this prefix, there are B+1 unacknowledged messages from p to q.

Conversely, suppose there is a prefix of M with B+1 unackowledged messages from p to q. Let e_p be the maximum p-event in this prefix. In the partial computation $\downarrow e_p$, there are B+1 messages in the channel from p to q. □

The time-stamping protocol in [12] is described in terms of a deterministic message-passing automaton (MPA). An MPA $\mathcal{A} = \{\mathcal{A}_p\}_{p \in P}$ consists of a collection of automata. Each component is a finite state automaton $\mathcal{A}_p = (S_p, \Sigma_p, \rightarrow_p, S_p^{in}, S_p^f)$. The local transitions of $\mathcal{A}_p$ are of the form $s \xrightarrow{a} s'$ where a is either a send action $p!q(m)$ or a receive action $p?q(m)$. To describe the behaviour of an MPA, we associate a FIFO channel (p, q) with each pair of distinct processes $p, q \in \mathcal{P}$. A send action $p!q(m)$ appends m to the channel (p, q) whereas a receive action $p?q(m)$ consumes the message at the head of the channel (p, q), if the channel is nonempty, and blocks otherwise. A more formal description of MPAs can be found in [8].

Let $\mathcal{A}^T$ be the MPA corresponding to the time-stamping protocol in [12]. The automaton $\mathcal{A}^T$ has the following properties.

- Every send event e is assigned a label ℓ_e from a set of labels whose size is bounded. This bound depends only on the number of processes in the system and the bound on the channel size and is independent of the length of the computation. The label ℓ_e forms part of the time-stamp t_e sent at e.
- The time-stamp t_e sent at any event e includes the labels of all the *unacknowledged* send events in $\downarrow e$. Note that this set is bounded if the channels are B-bounded.
- The time-stamp t_e sent at any event e additionally includes the labels of a bounded set of send events in $\downarrow e$. These extra labels are used by the receiving process to decide whether the information in t_e is "fresher" than its existing knowledge, in which case it updates its local state with the information in t_e.
- The automaton $\mathcal{A}^T$ is deterministic.
  - If the p-component $\mathcal{A}_p^T$ of $\mathcal{A}^T$ is in state s and it receives a time-stamp t, the new state s' assumed by $\mathcal{A}_p^T$ is uniquely determined by s and t.
  - If the p-component $\mathcal{A}_p^T$ of $\mathcal{A}^T$ is in state s and it sends a time-stamp t, the content of t is uniquely determined by s. Moreover, $\mathcal{A}_p^T$ moves to a new state s' that is also uniquely determined by s.

4.2 Time-Stamped unfolding of an MSG

Given an MSG $\mathcal{G} = (Q, \rightarrow, Q_{in}, F, \Phi)$, we construct a time-stamped MSG $\mathcal{G}^T = (Q^T, \rightarrow^T, Q_{in}^T, F^T, \Phi^T)$ by unfolding $\mathcal{G}$ so that each message is tagged with the appropriate time-stamp, as generated by the time-stamping automaton $\mathcal{A}^T$. Each state $\bar{q} \in Q^T$ is a path over Q in $\mathcal{G}$. The MSC associated with $q_0 q_1 \ldots q_j$ in $\mathcal{G}^T$ is a time-stamped version of the MSC associated with q_j in $\mathcal{G}$, with time-stamps as generated by $\mathcal{A}^T$ after the sequence $q_0 q_1 \ldots q_{j-1}$. We cut off a path in the unfolding whenever we generate two copies of an MSC corresponding to the same node q in $\mathcal{G}$ with identical time-stamps. Formally, we construct $\mathcal{G}^T$ inductively.

- $Q_{in}^T = \{q \mid q \in Q_{in}\}$. For each $q \in Q_{in}^T$, $\Phi^T(q) = \Phi(q)^T$, where $\Phi(q)^T$ is a time-stamped version of $\Phi(q)$ in which the time-stamps are generated assuming that the time-stamping automaton $\mathcal{A}^T$ was in its initial state at the beginning of $\Phi(q)$.
- Let $\bar{q} = q_0 q_1 \ldots q_j \in Q^T$ and let $q_j \rightarrow \hat{q}$ in $\mathcal{G}$, with $\widehat{M} = \Phi(\hat{q})$.

