Comparing Channel Restrictions of Communicating State Machines, High-level Message Sequence Charts, and Multiparty Session Types

Felix Stutz Damien Zufferey
MPI-SWS, Kaiserslautern, Germany
{fstutz,zufferey}@mpi-sws.org

Communicating state machines provide a formal foundation for distributed computation. Unfortunately, they are Turing-complete and, thus, challenging to analyse. In this paper, we classify restrictions on channels which have been proposed to work around the undecidability of verification questions. We compare half-duplex communication, existential B-boundedness, and k-synchronisability. These restrictions do not prevent the communication channels from growing arbitrarily large but still restrict the power of the model. Each restriction gives rise to a set of languages so, for every pair of restrictions, we check whether one subsumes the other or if they are incomparable. We investigate their relationship in two different contexts: first, the one of communicating state machines, and, second, the one of communication protocol specifications using high-level message sequence charts. Surprisingly, these two contexts yield different conclusions. In addition, we integrate multiparty session types, another approach to specify communication protocols, into our classification. We show that multiparty session type languages are half-duplex, existentially 1-bounded, and 1-synchronisable. To show this result, we provide the first formal embedding of multiparty session types into high-level message sequence charts.

Acknowledgements and Funding. The authors would like to thank Emanuele D'Osualdo, Georg Zetzsche and the anonymous reviewers for their feedback and suggestions. This research was funded in part by the Deutsche Forschungsgemeinschaft project 389792660-TRR 248.

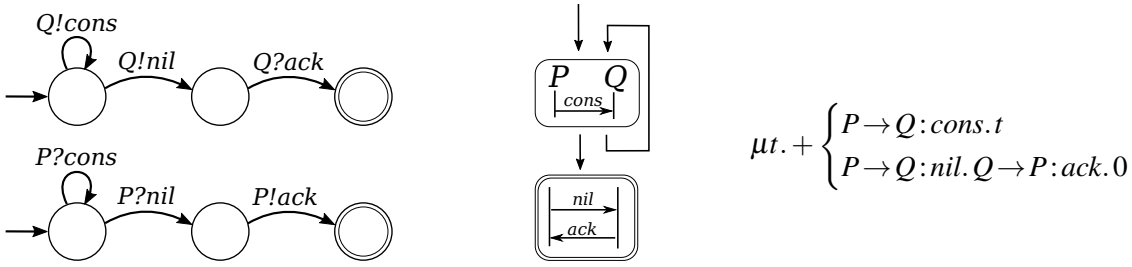
Extended Version: http://arxiv.org/abs/2208.05559

1 Introduction

Communicating state machines (CSMs) are one of the foundational models of message-passing concurrency. Unfortunately, the combination of multiple processes and unbounded FIFO channels yields a Turing-complete model of computation even when the processes are finite-state [14]. The communication channels can be used as memory and, therefore, most verification questions for CSMs are not algorithmically solvable. To regain decidability, one needs to exploit properties of specific systems. For instance, if all the runs of some communicating state machine use finite memory, it is possible to verify this system. This restriction, known as universal boundedness [24], admits only systems with finitely many reachable states.

In this paper, we compare three channel restrictions which allow infinite state systems while making interesting verification questions decidable. We compare half-duplex communication [17], existential B-boundedness [24], and k-synchronisability [13, 28].

D. Della Monica and P. Ganty (Eds.): 13th International Symposium on Games, Automata, Logics and Formal Verification (GandALF 22)
EPTCS 370, 2022, pp. 194–212, doi:10.4204/EPTCS.370.13



(a) Communicating state machine: one state machine for P (top) and one for Q (bottom)

(b) High-level message sequence chart

(c) Multiparty session type

Figure 1: Sending a list expressed in different formalisms. The left part is an implementation of the protocol specified in the middle and right parts.

We explain all three restrictions with the CSM in Fig. 1a. There, a process P sends a list, element by element, to a process Q. After receiving the list's end, Q sends an acknowledgement back to P.

Half-duplex communication requires that, at all times, at least one of both channels between two processes is empty. While P sends the list, the channel can grow arbitrarily large. However, Q always receives all the messages until *nil* before replying. When Q replies, the channel from P to Q is empty. Hence, the CSM is half-duplex.

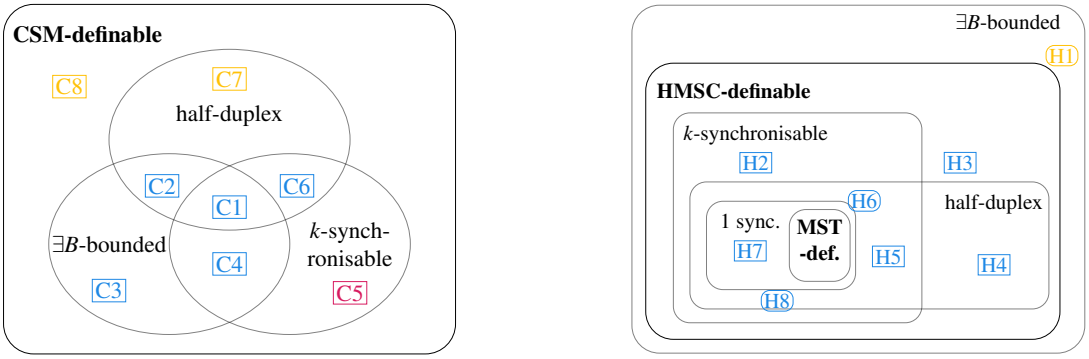
Existential B-boundedness means that, for every execution, we can reorder the sends and receptions such that the channels carry at most B messages. This CSM is existentially 1-bounded. Each reception is possible directly after the send.

k-synchronisability requires that every execution can be reordered and split into phases where up to k messages are first sent and then received. This CSM is 1-synchronisable because every message can be received directly after it was sent.

The original definitions of channel restrictions are phrased in terms of executions of a CSM. We present a characterisation for each restriction which only considers the generated language. This also allows us to reason about languages specified or generated in different ways. We consider languages given by protocol specifications and implementations. For implementations, we consider *CSM-definable languages*, i.e., languages which can be generated by a CSM.

Interestingly, for CSMs, these channel restrictions have not yet been compared thoroughly. In this paper, we close this gap and provide a classification of channel restrictions for CSM-definable languages. For instance, this answers a question for the FIFO point-to-point setting which has been posed for the mailbox setting by Bouajjani et al. [13] as we prove that existential B-boundedness and k-synchronisability are incomparable for CSM-definable languages. Overall, we give examples for every possible intersection and, thus, prove that none of the restrictions subsumes another one in this context. Our results for CSM-definable languages are summarised in Fig. 2a. In fact, we disprove one of the three known results from the literature [35, Thm. 7.1] which has been cited recently as part of a summary [12, Prop. 41]. This indicates that, despite their simplicity, these definitions hide some subtleties. Our classification provides a careful treatment — giving minimal examples for the sake of understandability.

Such a classification is interesting as focusing on languages or systems adhering to one of the channel restrictions can be key for solving verification problems algorithmically. For instance, control-state reachability and model checking LCPDL (propositional dynamic logic with loop and converse) formulas are decidable for k-synchronisable systems [28, 12]. Later, we highlight the impact of channel restrictions on verification questions and whether one can check if a system adheres to a restriction.



(a) CSM-definable Languages

(b) HMSC-definable Languages

Figure 2: Comparing half-duplex, existential B-bounded, and k-synchronisable systems. The results are known results, results are new, and the result disproves an existing result. Hypotheses with (rounded) corners indicate inclusions while [pointed] corners indicate incomparability results.

Protocol Specifications. Instead of considering arbitrary CSMs, it is possible to start with a global description written in a dedicated protocol specification formalism such as High-level Message Sequence Charts (HMSCs) [6, 26], Multiparty Session Types (MSTs) [32, 33], or Choreography Automata (CA) [7]. A protocol is a global specification of all the processes' actions together while an implementation only gives the local actions of each process. Fig. 1 shows, along the CSM, two protocol specifications. The key difference between a protocol specification and an implementation is that the protocol specification explicitly connects a send event to the corresponding receive event. In the HMSC (Fig. 1b), the arrows connect sends to receptions. The MST[1] (Fig. 1c) specifies communication by $sender \to receiver : message$. The CSM (Fig. 1a) does not specify this connection upfront and it may not exist. This makes CSMs strictly more general than protocols. For instance, an incorrect implementation of the protocol could have P terminate before receiving the acknowledgement.

