The Structure of Local-Choice in High-Level Message Sequence Charts (HMSC)

Blaise Genest and Anca Muscholl

LIAFA, Université Paris VII
2, pl. Jussieu, case 7014, 75251 Paris cedex 05, France

Abstract. As a specification formalism for communication protocols, (high-level) message sequence charts (H)MSCs offer a powerful visual notation for communication protocols. We introduce extended local-choice HMSCs and show that they are equivalent to local-choice HMSCs. Studying the structure of extended local-choice HMSC allows us to check for an HMSC to be equivalent to an extended local-choice HMSC in PSPACE. Finally, a successful test allows to construct effectively an equivalent (extended) local-choice HMSC, hence is implementable with additional data. If no additional data is allowed, we can test in PSPACE whether an extended local-choice HMSC is implementable without deadlock.

1 Introduction

As a specification formalism for communication protocols [1], high-level message sequence charts (HMSCs for short) offer a powerful visual notation, that describes the interaction of asynchronously evolving, communicating processes. An HMSC specification usually refers to the basic behavior of the system, leaving several details to the fully developed system (e.g., local and message data, timing, etc). The visual nature of the specification raises the question of realizability of HMSC specifications by some machine-like model, which can be seen as an abstract model of the final system. The most natural state-based model in this framework are communicating finite-state machines (CFMs for short), i.e., finite automata that communicate over point-to-point FIFO channels.

The question of realizability of HMSC specifications by CFMs was first raised by [2, 3]. Their motivation was to test whether a given specification is complete with respect to the future system implementation. That is, the future implementation can include more behaviors than those specified by the HMSC, which follows from the visual nature of the HMSC. Checking whether an HMSC specification is CFM-realizable is a basic test during the design process. Being able to construct easily an equivalent CFM is also important for the purpose of testing or simulating the behavior of the specification.

The realizability notion introduced by [2, 3] is a simple one. In that paper the CFM is fully determined by the HMSC specification, by defining the behavior of each process as the projection of the HMSC executions on that process. Although this definition of realizability is quite natural, it does not take into account that the design process can refine the specification by adding data or messages. Moreover, the realizability notion of [3] turns out to be undecidable, even for HMSC specifications that are finite-state (also called bounded, [4]). In practice, implementations are required to be deadlock-free, which is called *safe realizability* in [2]. Testing *safe realizability* of regular HMSC specifications is shown to be decidable in exponential space in [3] (the matching lower bound is proved in [10]).

The realizability notion used in [11, 7] allows to add bounded data to messages specified in the HMSC. In [11] the authors consider only regular (finite state) HMSC specifications. The framework is extended in [7], where two classes of unbounded HMSCs are shown to be always CFM-realizable by adding constant data in messages. Moreover, one of these classes – *local-choice* HMSCs [5, 8] – has deadlock-free CFM implementations of linear size, [7]. It is worth noting that local-choice HMSCs are not always implementable with respect to the realization notion of [2, 3] (i.e., without adding data), which is an important drawback of the definition of the simple implementation. Anyway, we show that such an implementation can be checked in PSPACE for (extended) local-choice HMSC.

The main problem considered in this paper is to find a semantical characterization of deadlock-free realizable HMSC specifications. We are facing two kinds of questions. First, we can ask whether an HMSC has the local-choice property. This is an easy syntactic check, but probably many specifications would fail the test, although they might be CFM-realizable. Second, we can ask whether a given HMSC has some deadlock-free CFM-realization with additional bounded data. As far as the amount of additional data is not known, this kind of question is undecidable and has no practical relevance. What we aim at is to define a class of unbounded HMSC specifications that guarantees the existence of deadlock-free CFM-realizations (using additional bounded message data). Precisely, we ask the question whether an HMSC is equivalent to some *local-choice* HMSC. If the answer is positive, we want to construct such an equivalent HMSC.

The local-choice property is a natural restriction of HMSC: while HMSC are controlled globally, every local-choice HMSC are locally controlled, that is each choice node is controlled by only one process. Algorithms for local-choice HMSC are easier than for HMSC [7]. Hence, local-choice HMSC are of great interest both from an implementation and an algorithmic point of view, and can be used to model many real life examples. Nevertheless, since the restriction concerns only choice node, local-choice property is too structural and not handy to be checked semantically. We introduce then the extended local-choice property meaning that each execution after a node is controlled locally, i.e. by a single process (that must not be the same for different alternatives, as in the local-choice case). An important result is that any extended local-choice HMSC can be transform into a local-choice HMSC with a small blowup.

We have studied the structure of executions of extended local-choice HMSC. The main result of this study is that we can decide whether an HMSC is equivalent to an extended local-choice HMSC in polynomial space. If the test succeeds, we show how to construct an equivalent (extended) local-choice HMSC, and hence how to implement it without deadlock.

2 Message Sequence Charts

In this section we recall the notation of message sequence charts (MSC) and high-level message sequence charts (HMSC, MSC-graphs) based on the ITU standard Z.120 [1]. An MSC describes a communication scenario, by specifying the messages sent and received, the local events, and the (partial) ordering between them. The event ordering is defined by a total event order on each process, together with the message ordering. In the visual description of MSCs, each process is represented by a vertical line, expressing the event order on that process. Messages are represented by horizontal or slanted arrows from the sending process to the receiving one.

MSC. An MSC over process set $\mathcal{P}$ is a tuple $M = \langle E, <, \mathcal{P}, m, \mathcal{C}, c\rangle$ where:

- $E = \biguplus_{p\in\mathcal{P}} E_p$ is a finite set of events, each located on some process from the set $\mathcal{P}$, with E_p denoting the set of events of process p. We denote by $P(e) \in \mathcal{P}$ the process to which event e belongs.
- Every event is either a communication event (send or receive) or a local event. We write $E = S \uplus R \uplus L$ as a disjoint union, with S denoting the sends, R the receives and L the local events.
- $\mathcal{C}$ is a finite set of message contents (names) and local action names.
- $m : S \longrightarrow R$ is a bijection that pairs up send and receive events (*matching* function). With each message $(s, m(s))$, we associate its content $c(s, m(s)) \in \mathcal{C}$.
- $< \subseteq E \times E$ is the least acyclic relation satisfying the following requirements:
  - The restriction of $<$ to E_p is a total order, for every process $p \in \mathcal{P}$.
  - For all $e, f \in E$, $m(e) = f$ implies $e < f$.

