Verifying Asynchronous Interactions via Communicating Session Automata

Julien Lange[1] and Nobuko Yoshida[2]

[1] University of Kent, Canterbury, United Kingdom
[2] Imperial College London, London, United Kingdom

Abstract. This paper proposes a sound procedure to verify properties of communicating session automata (CSA), i.e., communicating automata that include multiparty session types. We introduce a new *asynchronous* compatibility property for CSA, called k-multiparty compatibility (k-MC), which is a strict superset of the synchronous multiparty compatibility used in theories and tools based on session types. It is decomposed into two bounded properties: (*i*) a condition called *k-safety* which guarantees that, within the bound, all sent messages can be received and each automaton can make a move; and (*ii*) a condition called *k-exhaustivity* which guarantees that all k-reachable send actions can be fired within the bound. We show that k-exhaustivity implies existential boundedness, and *soundly and completely* characterises systems where each automaton behaves equivalently under bounds greater than or equal to k. We show that checking k-MC is PSPACE-complete, and demonstrate its scalability empirically over large systems (using partial order reduction).

1 Introduction

Communicating automata are a Turing-complete model of asynchronous interactions [10] that has become one of the most prominent for studying point-to-point communications over unbounded first-in-first-out channels. This paper focuses on a class of communicating automata, called *communicating session automata* (CSA), which strictly includes automata corresponding to *asynchronous multiparty session types* [28]. Session types originated as a typing discipline for the π-calculus [27, 66], where a session type dictates the behaviour of a process wrt. its communications. Session types and related theories have been applied to the verification and specification of concurrent and distributed systems through their integration in several mainstream programming languages, e.g., Haskell [44, 55], Erlang [49], F♯ [48], Go [11, 37, 38, 51], Java [30, 31, 34, 65], OCaml [56], C [52], Python [16, 47, 50], Rust [32], and Scala [61, 62]. Communicating automata and asynchronous multiparty session types [28] are closely related: the latter can be seen as a syntactical representation of the former [17] where a sending state corresponds to an internal choice and a receiving state to an external choice. This correspondence between communicating automata and multiparty session types has become the foundation of many tools centred on session types, e.g., for generating communication API from multiparty session (global) types [30,31,48,61],

for detecting deadlocks in message-passing programs [51, 67], and for monitoring session-enabled programs [5, 16, 47, 49, 50]. These tools rely on a property called *multiparty compatibility* [6, 18, 39], which guarantees that communicating automata representing session types interact correctly, hence enabling the identification of correct protocols or the detection of errors in endpoint programs. Multiparty compatible communicating automata validate two essential requirements for session types frameworks: every message that is sent can be eventually received and each automaton can always eventually make a move. Thus, they satisfy the *abstract* safety invariant φ for session types from [63], a prerequisite for session type systems to guarantee safety of the typed processes. Unfortunately, multiparty compatibility suffers from a severe limitation: it requires that each execution of the system has a synchronous equivalent. Hence, it rules out many correct systems. Hereafter, we refer to this property as *synchronous multiparty compatibility* (SMC) and explain its main limitation with Example 1.

Example 1. The system in Figure 1 contains an interaction pattern that is *not* supported by any definition of SMC [6, 18, 39]. It consists of a client (`c`), a server (`s`), and a logger (`l`), which communicate via unbounded FIFO channels. Transition `sr`!a denotes that `s`ender puts (asynchronously) message a on channel `sr`; and transition `sr`?a denotes the consumption of a from channel `sr` by `r`eceiver. The `c`lient sends a *req*uest and some *data* in a fire-and-forget fashion, before waiting for a response from the `s`erver. Because of the presence of this simple pattern, the system cannot be executed synchronously (i.e., with the restriction that a send action can only be fired when a matching receive is enabled), hence it is rejected by all definitions of SMC from previous works, even though the system is safe (all sent messages are received and no automaton gets stuck).

Synchronous multiparty compatibility is reminiscent of a strong form of existential boundedness. Among the existing sub-classes of communicating automata (see [46] for a survey), existentially k-bounded communicating automata [22] stand out because they can be model-checked [8, 21] and they restrict the model in a natural way: any execution can be rescheduled such that the number of pending messages *that can be received* is bounded by k. However, existential boundedness is generally *undecidable* [22], even for a fixed bound k. This shortcoming makes it impossible to know when theoretical results are applicable.

To address the limitation of SMC and the shortcoming of existential boundedness, we propose a (decidable) sufficient condition for existential boundedness, called *k-exhaustivity*, which serves as a basis for a wider notion of new compatibility, called *k-multiparty compatibility* (k-MC) where $k \in \mathbb{N}_{>0}$ is a bound on the number of pending messages in each channel. A system is k-MC when it is (*i*) *k-exhaustive*, i.e., all k-reachable send actions are enabled within the bound, and (*ii*) *k-safe*, i.e., within the bound k, all sent messages can be received and each automaton can always eventually progress. For example, the system in Figure 1 is k-multiparty compatible for any $k \in \mathbb{N}_{>0}$, hence it does not lead to communication errors, see Theorem 1. The k-MC condition is a natural constraint for real-world systems. Indeed any finite-state system is k-exhaustive

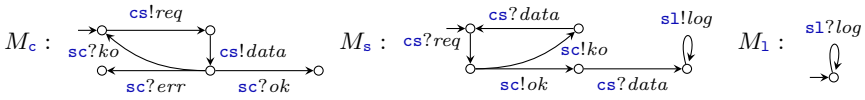


Fig. 1. Client-Server-Logger example.

(for k sufficiently large), while any system that is not k-exhaustive (resp. k-safe) for any k is unlikely to work correctly. Furthermore, we show that if a system of CSA validates k-exhaustivity, then each automaton locally behaves equivalently under any bound greater than or equal to k, a property that we call *local bound-agnosticity*. We give a *sound and complete* characterisation of k-exhaustivity for CSA in terms of local bound-agnosticity, see Theorem 3. Additionally, we show that the complexity of checking k-MC is PSPACE-complete (i.e., no higher than related algorithms) and we demonstrate empirically that its cost can be mitigated through (sound and complete) partial order reduction.

In this paper, we consider *communicating session automata* (CSA), which cover the most common form of asynchronous multiparty session types [15] (see *Remark 3*), and have been used as a basis to study properties and extensions of session types [6, 7, 18, 30, 31, 41, 42, 47, 49, 50]. More precisely, CSA are deterministic automata, whose every state is either sending (internal choice), receiving (external choice), or final. We focus on CSA that preserve the intent of internal and external choices from session types. In these CSA, whenever an automaton is in a sending state, it can fire any transition, no matter whether channels are bounded; when it is in a receiving state then at most one action must be enabled.

Synopsis In § 2, we give the necessary background on communicating automata and their properties, and introduce the notions of output/input bound independence which guarantee that internal/external choices are preserved in bounded semantics. In § 3, we introduce the definition of k-multiparty compatibility (k-MC) and show that k-MC systems are safe for systems which validate the bound independence properties. In § 4, we formally relate existential boundedness [22, 35], synchronisability [9], and k-exhaustivity. In § 5 we present an implementation (using partial order reduction) and an experimental evaluation of our theory. We discuss related works in § 6 and conclude in § 7.

See [43] for a full version of this paper (including proofs and additional examples). Our implementation and benchmark data are available online [33].

2 Communicating Automata and Bound Independence

This section introduces notations and definitions of communicating automata (following [12, 39]), as well as the notion of output (resp. input) bound independence which enforces the intent of internal (resp. external) choice in CSA.

Fix a finite set $\mathcal{P}$ of *participants* (ranged over by $\mathtt{p}$, $\mathtt{q}$, $\mathtt{r}$, $\mathtt{s}$, etc.) and a finite alphabet Σ. The set of *channels* is $\mathcal{C} \stackrel{\text{def}}{=} \{\mathtt{pq} \mid \mathtt{p}, \mathtt{q} \in \mathcal{P} \text{ and } \mathtt{p} \neq \mathtt{q}\}$,

$\mathcal{A} \stackrel{\text{def}}{=} \mathcal{C} \times \{!, ?\} \times \Sigma$ is the set of *actions* (ranged over by ℓ), Σ^* (resp. $\mathcal{A}^*$) is the set of finite words on Σ (resp. $\mathcal{A}$). Let w range over Σ^*, and ϕ, ψ range over $\mathcal{A}^*$. Also, ϵ ($\notin \Sigma \cup \mathcal{A}$) is the empty word, $|w|$ denotes the length of w, and $w \cdot w'$ is the concatenation of w and w' (these notations are overloaded for words in $\mathcal{A}^*$).