  Define $\widehat{M}^T$ to be the time-stamped version of $\widehat{M}$, assuming that $\mathcal{A}^T$ begins assigning time-stamps to $\widehat{M}$ starting from the global state $\{s_p\}_{p \in \mathcal{P}}$ after the MSC $\Phi^T(q_0) \circ \Phi^T(q_1) \circ \cdots \circ \Phi^T(q_j)$. (Note that after any sequence of complete MSCs, all messages that have been sent have been received, so the configuration reached by $\mathcal{A}^T$ will have all channels empty.)

  If $\hat{q} = q_i$ for some $i \in \{0, 1, \ldots, j\}$ and $\widehat{M}^T = \Phi^T(q_0 q_1 \ldots q_i)$, then add the edge $q_0 q_1 \ldots q_j \rightarrow^T q_0 q_1 \ldots q_i$ in $\mathcal{G}^T$. This corresponds to cutting off the current path.

  Otherwise, add a new node $q_0 q_1 \ldots q_j \hat{q}$ to $\mathcal{G}^T$, add an edge $q_0 q_1 \ldots q_j \rightarrow^T q_0 q_1 \ldots q_j \hat{q}$ and set $\Phi^T(q_0 q_1 \ldots q_j \hat{q}) = \widehat{M}^T$.

Since $\mathcal{A}^T$ uses a bounded set of time-stamps, we can only generate a bounded number of distinct time-stamped copies of any MSC M that appears in $\mathcal{G}$.

Hence, every path in $\mathcal{G}^T$ will be cut off at a finite depth, so $\mathcal{G}^T$ is a finite MSG. Moreover, any loop $\hat{\pi}$ in the unfolded MSG $\mathcal{G}^T$ also corresponds to a loop π in the original MSG $\mathcal{G}$. The communication graphs $CG_{\Phi^T(\hat{\pi})}$ and $CG_{\Phi(\pi)}$ are identical, so $\mathcal{G}^T$ is also locally synchronized and $L(\mathcal{G}^T)$ is a regular MSC language.

In many cases, time-stamps directly rule out implied scenarios—for instance, the example shown in Figure 4 no longer constitutes an implied scenario. The time-stamps generated for the two instances of the message $p!s(m)$ in M_1 and M_2 are different, and this will result in an inconsistent view for s in M'.

The time-stamp generated at a send event e is determined by its causal past $\downarrow e$. For an MSC M and a process p, p's causal view of M is $\downarrow e_p$, where e_p is p's last event in M. An MSC M is causally implied by an MSC language L if for every procees p, p's causal view of M matches p's causal view of some MSC M_p in L. Any causally implied scenario is also an implied scenario in our setting. However, since time-stamps are generated only at send events and not at receive events, a scenario may be implied in our setting without being causally implied.

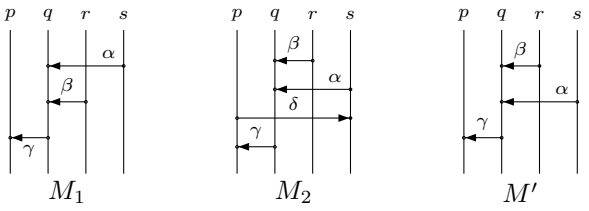

Figure 8. An implied scenario with time-stamps

In the example shown in Figure 8, all messages are implicitly labelled m. The messages from r to q and s to q have the same time-stamps in M_1 and M_2 because they constitute the first messages sent by r and s. To see why the message from q to p is labelled γ in both M_1 and M_2, we note that the time-stamping protocol in [12] only records information about the relative ordering of time-stamped events. Thus, the time-stamp generated by q only depends on the fact that q has received two messages labelled α and β sent independently, and *not* on the order in which these messages were received. From this, it follows that the MSC M' is implied by M_1 and M_2. For p and s, M' looks like M_1 while for q, M' looks like M_2, and for r both M_1 and M_2 are compatible with M'. We can enhance the time-stamping protocol to also record the order in which messages are received, which will rule out this implied scenario.

