Infinite-State High-Level MSCs: Model-Checking and Realizability ⋆

Blaise Genest

LIAFA, Université Paris 7 2, pl. Jussieu, case 7014 F-75251 Paris cedex 05, France

Anca Muscholl

LIAFA, Université Paris 7 2, pl. Jussieu, case 7014 F-75251 Paris cedex 05, France

Helmut Seidl [1]

FB IV, Universität Trier D-54286 Trier, Germany

Marc Zeitoun

LIAFA, Université Paris 7 2, pl. Jussieu, case 7014 F-75251 Paris cedex 05, France

Abstract

Message sequence charts (MSC) and High-Level MSC (HMSC) is a visual notation for asynchronously communicating processes and a standard of the ITU. They usually represent incomplete specifications of required or forbidden properties of communication protocols. We consider in this paper two basic problems concerning the automated validation of HMSC specifications, namely model-checking and synthesis. We identify natural syntactic restrictions of HMSCs for which we can solve the above questions. We show first that model-checking for globally-cooperative (and locally-cooperative) HMSCs is decidable within the same complexity as for the restricted class of bounded HMSCs. Furthermore, model-checking local-choice HMSCs turns out to be as efficient as for finite-state (sequential) systems. The study of locally-cooperative and local-choice HMSCs is motivated by the synthesis question, i.e., the question of implementing HMSCs through communicating finite-state machines (CFM) with additional message data. We show that locally-cooperative and local-choice HMSCs are always implementable. Furthermore, the implementation of a local-choice HMSC is deadlock-free and of linear-size w.r.t. the additional data.

Key words: Concurrency, MSC, Model-Checking, Synthesis, Communicating Automata.

1 Introduction

Message sequence charts (MSC for short) is a visual notation for asynchronously communicating processes and a standard of the ITU [14]. 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 for short) combine basic MSCs using choice and iteration, thus describing possibly infinite collections of scenarios. They have a graphical representation by means of directed graphs, with nodes labeled by finite MSCs.

Model-checking and synthesis are the two basic problems considered in this paper. High-level MSCs are infinite-state systems, since the semantics implies that communication channels are unbounded. Another important feature is that high-level MSCs have a global control structure, that comes from the diagram representation of MSCs and the graph structure of the HMSC. This makes the model-checking problem undecidable, and it also raises serious problems for synthesis, where the control must be distributed. Our goal is to propose relaxed restrictions of high-level MSCs that ensure both decidability of model-checking and synthesis, while preserving the infinite-state character of high-level MSCs, i.e. without restricting the channels.

Model-checking. The detection of possible design failures of a protocol at early stages is of critical importance, and the utility of HMSCs can be greatly enhanced by automatic validation methods. A preliminary specification of a protocol can suffer from several deficiencies, either related to the partial order of events (e.g. race conditions [3,25]) or to the violation of user-defined properties specified for instance in logics such as LTL or MSO. However, model-checking HMSCs against such logics is either undecidable (for LTL see [4]) or extremely time consuming (non elementary complexity for MSO interpreted on MSCs [19]). A common approach is to specify the property by a set of MSCs given by an HMSC, as usually done by engineers. The property HMSC can be interpreted as the bad behaviors which have to be avoided by the model [4,25].

* Work partly supported by the European Community Research Training Network "Games and Automata for Synthesis and Validation" (GAMES) and the ACI Versydis.
[1] Current affiliation: TU München, I2, Boltzmannstr. 2, 85748 Garching, Germany.

Unfortunately, this kind of model-checking is undecidable, even if we impose bounded communication channels. This undecidability result motivated the definition of regular (bounded) HMSCs [4,25,13]. A regular HMSC has the property that the set of all linearizations of its MSCs is regular. With this restriction, model-checking HMSC or LTL properties becomes of course decidable. However, the situation is not at all satisfactory since regularity imposes a bound on message channels, or equivalently, that each message has to be somehow acknowledged. This is a strong restriction, excluding very simple HMSCs such as a producer-consumer scenario, where a message is sent an arbitrary number of times from a producer to a consumer. Such a simple scenario is often needed, for instance to describe asynchronous transfers (see e.g. the usb 1.1 specification [29]). More generally, any protocol involving a read- or write-only process cannot be described by regular HMSCs.

The first objective of the paper is to show that we do not need to restrict the channel size for obtaining a class with a decidable model-checking problem. We propose in this paper *globally-cooperative HMSCs* and show that their model-checking problem is decidable, actually within the same asymptotic complexity bounds as for regular HMSCs.

Synthesis. A second basic validation step in protocol design is to know whether the specification is implementable in a machine-oriented model - the *synthesis problem*, see e.g. [28]. For HMSC specifications, synthesis has a specific flavor. First, we seek for fully distributed implementations, which are in general much harder to obtain than sequential ones [28,20,17]. Second, HMSCs usually describe a set of possibly incomplete requirements, which means that the model can be refined compared to the specification. Notice finally that an implementable model can be itself model-checked using for instance SDL tools (*Specification and description languages, ITU Z.100*). There is a large body of papers considering the implementability of HMSC specifications. Previous work considered implementations by statecharts as state-based model [16,10]. Another line of research used communicating finite state machines (CFM or message passing automata) [1,2,24], which is also the model used in this paper. In both models, no global control is available, contrary to an HMSC description. In order to install a distributed control, the machine realization may need further message data or even exchange additional control or synchronization messages.

Our second goal is to exhibit general techniques for synthesizing such a distributed control. For this, we adopt a moderate view of implementation: we allow additional message contents while ruling out extra control messages. The reason is that additional messages mean additional process synchronization, which is not desirable (or even impossible) in a given environment. On the other hand, additional message contents make sense since message data (e.g. call parameters) is often abstracted away in the specification. Still, our implementation semantics by CFMs is more general than the one introduced in [1] and used in [2,23] where a parallel product of communicating finite-state automata is employed to realize the (linear) behavior of each process of the given HMSC. In particular, the possibility of adding information through messages is explicitly ruled out in [1]. Actually, the implementation semantics of [1,2] was shown to be undecidable even for regular HMSCs, which was the motivation for considering safe implementations in [1,18].

We propose in this paper three classes of HMSCs that do not restrict the communication channels: globally-cooperative and two subclasses thereof, locally-cooperative and local-choice HMSCs. Globally-cooperative HMSCs have been introduced independently in [23], whereas locally-cooperative HMSCs are defined in this paper. The local-choice property has been considered in [11]. Globally-cooperative HMSCs extend the well studied class of regular HMSCs [25,4,13]. In a nutshell, regular HMSCs correspond exactly to globally-cooperative HMSCs that use only bounded communication channels.

In the first part of the paper (Section 4) we consider the model-checking problem stated as intersection (negative property) or inclusion (positive property) of HMSCs. That is, the property to be tested is also described by an HMSC. We show that negative and positive model-checking for globally-cooperative HMSCs are PSPACE- and EXPSPACE-complete, respectively, which is as good as the model-checking for regular HMSCs while being applicable to a much larger class of infinite-state HMSCs. For locally-cooperative HMSCs, negative model-checking is still PSPACE-complete, whereas positive model-checking lies between PSPACE and EXPSPACE. For the third subclass, local-choice HMSCs, we are able to obtain better complexities. Namely, we show that negative model-checking can be solved in 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 5.1 and 5.2). We show that both HMSCs classes are *always implementable* by CFMs, however the quality of the implementations differs considerably. Locally-cooperative HMSCs can be implemented with an exponential overhead in the finite control and the message contents, and the implementation is in general not deadlock-free. For local-choice HMSCs we present a linear-size, deadlock-free implementation by CFMs. For globally-cooperative HMSCs we conjecture that they are implementable, too, albeit with a lot of deadlocks.

The last contribution of this paper considers the question whether a CFM is deadlock-free (Section 6). We note that CFMs obtained from HMSCs have the property that for each execution there is an equivalent one that uses only bounded channels. We call such CFMs *existentially-bounded* and we show that reachability and deadlock-freeness for existentially-bounded CFMs is decidable (PSPACE-complete). In contrast, both problems are undecidable for general CFMs [5]. Therefore, we are able to test whether a CFM implementation of an HMSC is deadlock-free or not.

A preliminary version of this paper was presented in [9].

Related work. Model-checking regular HMSCs against HMSC properties was shown to be decidable in [4,25]. For partial-order logics, decidability of model-checking was obtained w.r.t. MSO [19] and TLC [27].

Synthesis (realizability, inference) of CFM from MSC specifications has been introduced in [1], where it is shown that the problem is co-NP complete for finite sets of finite MSCs, however solvable in polynomial time for deadlock-free (safe) realizability. In [2] this question is considered for HMSCs and it is shown that realizability is in general undecidable (even for regular HMSCs), however deadlock-free realizability of regular HMSCs is in EXPSPACE. The matching lower bound is shown in [18], where it is also shown that the upper bound still holds for globally-cooperative HMSCs. The implementation model considered in these papers consists of CFMs with no additional message data, i.e. a parallel product of communicating finite-state machines corresponding to the exact behavior of the single processes. A weaker framework (only messages with same content are FIFO-ordered) is used in [23], where it it is shown that it is decidable whether a given globally-cooperative HMSC is realizable. Our approach is similar to the one used in [24] for implementing regular HMSCs. Finally, [6] considers the implementation of HMSCs by Petri nets with a larger set of behaviors and [11] identifies particular classes of HMSCs that are implementable by CFMs without additional message data.

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 [14]. Each message sequence chart describes a scenario or an execution of a communication protocol in which processes communicate with each other over point-to-point, error-free FIFO channels. An MSC scenario consists in 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 gives a total order on the events belonging to that process. Messages are usually represented by horizontal or slanted arrows from the sending process to the receiving one.

Definition 1 *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 the disjoint union of the sets E_p, comprising the events located on process p. We denote by $P(e) \in \mathcal{P}$ the location of event e.*
- *Every event is either a communication event (send or receive) or a local event. We write $E = S \cup R \cup L$ 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 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). If $m(e) = f$, then $t(e) = p!q(a)$ and $t(f) = q?p(a)$ for some $p, q \in \mathcal{P}$ and $a \in \mathcal{C}$.*
- *$< \subseteq\ E \times E$ is the relation defined by the two conditions below:*
  - *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 relation $<$ is required to be acyclic and is called visual order.

A *message* (e, f) is a pair of matching send and receive events, i.e., $m(e) = f$. We assume that channels are FIFO, that is, whenever $m(e) = f$, $m(e') = f'$, $e < e'$ and $t(e) = p!q(a)$, $t(e') = p!q(a')$ for some $p, q \in \mathcal{P}$, $a, a' \in \mathcal{C}$, we also have $f < f'$.