The CSM, HMSC, and MST all have the same language. Thus, our observations on channel restrictions also hold for the HMSC and the MST. We also say that the CSM *implements* the protocol specified by the HMSC (or the MST) as they accept the same language and the CSM is deadlock free. In general, there are several approaches to obtain a CSM which implements a protocol (if one exists). For instance, a protocol specification can be projected on to each process. In this paper, we do not consider this problem. A protocol specification gives rise to a language, i.e., the protocol. We only need the protocol as our definitions for channel restrictions apply to languages, e.g., *HMSC-* and *MST-definable languages*.

For protocols, the classification of channel restrictions was less studied than for CSMs. Fig. 2b summarises our results. It was only known that each HMSC-definable language is existentially B-bounded for some B [24]. Surprisingly, the classification changes in the context of protocols. For restrictions which differ ([H2] to [H5], and [H7]), we give distinguishing examples. When one restriction subsumes another one ((H1), (H6), and (H8)), we prove it. For instance, (H6) proves that 1-synchronisability entails half-duplex communication while [H5] is an example which is half-duplex, existentially B-bounded, k-synchronisable but not 1-synchronisable.

Embedding MSTs into HMSCs. In addition to our results about CSM- and HMSC-definable languages, we provide the first formal embedding from MSTs into HMSCs. The contribution is two-fold. First, we situate MSTs in the picture of common channel restrictions and prove that languages specified by

[1] We actually present a global type in an MST framework here but only use the term after its formal introduction in Section 5

multiparty session types are half-duplex, existentially 1-bounded, and 1-synchronisable. This sheds a new light on why MSTs are effectively analysable. Second, we did recently show that using insights from the domain of HMSCs in the domain of MSTs is a promising research direction as we made the effective MST verification techniques applicable to patterns from distributed computing [38]. Hence, our formal embedding can act as a crucial building block for further advances which are facilitated by insights from both domains.

Contributions. In this paper, we make three main contributions. (1) We provide an exhaustive classification of channel restrictions for CSM-definable languages. In this process, we disprove a recent result from the literature. (2) We provide an exhaustive classification of channel restrictions for HMSC- and MST-definable languages. (3) We give the first formal embedding of MSTs into HMSCs.

Outline. After providing some preliminary definitions in Section 2, we define the channel restrictions formally in Section 3 and summarise their impact on the decidability of verification questions. Subsequently, we establish our results on HMSCs (Section 4), MSTs (Section 5), and CSMs (Section 6). We discuss related work in Section 7.

2 Preliminaries

Finite and Infinite Words. For an alphabet Σ, the set of finite words over Σ is denoted by Σ^*, the set of infinite words by Σ^ω, while we write $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ for their union. For two strings $u \in \Sigma^*$ and $v \in \Sigma^\infty$, u is said to be a *prefix* of v, denoted by $u \leq v$, if there is some $w \in \Sigma^\infty$ such that $u \cdot w = v$. For two alphabets Σ and Δ with $\Delta \subseteq \Sigma$, the *projection* of $w \in \Sigma^\infty$ on to Δ, denoted by $w{\Downarrow_\Delta}$, is the word which is obtained by omitting every letter in w that does not belong to Δ.

Message Alphabet. $\mathscr{P}$ is a finite set of processes, ranged over by $P, Q, R, \ldots$, and $\mathscr{V}$ a finite set of messages. For a process P, we define the alphabet $\Sigma_P = \{P \triangleright Q!m, P \triangleleft Q?m \mid Q \in \mathscr{P},\ m \in \mathscr{V}\}$ of events. The event $P \triangleright Q!m$ denotes process P sending a message m to Q, and $P \triangleleft Q?m$ denotes process P receiving a message m from Q. Note that the process performing the action is always the first one, e.g., the receiver P in $P \triangleleft Q?m$. The alphabet $\Sigma = \bigcup_{P \in \mathscr{P}} \Sigma_P$ denotes all send and receive events while $\Sigma_{sync} = \{P \to Q : m \mid P, Q \in \mathscr{P} \text{ and } m \in \mathscr{V}\}$ is the set where sending and receiving a message is specified at the same time. We fix $\mathscr{P}$, $\mathscr{V}$, Σ, and Σ_{sync} in the rest of the paper. We write $w{\Downarrow_{P \triangleright Q!\_}}$ to select all send events in w where P sends a message to Q and $\mathscr{V}(w)$ to project the send and receive events to their message values.

Distributed Executions. We use these specialised alphabets to model specifications in which multiple distributed processes communicate by exchanging messages. Furthermore, these executions cannot be any word but need to comply with conditions that correspond to the asynchronous communication over reliable FIFO channels. We call such words channel-compliant.

Definition 1 ([38]) *A* protocol *is a set of complete channel-compliant words where:*

1. Channel-compliant: *A word $w \in \Sigma^\infty$ is channel-compliant if messages are received after they are sent and, between two processes, the reception order is the same as the send order. Formally, for each prefix w' of w, we require $\mathscr{V}(w'{\Downarrow_{Q \triangleleft P?\_}})$ to be a prefix of $\mathscr{V}(w'{\Downarrow_{P \triangleright Q!\_}})$, for every $P, Q \in \mathscr{P}$.*
2. Complete: *A channel-compliant word $w \in \Sigma^\infty$ is* complete *if it is infinite or the send and receive events match: if $w \in \Sigma^*$, then $\mathscr{V}(w{\Downarrow_{P \triangleright Q!\_}}) = \mathscr{V}(w{\Downarrow_{Q \triangleleft P?\_}})$ for every $P, Q \in \mathscr{P}$.*

To pinpoint the corresponding send and receive events, we define a notion of *matching*.

Definition 2 (Matching Sends and Receptions) *In a word $w = e_1 \ldots \in \Sigma^\infty$, a send event $e_i = P \triangleright Q!m$ is* matched *by a receive event $e_j = Q \triangleleft P?m$, denoted by $e_i \mapsto e_j$, if $i < j$ and $\mathcal{V}((e_1 \ldots e_i)\Downarrow_{P \triangleright Q!\_}) = \mathcal{V}((e_1 \ldots e_j)\Downarrow_{Q \triangleleft P?\_})$. A send event e_i is* unmatched *if there is no such receive event e_j.*

If a sequence of events is channel-compliant, it is trivial that for each channel between two processes, either all send events are matched or there is an index from which all send events are unmatched.

In this paper, we consider protocols that can be specified with high-level messages sequence charts. We define *prefix message sequence charts* to allow unmatched send events, inspired by the work of Genest et al. [24, Def. 3.1]. The definition of a (prefix) MSC can look intimidating. In Fig. 3, we show pictorially what each component corresponds to.

Definition 3 ((Prefix) Message Sequence Charts) *A* prefix message sequence chart *is a 5-tuple $M = (N, p, f, l, (\leq_P)_{P \in \mathscr{P}})$ where*

- *N is a set of send (S) and receive (R) event nodes ($N = S \uplus R$),*
- *$p \colon N \to \mathscr{P}$ maps each event node to the process acting on it,*
- *$f \colon S \rightharpoonup R$ is an injective partial function linking corresponding send and receive event nodes,*
- *$l \colon N \to \Sigma$ labels every event node with an event, and*
- *$(\leq_P)_{P \in \mathscr{P}}$ is a family of total orders for the event nodes of each process: $\leq_P \subseteq p^{-1}(P) \times p^{-1}(P)$.*

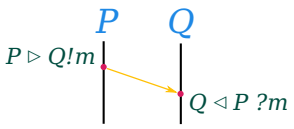


Figure 3: Highlighting the elements of a (prefix) MSC: $(N, p, f, l, (\leq_P)_{P \in \mathscr{P}})$

A prefix MSC M induces a partial order $\leq_M$ on N that is defined co-inductively[2]:

$$\frac{e \leq_P e'}{e \leq_M e'}\ \text{PROC} \qquad \frac{s \in S}{s \leq_M f(s)}\ \text{SND-RCV} \qquad \frac{}{e \leq_M e}\ \text{REFL} \qquad \frac{e \leq_M e' \qquad e' \leq_M e''}{e \leq_M e''}\ \text{TRANS}$$

The labelling function l respects the function f between S and R: for every pair of event nodes $e, e' \in N$ with $f(e) = e'$, we have $l(e) = p(e) \triangleright p(e')!m$ and $l(e') = p(e') \triangleleft p(e)?m$ for some $m \in \mathscr{V}$ and for every e where $f(e)$ is undefined, we have $l(e) = p(e) \triangleright P!m$ for some $P \neq p(e)$ according to its destination.

We say that M is *degenerate* if there is some P and Q such that there are $e_1, e_2 \in p^{-1}(P)$ with $e_1 \neq e_2$, $l(e_1) = l(e_2)$, $e_1 \leq_P e_2$ and $f(e_2) \leq_Q f(e_1)$. We say that M *respects FIFO order* if M is not degenerate and for every pair of processes P, Q, and for every two event nodes $e_1 \leq_M e_2$ with $l(e_i) = P \triangleright Q!\_$ for $i \in \{1, 2\}$, it holds that $f(e_2)$ is undefined if $f(e_1)$ is undefined as well as that it holds that $\mathcal{V}(w_P) = \mathcal{V}(f(w_P))$ where w_P is the (unique) linearisation of $p^{-1}(P)$.