Definition 1 (Communicating automaton). *A* communicating automaton *is a finite transition system given by a triple $M = (Q, q_0, \delta)$ where Q is a finite set of* states*, $q_0 \in Q$ is the initial state, and $\delta \subseteq Q \times \mathcal{A} \times Q$ is a set of* transitions*.*

The transitions of a communicating automaton are labelled by actions in $\mathcal{A}$ of the form $\mathtt{sr}!a$, representing the *emission* of message a from participant $\mathtt{s}$ to $\mathtt{r}$, or $\mathtt{sr}?a$ representing the *reception* of a by $\mathtt{r}$. Define $subj(\mathtt{pq}!a) = subj(\mathtt{qp}?a) = \mathtt{p}$, $obj(\mathtt{pq}!a) = obj(\mathtt{qp}?a) = \mathtt{q}$, and $chan(\mathtt{pq}!a) = chan(\mathtt{pq}?a) = \mathtt{pq}$. The projection of ℓ onto $\mathtt{p}$ is defined as $\pi_{\mathtt{p}}(\ell) = \ell$ if $subj(\ell) = \mathtt{p}$ and $\pi_{\mathtt{p}}(\ell) = \epsilon$ otherwise. Let $\dagger$ range over $\{!, ?\}$, we define: $\pi^{\dagger}_{\mathtt{pq}}(\mathtt{pq}\dagger a) = a$ and $\pi^{\dagger'}_{\mathtt{pq}}(\mathtt{sr}\dagger a) = \epsilon$ if either $\mathtt{pq} \neq \mathtt{sr}$ or $\dagger \neq \dagger'$. We extend these definitions to sequences of actions in the natural way.

A state $q \in Q$ with no outgoing transition is *final*; q is *sending* (resp. *receiving*) if it is not final and all its outgoing transitions are labelled by send (resp. receive) actions, and q is *mixed* otherwise. $M = (Q, q_0, \delta)$ is *deterministic* if $\forall (q, \ell, q'), (q, \ell', q'') \in \delta : \ell = \ell' \implies q' = q''$. $M = (Q, q_0, \delta)$ is *send* (resp. *receive*) *directed* if for all sending (resp. receiving) $q \in Q$ and $(q, \ell, q'), (q, \ell', q'') \in \delta : obj(\ell) = obj(\ell')$. M is *directed* if it is send and receive directed.

Remark 1. In this paper, we consider only deterministic communicating automata without mixed states, and call them *Communicating Session Automata* (CSA). We discuss possible extensions of our results beyond this class in Section 7.

Definition 2 (System). *Given a communicating automaton $M_{\mathtt{p}} = (Q_{\mathtt{p}}, q_{0\mathtt{p}}, \delta_{\mathtt{p}})$ for each $\mathtt{p} \in \mathcal{P}$, the tuple $S = (M_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ is a* system*. A* configuration *of S is a pair $s = (\boldsymbol{q}; \boldsymbol{w})$ where $\boldsymbol{q} = (q_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ with $q_{\mathtt{p}} \in Q_{\mathtt{p}}$ and where $\boldsymbol{w} = (w_{\mathtt{pq}})_{\mathtt{pq} \in \mathcal{C}}$ with $w_{\mathtt{pq}} \in \Sigma^*$; component $\boldsymbol{q}$ is the* control state *and $q_{\mathtt{p}} \in Q_{\mathtt{p}}$ is the* local state *of automaton $M_{\mathtt{p}}$. The* initial configuration *of S is $s_0 = (\boldsymbol{q_0}; \boldsymbol{\epsilon})$ where $\boldsymbol{q_0} = (q_{0\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ and we write $\boldsymbol{\epsilon}$ for the $|\mathcal{C}|$-tuple $(\epsilon, \ldots, \epsilon)$.*

Hereafter, we fix a communicating session automaton $M_{\mathtt{p}} = (Q_{\mathtt{p}}, q_{0\mathtt{p}}, \delta_{\mathtt{p}})$ for each $\mathtt{p} \in \mathcal{P}$ and let $S = (M_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ be the corresponding system whose initial configuration is s_0. For each $\mathtt{p} \in \mathcal{P}$, we assume that $\forall (q, \ell, q') \in \delta_{\mathtt{p}} : subj(\ell) = \mathtt{p}$. We assume that the components of a configuration are named consistently, e.g., for $s' = (\boldsymbol{q}'; \boldsymbol{w}')$, we implicitly assume that $\boldsymbol{q}' = (q'_{\mathtt{p}})_{\mathtt{p} \in \mathcal{P}}$ and $\boldsymbol{w}' = (w'_{\mathtt{pq}})_{\mathtt{pq} \in \mathcal{C}}$.

Definition 3 (Reachable configuration). *Configuration $s' = (\boldsymbol{q}'; \boldsymbol{w}')$ is* reachable *from configuration $s = (\boldsymbol{q}; \boldsymbol{w})$ by* firing transition *ℓ, written $s \xrightarrow{\ell} s'$ (or $s \rightarrow s'$ when ℓ is not relevant), if there are $\mathtt{s}, \mathtt{r} \in \mathcal{P}$ and $a \in \Sigma$ such that either:*

1. *(a) $\ell = \mathtt{sr}!a$ and $(q_{\mathtt{s}}, \ell, q_{\mathtt{s}}') \in \delta_{\mathtt{s}}$, (b) $q'_{\mathtt{p}} = q_{\mathtt{p}}$ for all $\mathtt{p} \neq \mathtt{s}$, (c) $w'_{\mathtt{sr}} = w_{\mathtt{sr}} \cdot a$ and $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ for all $\mathtt{pq} \neq \mathtt{sr}$; or*
2. *(a) $\ell = \mathtt{sr}?a$ and $(q_{\mathtt{r}}, \ell, q_{\mathtt{r}}') \in \delta_{\mathtt{r}}$, (b) $q'_{\mathtt{p}} = q_{\mathtt{p}}$ for all $\mathtt{p} \neq \mathtt{r}$, (c) $w_{\mathtt{sr}} = a \cdot w'_{\mathtt{sr}}$, and $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ for all $\mathtt{pq} \neq \mathtt{sr}$.*

Remark 2. Hereafter, we assume that any bound k is finite and $k \in \mathbb{N}_{>0}$.

We write $\rightarrow^*$ for the reflexive and transitive closure of $\rightarrow$. Configuration $(\boldsymbol{q}; \boldsymbol{w})$ is k-bounded if $\forall \mathtt{pq} \in \mathcal{C} : |w_{\mathtt{pq}}| \leqslant k$. We write $s_1 \xrightarrow{\ell_1 \cdots \ell_n} s_{n+1}$ when $s_1 \xrightarrow{\ell_1} s_2 \cdots s_n \xrightarrow{\ell_n} s_{n+1}$, for some $s_2, \ldots, s_n$ (with $n \geqslant 0$); and say that the *execution* $\ell_1 \cdots \ell_n$ is *k-bounded from* s_1 if $\forall 1 \leqslant i \leqslant n+1 : s_i$ is k-bounded. Given $\phi \in \mathcal{A}^*$, we write $\mathtt{p} \notin \phi$ iff $\phi = \phi_0 \cdot \ell \cdot \phi_1 \implies \mathit{subj}(\ell) \neq \mathtt{p}$. We write $s \xrightarrow{\phi}_k s'$ if s' is reachable with a k-bounded execution ϕ from s. The set of *reachable configurations of* S is $RS(S) = \{s \mid s_0 \rightarrow^* s\}$. The *$k$-reachability set of* S is the largest subset $RS_k(S)$ of $RS(S)$ within which each configuration s can be reached by a k-bounded execution from s_0.

Definition 4 streamlines notions of safety from previous works [6, 12, 18, 39] (absence of deadlocks, orphan messages, and unspecified receptions).

Definition 4 (k-Safety). *S is* k-safe *if the following holds $\forall (\boldsymbol{q}; \boldsymbol{w}) \in RS_k(S)$:*

(ER) *$\forall \mathtt{pq} \in \mathcal{C}$, if $w_{\mathtt{pq}} = a \cdot w'$, then $(\boldsymbol{q}; \boldsymbol{w}) \rightarrow_k{}^* \xrightarrow{\mathtt{pq}?a}_k$.*

(PG) *$\forall \mathtt{p} \in \mathcal{P}$, if $q_{\mathtt{p}}$ is* receiving*, then $(\boldsymbol{q}; \boldsymbol{w}) \rightarrow_k{}^* \xrightarrow{\mathtt{qp}?a}_k$ for $\mathtt{q} \in \mathcal{P}$ and $a \in \Sigma$.*

We say that S is safe *if it validates the unbounded version of k-safety (∞-safe).*

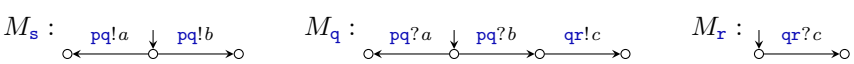
Property (ER), called *eventual reception*, requires that any sent message can always eventually be received (i.e., if a is the head of a queue then there must be an execution that consumes a), and Property (PG), called *progress*, requires that any automaton in a receiving state can eventually make a move (i.e., it can always eventually receive an *expected* message).

