Infinite-State High-Level MSCs: Model-Checking and Realizability
(Extended abstract)

Blaise Genest[1*], Anca Muscholl[1*], Helmut Seidl[2], and Marc Zeitoun[1*]

[1] LIAFA, Université Paris VII
2, pl. Jussieu, case 7014, 75251 Paris cedex 05, France
[2] FB IV, Universität Trier, 54286 Trier, Germany

Abstract. We consider three natural classes of infinite-state HMSCs: globally-cooperative, locally-cooperative and local-choice HMSCs. We show first that model-checking for globally-cooperative and locally-cooperative HMSCs has the same complexity as for the class of finite-state (bounded) HMSCs. Surprisingly, model-checking local-choice HMSCs turns out to be exponentially more efficient in space than for locally-cooperative HMSCs. We also show that locally-cooperative and local-choice HMSCs can be always implemented by communicating finite states machines, provided we allow some additional (bounded) message data. Moreover, the implementation of local-choice HMSCs is deadlock-free and of linear-size.

1 Introduction

Message sequence charts (MSC) is a visual notation for asynchronously communicating processes and a standard of the ITU [1]. The usual application of MSCs in telecommunication is for capturing requirements of communication protocols in form of scenarios at early design stages. MSCs usually represent incomplete specifications, obtained from a preliminary view of the system that abstracts away several details such as variables or message contents. High-level MSCs (HMSCs) combine basic MSCs using choice and iteration, thus describing possibly infinite collections of scenarios. From the viewpoint of automatic verification, high-level MSCs are infinite-state systems. Moreover, certain basic questions as model-checking against HMSC properties [5, 14] are undecidable.

A preliminary specification of a communication protocol can suffer from several deficiencies, either related to the partial order of events (e.g. race conditions [4, 14]) or to the violation of user-defined properties specified in some logics or HMSCs (model-checking [5]). The detection of possible failures in early design stages is of critical importance, and the utility of HMSCs can be greatly enhanced by automatic validation methods. A natural question for HMSC specifications is to test whether the specification is implementable (or realizable) [2]

* Work partly supported by the European research project IST-1999-29082 ADVANCE and by the INRIA/IRISA ARC project FISC.

by a communication protocol and to construct such an implementation. Since an abstract communication protocol is usually described by communicating finite-state machines (CFM), we test implementability against CFMs. Opposed to HMSC specifications, no global control is available in CFMs. In order to install a distributed control the CFM realization therefore may have to add further data to messages or even exchange additional (synchronization) messages. Our goal is to exhibit general techniques for synthesizing such distributed control. Once an implementation is available, one can easily simulate executions of the HMSC using the ITU-standard model SDL (Specification and Description Language) and use SDL tools for model-checking the HMSC specification.

We consider in this paper three natural classes of infinite-state HMSCs: globally-cooperative, locally-cooperative and local-choice HMSCs. Locally-cooperative and local-choice HMSCs are (syntactically incomparable) subclasses of globally-cooperative HMSCs. The crucial property of globally-cooperative HMSCs is there exists a *regular* set of representative behaviors, however buffers are unbounded (making the system infinite-state). For telecommunication applications it is of course essential to cope with unbounded communication buffers. Globally-cooperative HMSCs have been introduced independently in [12], whereas locally-cooperative HMSCs are defined in this paper. The local-choice property we use here has been considered in [8].

In the first part of the paper (Section 3) we consider the model-checking problem and we show that it is decidable for globally-cooperative HMSCs. The model-checking problem is stated as intersection (negative property) or inclusion (positive property) of HMSCs, that is, the property to be tested is also described by an HMSC. Recall that both questions are undecidable for unrestricted HMSCs, which has been the motivation for considering regular (finite-state) HMSCs [5, 14, 10]. Here, we show that negative and positive model-checking are PSPACE- and EXPSPACE-complete, respectively, for both globally-cooperative and locally-cooperative HMSCs — thus generalizing the complexity bounds for regular HMSCs to infinite-state HMSCs. For local-choice HMSCs we obtain better algorithms, specifically negative model-checking is quadratic time whereas positive model-checking is PSPACE-complete.