In this paper, we do only consider prefix message sequence charts that respect FIFO order.

If f is total, we omit the term *prefix* and call M a *message sequence chart (MSC)*. If N is finite for an MSC M, we call M a *basic MSC (BMSC)*. We denote the set of BMSCs by $\mathscr{M}$. When M is clear from context, we simply write $\leq$ instead of $\leq_M$. For a prefix MSC M, the language $\mathscr{L}(M)$ contains a sequence $l(w)$ for each linearisation w of N compatible with $\leq_M$. When unambiguous, we may refer to event nodes or sequences thereof by their (event) labels or omit the label function l.

A prefix MSC, in contrast to an MSC, allows send event nodes for any channel to be unmatched from some point on. The concatenation $M_1 \cdot M_2$, or simply $M_1 M_2$, of an MSC M_1 and a prefix MSC M_2 is

[2] Note that we cannot use the standard reflexive and transitive closure since we consider infinite sequences of events. Co-induction lifts the reflexive, transitive closure of the union of the send-receive relation and all process orders, i.e., $(\{(s, f(s)) \mid s \in S\} \cup \bigcup_{P \in \mathscr{P}} \leq_P)^*$, to infinite sets of event nodes.

defined as expected (see the technical report [45] for the formal definition). The concatenation requires that, for any individual process, all event nodes in M_1 happen before the event nodes in M_2. However, the induced partial order on N may permit linearisations in which an event node from M_2 of one process occurs before an event node from M_1 of another process.

For every channel-compliant word w, one can construct a unique prefix MSC M such that w is a linearisation of M.

Lemma 1 ($\mathrm{msc}(\text{-})$ **([24], Section 3.1))** *Let $w \in \Sigma^\infty$ be a channel-compliant word. Then, there is unique prefix MSC, denoted by* $\mathrm{msc}(w)$*, such that w is a linearisation of* $\mathrm{msc}(w)$*. In case the above conditions are not satisfied,* $\mathrm{msc}(w)$ *is undefined.*

All sequences of events we consider in this work are channel-compliant. For sequences from MSCs (considered in Section 4), this trivially holds, while for sequences from execution prefixes of CSMs (considered in Section 6), we prove this in the technical report [45].

3 Channel Restrictions

In this section, we present different channel restrictions and their implications on decidability of interesting verification questions. Their application is discussed subsequently: for HMSCs in Section 4.1, for MSTs in Section 5.3, and for CSMs in Section 6.1.

3.1 Definitions

3.1.1 Half-duplex Communication

Cécé and Finkel [17, Def. 8] introduced the restriction of half-duplex communication which intuitively requires that, for any two processes P and Q, the channel from P to Q is empty before Q sends a message to P. We define the restriction of half-duplex on sequences of events and show that it is equivalent to the original definition in the technical report [45].

Definition 4 (Half-duplex) *A sequence of events w is called* half-duplex *if for every prefix w' of w and pair of processes P and Q, one of the following holds: $\mathcal{V}(w'\Downarrow_{P\triangleright Q!\_}) = \mathcal{V}(w'\Downarrow_{Q\triangleleft P?\_})$ or $\mathcal{V}(w'\Downarrow_{Q\triangleright P!\_}) = \mathcal{V}(w'\Downarrow_{P\triangleleft Q?\_})$. A language $L \subseteq \Sigma^\infty$ is half-duplex if every word $w \in L$ is.*

3.1.2 Existential B-boundedness

While the previous property restricts the channel for at least one direction to be empty, one can also bound the size of channels and consider linearisations that are possible adhering to such bounds. On the one hand, one can consider a universal bound that applies for every linearisation. However, this yields finite-state systems [24] and disallows very simple protocols, e.g., the example in Fig. 1. On the other hand, one can consider an existential bound on the channels which solely asks that there is one linearisation of the distributed execution for which the channels are bounded. This allows infinite-state systems and admits the earlier example.

Definition 5 (B-bounded [24]) *Let $B \in \mathbb{N}$ be a natural number. A word w is* B-bounded *if for every prefix w' of w and pair of processes P and Q, it holds that $|w'\Downarrow_{P\triangleright Q!\_}| - |w'\Downarrow_{Q\triangleleft P?\_}| \leq B$.*

Definition 6 (Existentially B-bounded [24]) *Let $B \in \mathbb{N}$. A prefix MSC M is* existentially B-bounded *if there is a B-bounded linearisation for M. A sequence of events w is* existentially B-bounded *if* $\mathrm{msc}(w)$ *is defined and* existentially B-bounded*. A language L is* existentially B-bounded *if every word $w \in L$ is. We may use* not existentially bounded *as abbreviation for not existentially B-bounded for any B.*

3.1.3 k-synchronisability

The restriction of k-synchronisability was introduced for mailbox communication [13] and later refined and adapted to the point-to-point setting [28]. We define k-synchronisability following definitions by Giusto et al. [28, Defs. 6 and 7]. The definition of k-synchronisability builds upon the notion when a prefix MSC is k-synchronous. Its first condition requires that there is some linearisation of the prefix MSC while its second condition requires causal delivery to hold. In contrast to the mailbox setting, the first condition always entails the second condition for the point-to-point setting.

Point-to-point Communication implies Causal Delivery. We first adapt the definition of causal delivery [28, Def. 4] for point-to-point FIFO channels [28, Section 6]. Unfortunately, this discussion leaves room for interpreting what causal delivery exactly is for point-to-point systems. Based on the description that a process P can receive messages from two distinct processes Q and R in any order, regardless of the dependency between the corresponding send events, we decided to literally adapt the definition of causal delivery as follows.

Definition 7 (Causal delivery) *Let $M = (N, p, f, l, (\leq_P)_{P \in \mathscr{P}})$ be an MSC. We say that M satisfies* causal delivery *if there is a linearisation $w = e_1 \dots$ of N such that for any two events $e_i \leq_M e_j$ with $e_i = P \triangleright Q!\_$ and $e_j = P \triangleright Q!\_$, either e_j is unmatched in w or there are $e_{i'} \leq_M e_{j'}$ such that $e_i \vdash e_{i'}$ and $e_j \vdash e_{j'}$ in w.*

We show that $\mathrm{msc}(w)$ for every w (if defined) satisfies causal delivery (as proven in the technical report [45]).

Lemma 2 *Let $w \in \Sigma^\infty$ such that $\mathrm{msc}(w)$ is defined. Then, $\mathrm{msc}(w)$ satisfies causal delivery.*

In combination with the fact that, given a linearisation w of a prefix MSC M, $\mathrm{msc}(w)$ is isomorphic to M, this yields that causal delivery is satisfied if there is a linearisation.

Corollary 1 *Every prefix MSC with a linearisation satisfies causal delivery.*

With this, we can simplify the definition by omitting the second condition without changing its meaning. In addition, we extend it to apply for MSCs with infinite sets of event nodes.

Definition 8 (k-synchronous and k-synchronisable) *Let $k \in \mathbb{N}$ be a positive natural number. We say that a prefix MSC $M = (N, p, f, l, (\leq_P)_{P \in \mathscr{P}})$ is* k-synchronous *if*

1. *there is a linearisation of event nodes w compliant with $\leq_M$ which can be split into a sequence of* k-exchanges *(also called* message exchange *if k not given or clear from context), i.e., $w = w_1 \dots$ such that for all i, it holds that $l(w_i) \in S^{\leq k} \cdot R^{\leq k}$; and*
2. *for all e, e' in w such that $e \vdash e'$, there is some i with e, e' in w_i.*[3]

A linearisation w is k-synchronisable[4] *if $\mathrm{msc}(w)$ is k-synchronous. A language L is k-synchronisable if every word $w \in L$ is. We may use* not synchronisable *as abbreviation for not k-synchronisable for any k.*

[3] This is equivalent to the following: for all e and $f(e)$ in w, there is some i with e, $f(e)$ in w_i.

[4] One could distinguish between universal and existential k-synchronisability, i.e., to distinguish the existence of a k-synchronisable linearisation rather than all linearisations being k-synchronisable. However, the universal version does not make much sense in practice. Thus, we omit the term existential.

3.2 Algorithmic Verification and Channel Restrictions

It is important to note that we use the term *restriction* as a property of a system which occurs naturally and not something that is imposed on its semantics. However, both have a tight connection: a system *naturally* satisfies a restriction if *imposing* the restriction does not change its possible behaviours. If this is the case, one can exploit this for algorithmic verification and only check behaviours that satisfy the restriction without harming correctness.