The set of processes p in M with $E_p \neq \emptyset$ is denoted $P(M)$. The size of M is given as the number of events. We also denote by $\wp$ the cardinality of $\mathcal{P}$.

Since our implementation model assumes that communication is done over FIFO channels, we assume the FIFO condition for MSC, too. That is, we require that whenever two messages $m(e_1) = f_1$ and $m(e_2) = f_2$ are such that $P(e_1) = P(e_2)$ and $P(f_1) = P(f_2)$, we have $e_1 < e_2$ if and only if $f_1 < f_2$.

Since $<$ is required to be acyclic, its reflexive-transitive closure $<^*$ is a partial order on E. For sake of simplicity we will use the same notation $\leq$ for the partial order $<^*$. A *linearization* of $<$ is defined as usual, as a total order $\preceq$ extending $\leq$, i.e., $\leq \subseteq \preceq$. The future of an event e, denoted *future*(e), is the set of all events e' with $e \leq e'$.

Let M, N be two MSCs over the same set of processes. We say that M *is contained* in N (and we write $M \subseteq N$) if the partial order graph of M occurs as an induced subgraph in N. Similarly, for $M, N \subseteq O$ we write $M \cap N$ ($M \setminus N$, resp.) for the graph-theoretical intersection (difference, resp.) of the partial order subgraphs M, N of O. We write $e \parallel f$ for $e, f \in M$ whenever e and f are not ordered (incomparable) for $\leq_M$. By extention, for $E, F \subseteq M$, we write $E \parallel F$ iff $e \parallel f$ for all $e \in E, f \in F$. In particular, $P(E) \cap P(F) = \emptyset$. If an MSC cannot be written as $M_1 \parallel M_2$ with M_1, M_2 non-empty MSCs, then we call it a *connected* MSC.

Since the specification of a communication protocol includes many scenarios, a high-level description is needed for combining them together and defining infinite sets of (finite or infinite) scenarios. The standard description of the norm Z.120 uses non-deterministic branching, concatenation and iteration for defining finite or infinite sets of MSCs. Formally, a *high-level MSC* $G = \langle V, \rightarrow_G, V^0, V^f, \lambda \rangle$ (HMSC for short) is a finite transition system $(V, \rightarrow_G, V^0, V^f)$ with transition set $\rightarrow_G \subseteq V \times V$, set of initial nodes $V^0 \subseteq V$ (with no ingoing edges) and set of terminal nodes $V^f \subseteq V$ (with no outgoing edges). Each node v is labeled by the finite MSC $\lambda(v)$. We let $P(v) = P(\lambda(v))$, and we assume that $P(v) \neq \emptyset$ for each node v. We also assume that every node is accessible from V^0 and from each node there is a path to V^f. An *execution* of G is the MSC $\lambda(v_0)\lambda(v_1)\cdots\lambda(v_k)$ labeling some successful path $v_0, v_1, \ldots, v_k$ in G, i.e., $v_i \rightarrow_G v_{i+1}$ for every $0 \leq i < k$, $v_0 \in V^0$ and $v_k \in V^f$. The set of executions of G is denoted by $\mathcal{L}(G)$, the set of linearizations of executions of G is denoted by $\mathrm{Lin}(G)$. The *size* of an HMSC is the sum of the sizes of its nodes plus the number of transitions.

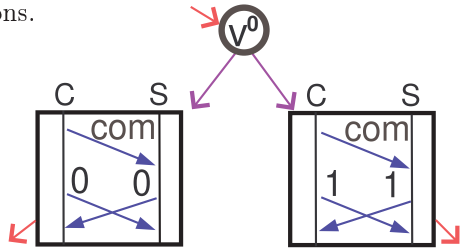


Fig. 1. an HMSC

The semantics of HMSCs depends on the definition of the MSC concatenation. We consider the usual *(weak) concatenation* of MSCs, as defined in the following. Let $M_1 = \langle E_1, <_1, \mathcal{P}, m_1, \mathcal{C}_1, c_1 \rangle$ and $M_2 = \langle E_2, <_2, \mathcal{P}, m_2, \mathcal{C}_2, c_2 \rangle$ be MSCs over the same set of processes $\mathcal{P}$. Their product $M_1 M_2$ is defined as the MSC $\langle E_1 \uplus E_2, <, \mathcal{P}, m_1 \cup m_2, \mathcal{C}_1 \cup \mathcal{C}_2, c_1 \cup c_2 \rangle$ over the disjoint union of events $E_1 \uplus E_2$, with the visual order given by:

$$< \; = \; <_1 \cup <_2 \cup \{(e, f) \in E_1 \times E_2 \mid P(e) = P(f)\}.$$

Note that it is possible that one process is still involved in some actions of M_1, while another process has advanced to an event of M_2. The notion of prefix is meant with respect to this definition of concatenation.

3 Communicating Finite-State Machines

The most natural implementation model for HMSCs are communicating finite state machines (CFM), as used for instance in the ITU standard specification language SDL.

A CFM $\mathcal{A}$ consists of a network of finite state machines $\mathcal{A} = (\mathcal{A}_p)_{p\in\mathcal{P}}$ that communicate over unbounded, error-free, FIFO buffers. The content of a buffer is a word over a finite alphabet $\mathcal{C}$. With each pair $(p, q) \in \mathcal{P}^2$ of distinct processes we associate a buffer $B_{p,q}$. Each finite state machine $\mathcal{A}_p$ is described by a tuple $\mathcal{A}_p = (S_p, A_p, \to_p, F_p)$ consisting of a set of local states S_p, a set of actions A_p, a set of local final states F_p and a transition relation $\to_p \subseteq S_p \times A_p \times S_p$. The computation begins in an initial state $s^0 \in \prod_{p\in\mathcal{P}} S_p$. The actions of $\mathcal{A}_p$ are either local actions or sending/receiving a message. Sending a message $a \in \mathcal{C}$ from process p to process q is denoted by $p!q(a)$ and it means that a is appended to the buffer $B_{p,q}$. Receiving message a by p from q is denoted by $p?q(a)$ and it means that a must be the first message in buffer $B_{q,p}$, which will be then removed from $B_{q,p}$. A local action m is denoted by $l_p(a)$. We denote a run of the CFM as *successful*, if each process p finishes the execution in some final state F_p and all buffers are empty. The set of successful runs (i.e., MSCs) of $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A})$. The *size* of $\mathcal{A}$ is $\sum_p |\mathcal{A}_p| + |\mathcal{C}|$.