The results of this paper are independent of this assumption. For non FIFO channels we have to add some information in the type $t(e)$ of an event e. Formally, we have to extend the set of types A to $A \times \mathbb{N}$ and 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 the visual order 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 a total order $\preceq$ extending $\leq$. For any MSC M we denote by $\mathrm{Lin}(M)$ the set of labeled linearizations of M:

$$\mathrm{Lin}(M) = \{t(e_1) \cdots t(e_k) \mid e_1 \cdots e_k \text{ is a linearization of } M\}.$$

Note that with the FIFO property, any labeled linearization $x \in \mathrm{Lin}(M)$ suffices to reconstruct the MSC M, since the type mapping $t : E \to A$ encodes all the information needed.

If the matching m is a partial, injective function then we speak about a *partial MSC*. For every $x \in A^*$ we denote by $\mathrm{msc}(x)$ the partial MSC defined by

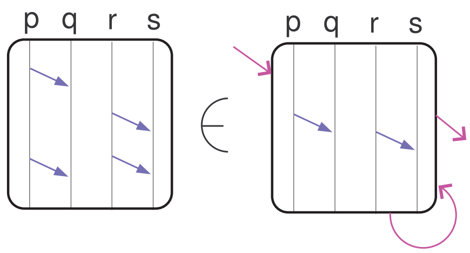

Fig. 1. The left part of the figure depicts an MSC on 4 processes p, q, r, s consisting of 4 messages. It is generated by the HMSC on the right side.

pairing the k-th event of type $p!q(a)$ with the k-th event of type $q?p(a)$ (if they exist).

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 (see the examples in Figures 1, 4). Formally, a *high-level MSC* (HMSC) $G = \langle V, R, v^0, v^f, \lambda \rangle$ is a finite transition system (V, R, v^0, v^f) with transition set $R \subseteq V \times V$, initial node v^0 and terminal node v^f. Each node v is labeled by the finite MSC $\lambda(v)$. We assume that each $\lambda(v)$ is non empty, 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 $0 \le i < k$. 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.

Of course, 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 $M_1 M_2$ is defined as the MSC $\langle E_1 \uplus E_2, <, \mathcal{P}, t_1 \cup t_2, \mathcal{C}_1 \cup \mathcal{C}_2, m_1 \cup m_2 \rangle$ over the disjoint union of event sets, 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 $M_1 M_2$.

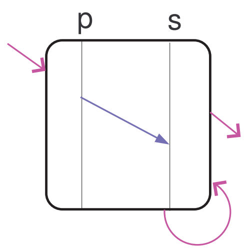

Fig. 2. A globally-cooperative HMSC.

3 A Panorama of HMSC Classes

In this section we introduce two of the subclasses of infinite-state HMSCs defined in this paper, globally-cooperative and locally-cooperative HMSCs. As mentioned in the introduction, both variants of model-checking HMSCs (intersection and inclusion) are undecidable for general HMSCs, even with bounded channels. The very reason for undecidability is that loops can simulate counting, as explained below.

Let us denote for an MSC M the set of processes that occur in M by $P(M)$. Clearly, we have $M_1M_2 = M_2M_1$ for any MSCs M_1, M_2 with $P(M_1) \cap P(M_2) = \emptyset$. We write in this case $M_1 \parallel M_2$ and we say that M_1, M_2 are *independent*. Notice that $(M_1M_2)^* = \{M_1^n M_2^n \mid n \geq 0\}$ if $M_1 \parallel M_2$.

The *communication graph* of an MSC M is a directed graph with a node p for each process $p \in P(M)$ that sends or receives a message, together with edges $p \longrightarrow q$ whenever M contains a message from p to q.

Definition 2 (globally-cooperative) *An HMSC $G = \langle V, R, v^0, v^f, \lambda \rangle$ is called* globally-cooperative*, if every MSC labeling a loop of the transition system (V, R) has a* weakly connected *communication graph.*

The HMSC in Figure 1 is not globally-cooperative, since the communication graph of the loop has two weakly connected components, one over $\{p, q\}$ and the other over $\{r, s\}$. The HMSC in Figure 2 is globally-cooperative since the communication graph of its loop consists of one edge $p \longrightarrow s$.

Compared with globally-cooperative HMSCs, the previously defined regular HMSCs impose a more severe restriction on communication.

Definition 3 (regular) *[25,4] An HMSC is called* regular*, if every MSC labeling a loop of the transition system has a* strongly connected *communication graph.*

For example, the HMSC in Figure 3 is regular, unlike the HMSC of Figure 2, which has no acknowledgment message from s to p.

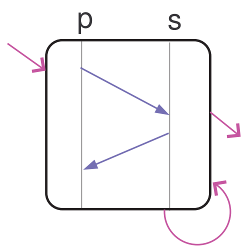

Fig. 3. A regular HMSC.

The link between globally-cooperative and regular HMSCs is summarized by the proposition below. An HMSC has *universally-bounded channels* if there exists some integer B such that for every $x \in \mathrm{Lin}(G)$ and every prefix $y \leq x$, the difference between the number of send symbols and receive symbols in y is at most B.

Proposition 4 *An HMSC G is regular if and only if it is globally-cooperative and it has universally bounded channels.*

PROOF. Assume first that G is regular. By definition, the communication graph of each loop is strongly connected, hence also weakly connected. Moreover, since $\mathrm{Lin}(G)$ is regular [4,25], the second property also follows.

Conversely, let G be globally-cooperative and assume that channels are universally bounded. Consider a loop in the transition system, say labeled by M. Since every state is reachable from the initial node, and the final node is reachable from every state, we find MSCs M_1, M_2 such that $M_1 M^* M_2 \subseteq \mathcal{L}(G)$. We want to show that the communication graph of M must be strongly connected.

Suppose by contradiction that this is not the case, and let $p \to q$ be an edge such that p, q do not belong to the same strongly connected component. The edge $p \to q$ corresponds to a message from p to q in M, say to (e, f). Consider the two occurrences of the message (e, f) in MM. The event f in the first M and the event e in the second M cannot be ordered, since otherwise there would be a path from q to p in the communication graph of M (which is the same as the graph of MM). Hence, MM has a linearization where the two occurrences of e appear before the two occurrences of f. Similarly, every M^k has a linearization where the k occurrences of e appear before the occurrences of f. Therefore, G does not have universally bounded channels, contradiction.

A natural subclass of globally-cooperative HMSCs is obtained by requiring that for every transition $(v, v') \in R$ the two MSCs labeling v, v' are not independent. Intuitively, this restriction avoids mixing the parallel product with the sequential transition relation of the underlying HMSC graph.

Definition 5 (locally-cooperative) *An HMSC* $G = \langle V, R, v^0, v^f, \lambda \rangle$ *is called* locally-cooperative*, if for every* $(v, v') \in R$*, the MSCs* $\lambda(v), \lambda(v'), \lambda(v)\lambda(v')$ *all have weakly connected communication graphs.*

Since locally- and globally-cooperativity, and regularity are structural restrictions, we can test whether a HMSC satisfies one of these restrictions:

Proposition 6 *Checking whether an HMSC is globally-cooperative (or regular) is co-NP-complete, whereas checking whether it is locally-cooperative can be done in linear time.*

PROOF. First we show how to check whether an HMSC is globally-cooperative. We guess a subgraph H of G. Then we check (in polynomial time) that there exists a loop passing through each node of H at least once. Moreover, the *communication graph* is checked to be not weakly connected. This algorithm is clearly co-NP.

For the lower bound we reduce 3-SAT to our problem. Let $\phi = C_1 \wedge \cdots \wedge C_m$ be a 3-SAT formula over n variables $x_1, \ldots, x_n$, and disjunctive clauses $C_i = l_i^1 \vee l_i^2 \vee l_i^3$. The formula ϕ is satisfiable iff there exists a valuation such that for every $i \leq m$, at least one literal among l_i^1, l_i^2, l_i^3 is true for the valuation.

The MSC-graph G contains two vertices NT_i, NF_i for each variable x_i, plus one initial and final node N_0. Transitions go from from each NX_i to each NX_{i+1}, for $X \in \{T, F\}$ and $1 \leq i < n$. Moreover, G contains transitions from N_0 to NT_1 and NF_1, resp. from NT_n and NF_n back to N_0. Hence, a simple loop of G around N_0 corresponds to a valuation of the variables: x_i true means that the loop goes through NT_i, and x_i false means that the loop goes through NF_i.

We describe now the MSCs used in the construction. There will be $3m + 2$ processes $(P_j^1, P_j^2, P_j^3)_{1 \leq j \leq m}$, where P_j^1, P_j^2, P_j^3 correspond to the clause C_j, plus two extra processes P_0, P_{m+1}.

Let M_0 be an MSC consisting of two local actions, one on P_0 and one on P_{m+1}, and let N_0 be labeled by M_0.

We define now MSCs LT_j^k, LF_j^1: The MSC LF_j^1 consists of a message from P_0 to P_j^1. The MSC LF_j^2 consists of a message from P_j^1 to P_j^2. The MSC LF_j^3 consists of a message from P_j^2 to P_{m+1}. The MSCs LT_j^1 and LT_j^2 are both empty, while LT_j^3 consists of one message from P_0 to P_j^2.

We label NT_i by the product of all MSCs LT_j^k such that $l_j^k = x_i$, and of all MSCs LF_j^k such that $l_j^k = \overline{x_i}$. Symmetrically, let NF_i be labeled by the product of all MSCs LT_j^k such that $l_j^k = \overline{x_i}$, and of all MSCs LF_j^k such that

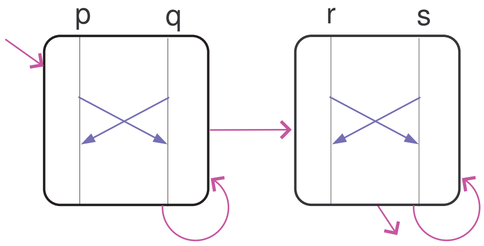

Fig. 4. Regular HMSC which is not locally-cooperative.

$l_j^k = x_i$.

Thus, the processes P_0 and P_{m+1} are connected in the communication graph of a simple loop around N_0 iff there exists a clause where all three literals are false. Moreover, notice that if P_0 and P_{m+1} are connected in some loop, then all processes occurring in that loop are connected (in the communication graph): if LF_j^1, LF_j^3 or LT_j^3 appear in the loop, their processes are connected either to P_0 or to P_{m+1}. Concerning LF_j^2, its processes are either connected to P_0 because of LT_j^3 (i.e., l_j^3 is true) or to P_{m+1} because of LF_j^3 (i.e., l_j^3 is false).