In the second part of the paper we consider the synthesis of communicating finite-state machines from locally-cooperative, resp. local-choice HMSCs (Sections 4 and 4.3). We adopt a moderate view by allowing additional message contents, while ruling out extra control messages. The reason is that additional messages mean additional process synchronization. This is not desirable, or even not realizable, in a given environment. Still, our implementation semantics by CFMs is more general than the one introduced in [2] and used in [3, 12] where a parallel product of finite-state automata communicating over FIFO channels is employed to realize the (linear) behavior of each process of the given HMSC. Moreover, implementability in this framework is undecidable even for bounded HMSCs (however, deadlock-free implementability is in EXPSPACE). Other notions of implementation are proposed in [7],[8],[13]. We show that both HMSCs classes are *always implementable* by CFMs. Moreover, the CFM implementation of local-choice HMSCs is deadlock-free and of linear-size.

The proofs use standard techniques from Mazurkiewicz trace theory as well as specific partial orders methods. Proofs are in most cases omitted in this abstract, due to lack of space.

2 Preliminaries

In this section we recall the specification formalism of message sequence charts (MSC) and high-level message sequence charts (HMSC) based on the ITU standard Z.120 [1]. An MSC describes a scenario or an execution of a communication protocol in which processes communicate with each other over point-to-point channels. Such a scenario is given by a description of the messages sent and received, the local events, and the ordering between them. The event ordering is based on a process ordering and a message ordering. In the visual description of MSCs, each process is represented by a vertical line, which shows the total order on the events belonging to 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 $M = \langle E, <, \mathcal{P}, t, \mathcal{C}, m \rangle$ where:

- $E = \dot{\bigcup}_{p \in \mathcal{P}} E_p$ 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 $E = S \dot{\cup} R \dot{\cup} L$ 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.
- $t : E \to A = \{p!q(a), p?q(a), l_p(a) \mid p, q \in \mathcal{P}, p \neq q,\ a \in \mathcal{C}\}$ labels each event by its *type* $t(e)$, with $t(e) = p!q(a)$ if $e \in E_p \cap S$ is a send event of message a from p to q, $t(e) = p?q(a)$ if $e \in E_p \cap R$ is a receive event of message a by p from q and $t(e) = l_p(a)$ if $e \in E_p \cap L$ is a p-local event describing the local action a.
- $m : S \longrightarrow R$ is a bijection that pairs up send and receive events (*matching* function). We have that $m(e) = f$ only if $t(e) = p!q(a)$, $t(f) = q?p(a)$ for some $p, q \in \mathcal{P}$ and $a \in \mathcal{C}$.
- $< \subseteq E \times E$ 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 $e < f$.

A *message* (e, f) is a pair of matching send and receive events, i.e., $m(e) = f$. Often one assumes that channels are FIFO, that is, there is no overtaking of messages in the channel from p to q, for every $p \neq q$. The results of this paper are depending on this assumption. For non-FIFO channels we just have to add some information in the type $t(e)$ of an event e. Formally, we extend the type of each event by an integer. We require that $m(e) = f$ only if $t(e) = (p!q(a), k)$ and $t(f) = (q?p(a), k)$ for some p, q, a and $k \in \mathbb{N}$.

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., $\leq \subseteq \preceq$. For any MSC M we denote by $\mathrm{Lin}(M)$ the set of labeled linearizations of M: $\mathrm{Lin}(M) = \{t(e_1) \cdots t(e_k) \mid e_1 \cdots e_k$ is a linearization of $M\}$. Note that any $x \in \mathrm{Lin}(M)$ suffices to reconstruct the MSC M, since the type mapping $t : E \to A$ encodes all the relevant information about M. If the matching m is a partial function then we speak about a *partial MSC*. For every $x \in A^*$ we denote by $\mathrm{msc}(x)$ the (partial) MSC defined by x, if it exists. The *size* of an MSC is the number of events it contains.

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* $G = \langle V, R, v^0, v^f, \lambda\rangle$ (HMSC for short) is a finite transition system (V, R, v^0, v^f) with transition set $R \subseteq V \times V$, initial node v^0 (with no ingoing edge) and terminal node v^f (with no outgoing edge). Each node v is labeled by the finite MSC $\lambda(v)$. We let $P(v) = P(\lambda(v))$, and we assume that $P(v) \neq \emptyset$, except possibly for $v = v^f$. 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 labeling $\lambda(v_0)\lambda(v_1) \cdots \lambda(v_k)$ of some path $v^0 = v_0, v_1, \ldots, v_k = v^f$ in G, i.e., $(v_i, v_{i+1}) \in R$ for every $0 \leq i < k$. 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.