A CFM-realization of an HMSC N will add in general data to the message contents of N, i.e., the CFM message contents are from a set $\mathcal{C}'$ and we have a projection from $\mathcal{C}'$ to $\mathcal{C}$. We will call $\mathcal{A}$ a *CFM-realization* of N if the set of MSCs defined by the successful runs of $\mathcal{A}$, where we project the message contents onto $\mathcal{C}$, is exactly $\mathcal{L}(N)$.

A CFM is *deadlock-free*, if each run starting in the initial configuration can be extended into a successful run.

4 Extended Local-Choice

4.1 Local-choice HMSCs and CFM-realizations

We use the framework of [7] for the realization problem. Among the HMSC classes described in that paper, a particular one turned out to be very suitable for obtaining a deadlock-free realization: local-choice HMSCs, defined in [5,8]. Roughly speaking, the local-choice property ensures that branching between executions is always controlled by a unique process. Figure 1 represents a local-choice HMSC.

Local-choice HMSCs. Let G be an HMSC $G = \langle V, \to_G, V^0, V^f, \lambda\rangle$. We denote an MSC with a unique minimal event as a *triangle*.

1. A node $v \in V$ of G is called *local* if there is a process $root(v)$ such that <u>every path</u> $w_1 w_2 \cdots$ with $v \to_G w_1$ is labeled by a triangle whose minimal event is located on $root(v)$.
2. The HMSC G is called local-choice if each node having at least two outgoing edges is local. Moreover, there exists some process p_0 such that each accepting MSC $M \in \mathcal{L}(G)$ has a unique minimal event located on p_0.

We recall here some basic results on model-checking and implementation of local-choice HMSC, see [7] for details. The property below uses the fact the final nodes of an HMSC have no outgoing edges.

Proposition 1 *Each local-choice HMSC is equivalent to a local-choice HMSC where all nodes are local (i.e., not only branching ones).*

Proposition 2 *Let G be a local-choice HMSC. Then G admits a deadlock-free realization of size linear in the size of G.*

Moreover, model-checking for local-choice HMSCs turns out to be as easy as for finite automata. The local-choice restriction is therefore of great interest for both model-checking and CFM-realization.

Proposition 3 *Let G, G' be two local-choice HMSCs. Checking whether $\mathcal{L}(G) \cap \mathcal{L}(G') = \emptyset$ is NLOGSPACE-complete, while $\mathcal{L}(G) \subseteq \mathcal{L}(G')$ is PSPACE-complete.*

A drawback of the local-choice definition is that it represents a syntactical property. Hence, many HMSC specifications might be realizable, although they are not local-choice. A natural question to ask is whether a given HMSC can be transformed into an equivalent local-choice HMSC. We will show in the last section how to decide this semantical property.

4.2 Why always the same?

The local-choice restriction that two executions starting after a given node must have the same minimal process is an obstacle for studying the structure of local-choice. We ask here what happens when we allow two executions starting after a given node to have each a unique minimal event on p, q respectively, with $p \neq q$.

Extended Local-Choice HMSCs. An HMSC G is called *extended local-choice* if every path (beginning in any node) of G is labeled by a triangle. Moreover there exists some process p_0 such that each MSC $M \in \mathcal{L}(G)$ has a unique minimal event located on p_0. We denote the minimal event of the MSC labeling node v by $\min(v)$, for any v.

An event e of an MSC M is called a *cut-event of M* if *future*(e) is a (complete) MSC, that is, for every receive in *future*(e), its matching send also belongs to *future*(e). Note that the minimal event of a triangle is always a cut-event.

By Proposition 1, each local-choice HMSC is equivalent to an extended local-choice HMSC G. Note also that for every transition $v \to_G w$ in an extended local-choice HMSC we have that $P(\min(w)) \in P(v)$, i.e., the minimal process of w occurs in v.

Actually, we can transform any extended local-choice HMSC into a local-choice HMSC. This shows that local-choice HMSCs and extended local-choice HMSCs have the same expressivity[1].

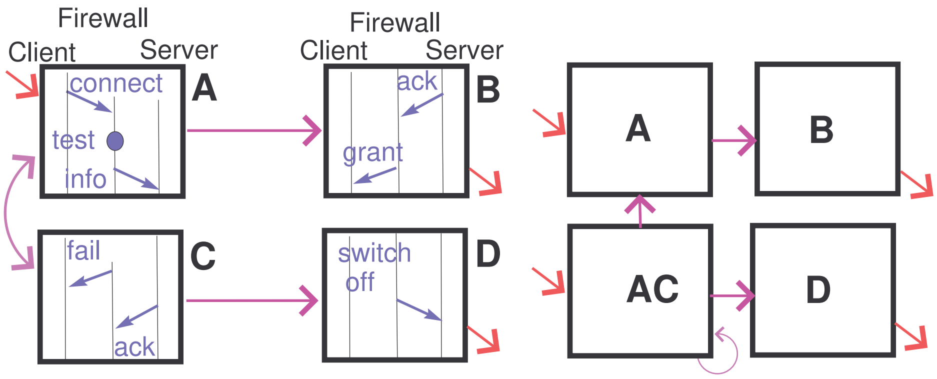
Theorem 1 *Let G be an extended local-choice HMSC. Then we can transform G into a local-choice HMSC of size at most quadratic in $|G|$.*

[1] This was noticed by an anonymous referee of an earlier version of the paper.