For each channel restriction, we recall known results about checking membership and which verification problems become decidable.

Half-duplex Communication. For CSMs with two processes, membership is decidable [17, Thm. 31]. The set of reachable configurations is computable in polynomial time which renders many verification questions like the *unspecified reception problem* decidable (see [17, Thm. 16] for a detailed list of verification problems) while model checking PLTL or CTL is still undecidable. Half-duplex CSMs with more than two processes are Turing-powerful [17, Thm. 38] so verification becomes undecidable and checking membership is of little interest.

Existential B-boundedness. For CSMs, membership is undecidable, unless CSMs are known to be deadlock free and B is given [24, Fig. 3]. For protocols, we will see that they are always existentially B-bounded for some B and thus a correct implementation of a protocol also is. It is quite straightforward that control-state reachability is decidable but not typically studied for these systems [12]. Intuitively, it can be solved by exhaustively enumerating the reachability graph of the CSM while pruning configurations exceeding the bound B. For HMSCs, model checking is undecidable for LTL [6, Thm. 3] and decidable for MSO [37, Thm. 1].

k-synchronisability. For CSMs, membership for a given k is decidable in EXPTIME [12, Rem. 30], originally shown decidable by Di Giusto et al. [28], while it is undecidable if k is not given [12, Thm. 22]. For HMSCs, both questions are decidable in polynomial time, while we show that MSTs are always 1-synchronisable. Model checking for k-synchronisable systems is decidable and in EXPTIME when formulas are represented in LCPDL. This follows from combining that such systems have bounded (special) tree-width [12, Prop. 28] and results by Bollig and Finkel [11]. Control-state reachability was shown to be decidable for k-synchronisable systems [28, Thm. 6].

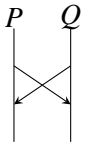
4 High-level Message Sequence Charts

Message sequence charts have been used as compact representation for executions of CSMs. The (prefix) message sequence charts obtained from different executions of CSMs can be analysed to determine which channel restriction is satisfied [24, 28]. In addition, we do also use message sequence charts as building blocks for high-level message sequence charts [39, 48] which specify protocols.

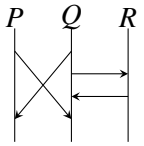
We define these following the presentation by Alur et al. [4, 5]. A BMSC corresponds to "straight line code" in which each process follows a single sequence of event nodes. A *high-level message sequence chart* (HMSC) adds a regular control structure (branching and loops).

Definition 9 (High-Level Message Sequence Charts) *A* high-level message sequence chart *(HMSC) is a structure (V, E, v^I, V^T, μ) where V is a finite set of vertices, $E \subseteq V \times V$ is a set of directed edges, $v^I \in V$ is an initial vertex, $V^T \subseteq V$ is a set of terminal vertices, and $\mu : V \to \mathscr{M}$ is a function mapping every vertex to a BMSC.*

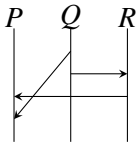
To obtain the language of an HMSC, we start with initial paths through the HMSC. As usual, we are interested only in maximal paths, i.e., either infinite or ending in a terminal vertex. We can expand



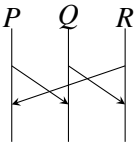
(a) ∃1-bounded, 2-synchronisable, and not half-duplex



(b) ∃1-bounded, not half-duplex, and not synchronisable



(c) half-duplex, ∃1-bounded, and not synchronisable



(d) half-duplex, ∃1-bounded, not 1- or 2-synchronisable but 3-synchronisable

Figure 4: BMSCs which satisfy different channels restrictions

each such path into a sequence of BMSCs, concatenate this sequence of BMSCs, and take the language of the resulting MSC. The language of the HMSC is the union of the languages of the MSCs generated by all its initial paths – see the technical report [45] for the formal definition. For simplicity, we assume every vertex in an HMSC is reachable from the initial vertex and every initial non-maximal path can be completed to a maximal one.

Example 1 *Figure 1b shows an HMSC composed of two BMSCs. MSCs of the HMSC are obtained by following the control structure and concatenating the corresponding BMSCs. The language contains all linearisations of these MSCs.*

4.1 Channel Restrictions of HMSCs

We say that an HMSC is half-duplex, existentially B-bounded or k-synchronisable respectively if its language is. It is straightforward that checking an HMSC for k-synchronisability amounts to checking its BMSCs.

Proposition 1 *An HMSC H is k-synchronisable iff all BMSCs of H are k-synchronous.*

For the presentation of our results, we follow the numbering laid out in Fig. 2b. Note that any BMSC can always be turned into a HMSC with a single initial and terminal vertex. Therefore, it is trivial that all BMSC examples also apply to HMSCs.

Lemma 3 ([24], Prop. 3.1) (H1): *Any HMSC H is existentially B-bounded for some B.*

We prove this result in a slightly different way in the technical report [45]. Basically, one computes the bound for the BMSC of every vertex in H and takes the maximum. This works since every MSC of H is a concatenation of individual BMSCs which can be scheduled in a way that the channels are empty after each BMSC.

Example 2 [H2]*: ∃B-bounded, k-synchronisable, and not half-duplex. Consider the BMSC in Fig. 4a. It is existentially* 1*-bounded as there is one message per channel,* 2*-synchronisable since the message exchange can be split into one phase of two sends and two subsequent receives and not half-duplex because both messages can traverse their channel at the same time.*

Example 3 [H3]*: ∃B-bounded, not half-duplex, and not synchronisable. It is obvious that the BMSC M in Fig. 4b is not half-duplex. We show that M is not k-synchronous for any k. Let us denote the event nodes for each process P with $p_1, \ldots$ as ordered by the total process order. It is straightforward that one of p_1 and q_1 has to be part of the first k-exchange. However, since the respective corresponding reception*

happens after the other's event node, both have to be a part of the first k-exchange. Since these receive event nodes (transitively) depend on all other event nodes, all event nodes have to be part of a single k-exchange for M. However, R first has to receive from Q in order to send back to it and therefore, there is no single k-exchange for M and M is not k-synchronous for any k.

Example 4 H4*: half-duplex, ∃B-bounded, and not synchronisable. Let us consider the BMSC in Fig. 4c. It is straightforward that it is half-duplex and existentially* 1*-bounded. However, it is not k-synchronisable for any k. In particular, the first and last event node (of any total order induced by the BMSC) must belong to the same message exchange but two more linearly dependent message exchanges need to happen in between.*

Example 5 H5*: half-duplex, ∃B-bounded, k-synchronisable but not* 1*-synchronisable. Consider the BMSC in Fig. 4d. It is easy to see that it is not* 1*- or* 2*-synchronisable but* 3*-synchronisable, half-duplex and existentially* 1*-bounded. Note that it is straightforward to amend the example such that it is still half-duplex but the parameters B and k need to be increased.*

Lemma 4 (H6): Every 1-synchronisable HMSC is half-duplex.

Intuitively, in any BMSC of an HMSC, every send event node has a corresponding receive event node. Therefore, a message that has been sent needs to have been received directly afterwards and the per-process order is total so any process has to receive a message before it sends a message back. The full proof can be found in the technical report [45].

5 Multiparty Session Types

In this section, we recall global types from Multiparty Session Types (MSTs) as a way to specify protocols. We present an embedding for MSTs into HMSCs, prove it correct, and use it to show that MSTs are half-duplex, existentially 1-bounded, and 1-synchronisable.

5.1 Specifying Protocols with Global Types

We now define global types in the framework of MSTs as a syntax for protocol specifications. The syntax of global types is defined following classical MST frameworks [44, Def. 3.2]. The calculus focuses on the core message-passing primitives of asynchronous MSTs and does not incorporate features like subsessions or delegation. However, it does incorporate a recent generalisation that allow a sender to send to different receivers upon branching [38].

Definition 10 (Syntax of Global Types [38]) Global types for MSTs *are defined by the grammar:*

$$G ::= 0 \mid \sum_{i \in I} P \to Q_i : m_i.G_i \mid \mu t.G \mid t$$

An expression $P \to Q : m$ stands for a send and receive event: $P \triangleright Q!m$ and $Q \triangleleft P?m$. Since global types always specify send and the receive events together, they specify complete channel-compliant sequences of events. For a *choice*, the sender process decides which branch to take and each branch of a choice needs to be uniquely distinguishable ($\forall i, j \in I.\, i \neq j \Rightarrow Q_i \neq Q_j \vee m_i \neq m_j$). If there is a single alternative (and no actual choice), we omit writing the sum operator. Loops are encoded by the least fixed point operator and recursion must be guarded, i.e., there is at least one message between μt and t. We assume, without loss of generality, that all occurrences of recursion variables t are bound and every variable t

is distinct. Recursion only happens at the tail (and there is no additional parameter) and, therefore, the language of a global type can be defined with an automaton – as expected by following the structure of a global type, splitting the message exchanges into send and receive events while not only accounting for finite but also infinite executions. We give one language as example and refer to the technical report [45] for a formal definition.