The semantics of HMSCs depends on the definition of the MSC product. We consider the usual *weak product* of MSCs, as defined in the following. Let $M_1 = \langle E_1, <_1, \mathcal{P}, t_1, \mathcal{C}_1, m_1\rangle$ and $M_2 = \langle E_2, <_2, \mathcal{P}, t_2, \mathcal{C}_2, m_2\rangle$ be MSCs over the same set of processes $\mathcal{P}$. Their product M_1M_2 is defined as the MSC $\langle E_1 \cup E_2, <, \mathcal{P}, t_1 \cup t_2, \mathcal{C}_1 \cup \mathcal{C}_2, m_1 \cup m_2\rangle$ over the disjoint union of events $E_1 \cup E_2$, with the visual order given by:

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

That is, events of M_1 precede the events of M_2 for each process, respectively. Note that there is no synchronization between different processes when moving from one node to the next one (this is called *weak sequencing*). Hence, 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. We also say that M_1 is a *prefix* of M_1M_2.

3 Model-Checking Cooperative HMSCs

In this section we introduce globally-cooperative HMSCs and the subclass of locally-cooperative HMSCs and we show that the model-checking problem for these infinite-state HMSCs is decidable.

For MSCs M_1, M_2 with $P(M_1) \cap P(M_2) = \emptyset$, we write $M_1 \parallel M_2$ and we say that M_1, M_2 are *independent*. Observe that $M_1 \parallel M_2$ implies $M_1M_2 = M_2M_1$.

An MSC M is called *linked* if it cannot be written as $M = M_1 M_2$ with $M_1 \parallel M_2$ both non empty MSCs. If v_1, v_2 are nodes of an HMSC $G = \langle V, R, v^0, v^f, \lambda\rangle$, we write $v_1 \parallel v_2$ if $\lambda(v_1) \parallel \lambda(v_2)$. The relation $\mathrm{Sync} = V \times V \setminus \parallel$ is called the *synchronization relation*.

Globally-cooperative and **locally-cooperative HMSCs**

Let $G = \langle V, R, v^0, v^f, \lambda\rangle$ be an HMSC with nodes labeled by linked MSCs.

1. G is called *globally-cooperative* if no strongly connected R-component $U \subseteq V$ can be partitioned as $U = U_1 \cup U_2$ such that $U_1, U_2 \neq \emptyset$ and $v_1 \parallel v_2$ for all $v_1 \in U_1$, $v_2 \in U_2$.
2. G is called *locally-cooperative* if $R \subseteq \mathrm{Sync}$.

In particular, any MSC labeling a loop of a globally-cooperative HMSC is linked. We will assume in the sequel that all nodes of an HMSC are labeled by linked MSCs. This is not a restriction, since any non-linked node can be split in a sequence of linked nodes. Note that this transformation preserves both classes of regular and globally-cooperative HMSCs.

The motivation behind the definition of globally-cooperative HMSCs comes from Mazurkiewicz trace theory. Let $\parallel \subseteq V \times V$ be a symmetric, irreflexive independence relation on V. A set $K \subseteq V^*$ is called $\parallel$-*closed* when $\sigma uv\sigma' \in K \Leftrightarrow \sigma vu\sigma' \in K$ for all $\sigma, \sigma' \in V^*$ and $u \parallel v$. The $\parallel$-*closure* of $K \subseteq V^*$ is the smallest $\parallel$-closed set containing K and is denoted $\mathrm{clos}_\parallel(K)$. In general, the closure does not preserve regularity. A sufficient condition for $\mathrm{clos}_\parallel(\mathcal{L}(A))$ being regular is that the set of edge labels $U \subseteq V$ occurring in any strongly connected component of $\mathcal{A}$ induces a connected subgraph of $(V, (V \times V) \setminus \parallel)$ [11, 15]. Moreover, the size of a non-deterministic automaton recognizing $\mathrm{clos}_\parallel(\mathcal{L}(\mathcal{A}))$ is at most $2^{O(n \cdot \wp)}$ [5, 14], where $n = |\mathcal{A}|$ and $\wp$ is the minimal number of cliques covering the graph $(V, (V \times V) \setminus \parallel)$. Testing that an automaton $\mathcal{A}$ satisfies the above condition is co-NP-complete, [5, 14]. This yields:

Proposition 1. *Checking whether an HMSC is globally-cooperative is co-NP-complete, whereas checking whether it is locally-cooperative is in P.*