Proof. Let us consider an extended local-choice HMSC $G = \langle V, \to_G, V^0, V^f, \lambda\rangle$. We define the HMSC H whose set of nodes is $V' = \{\langle v_1, v_2\rangle \in V^2 \mid v_1 \to_G v_2\}$. Moreover, node $\langle v_1, v_2\rangle$ is labeled by $\lambda(v_1)$, and we have a transition $\langle v_1, v_2\rangle \to_H \langle v_3, v_4\rangle$ iff $v_2 = v_3$. The initial nodes $\langle v_1, v_2\rangle$ are such that $v_1 \in V^0$. Additionally, we add the final nodes $v \in V^f$ of G as final nodes of H (with the same labeling as in G), and the transitions $\langle v_1, v\rangle \to_H v$.

It is easily seen that each transition $v' = \langle v_1, v_2\rangle \to_G v''$ is such that v'' is labeled by $\lambda(v_2)$. Hence, node v'' has a unique minimal event on a process in $P(v')$, so H is local-choice. Moreover, $\mathcal{L}(H) = \mathcal{L}(G)$. □

We will show later than we can automatically transform an HMSC into an extended local-choice HMSC when possible. The example below shows an HMSC that is not extended local-choice: for instance node C has two minimal events. Joining together nodes A, C leads to the equivalent, extended local-choice HMSC, in the right half of the picture. It is not local-choice since A and D that are successors of AC have not the same minimal process.



5 The Essence of Local-Choice

In this section, we study the structure of the executions of extended local-choice HMSCs. First, we exhibit in theorem 2 a semantical characterization of HMSCs that are equivalent to extended local-choice HMSCs. Consequently, we show in theorem 3 how to decide the inclusion $\mathcal{L}(G) \subseteq \mathcal{L}(H)$ for any extended local-choice HMSC H and G arbitrary.

First, we need some preliminary definitions. An HMSC class with nice properties with respect to model-checking has been introduced in [7], by restricting the MSCs labeling the loops. Formally, an HMSC is called globally-cooperative, if every loop is labeled by a *connected* MSC, i.e., an MSC that cannot be written as $M_1 \parallel M_2$ with M_1, M_2 non-empty MSCs. The HMSCs in the figure above are globally-cooperative. We recall that model-checking globally-cooperative HMSCs is decidable, although the linearization set $\mathrm{Lin}(G)$ is not necessarily regular. The reason is that we can construct a finite automaton $\mathcal{A}$ accepting for each $M \in \mathrm{Lin}(G)$ a representative linearization. More precisely, $\mathcal{A}$ accepts all linearizations from $\mathrm{Lin}(G)$ of the form $x = x_1 \cdots x_k$, where each x_i is a linearization of an atomic MSC. An MSC is called *atomic* [9] if it cannot be written as M_1M_2, with M_1, M_2 non-empty MSCs.

5.1 The generic HMSC H_n

Let n be an integer, then $\mathcal{T}_n$ denotes the set of triangles of size at most n. The generic HMSC H_n has exactly one node labeled by T for each $T \in \mathcal{T}_n$. Moreover, we have a transition $T \to T'$ iff the minimal event of T' is on a process of $P(T)$. Clearly H_n is an extended local-choiceHMSC.

First, one can notice that every extended local-choice HMSC G verifies $\mathcal{L}(G) \subseteq \mathcal{L}(H_n)$ for $n = |G|$. The theorem below shows that a globally-cooperative G with $\mathcal{L}(G) \subseteq \mathcal{L}(H_n)$ for some n allows to construct effectively an equivalent extended local-choice HMSC. Roughly speaking, we intersect G with H_n.

Theorem 2 *A globally-cooperative HMSC G is equivalent to some extended local-choice HMSC iff $\mathcal{L}(G) \subseteq \mathcal{L}(H_n)$ for some n. Moreover, if the condition holds, then we can effectively construct an equivalent extended local-choice HMSC of size exponential in $|G|, \wp, n$.*

Proof. Let $n > 0$ such that $\mathcal{L}(G) \subseteq \mathcal{L}(H_n)$. Since G is globally-cooperative we can construct a finite automaton $\mathcal{A}$ as described above (see also [7]), recognizing all atomic linearizations of $\mathcal{L}(G)$.

Now, the idea is to construct an extended local-choice HMSC B as the product of the generic HMSC H_n with $\mathcal{A}$. We have a node for each pair (T, s), where $T \in T_n$ and s is a state of $\mathcal{A}$. The node (T, s) is labeled by T. The transitions are given by $(T, s) \longrightarrow (T', s')$ if we have $T \longrightarrow T'$ in H_n and $s \xrightarrow{x} s'$, for some atomic linearization x of T. The initial nodes are of the form (T, s_0), where s_0 is an initial state of $\mathcal{A}$, and final nodes are of the form (T, s), where $s \xrightarrow{x} f$ is a transition labeled by an atomic linearization x of T leading to a final state f of $\mathcal{A}$. $\square$

Proposition 4 *Assume that G is an HMSC equivalent to some extended local-choice HMSC H. Then G is globally-cooperative.*

Proof. We suppose by contradiction that G contains a non-connected loop. That is, G contains accepting paths labeled by $M_1(N_1N_2)^nM_2$, for every n, such that M_1, M_2, N_1, N_2 are MSCs, $P(N_1) \cap P(N_2) = \emptyset$ and N_1, N_2 are both non-empty.

Assume now that $M_1(N_1N_2)^nM_2 = T_1 \cdots T_m$, where $T_1, \ldots, T_m$ is a path of H. For some large enough n we find some subpath $T_i, \ldots, T_j$ that contains events of both N_1, N_2 and no event from M_1, M_2. Clearly, this contradicts the fact that H is extended local-choice.

$\square$

Recall that checking the globally-cooperative property is co-NP-complete, [12]. Notice that every MSC $M \in \mathcal{L}(G)$ needs to be a triangle, if G is equivalent to some extended local-choice HMSC. We will show below (theorem 3) how to check that a given globally-cooperative HMSC satisfies $\mathcal{L}(G) \subseteq \mathcal{L}(H_n)$ in polynomial space. Let us begin by introducing condition (C_n), that is almost equivalent to having $\mathcal{L}(G) \subseteq \mathcal{L}(H_n)$. An HMSC G satisfies (C_n) if whenever $NM \in \mathcal{L}(G)$ is such that M is a triangle with minimal event $\min_M$, then either $|M| \leq n$ or the MSC M contains a cut-event $e \neq \min_M$ with $|M \setminus \mathit{future}(e)| \leq n$. Intuitively, if G satisfies (C_n), then we can decompose inductively any execution of G into a sequence of small triangles.