Example 6 *The type language for the global type in Fig. 1c,* $\mu t.\ (P \to Q : cons.t + P \to Q : nil.Q \to P : ack.0)$*, is the union of a set of finite executions and infinite executions:*
$\big(P \triangleright Q!cons.Q \triangleleft P?cons\big)^*.\ P \triangleright Q!nil.Q \triangleleft P?nil.Q \triangleright P!ack.P \triangleleft Q?ack \quad$ *and* $\quad \big(P \triangleright Q!cons.Q \triangleleft P?cons\big)^\omega$

Remark 1 *For readers familiar with MSTs, it may be strange that we do not define local types. In fact, one correctness criterion for local types requires that their composition generates the same language as the original global type. All channel restrictions are defined using languages, so it suffices to consider global types for our purposes.*

Example 7 *Consider the global type:* $P \to Q : m_1.R \to S : m_2$*. The type language for this type contains only the word* $P \triangleright Q!m_1.Q \triangleleft P?m_1.R \triangleright S!m_2.S \triangleleft R?m_2$*. On the other hand, if we want to describe the same protocol with a HMSC, it always allows any permutation of the events where* $P \triangleright Q!m_1$ *occurs before* $Q \triangleleft P?m_1$ *and* $R \triangleright S!m_2$ *before* $S \triangleleft R?m_2$*.*

Intuitively, some events in a distributed setting shall not be ordered since they are independent, e.g., happen on different processes as in the previous example. To this end, we recall an indistinguishability relation $\sim$ that captures the reordering allowed by CSMs with FIFO channels (which will be defined in Definition 12). In MSTs, similar reordering rules are applied (e.g., [33, Def. 3.2 and 5.3]).

Definition 11 (Indistinguishability relation $\sim$ [38]) *Let* $\sim_i \subseteq \Sigma^* \times \Sigma^*$*, for* $i \geq 0$*, be a family of indistinguishability relations. For all* $w \in \Sigma^*$*, we have* $w \sim_0 w$*. For* $i = 1$*, we define:*

(1) If $P \neq R$*, then* $w.P \triangleright Q!m.R \triangleright S!m'.u \ \sim_1 \ w.R \triangleright S!m'.P \triangleright Q!m.u$*.*

(2) If $Q \neq S$*, then* $w.Q \triangleleft P?m.S \triangleleft R?m'.u \ \sim_1 \ w.S \triangleleft R?m'.Q \triangleleft P?m.u$*.*

(3) If $P \neq S \wedge (P \neq R \vee Q \neq S)$*, then* $w.P \triangleright Q!m.S \triangleleft R?m'.u \ \sim_1 \ w.S \triangleleft R?m'.P \triangleright Q!m.u$*.*

(4) If $|w{\Downarrow_{P \triangleright Q!\_}}| > |w{\Downarrow_{Q \triangleleft P?\_}}|$*, then* $w.P \triangleright Q!m.Q \triangleleft P?m'.u \ \sim_1 \ w.Q \triangleleft P?m'.P \triangleright Q!m.u$*.*

Let w, w', w'' *be sequences of events such that* $w \sim_1 w'$ *and* $w' \sim_i w''$ *for some i. Then,* $w \sim_{i+1} w''$*. We define* $w \sim u$ *if there is n such that* $w \sim_n u$*.*

This formalises how messages can be swapped for finite executions of protocols. The infinite case requires special technical treatment for which we refer to the work by Majumdar et al. [38].

The relation is lifted to languages as expected. For a language L, we have:

$$\mathscr{C}^{\sim}(L) = \left\{ w' \mid \bigvee \begin{array}{l} w' \in \Sigma^* \wedge \exists w \in \Sigma^*.\ w \in L \text{ and } w' \sim w \\ w' \in \Sigma^\omega \wedge \exists w \in \Sigma^\omega.\ w \in L \text{ and } w' \preceq^\omega_\sim w \end{array} \right\}.$$

The indistinguishability relation $\sim$ does not change the order of send and receive events of a single process. The relation $\sim$ captures all reorderings which naturally appear when global types from MSTs are implemented with CSMs. For a global type G, its semantics is given by its *execution language* $\mathscr{C}^{\sim}(\mathscr{L}(G))$. Furthermore, the indistinguishability relation captures exactly the events that are independent in any HMSC. Phrased differently, HMSC include these reorderings by design.

Lemma 5 *Let H be any HMSC. Then,* $\mathscr{L}(H) = \mathscr{C}^{\sim}(\mathscr{L}(H))$*.*

We prove this in the technical report [45]. A similar result for CSMs has been proven [38, Lemma 21].

Theorem 1 *For channel-compliant words, the indistinguishability relation* $\sim$ *preserves satisfaction of half-duplex communication, existential B-boundedness, and k-synchronisability.*

The proof can be found in the technical report [45].

5.2 Encoding Global Types from MSTs into HMSCs

Global types from MSTs can be turned into HMSCs while preserving the protocol they specify. In this step, we account for the orders than can and cannot be enforced in an asynchronous point-to-point setting with the indistinguishability relation $\sim$. The main difference between the automata-based semantics of global types from MSTs and the semantics of HMSCs is that an automaton carries the events on the edges and an HMSC carries events as labels of the event nodes in the BMSCs associated with the vertices.

In the translation, we use the following notation. $M_\emptyset$ is the empty BMSC ($N = \emptyset$) and $M(P \to Q : m)$ is the BMSC with two event nodes: e_1, e_2 such that $f(e_1) = e_2$, $l(e_1) = P \triangleright Q!m$, and $l(e_2) = Q \triangleleft P?m$.

From a global type G, we construct an HMSC $H(G) = (V, E, v^I, V^T, \mu, \lambda)$ with

$$V = \{G' \mid G' \text{ is a subterm of } G\} \cup \{(\textstyle\sum_{i\in I} P \to Q_i : m_i.G_i, j) \mid \sum_{i\in I} P \to Q_i : m_i.G_i \text{ occurs in } G \wedge j \in I\}$$

$$\begin{aligned} E = {} & \{(\mu t.G', G') \mid \mu t.G' \text{ occurs in } G\} \cup \{(t, \mu t.G') \mid t, \mu t.G' \text{ occurs in } G\} \\ & \cup \{(\textstyle\sum_{i\in I} P \to Q_i : m_i.G_i, (\sum_{i\in I} P \to Q_i : m_i.G_i, j)) \mid (\sum_{i\in I} P \to Q_i : m_i.G_i, j) \in V\} \\ & \cup \{((\textstyle\sum_{i\in I} P \to Q_i : m_i.G_i, j), G_j) \mid (\sum_{i\in I} P \to Q_i : m_i.G_i, j) \in V\} \end{aligned}$$

$$v^I = G \qquad V^T = \{0\} \qquad \mu(v) = \begin{cases} M(P \to Q_i : m_j) & \text{if } v = (\sum_{i\in I} P \to Q_i : m_i.G_i\}, j) \\ M_\emptyset & \text{otherwise} \end{cases}$$

This translation does not yield the HMSC with the least number of vertices since vertices with a single successor could be merged to form larger BMSCs. Here, every BMSC contains at most one message exchange. We obtain the following correctness statement for the embedding:

Theorem 2 *For any global type G, it holds that $\mathscr{L}(G) \subseteq \mathscr{L}(H(G))$ and $\mathscr{C}^\sim(\mathscr{L}(G)) = \mathscr{C}^\sim(\mathscr{L}(H(G)))$.*

We provide the technical developments to show Theorem 2 the technical report [45].

Remark 2 *The first part of Theorem 2 uses $\subseteq$ instead of $=$ as HMSCs do not order indistinguishable events and we consider the type language of G. Example 7 shows that using the execution language rather than the type language in the second part is inevitable for equality and does not weaken the claim.*

5.3 Channel Restrictions of Global Types

Example 8 H7*: half-duplex, $\exists$1-bounded, 1-synchronisable but not in MSTs. Consider the HMSC in Fig. 5. It is straightforward that it is half-duplex, existentially 1-bounded, and 1-synchronisable. Both P and Q send the same message to R independently in each branch. Intuitively, R chooses which branch to take by the order it decides to receive both messages. Subsequently, it notifies Q about this choice. Such a communication pattern cannot be expressed in the MST framework. If one tried to model it with R actually choosing the branch, l and r would always occur before the receptions so the languages are different.*

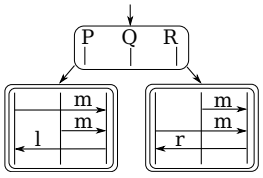


Figure 5: half-duplex, $\exists$1-bounded, and 1-synchronisable but not expressible in MSTs

We show that protocols specified as global types satisfy all discussed channel restrictions (with the minimal reasonable parameter).