Notice that globally-cooperative and locally-cooperative HMSCs have in general an infinite state space, thus they are not *bounded* in the sense of [5, 14]. Formally, the set of linearizations of executions cannot be described by finite automata. However, although globally-cooperative HMSCs are more general than bounded HMSCs, we are still able to do automata-based model-checking. The underlying idea is that the executions of a globally-cooperative HMSC can be captured by a regular set of representatives. As an example consider the HMSC G of Figure 1. The set $\mathrm{Lin}(G)$ of linearizations of executions of G is obviously non-regular (consider e.g. linearizations in the set $A_p^*(A_r + A_s)^* A_q^*$, where $A_p = \{t(e) \mid P(e) = p\}$ is the set of types of events located on p). A regular set of representatives is given by the set $\mathrm{Lin}(G) \cap (A_p(A_r + A_s)A_q)^*$.

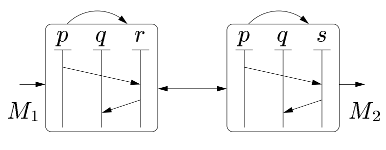

Fig. 1. A locally-cooperative HMSC

A non empty MSC is called *atomic* (atom, for short) if it cannot be written as $M = M_1 M_2$ for non empty MSCs M_1, M_2. It is not hard to see that any MSC M can be written as $M = M_1 \cdots M_k$ where each M_i is a non empty atom, and that this factorization is unique up to commutation of adjacent independent atoms. In [9], it is shown how to compute this factorization in linear time. Note that any atomic MSC is linked. Note that replacing any node by a sequence of atoms preserves globally-cooperative, but not locally-cooperative HMSCs, as shown through Proposition 2.

Let $G = \langle V, R, v^0, v^f, \lambda\rangle$ be an HMSC. We define $\mathrm{Atom}(G)$ as the set of atoms occurring in the decomposition of MSCs from $\lambda(V)$. Let $\mathrm{Lin}^a(G) = \mathrm{Lin}(G) \cap \mathrm{Lin}(\mathrm{Atom}(G))^*$, where $\mathrm{Lin}(\mathrm{Atom}(G)) = \bigcup_{M\in \mathrm{Atom}(G)} \mathrm{Lin}(M)$.

We show now that $\mathrm{Lin}^a(G)$ is a regular set of representatives for $\mathrm{Lin}(G)$.

Theorem 1. *Let G be a globally-cooperative HMSC. Then $msc(Lin^a(G)) = \mathcal{L}(G)$. Let s denote the size of G, $\wp$ the number of processes, and let μ be the maximal number of events on the same process in an MSC from $Atom(G)$. Then:*

1. *$Lin^a(G)$ is recognized by a non-deterministic finite automaton of size in $2^{O(\wp s)}(\mu + 1)^{\wp}$.*
2. *Moreover, if G is locally-cooperative, then the size of the automaton is in $s^{O(\wp)}(\mu + 1)^{\wp}(\wp + 1)^{\wp}$.*

The lower bounds below follow constructions similar to [14]:

Corollary 1. *Model-checking globally-cooperative HMSCs is decidable. Precisely, let G_1, G_2 be globally-cooperative HMSCs, then we have:*

1. *Deciding whether $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) \neq \emptyset$ is a PSPACE-complete problem.*
2. *Deciding whether $\mathcal{L}(G_1) \subseteq \mathcal{L}(G_2)$ is an EXPSPACE-complete problem.*

Both lower bounds hold also when G_1, G_2 are locally-cooperative.

We obtain better complexity bounds for the model-checking problems for locally-cooperative HMSCs if the nodes are labeled by *atomic* MSCs. For this we use the unicity of the decomposition of an MSC into atoms:

Proposition 2. *Let G_1, G_2 be locally-cooperative HMSCs such that each node is labeled by an atomic MSC. Then:*

1. *Deciding whether $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) \neq \emptyset$ is a NLOGSPACE-complete problem.*
2. *Deciding whether $\mathcal{L}(G_1) \subseteq \mathcal{L}(G_2)$ is an PSPACE-complete problem.*

3.1 Local-choice HMSCs

An important aspect of implementation is the absence of deadlocks. So we are led to consider HMSCs satisfying the *local-choice* property [8]. Roughly speaking, local-choice ensures that branching between executions is always controlled by a unique process.

Local-choice HMSCs. An HMSC $N = \langle V, R, v^0, v^f, \lambda\rangle$ is called *local-choice* if the following conditions are satisfied:

1. Every path starting in v^0 has a unique minimal event.
2. For each node $v \in V$ having at least two outgoing edges, there is a process $root(v)$ such that every path $w_1 w_2 \cdots$, starting in a node w_1 successor of v, has a unique minimal event located on $root(v)$.