Proposition 5 *Let G be an HMSC such that every MSC $M \in \mathcal{L}(G)$ is a triangle. Then $\mathcal{L}(G) \subseteq \mathcal{L}(H_{[n/\wp]}) \implies (C_n) \implies \mathcal{L}(G) \subseteq \mathcal{L}(H_n)$.*

Proof.
For the first implication let us assume that $\mathcal{L}(G) \subseteq \mathcal{L}(H_{[n/\wp]})$. Let $NM \in \mathcal{L}(G)$ with M a triangle. Then there is a decomposition $NM = v_1 \cdots v_t$ with $v_i \in \mathcal{T}_{n/\wp}$ and $P(\min(v_{i+1})) \in P(v_i)$ for each i. Let j be minimal such that $\min(v_j) \in M$. Then we have $v_j \cdots v_t \subseteq M$. Let $I = \{i \mid i < j, v_i \cap M \neq \emptyset\} = \{i_1, \ldots, i_s\}$. For each $k < s$, we have $v_{i_{k+1}} \setminus M \neq \emptyset$ has no process in common with $(v_{i_1} \cdots v_{i_k}) \cap M$. But since $v_{i_{k+1}}$ is connected, $v_{i_{k+1}} \setminus M$ and $v_{i_{k+1}} \cap M$ share a process that is not in $(v_{i_1} \cdots v_{i_k}) \cap M$. This gives $s \leq \wp$. Let e be the minimal event of v_j. Then, $|M \setminus \mathit{future}(e)| \leq sn/\wp \leq n$, which means that (C_n) is true.

For the second part, suppose (C_n). We will show that there exists no triangle $M \notin \mathcal{L}(H_n)$ with $NM \in \mathcal{L}(G)$ for some N. We can conclude that $\mathcal{L}(G) \subseteq \mathcal{L}(H_n)$. By absurd, let us take M of minimal size. Since $M \notin \mathcal{L}(H_n)$, $|M| > n$. Moreover, with (C_n) there exist a cut-event $e \neq \min_M$ of M with $P = M \setminus \mathit{future}(e)$ satisfies $|P| \leq n$. Then we can write $M = PS$ with S a triangle equals to $\mathit{future}(e)$. Because M was a suffix of $\mathcal{L}(G)$ of minimal size such that $M \notin \mathcal{L}(H_n)$, and $|S| < |M|$, we have $S \in \mathcal{L}(H_n)$. Then $M = PS \in \mathcal{L}(H_n)$, a contradiction. □

Proposition 6 *Checking that every MSC of $\mathcal{L}(G)$ is a triangle can be done in logarithmic space. Moreover, the condition (C_n) can be verified in polynomial space.*

Proof. First, we have to check that every accepting path of the graph is labeled by a triangle. If this is not the case, then there must be some process that sends some message without having received any message before. To look for such a case, for each process we delete all the nodes where this process receives a message before sending one. Then we want to know whether any of the states where the process sends a message before receiving one is accessible from an initial state in the new graph. Hence we can check that every MSC of $\mathcal{L}(G)$ is a triangle in logarithmic space.

To test that e is a cut event of some M, we need to compute its future $\mathit{future}(e)$. We check that no receive event $r \in \mathit{future}(e)$ is such that its associated send s satisfies $s \notin \mathit{future}(e)$. Note that we do not need to record all events in $\mathit{future}(e)$, but rather the set $F(e)$ of processes seen so far in $\mathit{future}(e)$, ie $F(e) = \{P(e') \mid e \leq e'\}$.

Next, we check the existence of some $PM \in \mathcal{L}(G)$ with M a triangle with minimal event $\min_M$ and $|M \setminus \mathit{future}(e)| > n$, for all cut events e in M. We just guess some send $\min_M$, which fixes M. We first have to verify that $\min_M$ is a cut-event. Additionally we have to guess for each send of *future*(M) whether it is a cut-event. The guess for e will be checked on the fly using the set $F(e)$ until we know that e cannot be a cut-event.

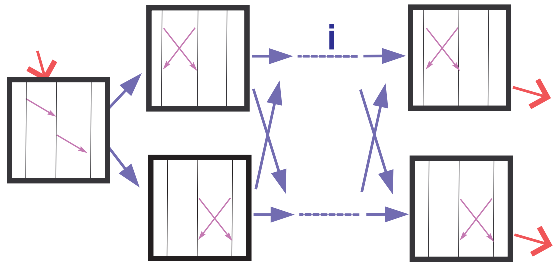
Finally, for every cut event e of M we check that $|future(\min_M)| - |future(e)| > n$. To compute this size, we only need the sets of processes $F(e)$ and $F(\min_M)$. We increment a counter at each event on a process $p \in F(\min_M) \setminus F(e)$.

Note also that if two events e, e' are such that $F(e) = F(e')$, then either both are cut-events or none of them. Moreover, at most $\wp$ different events e on the same process can have different sets $F(e)$. Hence we need to remember at most $\wp^2$ different sets of processes $F(e)$. □

Remark 1. We can show that (C_n) can be checked in co-NP, but the algorithm is rather involved. For our purposes, the PSPACE upper bound suffices.

By theorem 2, if G satisfies (C_n) then we can construct an equivalent extended local-choice HMSC that is exponential in the size of G, n. This upper bound is met by the globally-cooperative HMSC family $(G_i)_i$ given in the example below.

Example 1.



5.2 Using the generic HMSC H_n

The set of triangle $\mathcal{T}_n$ is quite useful when dealing with an extended local-choice HMSC G because we can use it as a basis for decomposing G. Although the decomposition is not unique, the main point is that we can test whether an HMSC G satisfies (C_n). Note also that (C_n) can be checked faster than the inclusion between globally-cooperative HMSCs, ie faster than EXSPACE. We show now three applications of H_n.

