Compositionality Everywhere: The end of MSC-Graphs?

Blaise Genest

LIAFA, Université Paris VII, 2 place Jussieu, 75251 Paris, France
genest@liafa.jussieu.fr

Abstract. Communicating Finite States Machines (CFMs) and Message Sequence Graphs (MSC-Graphs for short) are two popular specification formalisms for communicating systems. Since CFMs are close to distributed algorithms, they are the target of implementation. MSC-Graphs capture requirements (scenarios), hence they are the starting point of the design process. We exhibit an expressive subclass of Compositional MSC-Graphs, with a PSPACE-complete model-checking problem. It subsumes a non trivial class of CFMs. We give a tractable algorithm for implementation of this class into deadlock-free CFMs. These results extend the practical application of MSC-Graph since they improve results for MSC-Graphs in both qualitative complexity (considering an existential channel bound much smaller than universal bound) and expressivity.

1 Introduction

Specifying the behavior of software systems in such a way that formal methods can be applied and validation tasks can be automated, is a challenging goal. While research has brought strong results and tools for simple systems, complex systems still lack powerful techniques. For instance, concurrent systems such as message passing systems are still hard to cope with.

Concurrent languages such as Harel's Live Sequence Charts [10], UML sequence diagrams, interworkings..., have seen a growing interest this last decade. Among them, the ITU visual notation of *Message Sequence Charts* (*MSCs*, [13]) has received a lot of attention, both in the area of formal methods and in automatic verification [2, 12, 18, 17, 20]. MSCs can be considered as an abstract representation of communications between asynchronous processes. They are used as requirements, documentations, abstract test cases, and so on. *MSC-Graphs* (*MSGs* for short) propose a way of modeling set of behaviors, combining parallel composition (processes) with sequential composition (transition system). The main advantage of such a visual representation is to have a local, explicit description of the communication and the causalities appearing in the system. On the other hand, SDL (ITU norm Z100) brings another formalism, namely *Communicating Finite States Machines* (*CFM* for short) [5]. Being really close to distributed algorithms, CFMs are the ideal model for modelling parallel programs. Actually, any concurrent system which has to be implemented must be turned into a CFM at some point. However, unlimited channels of CFMs provide Turing machine equivalence, hence undecidability. A straightforward restriction that guarantiees decidability is to limit (universally) the size of message channels. That is, every execution of the system must forbid a process p to send to q if b messages from p to q are still in transit. This is a severe restriction (also used for regular MSGs [2, 12, 18]) which is not realistic in many settings. To preserve decidability, we consider in this paper the less restrictive existential b-bound on message channels, which means that every execution (linearization) can be retrieved from *some* b-bounded execution. Notice that any universally b-bounded system is existentially c-bounded, with c usually much smaller than b. With the usb 1.1 protocol [21], we will illustrate that existential boundedness is sufficiently general.

MSGs are meaningful at an early point of design where a CFM does not exist yet. The requirements a CFM designer has to face are given in form of a (infinite) set of scenarios, some being good scenarios that should be possible in the distributed system, some bad scenarios that must be avoided. Since scenarios are easily described as MSGs, the validation problem faced by engineers can be stated in form of two MSGs, a positive and a negative one. The challenge is then to be able to transform the positive MSG into a CFM that has empty intersection with the negative scenarios. We propose here a computer-aided framework, based on a class of MSGs that can be model-checked. A good expressivity of this class is extremely important. The first step is to help the designer describing his scenarios into this class (e.g. by adding messages). The second step is to modify the scenarios to prevent the bad behaviors, guided by counter-examples given by the model-checking algorithm. The last step is to help the designer modifying the system so that it can be automatically turned into a CFM.

Unluckily, MSGs are not expressive enough and model-checking is undecidable. To address the first issue, an extension of MSGs has been proposed, namely *Compositional MSGs* (*CMSG* for short [9]), that are more expressive than CFMs. To ensure the decidability of model-checking, a large subclass of MSGs has been exhibited, namely *globally-cooperative MSGs* (*gc-MSGs* for short)[8]. Another subclass, namely *local-choice MSGs* (*lc-MSGs* for short), is always implementable into CFMs [8]. In this paper, we study the combination of compositionality and cooperation, leading to *globally-cooperative CMSG* (gc-CMSG for short). Since finite MSC-generation is crucial for model-checking techniques for gc-MSGs (see [8]), decidability and complexity of the model-checking problem for gc-CMSG was an open problem. Moreover, the expressive power of gc-CMSG was unclear. We show that gc-CMSGs are more expressive than CFMs (unlike gc-MSGs) and allow model-checking in PSPACE, that is as good as model-checking the very restricted class of regular MSGs. We provide also a new heuristics to test in co-NP the implementability (and implement) a gc-CMSG into an exponential size CFM. It uses the easy implementation of *local-choice CMSG* (lc-CMSG for short) by deadlock-free CFMs. It improves in both complexity and expressivity an unpublished paper [6] considering the question whether a gc-MSG is equivalent to some lc-MSG.

We also provide a general explanation for several previous results. In short, we show that closure under partial order together with existentially-bounded channels suffices to do model-checking, and each property alone is not enough. Hence, we think that gc-CMSGs should replace MSGs as the central class for the modelization of scenarios.

2 Message Sequence Charts (MSCs)

Message Sequence Charts (MSC for short) is a scenario language standardized by the ITU ([13]). They represent simple diagrams depicting the activity and communications in a distributed system. The entities participating in the interactions are called instances (or processes) and are represented by vertical lines. Message exchanges are depicted by arrows from the sender to the receiver. In addition to messages, atomic actions can also be represented.

Figure 1 gives an example of an MSC M modeling messages sent between processes *Host* and *Function*.