It is easy to see that locally-cooperative and local-choice HMSCs are syntactically incomparable. However, local-choice HMSCs are globally-cooperative. Actually we can transform local-choice HMSCs into locally-cooperative, local-choice HMSCs of quadratic size, as shown in Proposition 3 below.

We call a node with at least two outgoing edges a *branching node*. Notice that every path $v_0 \cdots v_l$ in N where all of $v_1, \ldots, v_n$ are non-branching, is of length $l+1 \leq |V|$. Such a path will be called a *non-branching path*. Moreover, if the non-branching path $\sigma = v_0 \cdots v_l$ is maximal, then it is determined by v_0 and v_l is either branching or the terminal node v^f. We denote the maximal non-branching path starting in v by $\mathrm{NPath}(v)$. Consider now an accepting path σ of N. We decompose σ as $\sigma = \sigma_0 \sigma_1 \cdots \sigma_{k+1}$ where each σ_i is a maximal non-branching path (note that this decomposition is unique). Let v_i be the last node of σ_{i-1} (see Figure 2, where the triangles illustrate the partial order graphs of the subpaths σ_i). Note that v_i is branching for all $i \leq k$. Let also w_i be the first node of σ_i, hence $\sigma_i = \mathrm{NPath}(w_i)$. By definition, $p_i = root(v_i) \in P(w_i)$ is the process on which the minimal event of σ_i is located (which is the unique minimal event of the path $\sigma_i \cdots \sigma_k$). Moreover, the local choice condition applied to the branching node v_{i-1} ensures that p_i also belongs to $P(\sigma_{i-1})$. If $p_i \neq p_{i-1}$ then the first action of p_i in σ_{i-1} is a receive action.

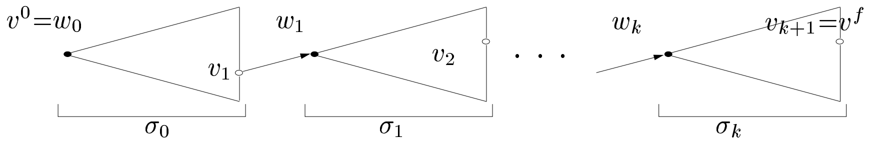

Fig. 2. Path decomposition in a local-choice HMSC.

The above decomposition of paths in a local-choice HMSC will be used by the implementation algorithm (Section 4.3).

Proposition 3. *For every local-choice HMSC we can construct an equivalent locally-cooperative, local-choice HMSC of quadratic size.*

The main technical argument establishing the model-checking algorithm for local-choice HMSCs is based on the following property. Consider an equality $M_1 \cdots M_k = M'_1 \cdots M'_l$, where every $M_i \cdots M_k$, resp. $M'_j \cdots M'_l$ has a unique minimal event, for all i, j. Then we can show that either M_1 is a prefix of M'_1 (or vice-versa) or $M_1 = X M'_2 \cdots M'_l$, $M'_1 = X M_2 \cdots M_k$ hold for some MSC X.

This observation allows to consider only configurations with a unique minimal event (instead of arbitrary configurations, which would require polynomial space).

Theorem 2. *Let G and G' be two local-choice HMSCs. Then we have:*

1. *Deciding whether $\mathcal{L}(G) \cap \mathcal{L}(G') \neq \emptyset$ is NLOGSPACE-complete. Moreover, this question can be decided in time $O(|G| \cdot |G'|)$.*
2. *Deciding whether $\mathcal{L}(G) \subseteq \mathcal{L}(G')$ is PSPACE-complete.*

4 Implementing HMSCs by Communicating Finite-State Machines

4.1 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 buffers. In general we assume that buffers are FIFO (if for instance the given HMSC is FIFO), but we can modify the semantics of receives if the MSCs contain overtaking of messages. 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 $\mathcal{A}_p = (S_p, A_p, \rightarrow_p, F_p)$ 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 $\rightarrow_p \subseteq S_p \times A_p \times S_p$. 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. We use the same notations as for MSCs. Sending 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 $B_{q,p}$ (supposing FIFO). In the non-FIFO case we specify the type of the message that can be received next (cf. the semantics of a receive in the *message queue* of UNIX system V). 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 $\sum_p |\mathcal{A}_p|$.