Hence, ϕ is satisfiable iff there exists a valuation such that for all $1 \leq j \leq m$, one of l_j^1, l_j^2, l_j^3 is true in the valuation, iff there exists a simple loop with a non connected communication graph.

Note that we can change slightly the construction for the lower bound for regular HMSCs. It suffices to replace each message by a pair of messages, back and forth.

We defined three subclasses of HMSCs, locally- and globally-cooperative HMSCs, and regular HMSCs. Any locally-cooperative or regular HMSC is also globally-cooperative by the definition of these classes. Locally-cooperative HMSCs and regular HMSCs are expressively incomparable, see e.g. Figure 2 that depicts a locally-cooperative HMSC which is not regular. On the other hand, Figure 4 depicts a regular HMSC which is not locally-cooperative.

We will show in Section 4 that the model-checking problem of the bigger class of globally-cooperative HMSCs is decidable within the very same complexity as for locally-cooperative or regular HMSCs. An implementation through CFMs (albeit with deadlocks) is known for regular HMSCs [24]. In Section 5 we show how to implement locally-cooperative HMSCs. However, the implementability for globally-cooperative HMSCs remains open.

We end this section by explaining some motivation behind the definition of globally-cooperative HMSCs. The main idea is that one can model-check an HMSC w.r.t. a property given by another HMSC, provided that we are able to obtain a regular set of representatives for the set $\mathrm{Lin}(H)$ of one HMSC H, that is compatible with the relation $\|$. We call $X \subseteq \mathrm{Lin}(H)$ a set of *representatives* for H, if for every MSC $M \in \mathcal{L}(G)$, we have $\mathrm{Lin}(M) \cap X \neq \emptyset$. We will explain in the next section how to provide for any globally-cooperative HMSC such a suitable set of representatives.

4 Model-Checking

Bounding message channels by forcing acknowledgments as required for regular HMSCs is a severe restriction, excluding such simple and common scenarios as the one depicted in Figure 2. Such a scenario is however a locally-cooperative HMSC. Although globally-cooperative HMSCs are much more general than regular HMSCs, we can show that we are still able to do automata-based model-checking with the same complexities. The underlying idea is that the executions of a globally-cooperative HMSC can be captured by a suitable regular set of representative linearizations. As an example reconsider the HMSC G of Figure 2. The set $\mathrm{Lin}(G)$ of linearizations of executions of G is obviously non regular. However, the representative set $X = (p!s\; s?p)^*$ captures all the information needed for doing model-checking against an HMSC property.

The plan of this section is as follows. First, we recall some basic facts from the theory of Mazurkiewicz traces [21,7], that are crucial for our definition of globally-cooperative HMSCs. Then we discuss the model-checking problem for globally-cooperative and locally-cooperative HMSCs, respectively.

4.1 Mazurkiewicz Traces

Mazurkiewicz traces were proposed as a formal model for concurrent executions, where the concurrency is made explicit by describing the independence between possible actions in the system. Formally, let $\| \subseteq A \times A$ be a symmetric, irreflexive independence (commutation) relation on the alphabet of actions A. A Mazurkiewicz trace is just a set of words that can be obtained from a given word $\sigma \in A^*$ by exchanging adjacent symbols a, b with $a \| b$.

We are interested here in computing closure sets under the independence relation $\|$. Let $K \subseteq A^*$ be a set of words. The $\|$-*closure* of $K \subseteq A^*$ is the smallest set $[K]_\|$ containing K such that $\sigma ab\sigma' \in [K]_\|$ iff $\sigma ba\sigma' \in [K]_\|$, for all $\sigma, \sigma' \in A^*$ and $a \| b$.

Unfortunately, the $\|$-closure does not preserve regularity in general. Consider for instance two letters a, b with $a \| b$. Then $[(ab)^*]_\|$ is the set of words over

$\{a, b\}$ with equally many a's and b's.

Mazurkiewicz trace theory provides a syntactical condition ensuring that the $\|$-closure of a regular language remains regular. Given an automaton $\mathcal{A}$, this condition states that each loop of $\mathcal{A}$ is labeled by a connected word [22,26]. A word w is called *connected* if its set of letters cannot be partitioned into non empty subsets X, Y such that $a \parallel b$ for all $a \in X$, $b \in Y$. If the condition is true, then we can construct a non deterministic automaton recognizing the set $[\mathcal{L}(\mathcal{A})]_{\|}$ which is of size at most $2^{O(n \cdot \wp)}$, where $n = |\mathcal{A}|$ and $\wp$ is the minimal number of cliques covering the graph $(A, (A \times A) \setminus \|)$ [25].

4.2 Model-checking globally-cooperative and locally-cooperative HMSCs

It is easy to obtain an automaton $\mathcal{A}$ from an HMSC G such that $\mathcal{L}(\mathcal{A}) \subseteq \mathrm{Lin}(G)$ is a set of representatives. For this, it suffices to replace each node by some linearization of the associated MSC and view the graph thus obtained as a (word) automaton. Unfortunately, this set does not suffice for deciding for instance whether $\mathcal{L}(G) \cap \mathcal{L}(H) \neq \emptyset$ for some HMSCs G, H. Consider for instance the HMSC G in Figure 1. The representative set obtained from G could be $X = (p!q\, q?p\, r!s\, s?r)^*$. However, for the MSC shown in the left part, one might choose the linearization $(p!q\, q?p)^2(r!s\, s?r)^2$, that does not belong to X.

The idea is to consider a sort of message alphabet with an independence relation on it, and apply the construction of [25] for closing it under commutation. Notice that defining the message alphabet is not easy. For instance, consider the MSC labeling one of the nodes in Figure 4. Its two messages cannot be splitted into two MSCs, and they must be considered together. Formally, a non empty MSC is called *atomic* (atom, for short) if it cannot be written as $M = M_1M_2$ for non empty MSCs M_1, M_2. For instance, the two MSCs in Figure 8 are not atomic, since the atoms correspond to single messages. It is not hard to see that any MSC has a unique factorization into atomic MSCs, up to commuting adjacent independent atoms, i.e. atoms M_1, M_2 such that $M_1 \parallel M_2$. The decomposition of M can be obtained in linear time by computing the strongly connected components in a directed graph obtained from the partial order graph of M by adding back edges from r to s whenever (s, r) represents a message, [12].

We are now ready to define the suitable set of representatives that allows us to perform model-checking of an HMSC w.r.t. a property expressed by another HMSC. For sake of simplicity and efficiency we rather define this set of representatives (denoted $\mathcal{L}^a(G)$ below) on a sort of message alphabet (atoms) instead of the event type alphabet.

Definition 7 *Let* $G = \langle V, R, v^0, v^f, \lambda\rangle$ *be an HMSC. We define* $Atom(G)$ *as the (finite) set of atoms occurring in the decomposition of MSCs from* $\lambda(V)$*. We view* $Atom(G)$ *as an alphabet, respectively* $\|$ *as an independence relation on this alphabet. Let also* $\mathcal{L}^a(G) = \{A_1 \cdots A_k \mid A_1 \cdots A_k \in \mathcal{L}(G)$ *and* $A_i \in Atom(G)$ *for all* $i\}$*.*

Consider as an example the HMSC G in Figure 4, and let a, b denote the atomic MSCs labeling the initial and final node, respectively. We have $\mathcal{L}^a(G) = \{a, b\}^+ \setminus (a^* \cup b^*)$, with $a \| b$.

Proposition 8 *Let* G *be a globally-cooperative HMSC of size* s *and over* $\wp$ *processes. Then* $\mathcal{L}^a(G)$ *is recognized by a non deterministic automaton of size at most* $2^{O(\wp s)}$*.*

PROOF. We first transform G into an equivalent automaton $\mathcal{B}$ with nodes labeled by symbols from Atom(G). Formally, each node v is first replaced by a path $v_1, \ldots, v_k$, with v_i labeled by an atom M_i such that $\lambda(v) = M_1 \cdots M_k$. It is easy to see that this transformation preserves the property of being globally-cooperative.

Note that $[L(\mathcal{B})]_\| = \mathcal{L}^a(G)$. In the example in Figure 4 we have for instance $\mathcal{L}(\mathcal{B}) = a^+b^+$ and $[L(\mathcal{B})]_\| = \{a, b\}^+ \setminus (a^* + b^*)$. Since G is globally-cooperative, each loop of $\mathcal{B}$ is labeled by a connected word, hence we can apply the construction of [25] mentioned in Section 4.1. We obtain a non deterministic automaton $\mathcal{A}$ recognizing $[L(\mathcal{B})]_\| = \mathcal{L}^a(G)$, of size at most $2^{O(\wp s)}$.

For locally-cooperative HMSCs G we obtain a smaller automaton recognizing $\mathcal{L}^a(G)$. More precisely, the size of the automaton is exponential in the number of processes, but polynomial in the size of the transition system.

Proposition 9 *Let* G *be a locally-cooperative HMSC with* n *nodes and* $\wp$ *processes. Let* k *be the maximal size of an MSC labeling a node of* G*. Then* $\mathcal{L}^a(G)$ *is recognized by a non deterministic automaton of size at most* $(kn)^{2\wp}(\wp+1)^\wp$*.*

PROOF. Let $G = \langle V, R, v^0, v^f, \lambda\rangle$. First note that we cannot replace G by an equivalent locally-cooperative HMSC with nodes labeled by atoms, since this does not preserve the locally-cooperative property. Let $x \in \mathcal{L}^a(G)$ and consider some accepting path $\sigma = v_1 \cdots v_m$ of G labeled by x. Let also x_1 be some prefix of x. We decompose the path $v_1 \cdots v_m \in V^*$ with respect to x_1 as follows: Let

$$\sigma = v_1 \cdots v_m = \alpha_1\beta_1\alpha_2 \cdots \beta_{k-1}\alpha_k\beta_k$$

with $\alpha_1, \beta_k \in V^*$, $\alpha_i, \beta_j \in V^+$ for all $1 < i \le k$, $1 \le j < k$, such that the nodes occurring in $\alpha_1 \cdots \alpha_k$ are precisely the nodes of σ that have at least one atom appearing in x_1. We call a node w of some subpath α_i *half-full*, if the MSC $\lambda(w)$ appears in x_1, but not completely. Otherwise it is called *full*. A node of a subpath β_i is called empty.

We show first that every subpath α_i, $i > 1$, starts with a half-full node. Indeed, with G being locally-cooperative, there can be no transition from an empty node to a full node, since two nodes linked by a transition share at least one process.

In a second step, we show that the number of half-full nodes is at most $\wp$ (see also Figure 5, where half-full nodes are grey, and full ones are black).

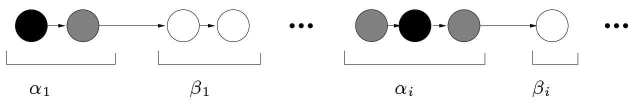

Fig. 5. Path decomposition in locally-cooperative HMSC.