We say that a configuration s is *stable* iff $s = (\boldsymbol{q}; \boldsymbol{\epsilon})$, i.e., all its queues are empty. Next, we define the *stable property* for systems of communicating automata, following the definition from [18].

Definition 5 (Stable). *S has the* stable property *(SP) if $\forall s \in RS(S) : \exists (\boldsymbol{q}; \boldsymbol{\epsilon}) \in RS(S) : s \rightarrow^* (\boldsymbol{q}; \boldsymbol{\epsilon})$.*

A system has the stable property if it is possible to reach a stable configuration from any reachable configuration. This property is called *deadlock-free* in [22]. The stable property implies the eventual reception property, but not safety (e.g., an automaton may be waiting for an input in a stable configuration, see Example 2), and safety does not imply the stable property, see Example 4.

Example 2. The following system has the stable property, but it is not safe.



Next, we define two properties related to *bound independence*. They specify classes of CSA whose branching behaviours are not affected by channel bounds.

Definition 6 (k-OBI). *S is* k-output bound independent *(k-OBI), if $\forall s = (\boldsymbol{q}; \boldsymbol{w}) \in RS_k(S)$ and $\forall \mathtt{p} \in \mathcal{P}$, if $s \xrightarrow{\mathtt{pq}!a}_k$, then $\forall (q_{\mathtt{p}}, \mathtt{pr}!b, q'_{\mathtt{p}}) \in \delta_{\mathtt{p}} : s \xrightarrow{\mathtt{pr}!b}_k$.*

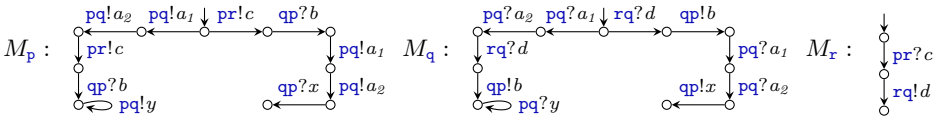


Fig. 2. Example of a *non*-IBI and *non*-safe system.

Definition 7 (k-IBI). *S is* k-input bound independent *(k-IBI), if* $\forall s = (\boldsymbol{q}; \boldsymbol{w}) \in RS_k(S)$ *and* $\forall \mathtt{p} \in \mathcal{P}$*, if* $s \xrightarrow{\mathtt{qp}?a}_k$*, then* $\forall \ell \in \mathcal{A} : s \xrightarrow{\ell}_k \wedge subj(\ell) = \mathtt{p} \implies \ell = \mathtt{qp}?a$.

If S is k-OBI, then any automaton that reaches a sending state is able to fire any of its available transitions, i.e., sending states model *internal choices* which are not constrained by bounds greater than or equal to k. Note that the unbounded version of k-OBI ($k = \infty$) is trivially satisfied for any system due to unbounded asynchrony. If S is k-IBI, then any automaton that reaches a receiving state is able to fire at most one transition, i.e., receiving states model *external choices* where the behaviour of the receiving automaton is controlled exclusively by its environment. We write IBI for the unbounded version of k-IBI ($k = \infty$).

Checking the IBI property is generally undecidable. However, systems consisting of (send and receive) *directed* automata are trivially k-IBI and k-OBI for all k, this subclass of CSA was referred to as *basic* in [18]. We introduce larger decidable approximations of IBI with Definitions 10 and 11.

Proposition 1. *(1) If S is send directed, then S is* k-OBI *for all* $k \in \mathbb{N}_{>0}$*. (2) If S is receive directed, then S is* IBI *(and* k-IBI *for all* $k \in \mathbb{N}_{>0}$*).*

Remark 3. CSA validating k-OBI and IBI strictly include the most common forms of asynchronous multiparty session types, e.g., the directed CSA of [18], and systems obtained by projecting Scribble specifications (global types) which need to be receive directed (this is called "consistent external choice subjects" in [31]) and which validate 1-OBI by construction since they are projections of synchronous specifications where choices must be located at a unique sender.

3 Bounded Compatibility for CSA

In this section, we introduce *k-multiparty compatibility* (k-MC) and study its properties wrt. safety of communicating session automata (CSA) which are k-OBI and IBI. Then, we soundly and completely characterise k-exhaustivity in terms of local bound-agnosticity, a property which guarantees that communicating automata behave equivalently under any bound greater than or equal to k.

3.1 Multiparty Compatibility

The definition of k-MC is divided in two parts: (*i*) *k-exhaustivity* guarantees that the set of k-reachable configurations contains enough information to make a

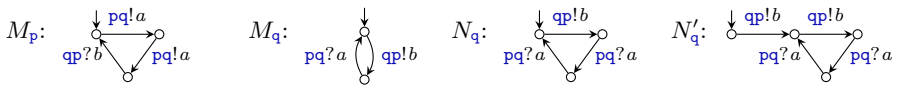


Fig. 3. $(M_{\mathtt{p}}, M_{\mathtt{q}})$ is non-exhaustive, $(M_{\mathtt{p}}, N_{\mathtt{q}})$ is 1-exhaustive, $(M_{\mathtt{p}}, N'_{\mathtt{q}})$ is 2-exhaustive.

sound decision wrt. safety of the system; and (*ii*) *k-safety* (Definition 4) guarantees that a subset of all possible executions is free of any communication errors. Next, we define k-exhaustivity, then k-multiparty compatibility. Intuitively, a system is k-exhaustive if for all k-reachable configurations, whenever a send action is enabled, then it can be fired within a k-bounded execution.

Definition 8 (k-Exhaustivity). *S is* k-exhaustive *if* $\forall(\boldsymbol{q}; \boldsymbol{w}) \in RS_k(S)$ *and* $\forall \mathtt{p} \in \mathcal{P}$*, if* $q_{\mathtt{p}}$ *is* sending*, then* $\forall(q_{\mathtt{p}}, \ell, q'_{\mathtt{p}}) \in \delta_{\mathtt{p}} : \exists \phi \in \mathcal{A}^* : (\boldsymbol{q}; \boldsymbol{w}) \xrightarrow{\phi}_k \xrightarrow{\ell}_k \wedge \mathtt{p} \notin \phi$.

Definition 9 (k-Multiparty compatibility). *S is* k-multiparty compatible *(k-MC) if it is k-safe and k-exhaustive.*

Definition 9 is a natural extension of the definitions of *synchronous* multiparty compatibility given in [18, Definition 4.2] and [6, Definition 4]. The common key requirements are that *every send* action must be matched by a receive action (i.e., send actions are universally quantified), while *at least one receive* action must find a matching send action (i.e., receive actions are existentially quantified). Here, the universal check on send actions is done via the eventual reception property (ER) and the k-exhaustivity condition; while the existential check on receive actions is dealt with by the progress property (PG).

Whenever systems are k-OBI and IBI, then k-exhaustivity implies that k-bounded executions are sufficient to make a sound decision wrt. safety. This is not necessarily the case for systems outside of this class, see Examples 3 and 5.

Example 3. The system $(M_{\mathtt{p}}, M_{\mathtt{q}}, M_{\mathtt{r}})$ in Figure 2 is k-OBI for any k, but not IBI (it is 1-IBI but not k-IBI for any $k \geqslant 2$). When executing with a bound strictly greater than 1, there is a configuration where $M_{\mathtt{q}}$ is in its initial state and *both* its *receive* transitions are enabled. The system is 1-safe and 1-exhaustive (hence 1-MC) but it is *not* 2-exhaustive nor 2-safe. By constraining the automata to execute with a channel bound of 1, the left branch of $M_{\mathtt{p}}$ is prevented to execute together with the right branch of $M_{\mathtt{q}}$. Thus, the fact that the y messages are not received in this case remains invisible in 1-bounded executions. This example can be easily extended so that it is n-exhaustive (resp. safe) but not n+1-exhaustive (resp. safe) by sending/receiving n+1 a_i messages.

Example 4. The system in Figure 1 is *directed* and 1-MC. The system $(M_{\mathtt{p}}, M_{\mathtt{q}})$ in Figure 3 is safe but *not* k-MC for any finite $k \in \mathbb{N}_{>0}$. Indeed, for any execution of this system, at least one of the queues grows arbitrarily large. The system $(M_{\mathtt{p}}, N_{\mathtt{q}})$ is 1-MC while the system $(M_{\mathtt{p}}, N'_{\mathtt{q}})$ is *not* 1-MC but it is 2-MC.