Note that, in practice, we do not actually have to explicitly construct $\mathcal{G}^T$ in order to perform local testing using time-stamps as tags. Also, we do not need to explicitly compute the upper bound B on the channel capacities, even though this can be done. We will address these points in Section 4.4.

4.3 Checking 1-testability of $\mathcal{G}^T$

We now consider the problem of determining whether $\mathcal{G}^T$, the time-stamped version of $\mathcal{G}$, is 1-testable. In other words, given a time-stamped MSG $\mathcal{G}^T$, is every MSC M^T that is 1-implied by $L(\mathcal{G}^T)$ already present in $L(\mathcal{G}^T)$? Henceforth, we just write implied scenario to mean a 1-implied scenario.

Our first observation is that the time-stamps that appear in an implied scenario are, in fact, consistent with the underlying MSC without time-stamps.

Lemma 9 *Let M be an implied scenario for $\mathcal{G}^T$. Let $\widehat{M}$ be the corresponding MSC in which the time-stamps have been stripped and let $\widehat{M^T}$ be the version of $\widehat{M}$ after adding time-stamps according to the time-stamping protocol of [12]. Then, M is identical to $\widehat{M^T}$.*

Proof: The result follows from the fact that the time-stamping MPA $\mathcal{A}^T$ is deterministic. For each p, let M_p be the witnessing MSC in $L(\mathcal{G}^T)$ such that $M\restriction_p = M_p\restriction_p$. Let $\widehat{M^T} = (E^T, \leq^T, \lambda_T)$, with $e \in E^T$ a p-event. Then, e corresponds to a p-event e_p in M_p. We show by induction on the size of $\downarrow e$ that the state of $\mathcal{A}^T_p$, the p-component of $\mathcal{A}^T$, before and after e in $\widehat{M^T}$ is the same as the state of $\mathcal{A}^T_p$ before and after e_p in M_p. Since $\mathcal{A}^T$ is deterministic, this also implies that the time-stamp generated with each message is the same in $\widehat{M^T}$ as in M_p, and hence the same as in M.

If $\downarrow e = \{e\}$, then $\downarrow e_p = \{e_p\}$ as well, and both must be send events with the same label $\lambda_T(e)$. In both cases, $\mathcal{A}^T_p$ is in its initial state before the event, so the time-stamp generated for the message sent at e and e_p is the one determined by this initial state, and the new state after the event is uniquely specified.

Let e be a p-event such that $|\downarrow e| = k{+}1$, where the induction hypothesis holds for all events e' such that $|\downarrow e'| \leq k$.

If e is a send event, let e^- be the previous p event. (There must be such an event, otherwise $\downarrow e = \{e\}$.) Inductively, the state of $\mathcal{A}^T_p$ after e^- in $\widehat{M^T}$ is the same as the state of $\mathcal{A}^T_p$ after the corresponding event e_p^- in M_p. Since $\mathcal{A}^T$ is deterministic, it is clear that the time-stamp and state change in $\widehat{M^T}$ will be the same as in M_p.

If e is a receive event, we can again use the induction hypothesis to argue that the state of $\mathcal{A}_p^T$ is the same before e as it is before the corresponding event e_p in M_p. Let f be the send event on some process q corresponding to the message received at e. Since $\downarrow f \subsetneq \downarrow e$, we can apply the induction hypothesis to conclude that the time-stamp generated at f is the same as the one generated at the corresponding event f_q in M_q. Thus, the time-stamp received by p in $\widehat{M_T}$ is the same as the one received in M and hence the same as the time-stamp received in M_p. This means that the new state of $\mathcal{A}_p^T$ after e in $\widehat{M^T}$ is the same as the state of $\mathcal{A}_p^T$ after e_p in M_p.

□

A corollary of this lemma is that in any implied scenario M^T generated by $\mathcal{G}^T$, no channel exceeds the bound B associated with $\mathcal{G}$.