Let w be a half-full node with $\lambda(w) = YZ$, where Y is the prefix of $\lambda(w)$ appearing in x_1. By definition, Y and Z are both non empty. Since the communication graph of $\lambda(w)$ is weakly connected there is some process $p \in P(Y) \cap P(Z)$. Suppose that w' is some half-full node occurring after w in σ, and let $\lambda(w') = Y'Z'$ be the decomposition of w' w.r.t. x_1. Then $P(Z) \cap P(Y') = \emptyset$, since Z is executed after Y' in x. Hence, $p \notin P(Y')$. This shows the claim.

We obtain a non deterministic automaton $\mathcal{B}$ recognizing $\mathcal{L}^a(G)$ as follows. A state of $\mathcal{B}$ records the first and the last node of each α_i, as well as the processes that occurred so far in each α_i. Moreover, we need to store the ordering and the configurations corresponding to the half-full nodes of the subpaths α_i. Each transition of $\mathcal{B}$ either adds an atom to a half-full node of some α_i, or creates a new subpath α_i (guessing a node). Whenever a node becomes full, the automaton can choose (non deterministically) to join two adjacent subpaths. The size of $\mathcal{B}$ is bounded by $k^{\wp} n^{2\wp} (\wp + 1)^{\wp}$, where $n^{2\wp}$ stands for the possible first/last node of each subpath α_i and the ordering of half-full nodes, $k^{\wp}$ stands for the configurations of half-full nodes and $(\wp+1)^{\wp}$ for the processes occurring in the α_i.

Theorem 10 *Model-checking globally-cooperative HMSCs is decidable. More precisely, let G_1, G_2 be globally-cooperative HMSCs. Then,*

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

Moreover, the lower bound for the intersection problem also holds when G_1, G_2 are locally-cooperative.

PROOF. Using the unique decomposition of MSCs into atoms it can be easily checked that:

(1) $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) \neq \emptyset$ iff $\mathcal{L}^a(G_1) \cap \mathcal{L}^a(G_2) \neq \emptyset$.
(2) $\mathcal{L}(G_1) \subseteq \mathcal{L}(G_2)$ iff $\mathcal{L}^a(G_1) \subseteq \mathcal{L}^a(G_2)$.

Hence, we obtain the upper bounds using the exponential-size, non deterministic automata recognizing $\mathcal{L}^a(G_i)$.

The lower bound of the inclusion problem for globally-cooperative HMSCs can be directly obtained from the EXPSPACE-hardness of the universality problem for loop-connected automata, see [25].

For the intersection problem we encode accepting computations of a polynomially space-bounded Turing machine T as executions in $\mathcal{L}(G_1) \cap \mathcal{L}(G_2)$, with G_1 and G_2 locally-cooperative. Let Γ be the tape alphabet of T and let w be an input of T. The proof idea is well-known: we encode a finite computation $C_0 \vdash C_1 \vdash \cdots \vdash C_m$ of T on w by using two copies C_i and $\bar{C}_i$ of each configuration and by grouping factors in two different ways (see Figure 6). The HMSC G_1 then ensures that C_i and $\bar{C}_i$ are identical. The HMSC G_2 ensures that for each i, $\bar{C}_i$ and C_{i+1} are successor configurations, and that C_0 (resp. $\bar{C}_m$) is the initial (resp. final) configuration.

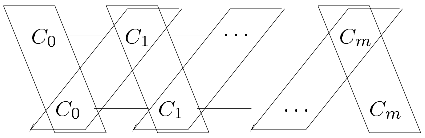

Fig. 6. TM computation.

The first issue is how to encode configurations. Let k be the length of any configuration (filled with blanks if necessary) reached by T on input w. We have $k = p(|w|)$ for some polynomial p. We use $k+2$ processes $P, \bar{P}, (P_i)_{1 \leq i \leq k}$. If the i-th symbol of a configuration C is a, then we encode it by the MSC M_a^i consisting of a single message named a from P_i to P. The encoding $h(C)$ of a configuration C is the concatenation of the encodings of symbols of C. Similarly, if the i-th symbol of $\bar{C}$ is $\bar{a}$, then we encode it by the MSC $\bar{M}_a^i$ consisting of a message from P_i to $\bar{P}$ named a, and we let $h(\bar{C})$ be the encoding of $\bar{C}$. The encoding of a sequence of configurations $C_0, \ldots, C_m$ is $h(C_0)h(\bar{C}_0) \cdots h(C_m)h(\bar{C}_m)$.

Let $M_i = \sum_a M_a^i$, $\bar{M}_i = \sum_a \bar{M}_a^i$ and $X_i = \sum_a M_a^i \bar{M}_a^i$, where the sum means a choice. We easily construct G_1 such that $\mathcal{L}(G_1) = (X_1 \cdots X_k)^*$ (see Figure 7). Its node set is the disjoint union of nodes of $X_1, \ldots, X_k$, the nodes of X_1 are initial, the nodes of X_k are final, and there is a transition from any node of X_j (resp. of X_k) to any node of X_{j+1} (resp. of X_1). Executions of G_1 are exactly encodings of finite sequences $C_0, \ldots, C_m$ with $C_i \in \Gamma^k$. Formally, we add a unique initial node comprising all processes, and a unique final node. Note that G_1 is locally-cooperative.

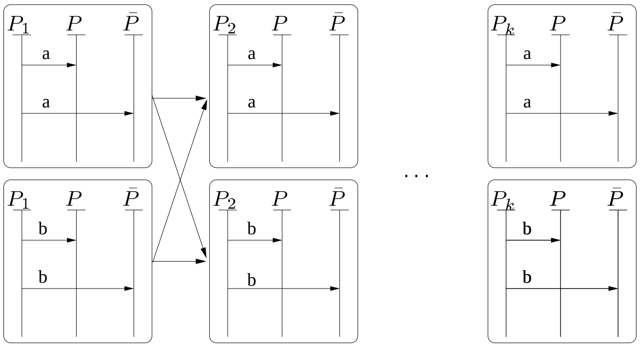

Fig. 7. The HMSC G_1 for the intersection problem

The definition of G_2 is slightly more complicated since we have to consider transitions of T. Assume that each configuration has the form $\lhd uqv \rhd$ where q is the current state of T, uv is written on the tape, and the head scans the first symbol of v. Thus, the transitions are of the form $aqb \longrightarrow qab'$ (left move) or $aqb \longrightarrow ab'q$ (right move). A transition $t = xyz \longrightarrow x'y'z'$ performed at head position i in a configuration is coded by the MSC $T_{t,i} = \bar{M}_x^{i-1}\bar{M}_y^{i}\bar{M}_z^{i+1}M_{x'}^{i-1}M_{y'}^{i}M_{z'}^{i+1}$, and we let $\delta_i = \{T_{t,i} \mid t \text{ transition of } T\}$. Denote by $s_1 \cdots s_k$ the initial configuration of T on w and by $f_1 \cdots f_k$ its accepting configuration. Let $Y_i = \sum_a \bar{M}_a^i M_a^i$. It is again straightforward to construct a locally-cooperative HMSC G_2 such that

$$\mathcal{L}(G_2) = M_{s_1}^1 \cdots M_{s_k}^k \left[\bigcup_{i=2}^{k-1} Y_1 \cdots Y_{i-2} \cdot \delta_i \cdot Y_{i+2} \cdots Y_k \right]^* \bar{M}_{f_1}^1 \cdots \bar{M}_{f_k}^k$$

Both G_1 and G_2 are of size $O(k)$. Finally, an execution of $\mathcal{L}(G_1) \cap \mathcal{L}(G_2)$ is an encoding of a sequence $C_0, \ldots, C_m$. Since it is an execution of G_2, C_0 is the initial configuration of T on w, C_m is the final configuration, and $C_i \vdash C_{i+1}$. Hence $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) \neq \emptyset$ iff the computation of T on w is accepting.

Note that the lower bounds for locally-cooperative HMSCs do not hold anymore if we assume that all nodes are labeled by *atomic* MSCs. In this case we can use the fact that any execution has a unique decomposition in atomic MSC, hence both questions can be rephrased in terms of finite word automata.

Proposition 11 *Let G_1, G_2 be locally-cooperative HMSCs such that each node is labeled by an atomic MSC. We have:*

(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.

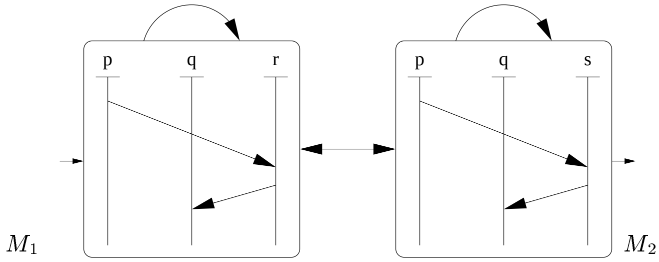

Fig. 8. A locally-cooperative HMSC G.

4.3 Local-Choice HMSCs

A potential deficiency of HMSC specifications is the distributed choice along branching paths, when a node has more than one successor, each with possibly several minimal processes. The *local-choice* property [11] ensures precisely that branching between executions is always controlled by a unique process. We define local HMSCs below, as a slightly different variant of local-choice HMSCs. Local HMSCs simplify the model-checking and implementation algorithms. Moreover, we show that our definition has the same expressivity as the original definition of [11].

Definition 12 (local HMSCs) *An HMSC $G = \langle V, R, v^0, v^f, \lambda\rangle$ is called* local *if*

(1) v^0 has a single minimal event
(2) for each node $v \in V$, there is a process $root(v) \in P(v)$ such that every node w with $(v, w) \in R$ has a unique minimal event, and this event is located on $root(v)$.

For instance, the HMSC of Figure 8 is local. Figure 7 depicts an HMSC which is not local although every node has a unique minimal event, since this event does not belong to the processes of each predecessor node. It is easy to see that any local HMSC is locally-cooperative.

Local-choice was defined by different syntactic conditions in [11,8], but all these definitions are easily seen to yield the same class of HMSCs. We have chosen the simplest definition here, since it is easier to handle and it is compatible with locally-cooperative. Proposition 14 shows how to transform a local-choice HMSC into a local one. Since this construction can be achieved with a quadratic blow-up only, the complexity results for model-checking will not be affected too much. Actually, the results stated for negative model-checking (intersection) can be seen to hold with the very same complexity for local-choice HMSC.

We call a node with at least two outgoing edges a *branching node*.