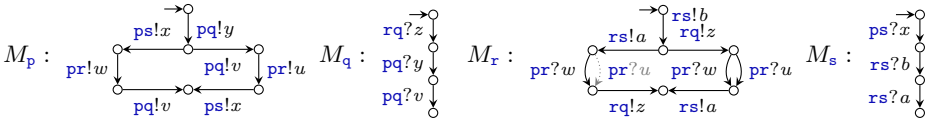


Fig. 4. Example of a system which is not 1-OBI.

Example 5. The system in Figure 4 (without the dotted transition) is 1-MC, but not 2-safe; it is not 1-OBI but it is 2-OBI. In 1-bounded executions, $M_{\mathtt{r}}$ can execute $\mathtt{rs}!b \cdot \mathtt{rp}!z$, but it cannot fire $\mathtt{rs}!b \cdot \mathtt{rs}!a$ (queue $\mathtt{rs}$ is full), which violates the 1-OBI property. The system with the dotted transition is not 1-OBI, but it is 2-OBI and k-MC for any $k \geqslant 1$. Both systems are receive directed, hence IBI.

Theorem 1. *If S is k-OBI, IBI, and k-MC, then it is safe.*

Remark 4. It is undecidable whether there exists a bound k for which an arbitrary system is k-MC. This is a consequence of the Turing completeness of communicating (session) automata [10, 20, 42].

Although the IBI property is generally undecidable, it is possible to identify sound approximations, as we show below. We adapt the dependency relation from [39] and say that action ℓ' depends on ℓ from $s = (\boldsymbol{q}; \boldsymbol{w})$, written $s \vdash \ell \prec \ell'$, iff $subj(\ell) = subj(\ell') \vee (chan(\ell) = chan(\ell') \wedge w_{chan(\ell)} = \epsilon)$. Action ℓ' depends on ℓ in ϕ from s, written $s \vdash \ell \prec_\phi \ell'$, if the following holds:

$$s \vdash \ell \prec_\phi \ell' \iff \begin{cases} (s \vdash \ell \prec \ell'' \wedge s \vdash \ell'' \prec_\psi \ell') \vee s \vdash \ell \prec_\psi \ell' & \text{if } \phi = \ell'' \cdot \psi \\ s \vdash \ell \prec \ell' & \text{otherwise} \end{cases}$$

Definition 10. *S is k-*chained input bound independent *(k-CIBI) if $\forall s = (\boldsymbol{q}; \boldsymbol{w}) \in RS_k(S)$ and $\forall \mathtt{p} \in \mathcal{P}$, if $s \xrightarrow{\mathtt{qp}?a}_k s'$, then $\forall (q_{\mathtt{p}}, \mathtt{sp}?b, q'_{\mathtt{p}}) \in \delta_{\mathtt{p}} : \mathtt{s} \neq \mathtt{q} \implies \neg(s \xrightarrow{\mathtt{sp}?b}_k) \wedge (\forall \phi \in \mathcal{A}^* : s' \xrightarrow{\phi}_k \xrightarrow{\mathtt{sp}!b}_k \implies s \vdash \mathtt{qp}?a \prec_\phi \mathtt{sp}!b)$.*

Definition 11. *S is k-*strong input bound independent *(k-SIBI) if $\forall s = (\boldsymbol{q}; \boldsymbol{w}) \in RS_k(S)$ and $\forall \mathtt{p} \in \mathcal{P}$, if $s \xrightarrow{\mathtt{qp}?a}_k s'$, then $\forall (q_{\mathtt{p}}, \mathtt{sp}?b, q'_{\mathtt{p}}) \in \delta_{\mathtt{p}} : \mathtt{s} \neq \mathtt{q} \implies \neg(s \xrightarrow{\mathtt{sp}?b}_k \vee s' \rightarrow_k{}^* \xrightarrow{\mathtt{sp}!b}_k)$.*

Definition 10 requires that whenever $\mathtt{p}$ can fire a receive action, at most one of its receive actions is enabled at s, and no other receive transition from $q_{\mathtt{p}}$ will be enabled until $\mathtt{p}$ has made a move. This is due to the existence of a dependency chain between the reception of a message ($\mathtt{qp}?a$) and the matching send of another possible reception ($\mathtt{sp}!b$). Property k-SIBI (Definition 11) is a stronger version of k-CIBI, which can be checked more efficiently.

Lemma 1. *If S is k-OBI, k-CIBI (resp. k-SIBI) and k-exhaustive, then it is* IBI.

The decidability of k-OBI, k-IBI, k-SIBI, k-CIBI, and k-MC is straightforward since both $RS_k(S)$ (which has an exponential number of states wrt. k) and $\rightarrow_k$ are finite, given a finite k. Theorem 2 states the space complexity of the procedures, except for k-CIBI for which a complexity class is yet to be determined. We show that the properties are PSPACE by reducing to an instance of the reachability problem over a transition system built following the construction of Bollig et al. [8, Theorem 6.3]. The rest of the proof follows from similar arguments in Genest et al. [22, Proposition 5.5] and Bouajjani et al. [9, Theorem 3].

Theorem 2. *The problems of checking the k-OBI, k-IBI, k-SIBI, k-safety, and k-exhaustivity properties are all decidable and* PSPACE*-complete (with $k \in \mathbb{N}_{>0}$ given in unary). The problem of checking the k-CIBI property is decidable.*

3.2 Local Bound-Agnosticity

We introduce local bound-agnosticity and show that it fully characterises k-exhaustive systems. Local bound-agnosticity guarantees that each communicating automaton behave in the same manner for any bound greater than or equal to some k. Therefore such systems may be executed transparently under a bounded semantics (a communication model available in Go and Rust).

Definition 12 (Transition system). *The k-bounded transition system of S is the labelled transition system (LTS) $TS_k(S) = (N, s_0, \Delta)$ such that $N = RS_k(S)$, s_0 is the initial configuration of S, $\Delta \subseteq N \times \mathcal{A} \times N$ is the transition relation, and $(s, \ell, s') \in \Delta$ if and only if $s \xrightarrow{\ell}_k s'$.*

Definition 13 (Projection). *Let $\mathcal{T}$ be an LTS over $\mathcal{A}$. The* projection *of $\mathcal{T}$ onto $\mathtt{p}$, written $\pi^{\epsilon}_{\mathtt{p}}(\mathcal{T})$, is obtained by replacing each label ℓ in $\mathcal{T}$ by $\pi_{\mathtt{p}}(\ell)$.*

Recall that the projection of action ℓ, written $\pi_{\mathtt{p}}(\ell)$, is defined in Section 2. The automaton $\pi^{\epsilon}_{\mathtt{p}}(TS_k(S))$ is essentially the *local* behaviour of participant $\mathtt{p}$ within the transition system $TS_k(S)$. When each automaton in a system S behaves equivalently for any bound greater than or equal to some k, we say that S is locally bound-agnostic. Formally, S is *locally bound-agnostic for k* when $\pi^{\epsilon}_{\mathtt{p}}(TS_k(S))$ and $\pi^{\epsilon}_{\mathtt{p}}(TS_n(S))$ are weakly bisimilar ($\approx$) for each participant $\mathtt{p}$ and any $n \geqslant k$. For k-OBI and IBI systems, local bound-agnosticity is a *necessary and sufficient* condition for k-exhaustivity, as stated in Theorem 3 and Corollary 1.

Theorem 3. *Let S be a system.*

(1) If $\exists k \in \mathbb{N}_{>0} : \forall \mathtt{p} \in \mathcal{P} : \pi^{\epsilon}_{\mathtt{p}}(TS_k(S)) \approx \pi^{\epsilon}_{\mathtt{p}}(TS_{k+1}(S))$, then S is k-exhaustive.
(2) If S is k-OBI, IBI, and k-exhaustive, then $\forall \mathtt{p} \in \mathcal{P} : \pi^{\epsilon}_{\mathtt{p}}(TS_k(S)) \approx \pi^{\epsilon}_{\mathtt{p}}(TS_{k+1}(S))$.

Corollary 1. *Let S be k-OBI and IBI s.t. $\forall \mathtt{p} \in \mathcal{P} : \pi^{\epsilon}_{\mathtt{p}}(TS_k(S)) \approx \pi^{\epsilon}_{\mathtt{p}}(TS_{k+1}(S))$, then S is locally bound-agnostic for k.*

Theorem 3 (1) is reminiscent of the (PSPACE-complete) checking procedure for existentially bounded systems with the stable property [22] (an *undecidable* property). Recall that k-exhaustivity is not sufficient to guarantee safety, see Examples 3 and 5. We give an effective procedure (based on partial order reduction) to check k-exhaustivity and related properties in [43].