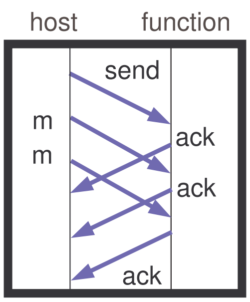


Fig. 1. MSC scenario of the bulk transaction of usb 1.1

Definition 1 *A* compositional *MSC (CMSC) is a tuple* $M = \langle \mathcal{P}, E, \mathcal{C}, t, m, < \rangle$ *where:*

- $\mathcal{P}$ *is a finite set of processes,*
- E_p *is a finite set of events on process* p*, with* $E = \bigcup_{p \in \mathcal{P}} E_p$
- $\mathcal{C}$ *is a finite set of names for messages and local actions,*
- $t : E \to \mathcal{T} = \{p!q(a), p?q(a), p(a) \mid p \neq q \in \mathcal{P}, a \in \mathcal{C}\}$ *labels an event with its* type*: either a send* $p!q(a)$ *of message* a *on process* p *to* q*, a receive* $p?q(a)$ *on* p *from* q*, or a local event* $p(a)$*. We partition* $E = S \bigcup R \bigcup L$ *into sends, receives and local events.*
- $m : S \to R$ *is a function matching a send to its corresponding receive. If* $m(s) = r$*, then* $t(s) = p!q(a)$ *and* $t(r) = p?q(a)$ *for some* $p, q \in \mathcal{P}, a \in \mathcal{C}$*.*
- $< \subseteq E \times E$ *is an acyclic relation between events consisting of:*
  - *a total order on* E_p*, for every process* $p \in \mathcal{P}$*, and*
  - $s < r$*, whenever* $m(s) = r$*.*

An MSC is a CMSC where the message function m *is a bijection.*

The event labeling t implicitly defines the process $P(e)$ for each event e: $P(e) = p$ if $t(e) \in \{p!q(a), p?q(a), p(a)\}$ for some $q \in \mathcal{P}, a \in \mathcal{C}$. We assume that channels are FIFO, i.e., there is no overtaking on messages sent on the same channel.

The relation $<$ is called the *visual* order on the CMSC, since it corresponds to its graphical representation. It is comprised of the process ordering and the message ordering. Since $<$ is required to be acyclic, its reflexive-transitive closure $<^*$ is a partial order on the set E of events, which we will denote by $\leq$. An extension of $\leq$ to a total order is called a linearization of M. We consider labeled linearizations $t(e_1)\cdots t(e_n)$, with $e_1 \cdots e_n$ a linearization on events. One can understand any linearization as some particular execution of the CMSC. Notice that because of the FIFO condition, one can retrieve an CMSC from any of its linearization.

Definition 2 *We say that a linearization $t(e_1)\cdots t(e_n)$ is b-*bounded *if, for each channel (p, q), the difference between the number of sends $p!q$ and the number of receives $q?p$ in any prefix $t(e_1)\cdots t(e_i)$ is at most b, for any i. We say that an CMSC M is* existentially *(respectively* universally*) b-*bounded *if some (resp. every) linearization of M is b-bounded [16].*

MSCs specify only finite behaviors. This leads to MSC-Graphs (MSGs, also called H(igh level) MSCs) [13], that are just transition systems with nodes labeled by MSCs. Their semantics is defined by the *sequential composition* of two MSCs M_1, M_2. Let $M_i = \langle \mathcal{P}_i, E_i, \mathcal{C}_i, t_i, m_i, <_i\rangle$. Their sequential composition is the MSC $M_1M_2 = \langle \mathcal{P}_1 \cup \mathcal{P}_2, E_1 \uplus E_2, \mathcal{C}_1 \cup \mathcal{C}_2, t_1 \cup t_2, m_1 \cup m_2, <\rangle$, where $<$ equals $<_1 \cup <_2 \cup \{(e_1, e_2) \in E_1 \times E_2 \mid P(e_1) = P(e_2)\}$ defined on the disjoint union of events $E_1 \uplus E_2$. Intuitively, the sequential composition glues two MSCs along their common process axis.

Definition 3 *An* MSG *is a labeled transition system $G = (V, \rightarrow, v^0, F, \lambda, L)$ with set of nodes V, set of MSCs L, transition relation $\rightarrow \in V \times V$, initial node v^0 and set of final nodes F. Each node v is labeled by an MSC $\lambda(v) \in L$. An accepting path of G is defined as a sequence of transitions $\rho = (v_1 \rightarrow v_2 \cdots \rightarrow v_k)$ with $v_1 = v^0$ and $v_k \in F$. An MSG G generates a set of MSCs $\mathcal{L}(H) = \{\lambda(v_1)\cdots\lambda(v_k) \mid v_1 \cdots v_k$ is an accepting path of $H\}$.*

Let G be an MSG. The MSG G is *finitely-MSC-generated* since any MSC generated by G is the composition of MSCs in L. Hence, any $M \in \mathcal{L}(G)$ is existentially b-bounded, where b is at most the size of the largest MSC in L. We call G an $\exists$-*b-bounded* MSG.

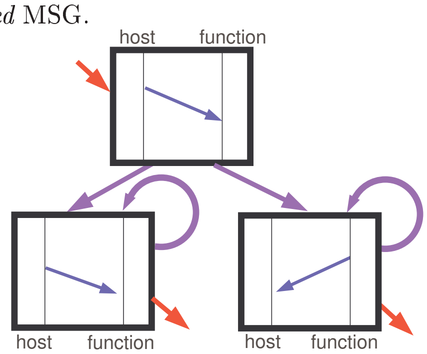


Fig. 2. MSG depicting the Isochronous transactions of usb 1.1

A CFM $\mathcal{A} = (\mathcal{A}_p)_{p\in\mathcal{P}}$ [5] consists of finite automata $\mathcal{A}_p$ associated with processes $p \in \mathcal{P}$, that communicate over unbounded, error-free, FIFO channels. 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 $C_{p,q}$. Each $\mathcal{A}_p$ 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 $p!q(a)$ means that a is appended to the channel $C_{p,q}$. Receiving message $p?q(a)$ means that a must be the first message in $C_{q,p}$, which will be then removed from $C_{q,p}$. A local action a on process p is denoted by $l_p(a)$. A run of a CFM is any MSC M such that the projection of M on process p is a run of A_p, for all p. We denote a run of the CFM as *successful*, if each process p can finish the run in some final state of F_p. The set of successful runs (MSCs) generated by $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A})$. We will denote a CFM A as $\exists$-b-bounded CFM if $\mathcal{L}(A)$ consists only of existsentially-b-bounded MSCs. Note that CFMs and MSGs are incomparable models. For instance, CFM are a priori not existentially-bounded.