A CFM implementation of an HMSC N will add in general data to the message contents of N. We will call $\mathcal{A}$ a *CFM implementation* of N if the MSCs defined by the successful runs of $\mathcal{A}$ with the additional data removed, correspond precisely to the executions of N.

4.2 Implementing locally-cooperative HMSCs

The simplest realization of an HMSC N by a CFM is the one where the automaton $\mathcal{A}_p$ corresponding to process p generates the projection of $\mathcal{L}(N)$ on p. This approach is used in [3, 12]. Consider again the HMSC G_1 of Figure 1 (page 5), and let M be the MSC given by the projections $\pi_p(M) = p!r\ p!r\ p!s\ p!s$, $\pi_q(M) = q?r\ q?s\ q?r\ q?s$, $\pi_r(M) = r?p\ r!q\ r?p\ r!q$ and $\pi_s(M) = s?p\ s!q\ s?p\ s!q$. Then M

does not belong to $\mathcal{L}(G_1)$. However, it is easy to verify that $\pi_t(M) \in \pi_t(\mathcal{L}(G_1))$ for all $t \in \{p, q, r, s\}$. Hence G_1 is not realizable according to [3].

We describe our implementation of locally-cooperative HMSCs first on our example G_1. One can observe that G_1 can be implemented if process p anticipates the next choice and sends the prediction with the current message. Processes r and s then forward the prediction to q. In this way, process q knows whether the next message should be received from r or from s. The general solution will involve a leader process (p in the example) for each transition, i.e., a process that occurs in both nodes and decides about certain nodes in the future (prediction), as described below.

For a node $v \in V$ of $N = \langle V, R, v^0, v^f, \lambda \rangle$, let $P(v)$ denote the processes occurring in $\lambda(v)$. For a path $\sigma = v_0 v_1 \cdots v_k$ of N let $P(\sigma) = \bigcup_i P(v_i)$ be the processes occurring in σ. Moreover, we define $\mathsf{first}(\sigma, p)$ for all $p \in \mathcal{P}$ as the first node containing p in σ: $\mathsf{first}(\sigma, p) = \perp$ if $p \notin P(\sigma)$, and $\mathsf{first}(\sigma, p) = v_j$, where $j = \min\{k \geq 0 \mid p \in P(v_k)\}$ otherwise. Similarly, if σ has at least $i + 1$ nodes, let $\mathsf{last}(\sigma, i, p)$ be the last node among the first $(i + 1)$ nodes of σ containing process p (resp., $\mathsf{first}(\sigma, p) = \perp$ if $p \notin P(v_0 \cdots v_i)$).

Let $N = \langle V, R, v^0, v^f, \lambda \rangle$ be a locally-cooperative HMSC. A triple $(v, \nu, l) \in V \times (V \cup \{\perp\})^{\mathcal{P}} \times \mathcal{P}$ is a *realizable prediction* if either $\nu(p) = \perp$ for all $p \in \mathcal{P}$ and $v = v^f$, or if all conditions below hold:

1. There exists a path $\sigma = v_0 v_1 \cdots$ in N such that $v_0 = \nu(l)$ and $\nu(p) = \mathsf{first}(\sigma, p)$ for each process p;
2. $(v, \nu(l)) \in R$;
3. $l \in P(v) \cap P(\nu(l))$.

The process l is called the *leader* of the transition $(v, \nu(l))$ with respect to (v, ν, l).

From a locally-cooperative HMSC $N = \langle V, R, v^0, v^f, \lambda \rangle$, we build a communicating automaton $\mathcal{A}_N$ as follows. Each process is initialized with the same input $i_0 = (v_0, \nu_0, l_0)$ which is some realizable prediction with $v_0 = v^0$. The algorithm for process p is described below:

```
(v,ν,ℓ) =  (v₀,ν₀,ℓ₀);
while (true)
{ m = (v,ν,ℓ);
if (p ∈ P(v))     // test useful only for the first node of p
execute(v,m);
v' = ν(p);
if (v' == ⊥)
halt();
if (v' == ν(ℓ))        // v' is the successor of v
(ν',ℓ') = guess_next(v',ν);
else (ν',ℓ') = guess(v');
v = v'; ν = ν'; ℓ = ℓ';}
```