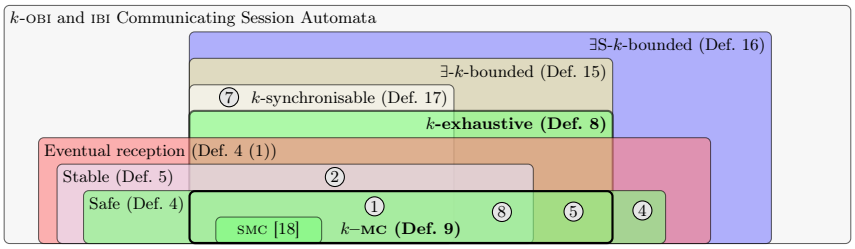


Fig. 5. Relations between k-exhaustivity, existential k-boundedness, and k-synchronisability in k-OBI and IBI CSA (the circled numbers refer to Table 1).

4 Existentially Bounded and Synchronisable Automata

4.1 Kuske and Muscholl's Existential Boundedness

Existentially bounded communicating automata [21, 22, 35] are a class of communicating automata whose executions can always be scheduled in such a way that the number of pending messages is bounded by a given value. Traditionally, existentially bounded communicating automata are defined on communicating automata that feature (local) accepting states and in terms of *accepting runs*. An accepting run is an execution (starting from s_0) which terminates in a configuration $(\boldsymbol{q}; \boldsymbol{w})$ where each $q_{\mathtt{p}}$ is a local accepting state. In our setting, we simply consider that every local state $q_{\mathtt{p}}$ is an accepting state, hence any execution ϕ starting from s_0 is an accepting run. We first study existential boundedness as defined in [35] as it matches more closely k-exhaustivity, we study the "classical" definition of existential boundedness [22] in Section 4.2.

Following [35], we say that an execution $\phi \in \mathcal{A}^*$ is *valid* if for any prefix ψ of ϕ and any channel $\mathtt{pq} \in \mathcal{C}$, we have that $\pi^?_{\mathtt{pq}}(\psi)$ is a prefix of $\pi^!_{\mathtt{pq}}(\psi)$, i.e., an execution is valid if it models the FIFO semantics of communicating automata.

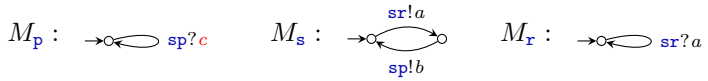
Definition 14 (Causal equivalence [35]). *Given $\phi, \psi \in \mathcal{A}^*$, we define: $\phi \leftrightarrows \psi$* iff *$\phi$ and ψ are* valid *executions and $\forall \mathtt{p} \in \mathcal{P} : \pi_{\mathtt{p}}(\phi) = \pi_{\mathtt{p}}(\psi)$. We write $[\phi]_{\leftrightarrows}$ for the equivalence class of ϕ wrt. $\leftrightarrows$.*

Definition 15 (Existential boundedness [35]). *We say that a valid execution ϕ is* k-match-bounded *if, for every prefix ψ of ϕ the difference between the number of* matched *events of type* $\mathtt{pq}!$ *and those of type* $\mathtt{pq}?$ *is bounded by k, i.e., $min\{|\pi^!_{\mathtt{pq}}(\psi)|, |\pi^?_{\mathtt{pq}}(\phi)|\} - |\pi^?_{\mathtt{pq}}(\psi)| \leqslant k$.*
Write $\mathcal{A}^*|_k$ for the set of k-match-bounded words. An execution ϕ is existentially k-bounded *if $[\phi]_{\leftrightarrows} \cap \mathcal{A}^*|_k \neq \varnothing$. A system S is existentially k-bounded, written $\exists$-k-bounded, if each execution in $\{\phi \mid \exists s : s_0 \xrightarrow{\phi} s\}$ is existentially k-bounded.*

Example 6. Consider Figure 3. $(M_{\mathtt{p}}, M_{\mathtt{q}})$ is *not* existentially k-bounded, for any k: at least one of the queues must grow infinitely for the system to progress. Systems $(M_{\mathtt{p}}, N_{\mathtt{q}})$ and $(M_{\mathtt{p}}, N'_{\mathtt{q}})$ are existentially bounded since any of their executions can be scheduled to an $\leftrightarrows$-equivalent execution which is 2-match-bounded.

The relationship between k-exhaustivity and existential boundedness is stated in Theorem 4 and illustrated in Figure 5 for k-OBI and IBI CSA, where SMC refers to synchronous multiparty compatibility [18, Definition 4.2]. The circled numbers in the figure refer to key examples summarised in Table 1. The strict inclusion of k-exhaustivity in existential k-boundedness is due to systems that do not have the eventual reception property, see Example 7.

Example 7. The system below is ∃-1-bounded but is *not* k-exhaustive for any k.



For any k, the channel sp eventually gets full and the send action sp!b can no longer be fired; hence it does *not* satisfy k-exhaustivity. Note that each execution can be reordered into a 1-match-bounded execution (the b's are never matched).

Theorem 4. *(1) If S is k-*OBI, IBI*, and k-exhaustive, then it is ∃-k-bounded. (2) If S is ∃-k-bounded and satisfies eventual reception, then it is k-exhaustive.*

4.2 Existentially Stable Bounded Communicating Automata

The "classical" definition of existentially bounded communicating automata as found in [22] differs slightly from Definition 15, as it relies on a different notion of accepting runs, see [22, page 4]. Assuming that all local states are accepting, we adapt their definition as follows: a *stable accepting run* is an execution ϕ starting from s_0 which terminates in a *stable* configuration.

Definition 16 (Existential stable boundedness [22]). *A system S is* existentially stable k-bounded*, written ∃S-k-bounded, if for each execution ϕ in $\{\phi \mid \exists(\boldsymbol{q};\boldsymbol{\epsilon}) \in RS(S) : s_0 \xrightarrow{\phi} (\boldsymbol{q};\boldsymbol{\epsilon})\}$ there is ψ such that $s_0 \xrightarrow{\psi}_k$ with $\phi \precsim \psi$.*

A system is existentially stable k-bounded if each of its executions leading to a *stable* configuration can be re-ordered into a k-bounded execution (from s_0).

Theorem 5. *(1) If S is existentially k-bounded, then it is existentially* stable *k-bounded. (2) If S is existentially* stable *k-bounded and has the stable property, then it is existentially k-bounded.*

We illustrate the relationship between existentially stable bounded communicating automata and the other classes in Figure 5. The example below further illustrates the strictness of the inclusions, see Table 1 for a summary.

Example 8. Consider the systems in Figure 3. $(M_{\mathtt{p}}, M_{\mathtt{q}})$ and $(M_{\mathtt{p}}, N'_{\mathtt{q}})$ are (trivially) existentially stable 1-bounded since none of their (non-empty) executions terminate in a stable configuration. The system $(M_{\mathtt{p}}, N_{\mathtt{q}})$ is existentially stable 2-bounded since each of its executions can be re-ordered into a 2-bounded one. The system in Example 7 is (trivially) ∃S-1-bounded: none of its (non-empty) executions terminate in a stable configuration (the b's are never received).

Theorem 6. *Let S be an ∃(S)-k-bounded system with the stable property, then it is k-exhaustive.*

Table 1. Properties for key examples, where direct. stands for directed, OBI for k-OBI, SIBI for k-SIBI, ER for eventual reception property, SP for stable property, exh. for k-exhaustive, $\exists$(S)-b for $\exists$(S)-bounded, and syn. for n-synchronisable (for some $n \in \mathbb{N}_{>0}$).

#	System	Ref.	k	direct.	OBI	SIBI	safe	ER	SP	exh.	$\exists$S-b	$\exists$-b	syn.
1	(M_c, M_s, M_1)	Fig. 1	1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
2	(M_s, M_q, M_r)	Ex. 2	1	yes	yes	yes	no	yes	yes	yes	yes	yes	yes
3	(M_p, M_q, M_r)	Fig. 2	$\geqslant 2$	no	yes	no	no	no	no	no	yes	yes	no
4	(M_p, M_q)	Fig. 3	any	yes	yes	yes	yes	yes	no	no	yes	no	no
5	(M_p, N'_q)	Fig. 3	2	yes	yes	yes	yes	yes	no	yes	yes	yes	no
6	(M_p, M_q, M_r, M_s)	Fig. 4	2	no	yes	yes	yes	yes	no	yes	yes	yes	no
7	(M_s, M_r, M_p)	Ex. 7	any	yes	yes	yes	no	no	no	no	yes	yes	yes
8	(M_p, M_q)	Ex. 9	1	yes	yes	yes	yes	yes	yes	yes	yes	yes	no

4.3 Synchronisable Communicating Session Automata

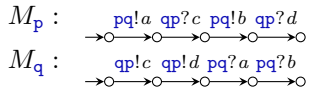
In this section, we study the relationship between synchronisability [9] and k-exhaustivity via existential boundedness. Informally, communicating automata are synchronisable if each of their executions can be scheduled in such a way that it consists of sequences of "exchange phases", where each phase consists of a bounded number of send actions, followed by a sequence of receive actions. The original definition of k-synchronisable systems [9, Definition 1] is based on communicating automata with *mailbox* semantics, i.e., each automaton has one input queue. Here, we adapt the definition so that it matches our point-to-point semantics. We write $\mathcal{A}_!$ for $\mathcal{A} \cap (\mathcal{C} \times \{!\} \times \Sigma)$, and $\mathcal{A}_?$ for $\mathcal{A} \cap (\mathcal{C} \times \{?\} \times \Sigma)$.