Theorem 3 H8: *The execution language $\mathscr{C}^\sim(\mathscr{L}(G))$ is half-duplex, existentially 1-bounded, and 1-synchronisable for any global type G.*

The proof uses the embedding to obtain an HMSC built of BMSCs with at most one message exchange and exploits previously shown properties about HMSCs. Details can be found in the technical report [45].

Remark 3 (Choreography automata are half-duplex, $\exists 1$-bounded, and 1-synchronisable) *In this section, we looked at MSTs which are rooted in process algebra. With* choreography automata *[7], a similar concept has been studied from automata theory perspective. Basically, a protocol specification is an automaton whose transitions are labelled by $P \to Q : m$. In contrast to global types from MSTs, they do not impose constraints on choice, i.e., there does not need to be a unique process chooses which branch to take next and do not employ an indistinguishability relation but require to explicitly spell out all possible reorderings. This feature can lead to complications w.r.t. implementing such protocols but does not change the satisfaction of channel restrictions. In fact, protocols specified by choreography automata are also half-duplex, existentially 1-bounded, and 1-synchronisable.*

6 Communicating State Machines

In this section, we first present communicating state machines (CSMs) as formal model for distributed processes which communicate messages asynchronously via reliable point-to-point FIFO channels. If a CSM implements a protocol specification, both languages are the same – modulo $\sim$ which does not alter satisfaction of channel restrictions. This entails that CSMs implementing protocol specifications satisfy the same channel restrictions as presented in previous sections. Here, we investigate the channel restrictions of general CSMs which might not implement a protocol specified as global type or HMSC.

Definition 12 (Communicating state machines) *A* state machine $A = (Q, \Delta, \delta, q_0, F)$ *is a 5-tuple where Q is a finite set of states, Δ is an alphabet, $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ is a transition relation, $q_0 \in Q$ is an initial state, and $F \subseteq Q$ is a set of final states. We write $q \xrightarrow{a} q'$ for $(q, a, q') \in \delta$. A* run *of A is a sequence $\rho = q_0 \xrightarrow{w_0} q_1 \xrightarrow{w_1} \ldots$, with $q_i \in Q$ and $w_i \in \Delta \cup \{\varepsilon\}$ for $i \geq 0$, such that q_0 is the initial state, and for each $i \geq 0$, it holds that $(q_i, w_i, q_{i+1}) \in \delta$. The* trace *of the run is the finite or infinite word $w_0 w_1 \ldots \in \Sigma^\infty$. The* path *of the run is the finite or infinite sequence $q_0 q_1 \ldots \in Q^\infty$. A run is called maximal if it is infinite or ends at a final state. Accordingly, the corresponding trace and path are called maximal. The* language *$\mathscr{L}(A)$ of A is the set of its maximal traces.*

We call $\mathscr{A} = \{\{A_P\}\}_{P \in \mathscr{P}}$ a communicating state machine *(CSM) over $\mathscr{P}$ and $\mathscr{V}$ if A_P is a finite state machine with alphabet Σ_P for every $P \in \mathscr{P}$. The state machine for P is denoted by $(Q_P, \Sigma_P, \delta_P, q_{0,P}, F_P)$. Intuitively, a CSM allows a set of state machines, one for each process in $\mathscr{P}$, to communicate by sending and receiving messages. For this, each pair of processes $P, Q \in \mathscr{P}$, $P \neq Q$, is connected by two directed* message channels*. A transition $q_P \xrightarrow{P \triangleright Q!m} q'_P$ in the state machine of P denotes that P sends message m to Q if P is in the state q and changes its local state to q'. The channel $\langle P, Q \rangle$ is appended by message m. For receptions, a transition $q_Q \xrightarrow{Q \triangleleft P?m} q'_Q$ in the state machine of Q corresponds to Q retrieving the message m from the head of the channel when its local state is $\hat{q}$ which is updated to $\hat{q}'$. The run of a CSM always starts with empty channels and each finite state machine is its respective initial state. The formalisation of this intuition is standard and can be found in the technical report [45].*

As for HMSCs, the language of a CSM is closed under $\sim$.

Lemma 6 ([38], Lemma 21) *Let $\mathscr{A}$ be a CSM. Then $\mathscr{L}(\mathscr{A}) = \mathscr{C}^{\sim}(\mathscr{L}(\mathscr{A}))$.*

Implementing Protocol Specifications. Given a protocol specification, one can try to generate an implementation which admits the same language. This problem is known as implementability or realisability in the HMSC setting [27, 5, 36]. In the MST setting [32], this is done in two steps. First, the global type is projected on to local types. Second, a type system ensures that the implementations follow the local types, i.e., a refinement check. However, one can design CSMs from scratch that yield systems which cannot be captured by protocol specifications like HMSCs or global types from MSTs.

6.1 Channel Restrictions of CSMs

We say that an CSM is half-duplex, existentially B-bounded, or k-synchronisable respectively if its language is. Again, we follow the outline presented in Fig. 2a.

Example 9 C1: *half-duplex, $\exists B$-bounded, and k-synchronisable. The CSM in Fig. 1a is $\exists 1$-bounded, 1-synchronisable, and half-duplex.*

Any BMSC can easily be implemented with an CSM by simple letting each process follow its linear trajectory of eventnodes. We call this projection. Therefore, we can use three of the BMSCs presented in Fig. 4 to show the hypotheses for CSMs:

Example 10 *For, C2, the projection of Fig. 4c (used to show H4) is half-duplex, $\exists B$-bounded, and not synchronisable. For C3, the projection of Fig. 4b (used to show H3) is $\exists B$-bounded, not half-duplex, and not synchronisable. For C4, the projection of Fig. 4a (used to show H2) is $\exists B$-bounded, k-synchronisable, and not half-duplex.*

Example 11 C5: *k-synchronisable, not half-duplex and not $\exists$-bounded;* C6: *k-synchronisable, half-duplex and not $\exists$-bounded.*
We consider two CSMs constructed from the state machines in Fig. 6. For C5, we consider the CSM consisting of both state machines. It is 1-synchronisable but not existentially bounded and not half-duplex. It is 1-synchronisable because every linearisation can be split into single send events that constitute 1-exchanges. It is neither existentially B-bounded for any B nor half-duplex since none of the messages will be received so both channels can grow arbitrarily. For C6, it can easily be turned into a half-duplex CSM by removing one of the send events. Then, the CSM is 1-synchronisable and half-duplex but not existentially bounded.

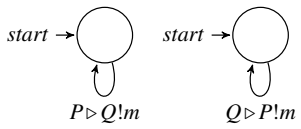


Figure 6: CSM with FSMs for P (left) and for Q (right)

This example disproves a result from the literature [35, Thm. 7.1], which states that every k-synchronisable system is existentially B-bounded for some B and has been cited recently as part of a summary [12, Prop. 41]. In the proof, it is neglected that unreceived messages remain in the channels after a message exchange. Our example satisfies their assumption that CSMs do not have states with mixed choice, i.e., each state either is final, has send options to choose from, or receive options to choose from. We do not impose any assumptions on mixed choice in this work. Still, all the presented examples do not have states with mixed choice so the presented relationships also hold for this subset of CSMs.

Corollary 2 *Existential B-boundedness and k-synchronisability for CSMs are incomparable.*

The previous result follows immediately from the CSMs constructed in Example 11. Our result considers the point-to-point FIFO setting. For the mailbox setting, the analogous question is an open problem [13].

Turing-powerful Encodings. On the one hand, it is well-known that CSMs are Turing-complete [14] and Cécé and Finkel [17, Thm. 36] showed that half-duplex communication does not impair expressiveness of CSMs with more than two processes. On the other hand, each of existential B-boundedness and k-synchronisability render some verification questions decidable. Therefore, the encodings of Turing-completeness [14, 17] are examples for CSMs which are not existentially B-bounded for any B nor k-synchronisable for any k and either half-duplex (C7) or not half-duplex (C8).

7 Related Work

We now cover related work which is not already cited in the earlier sections.