MSGs are not expressive enough, which is a real drawback. For instance, the bulk transaction of usb 1.1 (see figure 3) is a protocol that is not finitely MSC-generated, hence not modelizable by MSGs. We follow here the solution of [9], proposing CMSG, that is MSG whose nodes can be instantiated by CMSC.

Definition 4 *A* CMSG *is a labeled transition system $G = (V, \rightarrow, v^0, F, \lambda, L)$ with set of nodes V, set of CMSCs L, transition relation $\rightarrow \in V \times V$, initial node v^0 and set of final nodes F. Each node v is labeled by a CMSC $\lambda(v) \in L$. An accepting path of G is defined as a sequence of transitions $\rho = (v_1 \rightarrow v_2 \cdots \rightarrow v_k)$ with $v_1 = v^0$ and $v_k \in F$.*

Definition 5 *A composition of CMSCs $M_1 \cdots M_n$, where $M_i = \langle \mathcal{P}, E_i, A_i, t_i, m_i, <_i\rangle$ is a CMSC $M = \langle \mathcal{P}, E = \bigcup_i E_i, \mathcal{A} = \bigcup \mathcal{A}_i, t = \bigcup t_i, m, <\rangle$ such that:*

- *The message function m extends each m_i and it is required that m preserves the FIFO restriction on matched events.*
- *The visual order $<$ is the union of $<_i$ and the set of (e, f) with $m(e) = f$ or $P(e) = P(f)$ and $e \in E_i, f \in E_j$, $j > i$.*

Notice that a sequence of given CMSCs $M_1 \dots M_n$ can admit several compositions. However, there can be at most one such composition that is an MSC, since the k-th send from p to q of an MSC is matched by the k-th receive on q from p. For instance, there exists only one MSC which is a composition of CMSCs seen along the loop-free path of figure 3. This MSC is depicted in Figure 1. Since $\mathcal{L}(G)$ consists only of (complete) MSCs, we only consider this unique composition (if it exists), for each accepting path. We denote the set of all MSCs as MSC.

Definition 6 *Let G be a CMSG. The language of G is $\mathcal{L}(G) = \{\lambda(v_1) \cdots \lambda(v_k) \in MSC \mid v_1 \cdots v_k$ is an accepting path of $G\}$.*

A CMSG may or may not be finitely MSC-generated. Notice that finite CMSC-generation is not a relevant notion, since any CFM or CMSG is finitely generated by its set of events, which yields no result at all about bound on message channels. The size $|M|$ of a (C)MSC M is its number of events. The size $|G|$ of a (C)MSG G is the sum of the sizes $|M|$ of (C)MSCs $M \in L$. The size of $\mathcal{P}$ is the number $\wp$ of processes.

3 When Expressivity meets Model-Checking

3.1 Existential Bounds on Channels

In [9], the authors show how to represent any CFM by a CMSG. Figure 3 depicts (parts of) the bulk transactions of the protocol usb 1.1 as a CMSG. The drawback of this power of expressivity is that basically, any non trivial algorithmic problem for CMSGs (e.g. pattern matching [7]) is undecidable, as this is already the case for CFMs. To overcome the undecidability for CFMs, one solution is to limit universally the size of the message channels. However, this restriction decreases severely the expressive power: universally bounded CFMs cannot model even simple protocols. For instance, the isochronous transaction of usb 1.1 (see figure 2) consists of a sequence of messages from process host to process function without requiring acknowledgments, which makes it non universally-bounded. Second, algorithms on bounded CFMs are often exponential in the maximal channel size b (e.g. pattern matching [7]). Hence, the bound b can seldom be defined large enough to take care of every linearization of a b-bounded CFM. For instance, the control transfer of usb 1.1 is basically a sequence of at most 1024 messages from host to function, followed by one message from function to host. Thus, it is universally 1024-bounded.

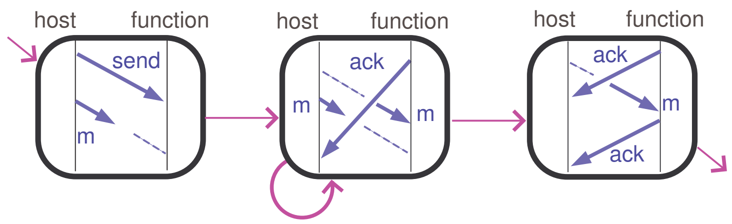


Fig. 3. CMSG depicting Bulk transactions of usb 1.1

The solution applied here is to consider only representative linearizations rather than every linearization: we will define these representatives so that they are b-bounded, with a small value of b. Recall that an MSC M is *existentially-b-bounded* if M has a linearization such that number of sends of a given channel p, q never exceeds b plus the number of receives on that channel. For instance, in the case of isochronous transfer of usb 1.1, the linearizations that receive a message immediately after it was sent are 1-bounded. Therefore, the isochronous transfer is existentially-1-bounded. Recall that a CFM is $\exists$-*b-bounded* if it generates only existentially-b-bounded MSCs.

To overcome the undecidability for several questions on CMSGs, the authors proposed a restriction called *safe CMSGs* (they called it *realizable CMSG* which is ambiguous wrt. the implementation problem). This subclass still contains all MSGs, and the restriction can be tested syntactically in polynomial time, [9].

Definition 7 *A CMSG G is safe if every sequence of CMSCs labeling an accepting path of G admits one composition that is an MSC.*

Recall that some paths of a (non safe) CMSG may not admit a composition that is an MSC (e.g. because number of sends and receives from p to q are not equal). For instance, figure 2 depicts a safe CMSG. Being safe implies that each loop of the CMSG G is labeled by the same number of sends and receives from p to q, for each pair of processes p, q. This is the key argument for the syntactical characterization of safe CMSGs that can be checked in polynomial time. This also implies that a safe CMSG G is existentially-$|G|$-bounded. Notice however that the size $|G|$ of G can be significantly larger than the optimal bound. However, we can decide in NP whether G is $\exists$-b-bounded [16].