Corollary 10 *All implied scenarios of $\mathcal{G}^T$ are B-bounded, where B is the bound associated with the locally synchronized MSG $\mathcal{G}$.*

Proof: Suppose M^T is an implied scenario in which there are more than B messages in some channel, say (p, q). When p sends message $B{+}1$ in this sequence, the time-stamp associated with this message will indicate that there are $B{+}1$ unacknowledged messages on this channel. This time-stamp cannot be consistent with any time-stamp in $\mathcal{G}^T$ since $\mathcal{G}^T$ never has more than B unacknowledged messages, which contradicts the assumption that M^T is an implied scenario. □

The key feature that makes the preceding Lemma and Corollary go through is that the time-stamping scheme of [12] is local and deterministic. Locally deterministic components can combine together in only one way. It is clearly not essential that we use exactly the same time-stamping scheme as [12]—we just need a local, deterministic tagging mechanism that also reflects, directly or indirectly, the capacities of the channels.

Following [3, 4], we can construct an MPA $\{\mathcal{A}_P\}_{p\in\mathcal{P}}$ that accepts all the implied scenarios of $\mathcal{G}^T$ as follows: each process $\mathcal{A}_p$ is constructed by projecting $\mathcal{G}^T$ onto p and creating a new state between each pair of consecutive events along p. Let L' be the language of implied scenarios accepted by this automaton. From Corollary 10, it follows that L' is B-bounded for some B, so L' is in fact a regular MSC language. This immediately gives us the following theorem.

Theorem 11 *Given any locally synchronized MSG $\mathcal{G}$, it is decidable whether $\mathcal{G}^T$, the time-stamped version of $\mathcal{G}$, is 1-testable.*

Proof: We know that the MSC language $L(\mathcal{G}^T)$ is regular. On the other hand, from the preceding observations, the MSC language L' of implied scenarios generated by $\mathcal{G}^T$ is also regular. To check if $\mathcal{G}^T$ is 1-testable, we just have to check whether $L' \setminus L(\mathcal{G}^T)$ is empty, which can be done since both are regular MSC languages. □

4.4 Implementing local testing using time-stamps

To locally test a message-passing system (system under test, or SUT) $\mathcal{A}^I$, we can run the SUT in parallel with the time-stamping automaton $\mathcal{A}^T$. Let us denote this compound system $\mathcal{A}^I \parallel \mathcal{A}^T$. The state of each process p in $\mathcal{A}^I \parallel \mathcal{A}^T$ is a pair $\langle s_I, s_T\rangle$ where s_I is the current local state of p in $\mathcal{A}^I$ and s_T is the current local state of p in $\mathcal{A}^T$. If the implementation has a transition $s_I \xrightarrow{a} s_I'$ where $a = p!q(m)$, the compound system sends a message of the form $\langle m, t\rangle$, where $s_T \xrightarrow{p!q(t)} s_T'$ is the time-stamping move for $\mathcal{A}_p^T$ in state s_T. This takes the compound system to a new local state $\langle s_I', s_T'\rangle$. Symmetrically, if the compound system receives a message $\langle m, t\rangle$ from q in state $\langle s_I, s_T\rangle$, the new state of the system will be $\langle s_I', s_T'\rangle$ where $s_I \xrightarrow{p?q(m)} s_I'$ is a local move for p in the SUT and $s_T \xrightarrow{p?q(t)} s_T'$ is the move dictated by $\mathcal{A}_p^T$.

Local testing will observe the behaviour of the compound system at the interface of each process and compare this with the specification $\mathcal{G}^T$. Note that we do *not* have to explicitly construct $\mathcal{G}^T$ in order to perform the test. We can trace out a path through $\mathcal{G}^T$ by generating time-stamps on the fly, just as we do for the SUT.

We also do need to explicitly compute the precise upper bound B on the channel capacity that is enforced by the fact that the MSG $\mathcal{G}$ is locally synchronized. Given that such a bound B exists, we know that $\mathcal{G}^T$ is bounded. Hence, any local observation of $\mathcal{A}^I \parallel \mathcal{A}^T$ will, after a bounded number of steps, either diverge from the specification or hit a loop in $\mathcal{G}^T$. We do not need the value of B to determine when to terminate a test.