Definition 13 (local-choice HMSCs, [11]) *An HMSC* $G = \langle V, R, v^0, v^f, \lambda\rangle$ *is called* local-choice *if the following conditions are satisfied:*

(1) Each path starting in v^0 *has a single minimal event.*
(2) For each branching node $v \in V$ *and for* $v = v^f$*, there is a process* $root(v)$ *such that every path from a node* w *with* $(v, w) \in R$ *has a single minimal event, and that event is located on* $root(v)$*.*

It is easy to see that local HMSCs are local-choice HMSCs, while locally-cooperative and local-choice are syntactically incomparable.

An important observation is that every path in a local-choice HMSC G where all but the last node are non branching and different from v^f, is of length $\leq |V|$. Such a path will be called a *non branching path.* Moreover, for any node v, there is a unique maximal non branching path starting in v that ends either in a branching node or in the terminal node v^f. We denote this path by $\mathrm{NPath}(v)$. Consider now an accepting path σ of G. 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 also Figure 9, where the triangles represent the partial order graphs of the subpaths σ_i). Recall that v_i is branching (or v^f) 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. Moreover, the local-choice condition applied to the branching node v_{i-1} ensures that p_i also belongs to $P(\sigma_{i-1})$, since otherwise $\sigma_{i-1}\sigma_i \cdots \sigma_k$ would have more than one minimal event.

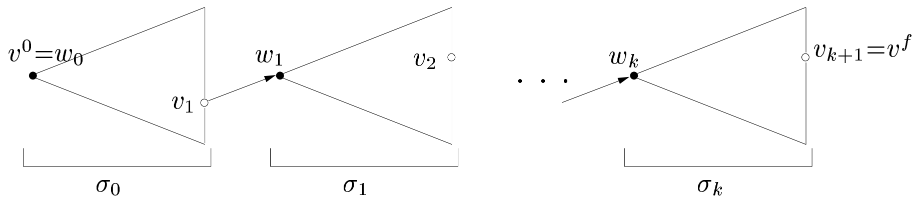

Fig. 9. Path decomposition in local-choice HMSC.

The above decomposition of paths in a local-choice HMSC will be used by the implementation algorithm (Section 5.2). Moreover, this decomposition yields a simple transformation of local-choice HMSC into local HMSCs:

Proposition 14 *For each local-choice HMSC we can construct an equivalent local HMSC of quadratic size.*

PROOF. Let $G = \langle V, R, v^0, v^f, \lambda\rangle$ be a local-choice HMSC. The nodes of the local-choice HMSC G' are all immediate successors of branching nodes, together with a new initial node w^0 and a new final node w^f. A successor v of a branching node is labeled by $\lambda(\mathrm{NPath}(v))$ (see e.g. $v = w_i$ in Figure 9). Similarly, w^0 is labeled by $\mathrm{NPath}(v^0)$. Finally, $\lambda(w^f)$ is the empty MSC. The transitions are given by:

- There is a transition $(v, w) \in R'$ in G' whenever $(x, w) \in R$, where x is the last node of $\mathrm{NPath}(v)$ (for $v = w^0$ we require that x is the last node of $\mathrm{NPath}(v^0)$).
- There is a transition $(v, w^f) \in R'$ for all v such that $\mathrm{NPath}(v)$ ends in v^f (for $v = w^0$ we require that $\mathrm{NPath}(v^0)$ ends in v^f).

It is obvious that the construction yields an equivalent, local HMSC G'. The number of nodes of G' is $\leq |V| + 2$ and the size of each MSC labeling a node of G' is at most the size of G.

4.4 Model-checking local HMSCs

For the model-checking algorithms on local HMSCs we will transform a local HMSC such that it is non decomposable according to Definition 15 below. Intuitively, we will refine the MSCs labeling a path as much as possible, while preserving the local property. This will ensure a kind of unicity (see Lemma 17) when comparing two or more paths to each other.

Given an event e of an MSC M, we denote by $e\!\downarrow$ the set of events $\{f \in M \mid e \leq f\}$ lying in the future of e.

Definition 15 *Let M be an MSC and p a process. We say that M is p-decomposable, if there exists an event e of M such that $e\!\downarrow$ is an MSC containing some event of process p, and $M \neq e\!\downarrow$. Note that $e\!\downarrow$ is required to contain for every receive event the matching send, and vice-versa.*

We say that a node v of an HMSC is p-decomposable, if the MSC labeling v is p-decomposable. A local HMSC is called non decomposable*, if no node v is p-decomposable, with $p = root(v)$.*

Proposition 16 *We can construct from a local HMSC G an equivalent, non decomposable local HMSC G' in time $O(|G|^2)$. Moreover, G' is of size $|G|$.*

PROOF. Let v be a node of $G = \langle V, R, v^0, v^f, \lambda\rangle$ and $M_1 = \lambda(v)$. We show how to decompose inductively the node v. The decomposition is applied repeatedly to an MSC M w.r.t. a process p. We start with M_1 and $p_1 = root(v)$ (if $v = v^f$ then we choose $p_1 \in P(M_1)$ arbitrarily). If M_1 is p_1-decomposable, then let $M_1 = M_2 N_1$ be a p_1-decomposition of M_1 with $|N_1|$ minimal. Hence

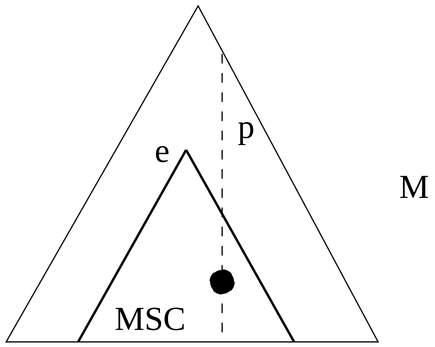

Fig. 10. p-decomposition of an MSC.

N_1 is not p_1-decomposable. Then we repeat the process with $M = M_2$ and p_2 being the process of the minimal event of N_1. The decomposition process stops as soon as M is not p-decomposable for the current process p. We obtain in this way a sequence $N_l, \ldots, N_1$ of MSCs where $M_1 = N_l \cdots N_1$ and every N_i has a unique minimal event located on process p_{i+1}. Moreover, $p_i \in P(N_i)$ for all $l > i \geq 1$. We replace now v by a new path $w_l, \ldots, w_1$, where w_i is labeled by the MSC N_i just defined. Note that the new path $w_l, \ldots, w_1$ satisfies the requirement of local HMSCs. Moreover, the MSCs labeling the nodes of the new path are all non decomposable, by the minimality condition in each decomposition step.

Applying this algorithm for each node v of G, we obtain an equivalent non decomposable local HMSC G'. Since each event of G belongs to one and only one node of G', G' has at most $|G|$ nodes. Since each decomposition step needs $O(|G|)$ time, the algorithm is quadratic.

The next lemma says that for deciding whether two paths $v_1, \ldots, v_k$ and $w_1, \ldots, w_l$ of a non decomposable local HMSC are labeled by the same MSCs, we can proceed as follows. Assume that v_i is labeled by M_i and w_j is labeled by N_j. Assume that there exists some index i such that $M_j = N_j$ for all $j \leq i$, but $M_{i+1} \neq N_{i+1}$. In this case, we show that $N_{i+2} \cdots N_l$ is a suffix of M_{i+1}, resp. $M_{i+2} \cdots M_k$ is a suffix of N_{i+1}. Notice that given v_{i+1} and w_{i+1}, we can decide effectively whether such factorizations of $\lambda(v_{i+1})$ and $\lambda(w_{i+1})$ exist.

Lemma 17 *Let $v_1, \ldots, v_k$ and $w_1, \ldots, w_l$ be two paths of a non decomposable local HMSC G and suppose that v_i is labeled by M_i and w_j is labeled by N_j. Assume further that $M_1 \cdots M_k = N_1 \cdots N_l$. Then one of the following conditions is satisfied:*

(1) $M_1 = N_1$.
(2) $M_1 = X N_2 \cdots N_l$ *and* $N_1 = X M_2 \cdots M_k$*, for some MSC* X*. Moreover,* $P(M_2 \cdots M_k) \cap P(N_2 \cdots N_l) = \emptyset$.

PROOF. If $l = 1$ or $k = 1$, then we are in the second case of the claim. Else, let $X = M_1 \cap N_1$ be the intersection of M_1, N_1, i.e., the greatest common prefix of M_1, N_1. We can write $M_1 = XY$, $N_1 = XY'$, where $P(Y) \cap P(Y') = \emptyset$ and X is an MSC. We can also write $N_2 \cdots N_l = YZ$ for some MSC Z. Then, $M_2 \cdots M_k = Y'Z$.

We show first that at least one of the MSCs Y, Y', Z must be empty. Suppose by contradiction that all of Y, Y', Z are non empty. Let e be a minimal event of Z. Since $YZ = N_2 \cdots N_l$ has a unique minimal event we must have $P(e) \in P(Y)$. By symmetry, $P(e) \in P(Y')$. But then $P(Y) \cap P(Y') \neq \emptyset$, a contradiction.

The case where Z is empty corresponds to the second case of the claim.

By symmetry we suppose now that Y is empty, thus M_1 is a prefix of N_1. We want to show that Y' is also empty, i.e., $M_1 = N_1$. We first show that $N_1 = M_1 \cdots M_j$ for some j, and finally that $j = 1$.

Let j be the first index such that $N_1 \subseteq M_1 \cdots M_j$. By contradiction, assume that the minimal event e of N_2 belongs to some M_i, $i \leq j$. Let f be the minimal event of M_i. Suppose first that $f = e$. Using the local property of G, we infer that $M_i \cdots M_j$ is a prefix of $N_2 \cdots N_l$, hence $N_1 \subseteq M_1 \cdots M_{i-1}$, contradicting the minimality of j. Because $f < e$, the event f belongs to N_1. Since e is a send event, and $f < e$ in $N_1 N_2$, there exists some event $g \geq f$ in N_1 with $P(g) = P(e) = p$. But this is a contradiction, since N_1 is not p-decomposable. (Note that the restriction of the future of f to N_1 is an MSC since f is the minimal event of M_i.) Therefore, $N_1 = M_1 \cdots M_j$.

We have $l \geq 2$, thus $j < k$. Hence, $\mathit{root}(M_j) = \min(M_{j+1}) = \min(N_2) = \mathit{root}(N_1)$. Thus, there exists some event f in M_j with $p = P(f) = \mathrm{root}(M_j)$. So $e = \min(M_j) \leq f$ and events e, f belong to N_1. But this means that N_1 is p-decomposable, since $e{\downarrow}$ restricted to N_1 is an MSC. Therefore, we must have $e = \min(N_1)$, i.e. $j = 1$, and Y' is empty.

We can show now the main result of this section.

Theorem 18 *Let G and G' be two local HMSCs. Then we have:*

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

PROOF. Both lower bounds follow from classical results on finite word automata.