Compared with MSGs, CMSGs have lost the existential bound that comes together with the finite generation, which explains their undecidability. Hence, safeness of CMSGs can be interpreted as a restriction that ensures an existential bound, without imposing the finite generation. For instance, pattern matching is decidable for safe CMSGs [7].

Proposition 1 *Any safe CMSG is an $\exists$-bounded CMSG. Safe CMSGs are strictly more expressive than $\exists$-bounded CFMs.*

Proof. Any safe CMSG G generates only $|G|$-bounded MSCs.

Constructing a safe CMSG from an existentially-bounded CFM is easy. We omit the proof since we will show a stronger result in theorem 3. $\square$

3.2 Globally-Cooperative CMSGs can be Model-Checked

We are interested in negative model-checking, that is checking whether a given (positive) CMSG contains an MSC generated by a given (negative) CMSG. Since safe CMSGs extend MSGs, the undecidability of model-checking holds for this class [18, 2].

Definition 8 *The communication graph of a CMSC M is a directed graph whose vertices are the processes involved in M, and there is an edge between vertices p, q iff M contains both a send $p!q$ from p to q and a receive $q?p$ on q from p (these send and receive may not define the same message).*

For instance, the communication graph of the MSC made of one message from process 3 to 2 and one message from process 1 to 2 is weakly connected. We recall that a loop in a CMSG is a path starting and ending in the same node. For instance, we do not require that the loop is simple, that is, a loop can meet several times the same node.

Definition 9 *A CMSG is loop-connected if every loop is labeled by a CMSC whose communication graph is weakly connected.*

*A CMSG is globally-cooperative (*gc-CMSG *for short) if it is safe and loop-connected,*

That is, if there is a loop labeled by two groups of processes without communication between the two groups, then G is not loop connected, hence not globally-cooperative.

Restricting a CMSG to be globally-cooperative is not a huge restriction, since having a system with a non-connected loop would mean that two non-communicating patterns should both happen exactly n times, for any n, which is impossible to ensure with a CFM. For example, we can model the whole usb 1.1 protocol using a gc-CMSG: figures 2,3 are gc-CMSGs. Let us state some results about model-checking a CMSG G.

- If G is globally-cooperative, universally bounded and finitely MSC-generated, then G is equivalent to an finite automaton.
- If G is globally-cooperative and generated by the finite set of MSCs S, then we can build an automaton on the alphabet S generating a set of representatives for G.
- If G is globally-cooperative, then one can build an automaton generating a set of *bounded* representative for G.

The first class above is that of regular MSGs [18, 2]. The second class is that of globally-cooperative MSGs [8]. We show in proposition 2 the third result. It implies that model-checking is decidable, generalizing the first two results (see theorem 1).

Lemma 1. *Let G, H be two gc-CMSGs. Let b be the existential-bound of G, and let $L^b(X)$ be the set of b-bounded linearizations of $X \in \{G, H\}$.*

$\mathcal{L}(G) \cap \mathcal{L}(H) = \emptyset$ *iff* $L^b(G) \cap L^b(H) = \emptyset$,
$\mathcal{L}(G) \subseteq \mathcal{L}(H)$ *iff* $L^b(G) \subseteq L^b(H)$.

One direction of this lemma is obvious, and the other one relies on the fact that it is possible to retrieve an MSC from any of its linearization. We show now that $L^b(X)$ is effectively regular for any loop-connected CMSG X. The proof looks like the one used in [18, 8], except we encode our MSCs with Mazurkiewicz traces. More precisely, we show how to use for existentially bounded MSCs the trace alphabet [14] defined for universally bounded MSCs. Compared to alphabet used in [8], it has the drawback to make the system exponentially bigger. However, we will show later how to overcome this exponential blow-up.

From the set of event types $\mathcal{T}$ (see definition 1), we define a (Mazurkiewicz) trace alphabet $\Sigma = \mathcal{T} \times \{0, \dots, b-1\}$ with the dependence relation $(x, n_1)D(y, n_2)$ iff $P(x) = P(y)$ or $\{(x, n_1), (y, n_2)\} = \{(p!q(a), n), (q?p(a), n)\}$ for some $p, q \in \mathcal{P}$ and $0 \leq n < b$. We will denote by $I = \Sigma \times \Sigma \setminus D$ the independence relation. For a set L, we denote by $[L]_I$ its closure under I-commutation, that is the set of words $w' \equiv_I w$ that can be obtained from some $w \in L$ by commuting adjacent letters $(a,b) \in I$. Notice that $[L^b(G)]_I = L^b(G)$, that is $L^b(G)$ is closed under I-commutation. We construct now an automaton $\mathcal{A}$ generating representatives for $L^b(G)$, that is $[L(\mathcal{A})]_I = L^b(G)$. We need later to compute its closure to obtain $L^b(G)$.

We can assume without lack of generality that the CMSG G has only one event per node. Then we construct an automaton $\mathcal{A}$ over the alphabet Σ, remembering in each state the corresponding node in the CMSG and the number of sends $s_{p,q}$ and receives $r_{p,q}$, modulo b, for each channel (p,q). Hence a node x in G, labeled by $p!q(a)$, corresponds in $\mathcal{A}$ to the set of transitions $v \xrightarrow{(p!q(a),n)} w$, with $0 \leq n < b$ and v, w corresponding to transitions of G ending and starting in x respectively. We requiere that the numbers of sends and receives of v and w are equal, up to the number of sends from p to q, that equals $n-1$ in v and n in w. In order to keep only b-bounded linearizations, we forbid transitions labeled by an event $(p!q(a), n)$ from a state with $r_{p,q} = n$.

It suffices to compute $[L(\mathcal{A})]_I$. Since $\mathcal{A}$ is loop-I-connected and of size $|G|b^{\wp}$, we can generate $[L(\mathcal{A})]_I$ by an automaton of doubly exponential size in the number $\wp$ of processes. However, we show now that since $\mathcal{A}$ is built from a smaller loop-connected graph G, we can reduce the complexity.