The call guess_next(v$'$, ν) guesses nondeterministically a prediction and a leader (ν', ℓ') for the next node v', such that $(\mathtt{v}', \nu', \ell')$ is realizable and the new prediction ν' is *compatible* with the old prediction ν for processes not occurring in v', that is, $\nu_{|\mathcal{P}\setminus P(v')} = \nu'_{|\mathcal{P}\setminus P(v')}$. The call `guess(v')` guesses nondeterministically a pair (ν', ℓ') such that $(\mathtt{v}', \nu', \ell')$ is realizable. In this case, process p makes a prediction about a node p' that is not a direct R-successor of v. This prediction is needed since all processes of a node must agree on some future information. The call `halt()` terminates the execution of `p` in an accepting state. Finally, the call `execute(v,m)` consists in executing the actions of `p` of the MSC labeling `v`, but overloading the messages to be sent or received with `m`. Note that if two communicating processes do not choose the same value for `m`, then a deadlock occurs. By transitivity and weak connectivity of each MSC, the deadlock-free execution of a node means that all processes in the node have chosen the same value for `m`.

Proposition 4. *Let N be a locally-cooperative HMSC. Then $\mathcal{A}_N$ is a CFM implementation of $\mathcal{L}(N)$ of size $n^{O(\wp)}$, where n is the number of nodes of N and $\wp$ is the number of processes.*

Remark 1. Note that we can fix a leader for each transition of the HMSC beforehand. This would decrease the degree of non-determinism and deadlocks.

4.3 Implementing local-choice HMSCs

The implementation algorithm described in the previous section cannot avoid deadlocks for the resulting CFM, since the future predictions are chosen by each process separately. One reason is that branching in an HMSC is not controlled by a single process, as it is the case for local-choice HMSCs. The results of [8] give a sufficient condition (called *reconstructibility*) for a local-choice HMSC to be implementable with no addition of extra message data.

Recall from Section 3.1 that any accepting path σ of a local-choice HMSC has a canonical decomposition $\sigma = \sigma_0 \cdots \sigma_k$ in subpaths σ_i, such that the MSC execution of σ_i has an unique minimal process located in the first node w_i of σ_i (recall Figure 2 in Section 3.1). We can use the minimal nodes w_i as future predictions in the CFM implementation. Each process $q \in P(\sigma_i)$ transmits w_i with each send action. Recall that p_{i+1} is the minimal process of w_{i+1}. When process $p_{i+1} \in P(\sigma_i)$ finishes σ_i then it has to choose the starting node w_{i+1} of the next non-branching path σ_{i+1} such that $(v_{i+1}, w_{i+1}) \in R$ (if $v_{i+1} \neq v^f$). Process p_{i+1} will be the only process in σ_i that knows the next node (w_{i+1}) to be executed. Every other process $q \in P(\sigma_i)$ will execute $\pi_q(\sigma_i)$ and then get into a polling state in which it accepts any incoming message. The first message received by q will inform it about a node w_j, $j > i$. Knowing N and w_j, process q determines the path $\sigma_j = \mathrm{NPath}(w_j)$ that it is executed.

The algorithm $\mathcal{A}'_{\mathtt{p}}$ for process `p` is given below. The call execute_path(w) means that p has to execute $\pi_p(\sigma)$, where $\sigma = \mathrm{NPath}(w)$. Note that p must only remember w and its current node in σ. The call execute1_path(w) is similar, except for the fact that p executes $\pi_p(\sigma)$ without its first action. The call guess(w) guesses w' such that w' is a successor node of the last node of NPath(w). The call poll() means that process p is waiting for an incoming message from an arbitrary process. By receiving a message, process p gets the current node w' and also executes its first action in $\sigma' = \mathrm{NPath}(w')$ (a receive action).

```
w = v^0; not_polling = true;
  while (true)
  {  if (p ∈  P(NPath(w))  // test useful only when starting
  if (not_polling)  execute_path(w);
  else execute1_path(w);
  v = last node of NPath(w);
  if (p = p_min(v))
  { w' = guess(w); not_polling = true; }
  else { w' = poll();  not_polling = false; }
  w = w';}
```

Proposition 5. *Let N be a local-choice HMSC. Then $\mathcal{A}'_N$ is a deadlock-free CFM implementation of N. Moreover, the size of $\mathcal{A}'_N$ is linear in the size of N.*

5 Channel-Bounded CFM and Deadlock Detection

In this section we consider a subclass of communicating finite state machines, called *channel-bounded CFM*. Intuitively, a CFM is bounded if every execution can be simulated by an execution using bounded buffers. Since implementations of HMSCs yield channel-bounded CFMs, it is natural to ask whether a channel-bounded CFM is deadlock-free.