The origins of MSTs date back to 1993 when Honda et al. [31] proposed a binary version for typing communication in the domain of process algebra. In 2008, Honda et al. [32] generalised the idea to multiparty systems. While the connection of MSTs and CSMs has been studied soon after MSTs had been proposed [16, 19], we provide, to the best of our knowledge, the first formal connection of MSTs to HMSCs, even though HMSC-like visualisations have been used in the community of session types, e.g. [15, Fig. 1], [33, Figs. 1 and 2]. For binary session types, it is known how to compute the bound B of universally B-bounded types [20, 22]. Lange et al. [35] proposed k-multiparty consistency (k-MC) for CSMs as extension of multiparty consistency for MSTs. We did not consider k-MC in this work for two reasons. First, they assume an existential bound (of k) on channels. Second, as an extension of multiparty consistency, k-MC focuses on implementability rather than channel restrictions.

HMSCs and variants thereof have been extensively studied [26, 25, 23, 43].The connection to CSMs has been investigated in particular for different forms of implementability [27, 5, 36], also called realisability [5], which is undecidable in general [26, 5], and implied scenarios [41, 40] which arise when implementing HMSCs with CSMs. Several restrictions to check implementability adopted a limited form of choice [8, 29, 41, 40, 18] which is similar to the one in global types from MSTs. For more details, we refer to work by Majumdar et al. [38].

While we consider finite state machines as model for processes, research has also been conducted on communicating systems where processes are given more computational power, e.g., pushdown automata [30, 47, 3]. However, as noted before, our setting is already Turing-powerful. In Section 3.2, we surveyed how channel restrictions can yield decidability. Incomplete approaches consider subclasses which enable the effective computation of symbolic representations (of channel contents) for reachable states [9, 34]. Other approaches change the semantics of channels, e.g., by making them lossy [2, 1, 34], input-bounded [10], or by restricting the communication topology [42, 46].

While we build on the most recent definitions of synchronisability [28], we refer to the work by Finkel and Lozes [21] and Bouajjani et al. [13] for earlier work on synchronisability. Bollig et. al [12] studied the connection of different notions of synchronisability for MSCs and MSO logic which yields interesting decidability results. We refer to their work for more details but briefly point to the slightly different use of terminology: k-synchronisability is called weak (k-)synchronisability by Bollig where the omission of k indicates a system is synchronisable for some k; while strong (k-)synchronisability does solely apply to the mailbox setting.

8 Conclusion

We presented a comprehensive comparison of half-duplex, existential B-bounded, and k-synchronisable communication. We showed that the three restrictions are different for CSMs. For HMSCs, the half-duplex restriction and k-synchronisability are different and included in existential B-boundedness. Furthermore, all 1-synchronisable HMSC-definable languages are half-duplex. This subclass contains global types from MSTs which are also existentially 1-bounded. We established the first formal embedding of global types from MSTs into HMSCs which can be used to combine insights from both domains for further advances on implementing protocol specifications.

References

[1] Parosh Aziz Abdulla, C. Aiswarya & Mohamed Faouzi Atig (2016): *Data Communicating Processes with Unreliable Channels*. In Martin Grohe, Eric Koskinen & Natarajan Shankar, editors: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, ACM, pp. 166–175, doi:10.1145/2933575.2934535.

[2] Parosh Aziz Abdulla, Ahmed Bouajjani & Bengt Jonsson (1998): *On-the-Fly Analysis of Systems with Unbounded, Lossy FIFO Channels*. In Alan J. Hu & Moshe Y. Vardi, editors: *Computer Aided Verification, 10th International Conference, CAV'98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, *Lecture Notes in Computer Science* 1427, Springer, pp. 305–318, doi:10.1007/BFb0028754.

[3] C. Aiswarya, Paul Gastin & K. Narayan Kumar (2014): *Verifying Communicating Multi-pushdown Systems via Split-Width*. In Franck Cassez & Jean-François Raskin, editors: *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, *Lecture Notes in Computer Science* 8837, Springer, pp. 1–17, doi:10.1007/978-3-319-11936-6_1.

[4] Rajeev Alur, Kousha Etessami & Mihalis Yannakakis (2003): *Inference of Message Sequence Charts*. *IEEE Trans. Software Eng.* 29(7), pp. 623–633, doi:10.1109/TSE.2003.1214326.

[5] Rajeev Alur, Kousha Etessami & Mihalis Yannakakis (2005): *Realizability and verification of MSC graphs*. *Theor. Comput. Sci.* 331(1), pp. 97–114, doi:10.1016/j.tcs.2004.09.034.

[6] Rajeev Alur & Mihalis Yannakakis (1999): *Model Checking of Message Sequence Charts*. In Jos C. M. Baeten & Sjouke Mauw, editors: *CONCUR '99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands, August 24-27, 1999, Proceedings*, *Lecture Notes in Computer Science* 1664, Springer, pp. 114–129, doi:10.1007/3-540-48320-9_10.

[7] Franco Barbanera, Ivan Lanese & Emilio Tuosto (2020): *Choreography Automata*. In Simon Bliudze & Laura Bocchi, editors: *Coordination Models and Languages - 22nd IFIP WG 6.1 International Conference, COORDINATION 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020, Valletta, Malta, June 15-19, 2020, Proceedings*, *Lecture Notes in Computer Science* 12134, Springer, pp. 86–106, doi:10.1007/978-3-030-50029-0_6.

[8] Hanêne Ben-Abdallah & Stefan Leue (1997): *Syntactic Detection of Process Divergence and Non-local Choice inMessage Sequence Charts*. In Ed Brinksma, editor: *Tools and Algorithms for Construction and Analysis of Systems, Third International Workshop, TACAS '97, Enschede, The Netherlands, April 2-4, 1997, Proceedings*, *Lecture Notes in Computer Science* 1217, Springer, pp. 259–274, doi:10.1007/BFb0035393.

[9] Bernard Boigelot, Patrice Godefroid, Bernard Willems & Pierre Wolper (1997): *The Power of QDDs (Extended Abstract)*. In Pascal Van Hentenryck, editor: *Static Analysis, 4th International Symposium, SAS '97, Paris, France, September 8-10, 1997, Proceedings*, *Lecture Notes in Computer Science* 1302, Springer, pp. 172–186, doi:10.1007/BFb0032741.

[10] Benedikt Bollig, Alain Finkel & Amrita Suresh (2020): *Bounded Reachability Problems Are Decidable in FIFO Machines*. In Igor Konnov & Laura Kovács, editors: *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, *LIPIcs* 171, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 49:1–49:17, doi:10.4230/LIPIcs.CONCUR.2020.49.

[11] Benedikt Bollig & Paul Gastin (2019): *Non-Sequential Theory of Distributed Systems*. *CoRR* abs/1904.06942, doi:10.48550/arXiv.1904.06942. arXiv:1904.06942.

[12] Benedikt Bollig, Cinzia Di Giusto, Alain Finkel, Laetitia Laversa, Étienne Lozes & Amrita Suresh (2021): *A Unifying Framework for Deciding Synchronizability*. In Serge Haddad & Daniele Varacca, editors: *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, *LIPIcs* 203, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 14:1–14:18, doi:10.4230/LIPIcs.CONCUR.2021.14.

[13] Ahmed Bouajjani, Constantin Enea, Kailiang Ji & Shaz Qadeer (2018): *On the Completeness of Verifying Message Passing Programs Under Bounded Asynchrony*. In Hana Chockler & Georg Weissenbacher, editors: *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, *Lecture Notes in Computer Science* 10982, Springer, pp. 372–391, doi:10.1007/978-3-319-96142-2_23.

[14] Daniel Brand & Pitro Zafiropulo (1983): *On Communicating Finite-State Machines*. *J. ACM* 30(2), pp. 323–342, doi:10.1145/322374.322380.

[15] Marco Carbone, Kohei Honda, N. Yoshida, R. Milner, G. Brown & Steve Ross-Talbot (2005): *A Theoretical Basis of Communication-Centred Concurrent Programming*.

[16] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini & Luca Padovani (2012): *On Global Types and Multi-Party Session*. *Log. Methods Comput. Sci.* 8(1), doi:10.2168/LMCS-8(1:24)2012.

[17] Gérard Cécé & Alain Finkel (2005): *Verification of programs with half-duplex communication*. *Inf. Comput.* 202(2), pp. 166–190, doi:10.1016/j.ic.2005.05.006.

[18] Haitao Dan, Robert M. Hierons & Steve Counsell (2010): *Non-local Choice and Implied Scenarios*. In José Luiz Fiadeiro, Stefania Gnesi & Andrea Maggiolo-Schettini, editors: *8th IEEE International Conference on Software Engineering and Formal Methods, SEFM 2010, Pisa, Italy, 13-18 September 2010*, IEEE Computer Society, pp. 53–62, doi:10.1109/SEFM.2010.14.