Lemma 2. *For any path of $\mathcal{A}$ labeled by a sequence $t_0 u_1 \cdots t_{k-1} u_k t_k$ with t_i, u_j non empty words over $\Sigma^\star$, with $t_i I u_j$ for all $i < j$, we have $k < \wp^2 b|G| + \wp|G|$.*

Proof. By definition, any path of $\mathcal{A}$ is associated with a path of G. Assume by contradiction that we have a set J of $(\wp/2)^2 b + \wp$ indices i such that t_i corresponds to a subpath of G starting in the same node s of G. I.e., for any consecutive $i, i' \in J$, $t_i \cdots u_{i'-1}$ is a loop around s. There are at most $\wp$ such loops where $u_i \cdots u_{i'-1}$ and $t_i \cdots t_{i'-1}$ share a process, since $t_i I u_j$ for every $i < j$ (the same process cannot be shared in two different loops). Since each loop is connected, either there is a shared process between the t's and the u's, or there is some $p!q$ in the t's and some $q?p$ in the u's (or the opposite). Since there are $\wp^2 b$ such loops, there are at least b sends or receives on the same channel $\{p,q\}$. Thus one of these events of some t_i is ordered with some event on the same channel on u_j with $i < j$, contradicting $t_i I u_j$. □

We can use now the construction from [18] which shows that for an automaton which fulfills lemma 2, one can build an automaton of size $|G|^{O((\wp b)^2 |G|)}$ that generates $[L(\mathcal{A})]_I = L^b(G)$.

Proposition 2 *Let G be a loop-connected CMSG. Then $L^b(G)$ is generated by an automaton B of size at most $|G|^{O(b^2 \wp^4 |G|)}$, for any $b > b_g$.*

By computing the automaton B on-the-fly, we can solve the model-checking problem easily (lower bounds hold for the less expressive bounded MSGs [18]):

Theorem 1 *Let G_1, G_2 be two gc-CMSGs. Then,*

1. *Deciding whether $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) = \emptyset$ is PSPACE-complete.*

2. Deciding whether $\mathcal{L}(G_1) \subseteq \mathcal{L}(G_2)$ is EXPSPACE-complete.

We exhibited two necessary properties to test the non-empty intersection between two message passing systems. Namely, we need an (existential) bound on channels for one system, else we can encode *Post Corespondance Problem (PCP)* using channels. Moreover, we need to compute the closure under commutation (up to the bound on buffers) for either the same system or the other one. Else we can also encode PCP using the concurrency between events. Safeness ensures an existential bound, while loop-connection ensures the ability to compute the closure under commutation. Such a formalism helps to understand most previous results. First, partial orders logics (MSO, TLC^- and LTL over MSCs, Template MSCs) are closed under commutation, hence we can model-check them against any *safe* CMSG [20, 17, 7], which provides the existential bound on channels. Note that the authors of [17] defined CMSGs with our definition of safe CMSG, which can lead to some confusion. We can also state a new result: we can decide in PSPACE the model-checking of a safe CMSG against a CFM, since the safe CMSG is existentially-bounded while the CFM is closed under commutation. The result of this section is that we can model-check a gc-CMSG against a gc-CMSG or any property expressed by word automaton, such as MSO or LTL over words (and still of course over MSCs).

3.3 Globally-cooperative CMSGs are expressive

We show in this section that global-cooperativity is not a huge restriction. Actually we show how to build a gc-CMSG from any $\exists$-bounded CFM. We use the theorem of [19] (see also [4]), which is to consider only linearizations of (Mazurkiewicz) traces that are in lexicographic normal form (LNF for short). This generates an automaton with connected loops, only. We will hence use the same trace alphabet as before. We give a different proof of Ochmanski's Theorem so that we know the exact complexity.

Ochmanski's theorem (theorem 2) states that over a trace alphabet (Σ, I), one can build an equivalent loop-I-connected automaton B from any automaton $\mathcal{A}$ such that $L(\mathcal{A})$ is closed under I-commutation. A loop is I-connected if it is not labeled by w with $w \equiv_I xy$, xIy and non-empty x, y.

Theorem 2 *Let I be an independence relation over an alphabet Σ. Then one can transform any automaton $\mathcal{A}$ whose language is closed under I-commutation into a loop-I-connected automaton $\mathcal{B}$ such that $[L(\mathcal{B})]_I = L(\mathcal{A})$.[19]. Moreover, $\mathcal{B}$ can be obtained of size $O(|\Sigma|!\mathcal{A})$.*

Proof. Let $\prec \subseteq \Sigma \times \Sigma$ be a total order over the alphabet. We extend it by the following recursive definition to a relation between words of $\Sigma^\star$, with $x_1 \cdots x_n \prec y_1 \cdots y_n$ iff $x_1 \prec y_1$, or $x_1 = y_1$ and $x_2 \cdots x_n \prec y_2 \cdots y_n$. Note that $\prec$ is a total order restricted to a set of I-equivalent words. Then a word $x_1 \cdots x_n$ is in LNF iff it is minimal among its set of I-equivalent words.

Let $\mathcal{A}$ be an automaton whose language is closed under I-commutation We modify the automaton $\mathcal{A}$ to obtain $\mathcal{B}$. In $\mathcal{B}$, we keep track of the last event e_σ of type σ seen along a path, for every letter σ. Additionally, we remember the set S_σ of events seen so far in the linearization since e_σ[1]. Each time an event of type σ appears, e_σ is set to this event and S_σ is initialized to $\{e_\sigma\}$. The transitions in $\mathcal{B}$ corresponds to transitions in $\mathcal{A}$, except that we delete the transitions in $\mathcal{B}$ labeled by f if for some σ, $f \prec e_\sigma$ and fIg for all $g \in S_\sigma$. That is, if $f \prec e_\sigma$ and has no I-ancestor that was already checked against e_σ, then we should delete the arrow labeled by f.