As we have emphasized, earlier, we can think of $\mathcal{A}^T$ as a transport layer sitting above $\mathcal{A}^I$ to make local testing possible. The important point is that this transport layer is independent of the SUT and does not require any modifications to be made in the structure of the SUT itself. Also, while it tags the behaviour of the SUT in a deterministic fashion, it does not interfere with the SUT in any way. This is a crucial feature of our approach, since it is not reasonable to expect the

implementation to offer any specific facility for testing, nor it is acceptable to test a restricted system whose behaviour does not match that of the original SUT.

The number of bits in each time-stamp generated by the protocol of [12] is bounded by $O(B^2N^2(\log B + \log N))$, where B is the channel bound and N is the number of processes. This means that the transport layer used to implement local testing adds only polynomially many bits to each message.

5 Realizability

The problem of implied scenarios was first identified in [3] in the context of a problem called called *weak realizability*. Their aim was to look for simple realizations of MSC specifications in terms of communicating finite-state machines, by constructing each component as a projection of the corresponding line in the MSC. Clearly, such an implementation is faithful to the specification if and only if the specification does not admit any implied scenarios. Thus realizability corresponds to 1-testability in our framework and was shown to be undecidable for regular MSG specifications in [4].

The corresponding question for synchronous communication—MSCs in which each message exchange is a handshake—has been considered in [15]. In this setting, it is shown that one can construct an implementation that covers the specification and includes the minimum number of implied scenarios in doing so.

On the other hand, arbitrary regular MSC specifications can be implemented faithfully using message-passing automata with bounded channels if we are allowed to add additional control information to messages [8]. These control messages piggyback on existing system messages and use the time-stamping protocol of [12] to transfer information about the global state of the sytem to each process. However, this construction is rather complex and impractical. Moreover, the systems synthesized in this manner may have deadlocks.

We can view our results as providing a solution to the realizability problem that lies between these two extremes. Given an MSG $\mathcal{G}$, we obtain each component as local projection, as in weak realizability, but also run a tagging protocol in parallel to enforce additional coordination. Our results show that we can check whether such an implementation is faithful to $\mathcal{G}$. The tagging protocol is independent of $\mathcal{G}$ and quite efficient to construct on the fly, so this provides a significantly simpler synthesis algorithm than the one in [8] that avoids the negative features of the very naïve approach of weak realizability.

A related question is that of deadlock free, or *safe*, realizability. In [4], this question is shown to be decidable for regular MSG specifications. The precise complexity of the construction is computed in [10], where it is also shown that safe realizability is undecidable for arbitrary MSG specifications. In our setting, the only additional machinery we introduce is the tagging automaton that runs in parallel with the implementation. Since this automaton is local and deterministic, it does not introduce any deadlocks, so our approach will produce a deadlock-free implementation whenever the approach of [4] does.

Another interesting connection is to causal realizability. Our approach is motivated by the fact that checking for causally implied MSCs is decidable for regular MSC languages [1]. The causal closure of L is the set of all MSCs that are causally implied by L. It turns out that the causal closure of a regular MSC language is always regular and can be constructed using bounded time-stamps.

The time-stamping used to compute causal closure is woven into the state space of the original system, which is not desirable for testing, as we have already pointed out. Nevertheless, it is natural to compare our approach with causal realizability. As we observed earlier, any causally implied scenario is also an implied scenario in our setting. We do not know, however, how close to causal closure we can get using our loosely coupled timestamps.

6 Discussion

As we have observed, since we can tag the SUT and the MSG specification on the fly, performing a local test is relatively efficient. The main practical difficulty with our proposal is the explosion in the length of the tests to be performed. These are now proportional to paths in $\mathcal{G}^T$. Let M be the number of nodes in $\mathcal{G}$ and let K be the number of messages in the largest MSC labelling any of the nodes in $\mathcal{G}$. There are $O((B^2N^2(\log B+\log N))^K)$ ways of assigning time-stamps to this largest MSC, each of which could appear as a separate node in $\mathcal{G}^T$. In general, this blowup could occur for each node in $\mathcal{G}$, so the size of $\mathcal{G}^T$ could be as large as $O((B^2N^2(\log B + \log N))^{KM})$.