Definition 17 (Synchronisability). *A valid execution $\phi = \phi_1 \cdots \phi_n$ is a k-*exchange *if and only if: (1) $\forall 1 \leqslant i \leqslant n : \phi_i \in \mathcal{A}_!^* \cdot \mathcal{A}_?^* \wedge |\phi_i| \leqslant 2k$; and (2) $\forall \mathtt{pq} \in \mathcal{C} : \forall 1 \leqslant i \leqslant n : \pi^!_{\mathtt{pq}}(\phi_i) \neq \pi^?_{\mathtt{pq}}(\phi_i) \implies \forall i < j \leqslant n : \pi^?_{\mathtt{pq}}(\phi_j) = \epsilon$.*

We write $\mathcal{A}^* \|_k$ for the set of executions that are k-exchanges and say that an execution ϕ is k-synchronisable *if $[\phi]_{\fallingdotseq} \cap \mathcal{A}^* \|_k \neq \varnothing$. A system S is* k-synchronisable *if each execution in $\{\phi \mid \exists s : s_0 \xrightarrow{\phi} s\}$ is k-synchronisable.*

Condition (1) says that execution ϕ should be a sequence of an arbitrary number of send-receive phases, where each phase consists of at most $2k$ actions. Condition (2) says that if a message is not received in the phase in which it is sent, then it cannot be received in ϕ. Observe that the bound k is on the number of actions (over possibly different channels) in a phase rather than the number of pending messages in a given channel.

Example 9. The system below (left) is 1-MC and $\exists$(S)-1-bounded, but it is *not* k-synchronisable for any k. The subsequences of send-receive actions in the $\fallingdotseq$-equivalent executions below are highlighted (right).



$$\phi_1 = \underbrace{\mathtt{pq}!a \cdot \mathtt{qp}!c \cdot \mathtt{qp}?c}\cdot\underbrace{\mathtt{qp}!d \cdot \mathtt{pq}?a}\cdot\underbrace{\mathtt{pq}!b \cdot \mathtt{qp}?d \cdot \mathtt{pq}?b}$$

$$\phi_2 = \underbrace{\mathtt{pq}!a \cdot \mathtt{qp}!c \cdot \mathtt{qp}!d \cdot \mathtt{qp}?c \cdot \mathtt{pq}?a}\cdot\underbrace{\mathtt{pq}!b \cdot \mathtt{qp}?d \cdot \mathtt{pq}?b}$$

Execution ϕ_1 is 1-bounded for s_0, but it is not a k-exchange since, e.g., a is received outside of the phase where it is sent. In ϕ_2, message d is received outside

Table 2. Experimental evaluation. $|\mathcal{P}|$ is the number of participants, k is the bound, $|RTS|$ is the number of transitions in the *reduced* $TS_k(S)$ (see [43]), direct. stands for directed, Time is the time taken to check all the properties shown in this table, and GMC is `yes` if the system is generalised multiparty compatible [39].

Example	$\|\mathcal{P}\|$	k	$\|RTS\|$	*direct.*	k-OBI	k-CIBI	k-MC	Time	GMC
Client-Server-Logger	3	1	11	yes	yes	yes	yes	0.04s	no
4 Player game† [39]	4	1	20	no	yes	yes	yes	0.05s	yes
Bargain [39]	3	1	8	yes	yes	yes	yes	0.03s	yes
Filter collaboration [68]	2	1	10	yes	yes	yes	yes	0.03s	yes
Alternating bit† [59]	2	1	8	yes	yes	yes	yes	0.04s	no
TPMContract v2† [25]	2	1	14	yes	yes	yes	yes	0.04s	yes
Sanitary agency† [60]	4	1	34	yes	yes	yes	yes	0.07s	yes
Logistic† [54]	4	1	26	yes	yes	yes	yes	0.05s	yes
Cloud system v4 [24]	4	2	16	no	yes	yes	yes	0.04s	yes
Commit protocol [9]	4	1	12	yes	yes	yes	yes	0.03s	yes
Elevator† [9]	5	1	72	no	yes	no	yes	0.14s	no
Elevator-dashed† [9]	5	1	80	no	yes	no	yes	0.16s	no
Elevator-directed† [9]	3	1	41	yes	yes	yes	yes	0.07s	yes
Dev system [58]	4	1	20	yes	yes	yes	yes	0.05s	no
Fibonacci [48]	2	1	6	yes	yes	yes	yes	0.03s	yes
SAP-Negot. [48, 53]	2	1	18	yes	yes	yes	yes	0.04s	yes
SH [48]	3	1	30	yes	yes	yes	yes	0.06s	yes
Travel agency [48, 64]	3	1	21	yes	yes	yes	yes	0.05s	yes
HTTP [29, 48]	2	1	48	yes	yes	yes	yes	0.07s	yes
SMTP [30, 48]	2	1	108	yes	yes	yes	yes	0.08s	yes
gen_server (buggy) [67]	3	1	56	no	no	yes	no	0.03s	no
gen_server (fixed) [67]	3	1	45	no	yes	yes	yes	0.03s	yes
double buffering [45]	3	2	16	yes	yes	yes	yes	0.01s	no

of its sending phase. In the terminology of [9], this system is not k-synchronisable because there is a "*receive-send dependency*" between the exchange of message c and b, i.e., `p` must receive c before it sends b. Hence, there is no k-exchange that is $\precsim$-equivalent to ϕ_1 and ϕ_2.

Theorem 7. *(1) If S is k-synchronisable, then it is $\exists$-k-bounded. (2) If S is k-synchronisable and has the eventual reception property, then it is k-exhaustive.*

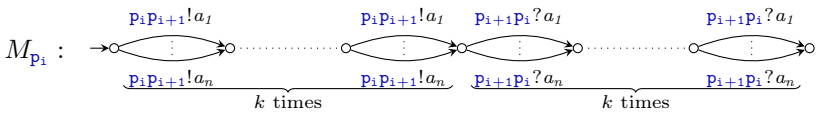
Figure 5 and Table 1 summarise the results of § 4 wrt. k-OBI and IBI CSA. We note that any finite-state system is k-exhaustive (and $\exists$(S)-k-bounded) for sufficiently large k, while this does not hold for synchronisability, see Example 9.

5 Experimental Evaluation

We have implemented our theory in a tool [33] which takes two inputs: (*i*) a system of communicating automata and (*ii*) a bound MAX. The tool iteratively checks whether the system validates the premises of Theorem 1, until it succeeds or reaches $k =$ MAX. We note that the k-OBI and IBI conditions are required for our soundness result (Theorem 1), but are orthogonal for checking k-MC. Each condition is checked on a *reduced bounded transition system*, called $RTS_k(S)$. Each verification procedure for these conditions is implemented in Haskell using a simple (depth-first-search based) reachability check on the paths of $RTS_k(S)$. We give an (optimal) partial order reduction algorithm to construct $RTS_k(S)$ in [43] and show that it preserves our properties.

We have tested our tool on 20 examples taken from the literature, which are reported in Table 2. The table shows that the tool terminates virtually instantaneously on all examples. The table suggests that many systems are indeed k-MC and most can be easily adapted to validate bound independence. The last column refers to the GMC condition, a form of *synchronous* multiparty compatibility (SMC) introduced in [39]. The examples marked with † have been slightly modified to make them CSA that validate k-OBI and IBI. For instance, we take only one of the possible interleavings between mixed actions to remove mixed states (taking send action before receive action to preserve safety), see [43].