We have built $\mathcal{B}$, and we show now that $[L(\mathcal{B})]_I = L(A)$ and $\mathcal{B}$ is loop-I-connected. Assume by contradiction that a loop of $\mathcal{B}$ is not I-connected, that is labeled by a word w s.t. $w \equiv_I xy$ with xIy. We have linearizations $x = a_1 \dots a_n$ and $y = b_1 \dots b_n$ such that $a_1 b_1 \dots a_n b_n$ labels the loop. We can assume a_i, b_i to be non-empty sequences of events (up to a_1, b_n). In particular, the sequence $a_1 b_1 \dots a_n b_n a_1 b_1 \dots a_n b_n$ labels a (non initial/final) path of B. We denote by σ_i, τ_i the types of the first events f_i, g_i of a_i, b_i, respectively. Since $e_i I b_i$, and $S_{\sigma_i} \subseteq b_i$, we have $\sigma_i \prec \tau_i$. In the same way, $\tau_i \prec \sigma_{i+1}$. By induction, we obtain a contradiction since $\sigma_1 \prec \dots \prec \sigma_1$ is not possible.

Let us show now that $[L(\mathcal{B})]_I = L(\mathcal{A})$. By construction, $[L(\mathcal{B})]_I \subseteq L(\mathcal{A})$. For the converse, we show that every linearization $a_1 \dots a_n$ in LNF labels a path of $\mathcal{B}$. By contradiction, let $a_1 \dots a_j$ be a prefix that is not a label of an initial path in B. Hence there must exist a_i, $i < j$, such that a_j is independent from $S_{P(a_i)}$ and $a_j \prec a_i$. That is $a_k I a_j$, for $i < k < j$. Hence $a_i \cdots a_{j-1} I a_j, a_j \prec a_i$. Then $a_1 \dots a_j a_i \dots a_n$ would be smaller for $\prec$ than an equivalent $a_1 \dots a_n$ in LNF, a contradiction.

For the complexity, it suffices to notice that the number of orders over Σ is precisely $|\Sigma|!$. $\square$

We apply again the construction of [14], as for gc-CMSG in the previous section. This time, we apply it to an $\exists$-b-bounded CFM. We work over the trace alphabet $\Sigma = \mathcal{T} \times \{0, \dots, b-1\}$, and the independence relation I is $(x, n_1) I (y, n_2)$ iff $P(x) \neq P(y)$ and $\{(x, n_1), (y, n_2)\} \neq \{(p!q, n), (q?p, n)\}$ for any $p, q \in \mathcal{P}$ and $n < b$. We build a b-bounded automaton $\mathcal{A}$ over the alphabet Σ from the CFM, by taking the product of the process automata and limiting the messages buffers to b. Note that $L(A)$ is closed under I-commutation (but not under the visual order). We can then apply Ochmanski's theorem and we obtain an equivalent loop-I-connected automaton B. We can interpret B as CMSG. Its loops are I-connected. Moreover, we can delete every non accessible state. Hence, every path in the graph corresponds to an accepting path in the CFM, that is every path is labeled by an MSC: $\mathcal{B}$ is safe. Notice that any I-connected loop w.r.t. Σ is loop-connected w.r.t. the original alphabet $\mathcal{T}$, hence G is a gc-CMSG.

Theorem 3 *The class of gc-CMSGs contains the class of $\exists$-bounded CFMs.*

First, we can state a direct consequence of [12]:

Theorem 4 *The class of universally-bounded gc-CMSGs (*regular CMSGs *for short) is expressively equivalent to the class of $\forall$-bounded CFMs.*

[1] We can encode the sets (S_σ) as an order on Σ

We conjecture that gc-CMSG are expressively equivalent to the class of $\exists$-bounded CFMs.

4 Implementation by CFMs?

It is easy to see that not every gc-CMSG is implementable by a deadlock-free CFM. Since any specification should be implemented, we need an algorithm to know whether a deadlock-free implementation is possible.

Definition 10 *A CMSG G is deadlock-free implementable iff there exists some CFM $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(G)$, and every reachable global state can reach a final state.*

We know an EXPSPACE-complete algorithm to test whether a gc-MSG is deadlock-free implementable [1, 15]. Anyway, there are two drawbacks in such an approach. First, the algorithm is obviously time-consuming. Last, implementing directly an MSG is too extreme, since some easily implementable MSGs are said not to be, as the gc-MSG of figure 2. The reason is that the data written in messages is abstracted away, hence both host and function can choose independently to send messages. The solution already used in [8, 12] is to allow data to be added to messages.

Definition 11 *A CMSG G is deadlock-free implementable with additional data iff there exists some data projection Proj, and some CFM $\mathcal{A}$ with $Proj(\mathcal{L}(A)) = \mathcal{L}(G)$, and every reachable global state can reach a final state.*

The problem is that we have no algorithm to test this implementability. Moreover, even in this case, it would probably be too time consuming. We propose then an alternative approach to the problem, trying to go through a class which is easily deadlock-free implementable with additional data. The reason for non-implementability of an MSG is the global control, whereas the choices in a CFM must be done locally. The idea is then to define MSGs that have only local choices, that is any node is controlled by a single process [3, 11].

Definition 12 *An MSG (resp CMSG) $G = \langle V, R, v^0, V_f, \lambda \rangle$ is called* local-choice *(*lc-(C)MSG *for short) if each (C)MSC labeling a node v of G is a* **Triangle***, that is it has a single minimal event* $\min(v)$ *for the partial order $\leq$. Moreover,* $\min(w)$ *must belong to the process set $P(v)$ of node v, whenever $v \to w$. A lc-CMSG is also required to be safe.*

Checking that a (C)MSG is local-choice can be done in polynomial time. Note that lc-CMSG is a subclass of gc-CMSG. The MSG in figure 2 is a lc-(C)MSG, while the CMSG in figure 3 is not (processes function and host both begin to send in the nodes on the loop). Anyway, we can transform this CMSG into a lc-CMSG generating the same MSCs (see figure 4). We want to give a generic algorithm to do that automatically. Note that a lc-CMSG can model the whole usb 1.1 protocol.

The next theorem is an easy adaptation of the result in [8].