The first easy corollary is that we can solve the positive model-checking problem $\mathcal{L}(G) \subseteq \mathcal{L}(H)$ in PSPACE, whenever H is extended local-choice.

Theorem 3 *Consider an HMSC G and an extended local-choice HMSC H. Then we can test in PSPACE whether $\mathcal{L}(G) \subseteq \mathcal{L}(H)$.*

Proof. Let $n = |H|$. We first test whether G satisfies $(C_{\wp n})$. If not, $\mathcal{L}(G) \nsubseteq \mathcal{L}(H_n)$ hence $\mathcal{L}(G) \nsubseteq \mathcal{L}(H)$.

Else, $\mathcal{L}(G) \subseteq \mathcal{L}(H_{\wp n})$, hence we can build an equivalent, exponential size extended local-choice HMSC G' from G.

Then we apply the algorithm of [7] for $\mathcal{L}(G') \subseteq \mathcal{L}(H)$, which is PSPACE in $|H|$ and LOGSPACE in $|G'|$. Hence, altogether we obtain a PSPACE algorithm. □

The second application is to test (and transform when possible) whether an HMSC is equivalent to some extended local-choice HMSC, see section 6. The last application is to decide whether an extended local-choice HMSC is realizable with respect to the simple realizability notion of [3], see the last section.

6 Transforming a globally-cooperative HMSC into an extended local-choice HMSC

We do not have any algorithm to answer whether a given HMSC is implementable with additional data and without deadlocks. Moreover, there exist implementable HMSCs that are not equivalent to any extended local-choice HMSC. However, we would want at least a weaker algorithm that tests whether an HMSC can be transformed into an extended local-choice HMSC. If the answer is yes, then we can apply the algorithm we have to implement a local-choice HMSC. A trivial test is to check whether $\mathcal{L}(G) \subseteq \mathcal{L}(H_n)$ for all n. A problem is that n is a priori arbitrarily large, so we cannot infer an algorithm yet.

We state now two properties of MSCs, that are of great use in the following.

Lemma 1. *Let M be an MSC and $e \parallel e'$ be two (incomparable) cut-events of M. Then $\mathit{future}(e) \cap \mathit{future}(e') = \emptyset$.*

Proof. Suppose by contradiction that some r exists with $r \in \mathit{future}(e) \cap \mathit{future}(e')$ and choose r minimal with this property. Hence r is a receive. Let s be the associated send. For the same reason, s is either in $\mathit{future}(e)$ or $\mathit{future}(e')$, but not in both (else r is not minimal). It means that either e or e' is not a cut-event, because its future is not an MSC (there is a receive but not the associated send), which is a contradiction. □

Lemma 2. *Let $LMN = B_0C_0B_1 \cdots B_{2\wp}C_{2\wp}B_{2\wp+1}$ with L, M, N, B_i, C_i MSCs. Moreover, assume that each C_i is connected and satisfies $C_i \cap M \neq \emptyset$. Then there exists p with $C_p \subseteq M$.*

Proof. We show that for at most $\wp$ indexes i we have $C_i \not\subseteq MN$ (resp. $C_i \not\subseteq LM$). Hence for at most $2\wp$ indexes i we have $C_i \not\subseteq LM \cap MN = M$. For all i, we can decompose $C_i = D_iE_i$ with $D_i \neq \epsilon$ the MSC containing the events belonging to MN, that is $D_i = C_i \cap MN$.

Assume by contradiction that for $\wp + 1$ indexes i, $E_i \neq \epsilon$. For simplicity, suppose that these indexes are $0, 1, \cdots \wp$. Since C_i is connected, D_i and E_i share a common process p_i. Moreover, we have $C_0 \cdots C_\wp = E_0 \cdots E_\wp D_0 \cdots D_\wp$ because MN is a suffix of LMN. That is E_i commutes with $D_0 \cdots D_{i-1}$, i.e., it shares no process with $D_0 \cdots D_{i-1}$. This means that $p_i \notin \{p_0, \cdots, p_{i-1}\}$. Hence $p_0, \cdots, p_\wp$

are pairwise different processes, which is a contradiction with $\wp$ being the total number of processes. □

We will apply lemma 2 to paths of an extended local-choice HMSC, where each C_i labels a node (see next two propositions), and to globally-cooperative HMSC where each C_i labels a loop (see proposition 11).

The next proposition states that in any MSC labeling a path of an extended local-choice HMSC, cut-events occur frequently.

Proposition 7 *Let G be an extended local-choice HMSC. Consider two triangles L, M with unique minimal events $\min_L, \min_M$ respectively, and an MSC N such that $LMN \in \mathcal{L}(G)$. Assume moreover that $\min_M$ is the unique cut-event of LMN within M. Then $|M| \leq 2\wp|G|$.*

We denote a triangle M as a *G-triangle without cut-events* if L, N exist satisfying the above assumptions. Notice that LM can have more than one cut-event within M, but they are not cut-event for LMN.

Proof. Assume by contradiction that $|M| > 2\wp|G|$. Then M intersects more than $2\wp$ nodes. We apply Lemma 2, yielding that (the minimal event of) a node v_i is in M. Hence M contains a cut-event of LMN different of $\min_M$, which is a contradiction. □

The next proposition shows that no execution of an extended local-choice have two long MSC factors in parallel.

Proposition 8 *Let G be an extended local-choice HMSC. Let $LMN \in \mathcal{L}(G)$, with $M = R \parallel S$ an MSC. Then either $|R| \leq 2\wp|G|$ or $|S| \leq 2\wp|G|$.*

We denote R, S as *two parallel MSC-factors in G* if L, N exist satisfying the above assumptions.

Proof. Assume by contradiction that $\lambda(\rho) = LMN$ for ρ a path of G. Moreover M is an MSC with $M = R \parallel S$, $|R| > 2\wp|G|$ and $|S| > 2\wp|G|$. Hence by lemma 2, R and S both contain (the minimal event of) some node of ρ. Since the minimal events of the nodes of ρ are totally ordered, we obtain a contradiction. □

An immediate consequence of the previous propositions is that a globally-cooperative HMSC containing an execution with an infinite MSC-factor without cut-event, or two infinite parallel MSCs, cannot be equivalent to an extended local-choice HMSC. We show now that the converse also holds.