[19] Pierre-Malo Deniélou & Nobuko Yoshida (2012): *Multiparty Session Types Meet Communicating Automata*. In Helmut Seidl, editor: *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, *Lecture Notes in Computer Science* 7211, Springer, pp. 194–213, doi:10.1007/978-3-642-28869-2_10.

[20] Manuel Fähndrich, Mark Aiken, Chris Hawblitzel, Orion Hodson, Galen C. Hunt, James R. Larus & Steven Levi (2006): *Language support for fast and reliable message-based communication in singularity OS*. In Yolande Berbers & Willy Zwaenepoel, editors: *Proceedings of the 2006 EuroSys Conference, Leuven, Belgium, April 18-21, 2006*, ACM, pp. 177–190, doi:10.1145/1217935.1217953.

[21] Alain Finkel & Étienne Lozes (2017): *Synchronizability of Communicating Finite State Machines is not Decidable*. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn & Anca Muscholl, editors: *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, *LIPIcs* 80, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 122:1–122:14, doi:10.4230/LIPIcs.ICALP.2017.122.

[22] Simon J. Gay & Vasco Thudichum Vasconcelos (2010): *Linear type theory for asynchronous session types*. *J. Funct. Program.* 20(1), pp. 19–50, doi:10.1017/S0956796809990268.

[23] Thomas Gazagnaire, Blaise Genest, Loïc Hélouët, P. S. Thiagarajan & Shaofa Yang (2007): *Causal Message Sequence Charts*. In Luís Caires & Vasco Thudichum Vasconcelos, editors: *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*, *Lecture Notes in Computer Science* 4703, Springer, pp. 166–180, doi:10.1007/978-3-540-74407-8_12.

[24] Blaise Genest, Dietrich Kuske & Anca Muscholl (2007): *On Communicating Automata with Bounded Channels*. *Fundam. Inform.* 80(1-3), pp. 147–167. Available at `http://content.iospress.com/articles/fundamenta-informaticae/fi80-1-3-09`.

[25] Blaise Genest & Anca Muscholl (2005): *Message Sequence Charts: A Survey*. In: *Fifth International Conference on Application of Concurrency to System Design (ACSD 2005), 6-9 June 2005, St. Malo, France*, IEEE Computer Society, pp. 2–4, doi:10.1109/ACSD.2005.25.

[26] Blaise Genest, Anca Muscholl & Doron A. Peled (2003): *Message Sequence Charts*. In Jörg Desel, Wolfgang Reisig & Grzegorz Rozenberg, editors: *Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This tutorial volume originates from the 4th Advanced Course on Petri Nets, ACPN 2003, held in Eichstätt, Germany in September 2003. In addition to lectures given at ACPN 2003, additional chapters have been*

commissioned], Lecture Notes in Computer Science 3098, Springer, pp. 537–558, doi:10.1007/978-3-540-27755-2_15.

[27] Blaise Genest, Anca Muscholl, Helmut Seidl & Marc Zeitoun (2006): *Infinite-state high-level MSCs: Model-checking and realizability*. *J. Comput. Syst. Sci.* 72(4), pp. 617–647, doi:10.1016/j.jcss.2005.09.007.

[28] Cinzia Di Giusto, Laetitia Laversa & Étienne Lozes (2020): *On the k-synchronizability of Systems*. In Jean Goubault-Larrecq & Barbara König, editors: *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, Lecture Notes in Computer Science 12077, Springer, pp. 157–176, doi:10.1007/978-3-030-45231-5_9.

[29] Loïc Hélouët & Claude Jard (2000): *Conditions for synthesis of communicating automata from HMSCs*. In: *In 5th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*.

[30] Alexander Heußner, Jérôme Leroux, Anca Muscholl & Grégoire Sutre (2012): *Reachability Analysis of Communicating Pushdown Systems*. *Log. Methods Comput. Sci.* 8(3), doi:10.2168/LMCS-8(3:23)2012.

[31] Kohei Honda (1993): *Types for Dyadic Interaction*. In Eike Best, editor: *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, Lecture Notes in Computer Science 715, Springer, pp. 509–523, doi:10.1007/3-540-57208-2_35.

[32] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty asynchronous session types*. In George C. Necula & Philip Wadler, editors: *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, ACM, pp. 273–284, doi:10.1145/1328438.1328472.

[33] Kohei Honda, Nobuko Yoshida & Marco Carbone (2016): *Multiparty Asynchronous Session Types*. *J. ACM* 63(1), pp. 9:1–9:67, doi:10.1145/2827695.

[34] Chris Köcher (2021): *Reachability Problems on Reliable and Lossy Queue Automata*. *Theory Comput. Syst.* 65(8), pp. 1211–1242, doi:10.1007/s00224-021-10031-2.

[35] Julien Lange & Nobuko Yoshida (2019): *Verifying Asynchronous Interactions via Communicating Session Automata*. In Isil Dillig & Serdar Tasiran, editors: *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, Lecture Notes in Computer Science 11561, Springer, pp. 97–117, doi:10.1007/978-3-030-25540-4_6.

[36] Markus Lohrey (2003): *Realizability of high-level message sequence charts: closing the gaps*. *Theor. Comput. Sci.* 309(1-3), pp. 529–554, doi:10.1016/j.tcs.2003.08.002.

[37] P. Madhusudan (2001): *Reasoning about Sequential and Branching Behaviours of Message Sequence Graphs*. In Fernando Orejas, Paul G. Spirakis & Jan van Leeuwen, editors: *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, Lecture Notes in Computer Science 2076, Springer, pp. 809–820, doi:10.1007/3-540-48224-5_66.

[38] Rupak Majumdar, Madhavan Mukund, Felix Stutz & Damien Zufferey (2021): *Generalising Projection in Asynchronous Multiparty Session Types*. In Serge Haddad & Daniele Varacca, editors: *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, LIPIcs 203, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 35:1–35:24, doi:10.4230/LIPIcs.CONCUR.2021.35.

[39] Sjouke Mauw & Michel A. Reniers (1997): *High-level message sequence charts*. In Ana R. Cavalli & Amardeo Sarma, editors: *SDL '97 Time for Testing, SDL, MSC and Trends - 8th International SDL Forum, Evry, France, 23-29 September 1997, Proceedings*, Elsevier, pp. 291–306.

[40] Arjan J. Mooij, Nicolae Goga & Judi Romijn (2005): *Non-local Choice and Beyond: Intricacies of MSC Choice Nodes*. In Maura Cerioli, editor: *Fundamental Approaches to Software Engineering, 8th International Conference, FASE 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, Lecture Notes in Computer Science 3442, Springer, pp. 273–288, doi:10.1007/978-3-540-31984-9_21.

[41] Henry Muccini (2003): *Detecting Implied Scenarios Analyzing Non-local Branching Choices*. In Mauro Pezzè, editor: *Fundamental Approaches to Software Engineering, 6th International Conference, FASE 2003,*

Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings, *Lecture Notes in Computer Science* 2621, Springer, pp. 372–386, doi:10.1007/3-540-36578-8_26.

[42] Wuxu Peng & S. Purushothaman (1992): *Analysis of a Class of Communicating Finite State Machines*. *Acta Informatica* 29(6/7), pp. 499–522, doi:10.1007/BF01185558.

[43] Abhik Roychoudhury, Ankit Goel & Bikram Sengupta (2012): *Symbolic Message Sequence Charts*. *ACM Trans. Softw. Eng. Methodol.* 21(2), pp. 12:1–12:44, doi:10.1145/2089116.2089122.

[44] Alceste Scalas & Nobuko Yoshida (2019): *Less is more: multiparty session types revisited*. *Proc. ACM Program. Lang.* 3(POPL), pp. 30:1–30:29, doi:10.1145/3290343.

[45] Felix Stutz & Damien Zufferey (2022): *Comparing Channel Restrictions of Communicating State Machines, High-level Message Sequence Charts, and Multiparty Session Types*. *CoRR* abs/2208.05559, doi:10.48550/arXiv.2208.05559. arXiv:2208.05559.

[46] Salvatore La Torre, P. Madhusudan & Gennaro Parlato (2008): *Context-Bounded Analysis of Concurrent Queue Systems*. In C. R. Ramakrishnan & Jakob Rehof, editors: *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, *Lecture Notes in Computer Science* 4963, Springer, pp. 299–314, doi:10.1007/978-3-540-78800-3_21.

[47] Tayssir Touili & Mohamed Faouzi Atig (2010): *Verifying parallel programs with dynamic communication structures*. *Theor. Comput. Sci.* 411(38-39), pp. 3460–3468, doi:10.1016/j.tcs.2010.05.028.

[48] International Telecommunication Union (1996): *Z.120: Message Sequence Chart*. Technical Report, International Telecommunication Union. Available at `https://www.itu.int/rec/T-REC-Z.120`.