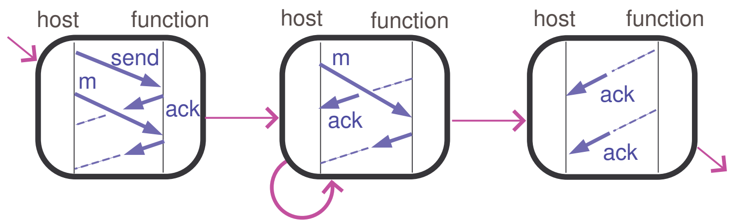


Fig. 4. A local-choice CMSG depicting Bulk transactions of usb 1.1

Proposition 3 *[8] Every lc-CMSG G is implementable by a deadlock-free CFM with additional data which is of size linear in $|G|$.*

Checking whether a gc-MSG is equivalent to an lc-MSG can be done in PSPACE and leads to an equivalent lc-MSG of doubly exponential size [6]. We show that we can transform a gc-CMSG into some lc-CMSG in co-NP and this leads to an equivalent lc-CMSG of exponential size. While local-choice is defined syntactically, we show that it corresponds to a semantic property. Triangles are of huge importance here. We first define a generic lc-CMSG H_n over Triangles of size bounded by n. The CMSG H_n has for each Triangle T of size at most n, one node v_T labeled by T. There is an edge $v_T \to v_{T'}$ if $P(\min(T)) \in P(T')$.

Proposition 4 *A gc-CMSG G is equivalent to some lc-CMSG iff there exists some n such that $\mathcal{L}(G) \subseteq \mathcal{L}(H_n)$. If this is the case, then we can obtain a lc-CMSG equivalent to G, of size exponential in $|G|$ and n.*

Proof. If $\mathcal{L}(G) \nsubseteq \mathcal{L}(H_n)$, then G cannot be equivalent to some lc-CMSG since a lc-CMSG H would satisfies this property for $n = |H|$.

If $\mathcal{L}(G) \nsubseteq \mathcal{L}(H_n)$, we compute an automaton $\mathcal{A}$ recognizing $L^b(G)$ using proposition 2, exponential in $|G|$. Remember that H_n is exponential in n too. We then synchronized $\mathcal{A}$ with H_n, yielding a lc-CMSG H of size $|\mathcal{A}| \times |H_n|$.

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

If the test in proposition 4 on G answers yes, then an equivalent lc-CMSG can be constructed by synchronizing G and H_n.

While it is PSPACE in $n, |G|$ to test whether $\mathcal{L}(G) \subseteq \mathcal{L}(H_n)$, the value of n is not bounded so far. Actually, we can fix the bound to be $b_0 = 2\wp|G| + 1$. If $\mathcal{L}(G) \nsubseteq \mathcal{L}(H_{b_0})$, we prove that $\mathcal{L}(G) \nsubseteq \mathcal{L}(H_n)$ for any n.

Let us take the decomposition of a Triangle T into sub-Triangles $T = T_1T_2$. The minimal event are named $\min(T) = \min(T_1) = e$ and $\min(T_2) = f$. There are at most 2 events g, h immediately after e (g on the same process of e, and if e is a send, the matching receive h). Obviously, either $f \geq g$ or $f \geq h$. In order to minimize the size of the T_1, an optimal choice is to take either $f = h$ or $f = g$. The Triangle T_2 is defined as being the set of events $F(f)$ in the future of f, and

T_1 is the set of events not in $F(f)$ but in $F(e)$, that is $G(f) = F(e) \setminus F(f)$. We show that if we can find a large enough $G(f)$, then we are able to find a loop we can iterate so that $G(f)$ becomes as large as we want.

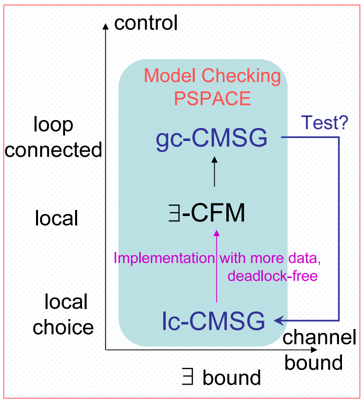
Theorem 5 *A gc-CMSG G is equivalent to some lc-CMSG iff $\mathcal{L}(G) \subseteq \mathcal{L}(H_{b_0})$, with $b_0 = 4\wp^2|G|^2 + 1$. Moreover, if it is the case, then an equivalent lc-MSG of exponential size in $|G|$ can be constructed. This test can be done in co-NP.*

Proof. Assume that $\mathcal{L}(G) \not\subseteq \mathcal{L}(H_{b_0})$. It means that there exists an MSC $M \in \mathcal{L}(G)$, a send event $e \in M$ such that for all events $f \in \{g, h\}$ immediately following e, we have $|G(f)| > b_0$. Else, we could decompose M in a sequence of triangle of size at most b_0. This test can be easily done in co-NP.

M is the labeling of a path in G. Since there are b_0 events in $G(g)$, there are at least $4\wp^2|G|+1$ occurences of the same event $e_g \in G(g)$ of G. That is, we can decompose M into a sequence $M = BB_1 \cdots B_n B'$ with B_i labeling a loop of G. Moreover, B_i begins (and ends) with e_g, and $n = 4\wp^2|G| + 1$.

Among these n loops, at most $2\wp^2|G|$ can change the processes in $F(g)$. More precisely, let $F_i(g) = (BB_1 \cdots B_i) \cap F(g)$ be the future of g at the beginning of B_i. There are at most $2\wp^2 B$ loops B_i with the Ω-types seen in $F_{i-1}(f)$ are strictly included in those of $F_i(f)$, since the types of $(F_i(f))_i$ is an increasing and bounded sequence. In the same way, at most $2\wp^2|G|$ loops can change the Ω-types in $F(h)$. So there is one loop at least that changes neither the Ω-types in $F(g)$ nor the processes in $F(h)$, so we can iterate it without decreasing the outcome of $G(g)$ and $G(h)$. It contains at least one element from $G(g)$. We act the same way for a decomposition of loops for $G(h)$. There is at least one loop we can iterate, and that contains one element from $G(h)$. Since G is safe, the CMSC obtained when iterating the loops is still an MSC, hence in $\mathcal{L}(G)$. We obtain for all k an MSC $M_k \in \mathcal{L}(G) \setminus \mathcal{L}(H_k)$. So G is not equivalent to any lc-CMSG. □

5 Conclusion



To sum up this paper, we combine in gc-CMSGs extension of MSG by means of compositionality and restriction to globally-cooperation for decidability of model-checking. Surprisingly, we show that this combination do very well, as we keep both expressivity and decidability results in same complexity! We defined also lc-CMSG that are included into deadlock-free ∃-bounded CFM.