Let $G = \langle V, R, v^0, v^f, \lambda\rangle$ and $G' = \langle V', R', v'^0, v'^f, \lambda'\rangle$.

Intersection problem We show that the intersection problem can be solved in time $O((|G|+|G'|)^2)$. By Proposition 16 applied to G and G', we can suppose that both HMSCs are local and non decomposable.

Assume that $\mathcal{L}(G) \cap \mathcal{L}(G') \neq \emptyset$. Then there exist two paths of G, G' labeled by $M = M_1 \cdots M_k$ and $N = N_1 \cdots N_l$ with $M = N$, thus satisfying the hypothesis of Lemma 17.

We define a graph $G \times G'$ with set of states $V \times V'$. We let $(v, v') \longrightarrow_{G \times G'} (w, w')$ in $G \times G'$ if $\lambda(v) = \lambda'(v')$, $(v, w) \in R$ in G and $(v', w') \in R'$ in G'.

We will check the second condition of Lemma 17 using a set of target nodes T of $G \times G'$. Let $(v, v') \in T$ iff:

(1) $\lambda(v) = XY$ and $\lambda'(v') = XY'$, where $P(Y) \cap P(Y') = \emptyset$.
(2) Y' labels a path from a successor of v to a final state of G (if Y' is empty then $v = v^f$ in G). Symmetrically, Y labels a path from a successor of v' to a final state of G' (if Y is empty then $v' = v'^f$ in G').

We can determine whether (v, v') belongs to the target set using depth-first search in G, G' for finding a path from a successor of v' (if v' is not final) that is labeled by Y, and symmetrically for Y'. Hence, the target set T can be built in time $O(|G \times G'|)$.

By Lemma 17, $\mathcal{L}(G) \cap \mathcal{L}(G') \neq \emptyset$ iff T is reachable from (v^0, v'^0). Hence, this problem can be solved in time linear in the number of nodes of $G \times G'$, thus in time $O(|G| \cdot |G'|)$. This leads to the overall bound $O((|G| + |G'|)^2)$.

For the NLOGSPACE upper bound, we sketch the main ingredients of the proof. As usual for accessibility problems, it suffices to store one node of $G \times G'$ at a time. Given two nodes v, w in a local HMSC such that $(v, w) \in R$ and v (resp. w) has e (resp. f) as minimal event, the MSC labeling v is uniquely determined by (e, f). Precisely, it comprises the events that are in the future of e but not in that of f. Thus, a node of $G \times G'$ can be represented by two pairs $(e, f), (e', f')$ of events, one in G and one in G'. With similar arguments, we can test whether $(v, v') \longrightarrow_{G \times G'} (w, w')$ holds in $G \times G'$ in NLOGSPACE. For the target set, note that each of the MSCs X, Y, Y' occurring in the definition of T has a single minimal event. Hence we can reason as above for deciding in NLOGSPACE whether a node of $G \times G'$ belongs to T.

Inclusion problem We show now that the inclusion problem is in PSPACE. We want to know whether there exists an accepting path in G labeled by M such that no path of G' labeled by M is accepting. This is done in the usual way, by guessing a path ρ in G and considering all possible paths in G' that may match. For this we define a graph $\widehat{G \times G'}$ from $G \times G'$, that corresponds roughly to a subset construction on G'. The guessed path ρ in G determines all possibly matching paths in G', each of which is in one of the two cases of Lemma 17. At any time, we consider a node v of G and a set W' of nodes of G' according to case 1 of Lemma 17. Whenever (v, v') with $v' \in W'$ belongs to the target set T, we store the suffix of v' which remains to be matched in G, according to case 2 of Lemma 17. Hence, a state of $\widehat{G \times G'}$ is a triple (v, W', S) where v is a node of G, W' is a subset of nodes of G' and S is a set of suffixes of MSCs of nodes in G'.

Formally, let Suff be the set of suffixes of MSCs labeling the nodes of G'. The state set of $\widehat{G \times G'}$ is $V \times 2^{V'} \times 2^{\mathrm{Suff}}$. Transitions are defined as $(v_1, W'_1, S_1) \longrightarrow (v_2, W'_2, S_2)$, with

- $(v_1, v_2) \in R$
- $W'_2 = \{v'_2 \mid \exists v'_1 \in W'_1,\ (v_1, v'_1) \longrightarrow_{G \times G'} (v_2, v'_2)$ [2] $\}$
- S_2 is defined by the following two conditions:

Case A: if $M = \lambda(v_1)Y$ for some $M \in S_1$, then $Y \in S_2$,

Case B: if $(v_2, v'_2) \in T$ for some $v'_2 \in W'_2$, and $\lambda(v'_2) = XY'$ with $X = \lambda(v_2) \cap \lambda'(v'_2)$, then $Y' \in S_2$.

Notice that in case B above, if $\lambda(v_2) = \lambda(v'_2)$, then the empty MSC belongs to S_2. We define a set of final nodes $\hat{T}$ of $\widehat{G \times G'}$: let $(v, W', S) \in \hat{T}$ if v is final in G and the empty MSC does not belong to S. The initial state is $(v^0, \{v'^0\}, \{Y'\})$, where $\lambda(v^0) = XY, \lambda'(v'^0) = XY'$ with $P(Y) \cap P(Y') = \emptyset$. If such an X does not exist, then clearly $\mathcal{L}(G) \not\subseteq \mathcal{L}(G')$.

We show now that a state of $\hat{T}$ is reachable if and only if $\mathcal{L}(G) \setminus \mathcal{L}(G') \neq \emptyset$. As this question is a reachability problem in a graph of exponential size, the PSPACE upper bound follows directly.

Assume that there exists an accepting path $\rho = v_0 \cdots v_l$ in G such that no matching path of G' is accepting. The path ρ uniquely defines a path $\hat{\rho} = \hat{v}_0 \cdots \hat{v}_l$ of $\widehat{G \times G'}$ with $\hat{v}_i = (v_i, W'_i, S_i)$, for all i. We show that $\hat{v}_l \in \hat{T}$. Assume by contradiction that the empty MSC belongs to S_l. Let us consider a transition $\hat{v}_k \rightarrow \hat{v}_{k+1}$ that generates a suffix Y' in S_{k+1} (Case B above) leading to the empty MSC in S_l on $\hat{\rho}$ (applying case A above). Thus, there exists a transition $(v'_k, v'_{k+1}) \in R'$ in G' such that the pair $(v_{k+1}, v'_{k+1}) \in T$. Moreover $\lambda(v'_{k+1}) = (\lambda(v_{k+1}) \cap \lambda'(v'_{k+1}))Y'$. By the definition of the transitions (W'-component), we have $\lambda(v_i) = \lambda'(v'_i)$ for all $i \leq k$. By the choice of k, $\lambda'(v'_{k+1} \cdots v'_m) = \lambda(v_{k+1} \cdots v_l)$, hence $\lambda'(v'_0 \cdots v'_m) = \lambda(\rho)$ with v'_m final in G' (due to the definition of the target set T), a contradiction.

[2] That is, $\lambda(v_1) = \lambda'(v'_1)$ and $(v'_1, v'_2) \in R'$.

Conversely, assume that there exists an accepting path

$$(v_0, W'_0, S_0), \ldots, (v_l, W'_l, S_l)$$

in $\widehat{G \times G'}$ and $(v_l, W'_l, S_l) \in \hat{T}$. Let $M = \lambda(v_0 \cdots v_l)$. Assume by contradiction that there exists an accepting path $v'_0 \cdots v'_m$ of G' labeled by M. Then we can apply Lemma 17. So there exists an index k such that $\lambda(v_i) = \lambda'(v'_i)$ for all $i \leq k$, and $(v_{k+1}, v'_{k+1}) \in T$. Then $v'_{k+1} \in W'_{k+1}$, and $Y' \in S_{k+1}$, where $\lambda'(v'_{k+1}) = (\lambda'(v'_{k+1}) \cap \lambda(v_{k+1}))Y'$. By the definition of T, the empty MSC belongs to S_l, a contradiction.

5 Implementing HMSC Specifications

The most natural implementation model for HMSCs are communicating finite state machines (CFM), as used for instance in the ITU Z.100 [15] 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 channels. In general we assume that channels 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 channel is a word over a finite alphabet $\mathcal{C}$. With each pair $(p, q) \in \mathcal{P}^2$ of distinct processes we associate a channel $B_{p,q}$. 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 channel $B_{p,q}$. Receiving message a by p from q is denoted by $p?q(a)$ and it means that a must be the first message in channel $B_{q,p}$, 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 a on process p is denoted by $l_p(a)$.

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 channels. The transition relation of the CFM is denoted by $\rightarrow$, its transitive-reflexive closure is denoted as usual by $\xrightarrow{*}$. The configuration with global state s^0 and empty channels 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 $\rightarrow$-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 msc(σ).

We denote an execution of the CFM as *successful*, if each process p ends in some final state from F_p and all channels are empty. The set of successful executions of $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A})$ and is a set of finite MSCs. A configuration C is a *deadlock* if there is no successful execution starting from C. The *size* of $\mathcal{A}$ is $\sum_p |\mathcal{A}_p|$.

A CFM implementation of an HMSC will refine the HMSC specification by adding for instance data to the message contents. We will call $\mathcal{A}$ a *CFM implementation* of the HMSC G if the projection of $\mathcal{L}(\mathcal{A})$ on the original message contents of G coincides with $\mathcal{L}(G)$.

5.1 Locally-cooperative HMSCs

The simplest realization of an HMSC G by a CFM is the one where the automaton $\mathcal{A}_p$ corresponding to process p generates the projection of $\mathcal{L}(G)$ on p. This approach is used in [2,23]. Consider again the HMSC G_1 of Figure 8 (page 18), and let M be the MSC given by the projections $\pi_p(M) = p!r\ p!s$, $\pi_q(M) = q?s\ q?r$, $\pi_r(M) = r?p\ r!q$ and $\pi_s(M) = s?p\ s!q$, pictured in Figure 11. Then M does not belong to $\mathcal{L}(G_1)$ although $\pi_t(M) \in \pi_t(\mathcal{L}(G_1))$ for all

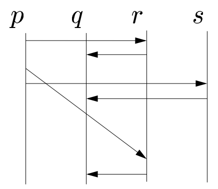

Fig. 11. An MSC $M \notin \mathcal{L}(G_1)$

$t \in \{p, q, r, s\}$. Hence G_1 is not realizable according to [2].

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 between the nodes M_1 and M_2 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. In the example above, this would mean that process p will send with the first message its next choice (node M_1). Since predictions are forwarded, this will prevent process q to receive the message from s that corresponds to M_2.