We have assessed the scalability of our approach with automatically generated examples, which we report in Figure 6. Each system considered in these benchmarks consists of $2m$ (directed) CSA for some $m \geqslant 1$ such that $S = (M_{\mathtt{p_i}})_{1 \leqslant \mathtt{i} \leqslant 2m}$, and each automaton $M_{\mathtt{p_i}}$ is of the form (when i is *odd*):



Each $M_{\mathtt{p_i}}$ sends k messages to participant $\mathtt{p_{i+1}}$, then receives k messages from $\mathtt{p_{i+1}}$. Each message is taken from an alphabet $\{a_1, \ldots, a_n\}$ ($n \geqslant 1$). $M_{\mathtt{p_i}}$ has the same structure when i is *even*, but interacts with $\mathtt{p_{i-1}}$ instead. Observe that any system constructed in this way is k-MC for any $k \geqslant 1$, $n \geqslant 1$, and $m \geqslant 1$. The shape of these systems allows us to assess how our approach fares in the worst case, i.e., large number of paths in $RTS_k(S)$. Figure 6 gives the time taken for our tool to terminate (y axis) wrt. the number of transitions in $RTS_k(S)$ where k is the least natural number for which the system is k-MC. The plot on the left in Figure 6 gives the timings when k is increasing (every increment from k=2 to k=100) with the other parameters fixed (n=1 and m=5). The middle plot gives the timings when m is increasing (every increment from m=1 to m=26) with k=10 and n=1. The right-hand side plot gives the timings when n is increasing (every increment from n=1 to n=10) with k=2 and m=1. The largest $RTS_k(S)$ on which we have tested our tool has 12222 states and 22220 transitions, and the verification took under 17 minutes.[3] Observe that partial order reduction mitigates the increasing size of the transition system on which k-MC is checked, e.g., these experiments show that parameters k and m have only a linear effect on the number of transitions (see horizontal distances between data points). However the number of transitions increases exponentially with n (since the number of paths in each automaton increases exponentially with n).

6 Related Work

Theory of communicating automata Communicating automata were introduced, and shown to be Turing powerful, in the 1980s [10] and have since then been

[3] All the benchmarks in this paper were run on an 8-core Intel i7-7700 machine with 16GB RAM running a 64-bit Linux.

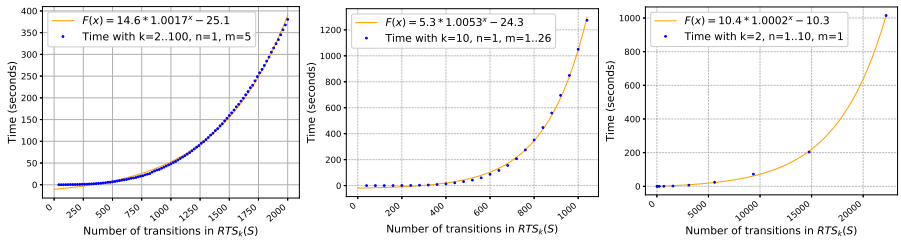


Fig. 6. Benchmarks: increasing k (left), increasing m (middle), and increasing n (right).

studied extensively, namely through their connection with message sequence charts (MSC) [46]. Several works achieved decidability results by using bag or lossy channels [1, 2, 13, 14] or by restricting the topology of the network [36, 57].

Existentially bounded communicating automata stand out because they preserve the FIFO semantics of communicating automata, do not restrict the topology of the network, and include infinite state systems. Given a bound k and an arbitrary system of (deterministic) communicating automata S, it is generally *undecidable* whether S is existentially k-bounded. However, the question becomes decidable (PSPACE-complete) when S has the stable property. The stable property is itself generally *undecidable* (it is called deadlock-freedom in [22, 35]). Hence this class is *not* directly applicable to the verification of message passing programs since its membership is overall undecidable. We have shown that k-OBI, IBI, and k-exhaustive CSA systems are (strictly) included in the class of existentially bounded systems. Hence, our work gives a sound *practical* procedure to check whether CSA are existentially k-bounded. To the best of our knowledge, the only tools dedicated to the verification of (unbounded) communicating automata are McScM [26] and Chorgram [40]. Bouajjani et al. [9] study a variation of communicating automata with *mailboxes* (one input queue per automaton). They introduce the class of synchronisable systems and a procedure to check whether a system is k-synchronisable; it relies on executions consisting of k-bounded exchange phases. Given a system and a bound k, it is decidable (PSPACE-complete) whether its executions are equivalent to k-synchronous executions. Section 4.3 states that any k-synchronisable system which satisfies eventual reception is also k-exhaustive, see Theorem 7. In contrast to existential boundedness, synchronisability does not include all finite-state systems. Our characterisation result, based on local bound-agnosticity (Theorem 3), is *unique* to k-exhaustivity. It does not apply to existential boundedness nor synchronisability, see, e.g., Example 7. The term "synchronizability" is used by Basu et al. [3, 4] to refer to another verification procedure for communicating automata with mailboxes. Finkel and Lozes [19] have shown that this notion of synchronizability is undecidable. We note that a system that is safe with a point-to-point semantics, may not be safe with a mailbox semantics (due to independent send actions), and vice-versa. For instance, the system in Figure 2 is safe when executed with mailbox semantics.

Multiparty compatibility and programming languages The first definition of multiparty compatibility appeared in [18, Definition 4.2], inspired by the work in [23], to characterise the relationship between global types and communicating automata. This definition was later adapted to the setting of communicating timed automata in [6]. Lange et al. [39] introduced a generalised version of multiparty compatibility (GMC) to support communicating automata that feature mixed or non-directed states. Because our results apply to automata without mixed states, k-MC is not a strict extension of GMC, and GMC is not a strict extension of k-MC either, as it requires the existence of *synchronous* executions. In future work, we plan to develop an algorithm to synthesise representative choreographies from k-MC systems, using the algorithm in [39].

The notion of multiparty compatibility is at the core of recent works that apply session types techniques to programming languages. Multiparty compatibility is used in [51] to detect deadlocks in Go programs, and in [30] to study the well-formedness of Scribble protocols [64] through the compatibility of their projections. These protocols are used to generate various endpoint APIs that implement a Scribble specification [30, 31, 48], and to produce runtime monitoring tools [47,49,50]. Taylor et al. [67] use multiparty compatibility and choreography synthesis [39] to automate the analysis of the `gen_server` library of Erlang/OTP. We can transparently widen the set of safe programs captured by these tools by using k-MC instead of synchronous multiparty compatibility (SMC). The k-MC condition corresponds to a much wider instance of the *abstract* safety invariant φ for session types defined in [63]. Indeed k-MC includes SMC (see [43]) and all finite-state systems (for k sufficiently large).

7 Conclusions

We have studied CSA via a new condition called k-exhaustivity. The k-exhaustivity condition is (i) the basis for a wider notion of multiparty compatibility, k-MC, which captures asynchronous interactions and (ii) the first practical, empirically validated, sufficient condition for existential k-boundedness. We have shown that k-exhaustive systems are fully characterised by local bound-agnosticity (each automaton behaves equivalently for any bound greater than or equal to k). This is a key requirement for asynchronous message passing programming languages where the possibility of having infinitely many orphan messages is undesirable, in particular for Go and Rust which provide *bounded* communication channels.

For future work, we plan to extend our theory beyond CSA. We believe that it is possible to support mixed states and states which do not satisfy IBI, as long as their outgoing transitions are independent (i.e., if they commute). Additionally, to make k-MC checking more efficient, we will elaborate heuristics to find optimal bounds and off-load the verification of k-MC to an off-the-shelf model checker.

Acknowledgements We thank Laura Bocchi and Alceste Scalas for their comments, and David Castro and Nicolas Dilley for testing the artifact. This work is partially supported by EPSRC EP/K034413/1, EP/K011715/1, EP/L00058X/1, EP/N027833/1, and EP/N028201/1.