Moreover, we define a new co-NP algorithm for implementing a gc-CMSG through some lc-CMSG, to achieve a deadlock-free CFM, which is at most exponentially larger than the gc-CMSG. We pointed out the very reasons for decidability of message passing system, namely existential bounds on channels and ability to compute representatives for closure under commutation. Hence we hope that similar results can be stated for other settings. The figure aside summarizes the results of this paper.

Acknowledgments I would like to thank an anonymous referee for pointing out a technical bug in an earlier version of the paper, and Anca Muscholl for fruitful discussions.

References

1. R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of MSC graphs. In *ICALP'01*, LNCS 2076, pp.797-808, 2001.
2. R. Alur and M. Yannakakis. Model checking of message sequence charts. In *CONCUR'99*, LNCS 1664, pp.114-129, 1999.
3. H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in MSCs. In *TACAS'97*, LNCS 1217, pp.259-274, 1997.
4. V. Diekert and G. Rozenberg, editors. *The Book of Traces.* World Scientific, Singapore, 1995.
5. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):pp.323-342, 1983.
6. B. Genest and A. Muscholl. The structure of local choice in HMSC Internal report LIAFA, 2003 Available at http://www.crans.org/~genest/report_lc.ps.
7. B. Genest, M. Minea, A. Muscholl, and D. Peled. Specifying and verifying partial order properties using template MSCs. In *FoSSaCS'04*, LNCS 2987, pp.195-210, 2004.
8. B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun. Infinite-state High-level MSCs: Model-checking and realizability. In *ICALP'02*, LNCS 2380, pp.657-668, 2002.
9. E. Gunter, A. Muscholl, and D. Peled. Compositional Message Sequence Charts. In International Journal on Software Tools for Technology Transfer (STTT), Volume 5, Springer, 2003.
10. D. Harel and R. Marelly. Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Springer 2003.
11. L. Hélouët and P. Le Maigat. Decomposition of Message Sequence Charts. In SAM2000, pp.46-60, 2000.
12. J. G. Henriksen, M. Mukund, K. Narayan Kumar, M. Sohoni and P. Thiagarajan. A Theory of Regular MSC Languages To appear In *IC*, 2004, available at http://www.comp.nus.edu.sg/~thiagu/icregmsc.pdf.
13. ITU-TS recommendation Z.120, Message Sequence Charts, Geneva, 1999.
14. D. Kuske. Regular sets of infinite message sequence charts. In *Information and Computation 187*, Academic Press, pp.80-109, 2003.
15. M. Lohrey. Safe realizability of High-level Message Sequence Charts. In *CONCUR'02*, LNCS 2421, pp.177-192, 2002.
16. M. Lohrey and A. Muscholl. Bounded MSC communication In *Information and Computation 189*, Academic Press, pp.135-263, 2004.
17. P. Madhusudan and B. Meenakshi. Beyond Message Sequence Graphs In FSTTCS'01, LNCS 2245, pp.256-267, 2001.
18. A. Muscholl and D. Peled. Message Sequence Graphs and decision problems on Mazurkiewicz traces. In *MFCS'99*, LNCS 1672, pp.81-91, 1999.
19. E. Ochmanski. Regular behaviour of concurrent systems in Bulletin of the EATCS 27, pp.56-67, 1985.
20. D. Peled. Specification and verification of Message Sequence Charts. In *FORTE/PSTV'00*, pp.139-154, 2000.
21. USB 1.1 specification, available at http://www.usb.org/developers/docs/usbspec.zip

Appendix

Here is an example for the which we need an exponential-blowup for coming from gc-CMSG to lc-CMSG. Since there is a loop on host, we need to put the n local events on the same node. Since we have 2 choices for each local events, it yields 2^n nodes in any equivalent lc-CMSGs.

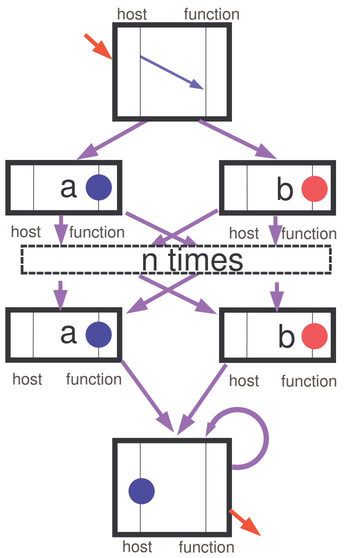


Fig. 5. gc-CMSG hard to turn into a lc-CMSG

Theorem 5 *Every lc-CMSG G is implementable by a deadlock-free CFM with additional data which is of size linear in $|G|$.*

The idea is to use a *leader process* and to let the current leader choose the current_node to be executed and the next_leader. The node is chosen among the nodes that follow the node being executed, and that begin with a minimal event belonging to the leader. The next_leader should be chosen among the minimal processes of nodes that follow the chosen node.

In the procedure polling_state below process p waits for a message informing it about the next node to execute and the next leader:

```
void polling_state()
{ while (true) {
     if p receives a message (a,v,q) then
            { current_node=v; next_leader=q; return;} } }
```

Initially, the current node is initialized by letting next_leader $= pr(\min((v^0))$. Before executing its event from current_node, process p goes to a polling state, unless it is the leader process. Here is the algorithm for process p:

```
initialization();
 while (true)
 { if (p ≠ next_leader) polling_state();
   else {current_node=guess(current_node);
          next_leader=guessp(current_node);}
   execute_path(current_node); }
```

The algorithm execute_path(current_node) above makes that process p executes its events from current_node, if any. In this case each message sent by p contains the additional data (current_node, next_leader).

Note that in a Triangle (see definition 12), every process but the minimal process begins by a receive. A process that is chosen to be the leader is always informed, since it occurs in the node where it is chosen.