The general solution will assign a leader process to each transition (p in the example) i.e., a process that occurs in both nodes of the transition, and decides about some nodes to be executed in the future (a kind of prediction).

For a node $v \in V$ of $G = \langle V, R, v^0, v^f, \lambda\rangle$, let $P(v)$ denote the processes occurring in $\lambda(v)$. For a path $\sigma = v_0 v_1 \cdots v_k$ of G 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:

$$\mathsf{first}(\sigma, p) = \begin{cases} \perp & \text{if } p \notin P(\sigma), \\ v_j, \text{ where } j = \min\{\ell \geq 0 \mid p \in P(v_\ell)\} & \text{otherwise.} \end{cases}$$

Similarly, if σ has at least $i+1$ nodes, let $\mathsf{last}(\sigma, i, p)$ be the last node among the first $(i+1)$ ones containing p:

$$\mathsf{last}(\sigma, i, p) = \begin{cases} \perp & \text{if } p \notin P(v_0 v_1 \cdots v_i), \\ v_j, \text{ where } j = \max\{\ell \leq i \mid p \in P(v_\ell)\} & \text{otherwise.} \end{cases}$$

Let $G = \langle V, R, v^0, v^f, \lambda\rangle$ 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 there exist a path $\sigma = v_0 v_1 \cdots v_k$ in G, and a transition $(v, v_0) \in R$ such that
  - $l \in P(v) \cap P(v_0)$, and
  - $\nu(p) = \mathsf{first}(\sigma, p)$ for each process p. In particular, $\nu(l) = v_0$.

The process l is called the *leader* of the transition $(v, \nu(l))$ with respect to (v, ν, l). Note that realizable predictions can be precomputed iteratively in polynomial time.

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

```
p:
(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);// deadlocks if p receives a prediction
                 // other than 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 $\texttt{guess\_next}(\mathtt{v}', \nu)$ chooses 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':

$$\nu_{|\mathcal{P}\setminus P(v')} = \nu'_{|\mathcal{P}\setminus P(v')} \tag{C}$$

The call `guess(v')` guesses nondeterministically a pair (ν', ℓ') such that (v', ν', ℓ') 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`. The weak connectivity of each MSC ensures that all processes executing a node must choose the same value for `m` in order to complete that node without deadlocking.

Proposition 19 *Let G be a locally-cooperative HMSC. Then $\mathcal{A}_G$ is a CFM implementation of $\mathcal{L}(G)$ of size $n^{O(\wp)}$, where n is the number of nodes of G and $\wp$ is the number of processes.*

PROOF. Since $\mathcal{A}_G$ can simulate any execution of G by guessing precisely the path corresponding to a given execution we see that every execution of G corresponds to a successful execution of $\mathcal{A}_G$. For the other direction let $x \in \mathcal{L}(\mathcal{A}_G)$ and let y be the MSC obtained from x by removing the additional message contents introduced by the algorithm. We will determine a path σ in G such that $y = \lambda(\sigma)$.

Since $x \in \mathcal{L}(\mathcal{A}_G)$, every process p executes a sequence of states, and halts when its prediction function ν satisfies $\nu(p) = \perp$. We define the path σ inductively.

Let $\sigma_0 = v_0 = v^0$. If $\nu_0 = \perp^{\mathcal{P}}$, then we let $\sigma = \sigma_0$; otherwise, $\nu_0 \neq \perp^{\mathcal{P}}$ implies $\nu_0(l_0) \neq \perp$, since (v_0, ν_0, l_0) is realizable. Assume now that we have constructed a sequence $(v_k, l_k, \nu_k)_{0 \leq k \leq i}$ such that:

a) $\sigma_i = v_0 v_1 \cdots v_i$ is a path in G;
b) for any process p, the sequence of events of process p along the path σ_i, $\pi_p(\sigma_i)$, is a prefix of $\pi_p(y)$;
c) the common leader chosen during the execution of v_i is $l_i \in P(v_i)$;
d) the common prediction function chosen during the execution of v_i is ν_i.

These conditions are fulfilled for $i = 0$. Assume that they hold for a given i, and that $\nu_i(l_i) \neq \perp$. Then $v_{i+1} = \nu_i(l_i)$ is a state of G and $(v_i, v_{i+1}) \in R$ by by the definition of a realizable prediction. Hence $\sigma_{i+1} = v_0 \cdots v_{i+1}$ is a path of G. Moreover, $l_i \in P(v_{i+1})$, and the algorithm for process l_i shows that l_i executes v_{i+1} immediately after having completed v_i.

We show first that every process $p \in P(v_{i+1})$ executes v_{i+1} after σ_i. Assume that p executes some state $v \neq v_{i+1}$ after its execution of σ_i. Let $v_j = \mathsf{last}(\sigma_i, i, p)$ be the last node executed by p in σ_i and let ν_j be the associated prediction. Then $j \neq i$ since $p \in P(v_i)$ would imply $v = v_{i+1}$ since the execution did not deadlock. By assumption, $v = \nu_j(p)$. The prediction functions ν_k, $j < k \leq i$ guessed by l_j satisfy the compatibility condition (C), so in particular $\nu_k(p) = \nu_j(p)$ for $j \leq k \leq i$, so $v = \nu_i(p)$. Since ν_i is realizable by a path τ of G and $p \in P(v_{i+1})$ we obtain that $v = \nu_i(p) = \mathsf{first}(\tau, p) = v_{i+1}$, a contradiction.

Since v_{i+1} is executed without deadlock, the new leader l_{i+1} and the new prediction function ν_{i+1} are well-defined.

Let now k be maximal such that σ_k satisfies conditions a) to d) above, and let $\sigma = \sigma_k$. We claim that $y = \lambda(\sigma)$. If this is not the case, $\pi_p(\sigma)$ would be a strict prefix of $\pi_p(y)$ for some $p \in \mathcal{P}$. Since p executes some actions after those of $\pi_p(\sigma)$, it has to execute at least the state $v = \nu_j(p) = \nu_k(p)$. But $\nu_k(l_k) = \perp$ by maximality of k, so this is a contradiction to σ_k being executed without deadlock.

Remark 20 *Note that we can fix a leader for each transition of the HMSC beforehand. This decreases the degree of non determinism and the possibility of deadlocking. We used a leader in our algorithm for simplifying the description of the implementation.*

5.2 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. An unavoidable problem is that branching in an HMSC is not controlled by a single process, as it is the case for local HMSCs.

Roughly speaking, the idea for implementing local HMSCs is that *before* the execution branches, the minimal process of the future execution knows that it has to choose the next node (through a call of `guess()` below). If a process is not the minimal process of the next branching, then its next action will be a receive. In this case, it stays in a *polling state* until it receives a message telling him what node it is currently executing (stored in a variable called `current_node`). A polling state means that the only possible actions are receives of arbitrary messages from arbitrary processes. Here is the algorithm for the polling state of process p.

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

Here is the algorithm for process p.