A configuration $C = (q, B)$ of a CFM $\mathcal{A} = (\mathcal{A}_p)_{p\in\mathcal{P}}$ is described by a global state q of $S = \prod_{p\in\mathcal{P}} S_p$ and the contents $B \in (\mathcal{C}^*)^{\mathcal{P}\times\mathcal{P}}$ of all buffers. The transition relation of the CFM is denoted by $\to$, its transitive-reflexive closure is denoted as usual by $\xrightarrow{*}$. The configuration with global state s^0 and empty buffers is the initial configuration. An *execution* $\sigma = C_1 \xrightarrow{a_1} C_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{m-1}} C_m$ of $\mathcal{A}$ is a finite $\to$-path. The labeling of the execution σ is the sequence $a_1 \cdots a_{m-1}$. Note that the labeling of an execution σ defines in a natural way a partial MSC $\mathrm{msc}(\sigma)$. Recall that an execution is *successful* if it ends with empty buffers and each process reaches some local final state. A configuration C is a *deadlock* if there is no successful execution starting from C. Let $A = \bigcup_{p\in\mathcal{P}} A_p$ be the set of possible actions of a CFM over process set $\mathcal{P}$. Two executions σ, σ' are *equivalent* (and we write $\sigma \sim \sigma'$) if $\mathrm{msc}(\sigma) = \mathrm{msc}(\sigma')$ and σ, σ' start in the same configuration.

An execution σ of a CFM is called *b-bounded*, if every configuration of σ is such that the size of every buffer is bounded by b. If $C \xrightarrow{*} C'$ is b-bounded, then we say that C' is *b-reachable* from C.

A CFM is *b-bounded* if every successful execution σ starting in the initial configuration admits some successful, b-bounded equivalent execution $\sigma' \sim \sigma$. Let $\mathcal{A}$ be a b-bounded CFM and let C be b-reachable from the initial configuration of $\mathcal{A}$. Then C is not a deadlock if and only if there is some b-bounded, successful execution starting from C. This allows to show the following proposition.

Proposition 6. *Reachability and deadlock detection for b-bounded CFMs are both PSPACE-complete (with b in unary representation).*

The last proposition allows to connect the model-checking problem and the implementation by CFMs.

Theorem 3. *Let $\mathcal{C}$ be a class of HMSCs that are CFM implementable. Then the model-checking problem for $\mathcal{C}$ (intersection and inclusion, resp.) is decidable.*

Conclusion. We have shown that model-checking a natural class of infinite-state HMSCs, globally-cooperative HMSCs, is decidable and of the same complexity as for regular HMSCs. For a natural subclass (local-choice HMSCs) the complexity of model-checking is the same as in the sequential case. The implementation of locally-cooperative HMSCs raises the question whether we can decide for a given HMSC if it can be implemented without deadlocks in our framework (with finite additional message contents).

References

1. ITU-TS recommendation Z.120, 1996.
2. R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *22nd Int. Conf. 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, G. H. Holzmann, and D. A. Peled. An analyzer for message sequence charts. *Software Concepts and Tools*, 17(2):70–77, 1996.
5. R. Alur and M. Yannakakis. Model checking of message sequence charts. In *CONCUR'99*, LNCS 1664, pages 114–129, 1999.
6. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.
7. B. Caillaud, P. Darondeau, L. Hélouët, and G. Lesventes. HMSCs as partial specifications... with PNs as completions. In *MOVEP*, 2000.
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. J. G. Henriksen, M. Mukund, K. Narayan Kumar, and P. Thiagarajan. On message sequence graphs and finitely generated regular msc languages. In *ICALP'00*, LNCS 1853, pages 675–686, 2000.
11. Y. Métivier. On recognizable subsets of free partially commutative monoids. *Theoretical Computer Science*, 58:201–208, 1988.
12. R. Morin. Recognizable Sets of Message Sequence Charts. In *STACS'02*, LNCS 2285, pages 523–534, 2002.
13. M. Mukund, K. Narayan Kumar, and M. Sohoni. Synthesizing distributed finite-state systems from MSCs. In *CONCUR'00*, LNCS 1877, pages 521–535, 2000.
14. A. Muscholl and D. Peled. Message sequence graphs and decision problems on Mazurkiewicz traces. In *MFCS'99*, LNCS 1672, pages 81–91, 1999.
15. E. Ochmański. Recognizable trace languages. In *The Book of Traces*, chapter 6, pages 167–204. World Scientific, Singapore, 1995.