References

1. P. A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In *CAV 1998*, pages 305–318, 1998.
2. P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *(LICS 1993)*, pages 160–170, 1993.
3. S. Basu and T. Bultan. Automated choreography repair. In *FASE 2016*, pages 13–30, 2016.
4. S. Basu, T. Bultan, and M. Ouederni. Deciding choreography realizability. In *POPL 2012*, pages 191–202, 2012.
5. L. Bocchi, T. Chen, R. Demangeon, K. Honda, and N. Yoshida. Monitoring networks through multiparty session types. *Theor. Comput. Sci.*, 669:33–58, 2017.
6. L. Bocchi, J. Lange, and N. Yoshida. Meeting deadlines together. In *CONCUR 2015*, pages 283–296, 2015.
7. L. Bocchi, W. Yang, and N. Yoshida. Timed multiparty session types. In *CONCUR 2014*, pages 419–434, 2014.
8. B. Bollig, D. Kuske, and I. Meinecke. Propositional dynamic logic for message-passing systems. *Logical Methods in Computer Science*, 6(3), 2010.
9. A. Bouajjani, C. Enea, K. Ji, and S. Qadeer. On the completeness of verifying message passing programs under bounded asynchrony. In *CAV 2018*, pages 372–391, 2018.
10. D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.
11. D. Castro, R. Hu, S. Jongmans, N. Ng, and N. Yoshida. Distributed programming using role-parametric session types in Go: statically-typed endpoint apis for dynamically-instantiated communication structures. *PACMPL*, 3(POPL):29:1–29:30, 2019.
12. G. Cécé and A. Finkel. Verification of programs with half-duplex communication. *Inf. Comput.*, 202(2):166–190, 2005.
13. G. Cécé, A. Finkel, and S. P. Iyer. Unreliable channels are easier to verify than perfect channels. *Inf. Comput.*, 124(1):20–31, 1996.
14. L. Clemente, F. Herbreteau, and G. Sutre. Decidable topologies for communicating automata with FIFO and bag channels. In *CONCUR 2014*, pages 281–296, 2014.
15. M. Coppo, M. Dezani-Ciancaglini, L. Padovani, and N. Yoshida. A Gentle Introduction to Multiparty Asynchronous Session Types. In *15th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Multicore Programming*, volume 9104 of *LNCS*, pages 146–178. Springer, 2015.
16. R. Demangeon, K. Honda, R. Hu, R. Neykova, and N. Yoshida. Practical interruptible conversations: distributed dynamic verification with multiparty session types and Python. *Formal Methods in System Design*, 46(3):197–225, 2015.
17. P. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP 2012*, pages 194–213, 2012.
18. P. Deniélou and N. Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP 2013*, pages 174–186, 2013.
19. A. Finkel and É. Lozes. Synchronizability of communicating finite state machines is not decidable. In *ICALP 2017*, pages 122:1–122:14, 2017.
20. A. Finkel and P. McKenzie. Verifying identical communicating processes is undecidable. *Theor. Comput. Sci.*, 174(1-2):217–230, 1997.

21. B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Inf. Comput.*, 204(6):920–956, 2006.
22. B. Genest, D. Kuske, and A. Muscholl. On communicating automata with bounded channels. *Fundam. Inform.*, 80(1-3):147–167, 2007.
23. M. G. Gouda, E. G. Manning, and Y. Yu. On the progress of communications between two finite state machines. *Information and Control*, 63(3):200–216, 1984.
24. M. Güdemann, G. Salaün, and M. Ouederni. Counterexample guided synthesis of monitors for realizability enforcement. In *ATVA 2012*, pages 238–253, 2012.
25. S. Hallé and T. Bultan. Realizability analysis for message-based interactions using shared-state projections. In *SIGSOFT 2010*, pages 27–36, 2010.
26. A. Heußner, T. L. Gall, and G. Sutre. McScM: A general framework for the verification of communicating machines. In *TACAS 2012*, pages 478–484, 2012.
27. K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP 1998*, pages 122–138, 1998.
28. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL 2008*, pages 273–284, 2008.
29. R. Hu. Distributed programming using Java APIs generated from session types. In *Behavioural Types: from Theory to Tools*. River Publishers, June 2017.
30. R. Hu and N. Yoshida. Hybrid session verification through endpoint API generation. In *FASE 2016*, pages 401–418, 2016.
31. R. Hu and N. Yoshida. Explicit connection actions in multiparty session types. In *FASE 2017*, pages 116–133, 2017.
32. T. B. L. Jespersen, P. Munksgaard, and K. F. Larsen. Session types for Rust. In *WGP@ICFP 2015*, pages 13–22, 2015.
33. KMC tool, 2019. https://bitbucket.org/julien-lange/kmc-cav19.
34. D. Kouzapas, O. Dardha, R. Perera, and S. J. Gay. Typechecking protocols with Mungo and StMungo. In *PPDP 2016*, pages 146–159, 2016.
35. D. Kuske and A. Muscholl. Communicating automata. Available at http://eiche.theoinf.tu-ilmenau.de/kuske/Submitted/cfm-final.pdf, 2014.
36. S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *TACAS 2008*, pages 299–314, 2008.
37. J. Lange, N. Ng, B. Toninho, and N. Yoshida. Fencing off Go: liveness and safety for channel-based programming. In *POPL 2017*, pages 748–761, 2017.
38. J. Lange, N. Ng, B. Toninho, and N. Yoshida. A static verification framework for message passing in Go using behavioural types. In *ICSE 2018*. ACM, 2018.
39. J. Lange, E. Tuosto, and N. Yoshida. From communicating machines to graphical choreographies. In *POPL 2015*, pages 221–232, 2015.
40. J. Lange, E. Tuosto, and N. Yoshida. A tool for choreography-based analysis of message-passing software. In *Behavioural Types: from Theory to Tools*. River Publishers, June 2017.
41. J. Lange and N. Yoshida. Characteristic formulae for session types. In *TACAS 2016*, pages 833–850, 2016.
42. J. Lange and N. Yoshida. On the undecidability of asynchronous session subtyping. In *FOSSACS 2017*, pages 441–457, 2017.
43. J. Lange and N. Yoshida. Verifying asynchronous interactions via communicating session automata. *CoRR*, abs/1901.09606, 2019. Available at https://arxiv.org/abs/1901.09606.
44. S. Lindley and J. G. Morris. Embedding session types in Haskell. In *Haskell 2016*, pages 133–145, 2016.

45. D. Mostrous, N. Yoshida, and K. Honda. Global principal typing in partially commutative asynchronous sessions. In *ESOP 2009*, pages 316–332, 2009.
46. A. Muscholl. Analysis of communicating automata. In *LATA 2010*, pages 50–57, 2010.
47. R. Neykova, L. Bocchi, and N. Yoshida. Timed Runtime Monitoring for Multiparty Conversations. *FAOC*, pages 1–34, 2017.
48. R. Neykova, R. Hu, N. Yoshida, and F. Abdeljallal. A Session Type Provider: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F♯. In *CC 2018*. ACM, 2018.
49. R. Neykova and N. Yoshida. Let It Recover: Multiparty Protocol-Induced Recovery. In *CC 2017*, pages 98–108. ACM, 2017.
50. R. Neykova and N. Yoshida. Multiparty Session Actors. *LMCS*, 13:1–30, 2017.
51. N. Ng and N. Yoshida. Static deadlock detection for concurrent Go by global session graph synthesis. In *CC 2016*, pages 174–184, 2016.
52. N. Ng, N. Yoshida, and K. Honda. Multiparty session C: safe parallel programming with message optimisation. In *TOOLS 2012*, pages 202–218, 2012.
53. Ocean Observatories Initiative. www.oceanobservatories.org.
54. OMG. Business Process Model and Notation, 2018. https://www.omg.org/spec/BPMN/2.0/.
55. D. A. Orchard and N. Yoshida. Effects as sessions, sessions as effects. In *POPL 2016*, pages 568–581, 2016.
56. L. Padovani. A simple library implementation of binary sessions. *J. Funct. Program.*, 27:e4, 2017.
57. W. Peng and S. Purushothaman. Analysis of a class of communicating finite state machines. *Acta Inf.*, 29(6/7):499–522, 1992.
58. R. Perera, J. Lange, and S. J. Gay. Multiparty compatibility for concurrent objects. In *PLACES 2016*, pages 73–82, 2016.
59. Introduction to protocol engineering. Available at http://cs.uccs.edu/~cs522/pe/pe.htm, 2006.
60. G. Salaün, L. Bordeaux, and M. Schaerf. Describing and reasoning on web services using process algebra. *IJBPIM*, 1(2):116–128, 2006.
61. A. Scalas, O. Dardha, R. Hu, and N. Yoshida. A linear decomposition of multiparty sessions for safe distributed programming. In *ECOOP 2017*, pages 24:1–24:31, 2017.
62. A. Scalas and N. Yoshida. Lightweight session programming in scala. In *ECOOP 2016*, pages 21:1–21:28, 2016.
63. A. Scalas and N. Yoshida. Less is more: multiparty session types revisited. *PACMPL*, 3(POPL):30:1–30:29, 2019.
64. Scribble Project homepage, 2018. www.scribble.org.
65. K. C. Sivaramakrishnan, M. Qudeisat, L. Ziarek, K. Nagaraj, and P. Eugster. Efficient sessions. *Sci. Comput. Program.*, 78(2):147–167, 2013.
66. K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *PARLE 1994*, pages 398–413, 1994.
67. R. Taylor, E. Tuosto, N. Walkinshaw, and J. Derrick. Choreography-based analysis of distributed message passing programs. In *PDP 2016*, pages 512–519, 2016.
68. D. M. Yellin and R. E. Strom. Protocol specifications and component adaptors. *ACM Trans. Program. Lang. Syst.*, 19(2):292–333, 1997.