Proposition 9 *Let G be a globally-cooperative HMSC and b an integer with the following properties:*

1. *Every $M \in \mathcal{L}(G)$ is a triangle.*
2. *If M is a G-triangle without cut-events, then $|M| \leq b$.*
3. *If M, N are two parallel MSC-factors in G, then either $|M| \leq b$ or $|N| \leq b$.*

Then $\mathcal{L}(G) \in \mathcal{L}(H_{\wp b})$.

Proof. Let G satisfying the assumptions of the theorem and $M \in \mathcal{L}(G)$. We show how to decompose M as a path ρ of $H_{\wp b}$.

Let E be the set of cut-events of M, except for min_M, and F the minimal events in E. Since all events in F are incomparable cut-events, their futures are in parallel due to Lemma 1. By the third condition, there exists $e \in F$ such that $|\mathit{future}(f)| \leq b$ for all $f \neq e$. Using the second condition we obtain $|M \setminus \mathit{future}(e)| \leq b + \sum_{f \neq e, f \in F} |\mathit{future}(f)| \leq \wp b$. Hence we define the first node of ρ as the node of $H_{\wp b}$ labeled by $M \setminus \mathit{future}(e)$. We proceed recursively with $\mathit{future}(e)$. □

Notice that if $G \in H_{\wp b}$, then we can apply Theorem 2 to build an extended local-choice HMSC equivalent to G. We now show how to test effectively the assumptions of proposition 9. This will provide a bound on b.

Proposition 10 *Let G be a globally-cooperative HMSC. We can check that every $M \in \mathcal{L}(G)$ has a unique minimal event in logarithmic space.*

Proposition 11 *Let G be a globally-cooperative HMSC. Assume that M, N are two parallel MSC-factors in G with $|M| > 2\wp|G|$ and $|N| > 2\wp|G|$. Then there exist two parallel infinite MSC-factors in G. Moreover, the condition can be checked in co-NP.*

Proof. Assume that $M = R \parallel S$ with $|R| > 2\wp|G|$ and $|S| > 2\wp|G|$. Then R intersects more than $2\wp$ loops of the globally-cooperative HMSC G. We then apply lemma 2, which provides us a loop of G labeled by events of R. In the same way, we obtain a loop of G labeled by events of S. We can iterate an infinite number of times these loops and achieve 2 infinite MSCs in parallel in G.

To test such a property, we just have to guess 2 simple loops $\mu \parallel \nu$ and a loop-free path between the two loops. □

Proposition 12 *Let G be a globally-cooperative HMSC. We can build effectively an automaton $\mathcal{A}(G)$ recognizing representative linearizations of each triangle of G without cut-event, and only them. This automaton is at most exponential in $|G|$, and allows to test the finiteness of G-triangle without cut-event in PSPACE.*

Hence, we obtain as corollary:

Theorem 4 *We can test in PSPACE whether an HMSC is equivalent to an extended local-choice HMSC.*

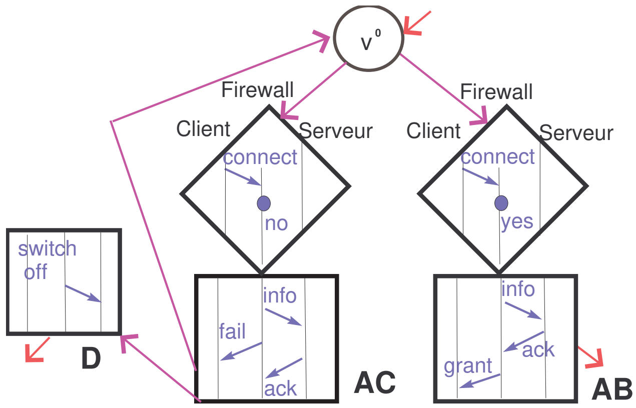
We recall that checking whether an HMSC is globally-cooperative is co-NP-hard.

Notice that the tests give us a bound b, at most exponential in $|G|$, s.t. $G \in H_{2b}$. We can effectively construct an extended local-choice HMSC exponential in b equivalent to G. This can yield a 2 exponential extended local-choice HMSC in the size of $|G|$. Anyway, we think this worst case is really unlikely.

7 Discussion about implementation as conclusion

(Extended) local-choice HMSC were introduced for their suitability for implementation, either by adding data [7], or not [8]. The reason is that HMSCs are drawn with a global control (eagle-eye), while practical implementations are locally controlled. Moreover, algorithms tend to be more efficient for (extended) local-choice HMSC than in the general case. Moreover we have shown how to test effectively whether an HMSC is equivalent to any local-choice HMSC, which makes this class of great practical innterest.

We recall in the appendix how to implement extended local-choice HMSCs without deadlocks when adding data on messages. To sum up, the way we implement local-choice HMSC is to declare the minimal event of a node as a leader. This can be sometimes unrealistic because this particular process is not allowed to choose for the others (a client cannot choose whether a server should grant him the connection). Anyway, this kind of problem can be ruled out by enhancing specification. We can for instance define nodes to be triggered nodes (drawn as diamond in the example), while others represent the reaction of the system (akin to triggered MSCs [13] and LSCs [6]).



In such a specification, the client would be in a triggered node since it does not decide for the system. Then we could enhance extended local-choice conditions to this kind of specification, namely

1. If H' is the HMSC obtained from H by deleting triggered nodes, then H' is extended local-choice.
2. For distinct nodes v, c, r, c', r' (c,c' triggered nodes associated with their respective reactions r, r') with $v \to c \to r, v \to c' \to r'$, the sequence of events on process $P(\min(r'))$ of $\lambda(c')$ is not a prefix of $\lambda(c)$ on this process.

With these conditions, such an HMSC is always implementable with polynomial additional data.