It would be interesting to find more efficient time-stamping schemes that still guarantee that the testability problem is decidable. We saw a simple tagging scheme in which we uniformly labelled messages in each node with the name of the node as the time-stamp. We can construct more elaborate tagging schemes in which the tag structure depends on the MSG. However, as we have seen, the key feature that we need from the tagging scheme is that it is local and deterministic, which

is not easy to achieve if the tags refer to the structure of the MSG.

Another, more theoretical, question is to explore the expressive power of loosely coupled time-stamps. As we observed in the context of the example in Figure 8, we can enhance the time-stamping protocol of [12] to eliminate some implied scenarios. In the limit, we approach causal closure, but how close can we get?

Acknowledgments We thank Philippe Darondeau, Paul Gastin, Blaise Genest and K. Narayan Kumar for their insightful comments on an earlier version of this paper.

References

[1] Adsul, B., Mukund, M., Narayan Kumar, K., and Narayanan, V.: Causal closure for MSC languages. *Proc. FSTTCS 2005*, Springer LNCS **3821** (2005) 335–347.

[2] Alur, R., and Yannakakis, M.: Model checking of message sequence charts. *Proc. CONCUR 1999*, Springer LNCS **1664** (1999) 114–129.

[3] Alur, R., Etessami, K.,and Yannakakis, M.: Inference of message sequence graphs. *IEEE Trans. Software Engg* **29(7)** (2003) 623–633.

[4] Alur, R., Etessami, K., and Yannakakis, M.: Realizability and Verification of MSC Graphs. *Theor. Comput. Sci.* **331(1)** (2005) 97–114.

[5] Bhateja, P., Gastin, P., Mukund, M., and Narayan Kumar, K.: Local testing of message sequence charts is difficult. *Proc. FCT 2007*, Springer LNCS **4639** (2007) 76–87.

[6] Booch, G., Jacobson, I., and Rumbaugh, J.: *Unified Modeling Language User Guide.* Addison-Wesley (1997).

[7] Harel, D., and Gery, E.: Executable object modeling with statecharts. *IEEE Computer*, July 1997 (1997) 31–42.

[8] Henriksen, J.G., Mukund, M., Narayan Kumar, K., Sohoni, M., and Thiagarajan, P.S.: A Theory of Regular MSC Languages. *Inf. Comp.*, **202(1)** (2005) 1–38.

[9] ITU-TS Recommendation Z.120: *Message Sequence Chart (MSC).* ITU-TS, Geneva (1997).

[10] Lohrey, M.: Realizability of high-level message sequence charts: closing the gaps. *Theor. Comput. Sci.* **309**(1–3) (2003), 529–554.

[11] Mauw, S., and Reniers, M. A.: High-level message sequence charts, *Proc SDL'97*, Elsevier (1997) 291–306.

[12] Mukund, M., Narayan Kumar, K., and Sohoni, M.: Bounded time-stamping in message-passing systems. *Theoretical Computer Science*, **290(1)** (2003) 221–239.

[13] Muscholl, A., and Peled, D.: Message sequence graphs and decision problems on Mazurkiewicz traces. *Proc. MFCS 1999*, Springer LNCS **1672** (1999) 81–91.

[14] Rudolph, E., Graubmann, P., and Grabowski, J.: Tutorial on message sequence charts. In *Computer Networks and ISDN Systems — SDL and MSC* **28** (1996)

[15] Uchitel, S., Kramer, J., and Magee, J.: Detecting implied scenarios in message sequence chart specifications. *Proc. ESEC/ SIGSOFT FSE*, ACM (2001) 74–82.

[16] Willcock, C., Deiß, T., Tobies, S., Keil, S., Engler, F., and Schulz, S.: *An Introduction to TTCN-3.* Wiley (2005).