```
  initialization();
  while (true)  {
  if (p ≠  root(current_node))
       polling();
  else { v = guess(current_node);
          current_node = v; }
  execute(current_node);
                  }
```

The current node is initialized in `initialization()` by letting the value of `current_node` be v^0. The call `execute`(w) means that process p executes $\pi_p(w)$. In this case each message sent by p contains in addition w.

The correctness proof of the algorithm is easy, since each node is connected and admits a unique minimal event on a process occurring in the previous node. Then each process (except for the minimal process of the current node) begins its execution by receiving a message, so it gets informed about the current node. We obtain an implementation of linear size for local HMSCs.

Proposition 21 *For any local HMSC, one can construct a deadlock-free CFM implementation of linear size.*

6 Existentially-Bounded CFM and Deadlock Detection

In this section we consider a subclass of communicating finite state machines, called *existentially-bounded CFM*. Intuitively, a CFM is existentially-bounded if every execution can be simulated using bounded channels. Since implementations of HMSCs yield existentially-bounded CFMs, it is natural to ask whether a existentially-bounded CFM is deadlock-free. We show below how to decide this question in polynomial space. Notice that for unrestricted CFMs this question is undecidable [5].

Recall that a CFM execution is *successful* if it ends with empty channels and each process reaches some local final state. A configuration C is a *deadlock* if there is no successful execution starting from C. Finally, for $x \in A^*$ we let $\mathrm{msc}(x)$ denote the partial MSC defined by pairing the k-th event of type $p!q(a)$ with the k-th event of type $q?p(a)$ (if they exist).

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 called *equivalent* if $\mathrm{msc}(\sigma) = \mathrm{msc}(\sigma')$ and σ, σ' start in the same configuration. In this case we write $\sigma \sim \sigma'$.

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

Definition 22 (existentially-b-bounded CFM) *Let $b > 0$. A CFM is existentially-b-*bounded *if every successful execution σ starting in the initial configuration admits some successful, b-bounded execution $\sigma' \sim \sigma$. A CFM is called existentially-bounded if it is existentially-b-bounded.*

Proposition 23 *Let $\mathcal{A}$ be a existentially-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.*

PROOF. We assume that C is not a deadlock configuration, that is, there exists some successful execution σ starting in C and labeled by $v \in A^*$. Moreover, let $\tau = C_0 \xrightarrow{u} C$ be a b-bounded execution, with $u \in A^*$. Since $\mathcal{A}$ is an existentially-b-bounded CFM there exists also a successful, b-bounded execution σ' such that $\sigma' \sim \tau\sigma$, that is, σ' and $\tau\sigma$ are equivalent. Let $w \in A^*$ be the labeling of σ', hence $\mathrm{msc}(w) = \mathrm{msc}(uv)$. Hence, we can write $w = u_0 v_1 u_1 \cdots v_k u_k v_{k+1}$ with $\mathrm{msc}(u) = \mathrm{msc}(u_0 \cdots u_k)$, and for all $i \leq j$ and every process p, either $\pi_p(u_j) = \epsilon$ or $\pi_p(v_i) = \epsilon$.

We show that the execution starting in C and labeled by $v_1 \cdots v_{k+1}$ is b-bounded. Note first that $v_1 \cdots v_{k+1}$ is well-defined starting from C, since for any pair p, q of processes, $|v_i|_a \neq \epsilon$ for some $a = p?q(c)$ implies $|u_i \cdots u_k|_{a'} = \epsilon$ for any $a' = p?q(c')$, thus a is possible after executing $u_0 \cdots u_k v_1 \cdots v_{i-1}$. Moreover, there can be no channel of size more than b in v_i after executing $u_0 \cdots u_k v_1 \cdots v_{i-1}$, since if $|v_i|_a \neq \epsilon$ for some $a = p!q(c)$ then $|u_i \cdots u_k|_{a'} = \epsilon$ for any $a' = p!q(c')$.

Proposition 24 *The following problem is PSPACE-complete:*

Input: *Integer b (unary representation) and a existentially-b-bounded CFM $\mathcal{A}$.*
Question: *Is $\mathcal{A}$ deadlock-free?*

PROOF. The upper bound is provided by Proposition 23. For the lower bound we give a reduction from the non empty intersection problem for finite automata. Assume that $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are finite automata over a mutual alphabet $\mathcal{M}$. Without restriction, we suppose that for every $\mathcal{A}_i$ and every state we can reach a final state. We use n processes $p_1, \ldots, p_n$ that exchange messages over the alphabet $\mathcal{M} \cup \{\#\}$. Process p_1 starts by sending a message # to p_n. Then p_1 starts simulating $\mathcal{A}_1$ such that for each transition it sends a message to p_2 corresponding to the transition label. Finally, from each final state of $\mathcal{A}_1$ it can go to a new state d_1 after sending # to p_2 (state d_1 has no successor). For each $i > 1$ process p_i simulates the automaton $\mathcal{A}_i$ such that for each transition labeled by some $a \in \mathcal{M}$ it must first receive the message a from p_{i-1} and then it sends a to p_{i+1}. Upon receiving # from p_{i-1} in a final state of $\mathcal{A}_i$ it can move to a new state d_i, after sending # to p_{i+1}. If $i = n$ then it also receives the message # from p_1. It can be easily checked that the intersection of $\mathcal{A}_1, \ldots, \mathcal{A}_n$ is non empty if and only if there is some successful, 1-bounded execution starting from the initial configuration.

Corollary 25 *The following problem is PSPACE-complete:*

Input: *Integer b (unary representation) and a existentially-b-bounded CFM $\mathcal{A}$.*
Question: *Is $\mathcal{L}(\mathcal{A}) \neq \emptyset$?*

The last proposition allows to connect the model-checking problem and the implementation by CFMs. We first define the synchronized product of two CFM implementations of HMSCs G_1, G_2. Let $\mathcal{A}_i = (\mathcal{A}^i_p)_{p \in \mathcal{P}}$, $\mathcal{A}^i_p = (S^i_p, \rightarrow^i_p, s^0_i, F^i_p)$, be a CFM implementation of G_i. The synchronized product $\mathcal{A}_1 \times \mathcal{A}_2$ is the CFM $\mathcal{A} = (\mathcal{A}_p)_{p \in \mathcal{P}}$ with $\mathcal{A}_p = (S_p, \rightarrow_p, s^0, F_p)$ given by:

- $S_p = S^1_p \times S^2_p$,
- $(s_1, s_2) \xrightarrow{a}_p (s'_1, s'_2)$ if $s_1, s'_1 \in S^1_p$, $s_2, s'_2 \in S^2_p$, $s_i \xrightarrow{a_i}{}^i_p s'_i$ for each $i = 1, 2$. The actions a, a_1, a_2 are either of the form $a_i = p!q(c_i)$ and $a = p!q(c_1, c_2)$, or $a_i = p?q(c_i)$ and $a = p?q(c_1, c_2)$. Moreover, the projection of c_1, c_2 on the message contents of G_1, G_2, must be identical.
- The initial state of $\mathcal{A}$ is $s^0 = (s^0_1, s^0_2)$, where s^0_i is the initial state of $\mathcal{A}_i$. Let also $F_p = F^1_p \times F^2_p$ be the set of local p-final states of $\mathcal{A}$.

Theorem 26 *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.*

PROOF. Suppose that $\mathcal{A}_i$ is a CFM implementation of G_i. Moreover, let b_i be the channel bound for G_i. That is, any atom in Atom(G_i) has some linearization such that the difference between send and receive symbols is at most b_i, for any prefix. Then the CFMs $\mathcal{A}_1, \mathcal{A}_2$ are existentially-b-bounded,

Class $\mathcal{C}$:	G.-Cooperative	L.-Cooperative	L.-Cooperative + atomic	local-choice
$\in \mathcal{C}$?	co-NP	P	P	P
$\cap\ = \emptyset$	PSPACE	NLOG if $\wp$ is constant	NLOG	NLOG
$\subseteq$	EXPSPACE	PSPACE if $\wp$ is constant	PSPACE	PSPACE
Implem.	?	$\lvert G\rvert^{O(\wp)}$	$\lvert G\rvert^{O(\wp)}$	$\lvert G\rvert$

Fig. 12. Model-checking and implementation infinite-state HMSCs.

with $b = \max(b_1, b_2)$.

Moreover, for the negative model-checking we have $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}_1 \times \mathcal{A}_2) \neq \emptyset$. For the positive model-checking we have $\mathcal{L}(G_1) \not\subseteq \mathcal{L}(G_2)$ iff $\mathcal{L}(\mathcal{A}_1 \times 2^{\mathcal{A}_2}) \neq \emptyset$, where $2^{\mathcal{A}}$ denotes the powerset CFM associated with $\mathcal{A}$, obtained by synchronizing on actions of same type and identical original contents.

Conclusion

We have shown that model-checking is decidable for a large class of infinite-state HMSCs, globally-cooperative HMSCs. Moreover, the complexity remains the same as for the more restricted class of regular HMSCs. For local-choice HMSCs, we have shown how to perform model-checking with the same complexity as for finite automata. The precise complexity of the inclusion problem for locally-cooperative HMSCs remains open, we only know that it lies between PSPACE and EXPSPACE. The implementation of locally-cooperative HMSCs raises the question whether we can decide for a given locally-cooperative HMSC if it can be implemented without deadlocks in our framework. The table below summarizes our results.

References

[1] R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *Proceedings of the 22nd International Conference on Software Engineering, Limerick (Ireland)*, pages 304–313. ACM, 2000.

[2] R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of MSC graphs. In *Proceedings of the 28th International colloquium on Automata, Languages and Programming (ICALP'01), Crete (Greece) 2001*, LNCS 2076, pages 797–808. Springer, 2001.

[3] R. Alur, G. H. Holzmann, and D. A. Peled. An analyzer for message sequence charts. *Software Concepts and Tools*, 17(2):70–77, 1996.

[4] R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proceedings of the 10th International Conference on Concurrency Theory CONCUR'99, Eindhoven (The Netherlands)*, LNCS 1664, pages 114–129. Springer, 1999.

[5] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[6] B. Caillaud, P. Darondeau, L. Hélouët, and G. Lesventes. HMSCs as partial specifications... with PNs as completions. In *Modeling and Verification of Parallel Processes, 4th Summer School, MOVEP 2000*, Nantes, France 2000.

[7] V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.

[8] B. Genest and A. Muscholl. Pattern matching and membership for hierarchical message sequence charts. In *Proceedings of the LATIN 2002: Theoretical Informatics, 5th Latin American Symposium, Cancun, Mexico*, LNCS 2286, pages 326–340. Springer, 2002.

[9] B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun. Infinite-state High-level MSCs: Model-checking and realizability. In *Proceedings of the 29th International colloquium on Automata, Languages and Programming (ICALP'02), Malaga (Spain), 2002*, LNCS 2380, pages 657–668. Springer, 2002.

[10] D. Harel and H. Kugler. Synthesizing State-Based Object Systems from LSC Specifications Int. J. of Foundations of Computer Science 13:1 (2002), pages 5–51.

[11] 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.

[12] L. Hélouët and P. Le Maigat. Decomposition of Message Sequence Charts. In *Proceedings of the 2nd Workshop on SDL and MSC (SAM2000), Col de Porte, Grenoble*, pages 46–60, 2000.

[13] J. G. Henriksen, M. Mukund, K. Narayan Kumar, and P. Thiagarajan. On message sequence graphs and finitely generated regular MSC languages. In *Proceedings of the 27th International colloquium on Automata, Languages and Programming (ICALP'00), Geneva (Switzerland), 2000*, LNCS 1853, pages 675–686. Springer, 2000.

[14] ITU-TS recommendation Z.120, 1996.

[15] ITU-TS recommendation Z.100, Specification and Description Languauge (SDL), Geneva 1999.

[16] I. Krüger, R. Grosu, P. Scholz, and M. Broy. From MSCs to Statecharts. In Franz J. Rammig (Ed.), Distributed and Parallel Embedded Systems. Kluwer Academic Publishers. 1999

[17] O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *Proceedings of the IEEE Symposium on Logic in Computer Science, LICS'01*, pages 389–398. IEEE, 2001.

[18] M. Lohrey. Safe realizability of high-level message sequence charts. In *Proceedings of the Concurrency Theory, 13th International Conference (CONCUR'02)*, LNCS 2421, pages 177–192. Springer, 2002.

[19] P. Madhusudan. Reasoning about sequential and branching behaviours of message sequence graphs. In *Proceedings of the 28th International colloquium on Automata, Languages and Programming (ICALP'01), Crete (Greece) 2001*, LNCS 2076, pages 809–820. Springer, 2001.

[20] P. Madhusudan and P. Thiagarajan. Distributed controller synthesis for local specification. In *Proceedings of the 28th International colloquium on Automata, Languages and Programming (ICALP'01), Crete (Greece) 2001*, LNCS 2076, pages 396–407. Springer, 2001.

[21] A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.

[22] Y. Métivier. On recognizable subsets of free partially comutative monoids. *Theoretical Computer Science*, 58:201–208, 1988.

[23] R. Morin. Recognizability vs realizability and mso-definability in unbounded high level message sequence charts. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS'02, Antibes, France)*, LNCS 2285, pages 523-534. Springer, 2002.

[24] M. Mukund, K. Narayan Kumar, and M. Sohoni. Synthesizing distributed finite-state systems from MSCs. In *Proceedings of the 11th International Conference on Concurrency Theory CONCUR'00, University Park, USA*, LNCS 1877, pages 521–535. Springer, 2000.

[25] A. Muscholl and D. Peled. Message sequence graphs and decision problems on Mazurkiewicz traces. In *Proceedings of the 24thSymposium on Mathematical Foundations of Computer Science (MFCS'99), Szklarska Poreba (Poland) 1999*, LNCS 1672, pages 81–91. Springer, 1999.

[26] E. Ochmański. Recognizable trace languages. In V. Diekert and G. Rozenberg, editors, *The Book of Traces*, chapter 6, pages 167–204. World Scientific, Singapore, 1995.

[27] D. Peled. Specification and verification of message sequence charts. In *Proceedings of Formal Techniques for Distributed System Development, FORTE/PSTV 2000, Pisa, Italy*, pages 139–154, 2000.

[28] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proceedings of the IEEE Symposium on Foundations of Computer Science, FOCS'90*, IEEE, pages 746–757, 1990.

[29] USB 1.1 specification, http://www.usb.org/developers/docs/usbspec.zip