For the case where additional data is not allowed, even local-choice HMSCs are not always implementable (see [8] or the local-choice HMSC of figure 1), which in some sense shows the lacks of this model of implementation (trivially implementable HMSCs are not captured by this implementation). Anyway, when such an implementation is possible, we should use it. The paper [8] shows a testable subclass of local-choice HMSC that is implementable without more data and deadlock. This implementation was called strong realizability in the framework of [3]. Later, the author of [10] gave an EXPSPACE-complete algorithm to test whether a globally-cooperative HMSC is strongly realizable. For extended local-choice HMSC, we obtain:

Theorem 5 *We can test whether an extended local-choice HMSC L is strongly realizable in PSPACE.*

Proof. Let $L|_p$ be the HMSC projection of L on process p for all p. Then L is strongly realizable iff $L = \prod_p L_p$. Because one inclusion is trivial, we just have to test the other using theroem 3. □

References

1. ITU-TS recommendation Z.120, 1996.
2. R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *Proceedings of the 22nd International Conference on Software Engineering*, pages 304–313. ACM, 2000.
3. R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of MSC graphs. In *ICALP'01*, LNCS 2076, pages 797–808, 2001.
4. R. Alur and M. Yannakakis. Model checking of message sequence charts. In *CONCUR'99*, LNCS 1664, pages 114–129, 1999.
5. H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In *TACAS'97*, LNCS 1217, pages 259–274, 1997.
6. W. Damm and D. Harell. LSCs: Breathing Life into Message Sequence Charts In *FMSD* 19, Kluwer Academic Publishers, 2001.
7. B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun. Infinite-state High-Level MSCs: Model-Checking and Realizability In *ICALP'02*, LNCS 2380, 2002.
8. L. Hélouët and C. Jard. Conditions for synthesis of communicating automata from HMSCs. In *5th Int. Workshop on Formal Methods for Ind. Crit. Systems*, 2000.
9. L. Hélouët and P. Le Maigat. Decomposition of Message Sequence Charts. In *SAM2000*, pages 46–60, 2000.
10. M. Lohrey. Safe realizability of High-Level Message Sequence Charts. In *CONCUR'02*, 2002.
11. M. Mukund, K. Narayan Kumar, and M. Sohoni. Synthesizing distributed finite-state systems from MSCs. In *CONCUR'00*, LNCS 1877, pages 521–535, 2000.
12. A. Muscholl and D. Peled. Message sequence graphs and decision problems on Mazurkiewicz traces. In *MFCS'99*, LNCS 1672, pages 81–91, 1999.
13. B. Sengupta and R. Cleaveland Triggered Message Sequence Charts. In SIGSOFT2002/FSE-10, November 2002. ACM Press.

Appendix

proposition Let G be a globally-cooperative HMSC. We can build effectively an automaton $\mathcal{A}(G)$ recognizing representative linearizations of each triangle of G without cut-event, and only them. This automaton is at most exponential in $|G|$, and allows to test the finiteness of G-triangle without cut-event in PSPACE.

Proof.

We replace each node of G by a succession of nodes labeled by a chosen representative linearization.

We define now the finite automaton $\mathcal{A}(G)$. Each state w of $\mathcal{A}(G)$ is of the form $[v, \min, X, Y, F]$, where $v \in V$ is a state of G, min is an event, X, Y are sets of at most $\wp^2, \wp$ events. The function F associates to each $x \in \{\min\} \cup X \cup Y$ the set of processes $F(x)$ seen so far in its future. The idea is to travel on G until guessing the minimal event min of M, then producing the events (set X) seen in the future of min with $F(\min)$. We keep track of $F(e)$ for all the events produces, and we show that there are at most $\wp^2$ events to remember in X. At some point we can guess that an event and its future is no more in M. There is at most $\wp$ such events (set Y). We have to check that min is a cut-event, but no events produced is a cut-event. This is done using F. We finish when v is a final state of G and the checks are successful.

We define now the transitions of $\mathcal{A}(G)$ as follows. We do not describe the update of F since it is obvious. Assume that $v \xrightarrow{e} v'$ is a transition of G and let $p = P(e)$. We let $[v, \min, X, Y, F] \xrightarrow{\ell} [v', X', F']$, with X', F', obtained through the following algorithm:

1. If $\min = \epsilon$,
   - If (guessed $\min = e$) then set $\min = e$ and $\ell = m$
   - else $\ell = \epsilon$
2. Else If $e \notin \bigcup_{y \in Y} F(y)$
   - If (guessed e is nomore in M) then set $Y = Y \cup \{e\}$ and $\ell = \epsilon$
   - else $X = X \cup \{e\}$ and $ell = e$
3. Else $\ell = \epsilon$.
4. If $p \in F(\min)$ and e is a receive associated to send s,
   - Check whether $s \in \mathit{future}(\min)$
   - If not, stop (min is not a cut-event).
5. For all $x, x' \in X$ s.t. $F(x) = F(x')$, remove x from X.

Moreover, a state $[v, \epsilon, \emptyset, \emptyset, \emptyset]$ is initial if v is. A state $[v, T, S, k, X]$ is final if v is and $F(x) = \emptyset$ for all $x \in X$ (there is no cut-event in M).

By construction, $|Y| < \wp$. Moreover, $X < \wp^2$ since there is at most $\wp$ different set $F(x)$ for events on the same process (theses sets are include one in others).

Testing whether an element of an automaton is NLOGSPACE, yielding a PSPACE algorithm for this problem. □

Here is the proof for implementation of a local-choice HMSC.

Proof. Roughly speaking, the idea of the implementation for *local-choice* HMSCs is that when the execution branches, the minimal process of the execution knows that it has to choose the next node:

If process p is not the minimal process, then it goes to a polling state until it receives a message, thus learning what node to execute currently. Here is the algorithm for the polling state of process p.

```
while (true) {
      if p receives a message containing v
            { current_node=v; return;} }
```

Initially, the current node is initialized by letting current_node $= v^0$.

The algorithm execute_path(w) used below means that process p executes $\pi_p(w)$ (its events in node w). In this case each message sent by p contains in addition w. Here is the algorithm for each process p.

```
initialization;
 while (true)
 { if (p ≠  root(current_node) ) go to polling state;
   else {v=guess(current_node); current_node=v;}
   execute_path(current_node);
 }
```

□