HAL open science

Stability of Asynchronously Communicating Systems

Gwen Salaün, Lina Ye

To cite this version:

Gwen Salaün, Lina Ye. Stability of Asynchronously Communicating Systems. [Research Report] RR-8561, INRIA. 2014. hal-01020777v2

HAL Id: hal-01020777

https://inria.hal.science/hal-01020777v2

Submitted on 20 Oct 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Stability of Asynchronously Communicating Systems

Gwen Salaün, Lina Ye

RESEARCH REPORT
N° 8561
July 2014
Project-Teams Convecs

ISSN 0249-6399 ISRN INRIA/RR--8561--FR+ENG

Stability of Asynchronously Communicating Systems

Gwen Salaün*, Lina Ye†

Project-Teams Convecs

Research Report n° 8561 — version 2 — initial version July 2014 — revised version October 2014 — 26 pages

Abstract: Recent software is mostly constructed by reusing and composing existing components. Software components are usually stateful and therefore described using behavioral models such as finite state machines. Asynchronous communication is a classic interaction mechanism used for such software systems. However, analyzing communicating systems interacting asynchronously via reliable FIFO buffers is an undecidable problem. A typical approach is to check whether the system is bounded, and if not, the corresponding state space can be made finite by limiting the presence of communication cycles in behavioral models or by fixing buffer sizes.
In this paper, our focus is on systems that are likely to be unbounded and therefore result in infinite systems. We do not want to restrict the system by imposing any arbitrary bounds. We introduce a notion of stability and prove that once the system is stable for a specific buffer bound, it remains stable whatever larger bounds are chosen for buffers. This enables us to check certain properties on the system for that bound and to ensure that the system will preserve them whatever larger bounds are used for buffers. We also prove that computing this bound is undecidable but we show how we succeed in computing these bounds for many typical examples using heuristics and equivalence checking.

Key-words: Communicating Systems, Labelled Transition Systems, Unbounded Buffers, Automated Verification

* Grenoble INP, Inria, France
† Inria

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Stabilité des systèmes communicants de façon asynchrone

Résumé : Le logiciel moderne est principalement construit par réutilisation et composition de composants existants. Les composants logiciels sont souvent décrits par des modèles comportementaux tels que des machines à états. La communication asynchrone est un mécanisme de communication classique pour les systèmes logiciels. Cependant, analyser des systèmes communicants intergaissant de façon asynchrone au travers de buffers FIFO est un problème indécidable. Une approche habituelle est de vérifier si le système est borné, et s'il ne l'est pas, l'espace d'états correspondant peut être rendu fini en limitant la présence de cycles de communication dans les modèles ou en fixant une taille arbitraire aux buffers.

Dans ce rapport, nous nous focalisons sur des systèmes non-bornés et donc infinis. Nous ne voulons pas imposer de restriction arbitraire au système. Nous introduisons une notion de stabilité et prouvons que lorsque le système est stable pour une borne spécifique, il reste stable pour des bornes plus élevées. Cela permet de vérifier que certaines propriétés sont satisfaites par le système pour cette borne, et assure que ces propriétés seront préservées pour des tailles supérieures de buffers. Nous avons aussi prouvé que calculer cette borne n'est pas décidable mais nous somme capables en pratique de trouver cette borne pour de nombreux exemples en utilisant des heuristiques et la vérification d'équivalence.

Mots-clés : Systèmes communicants, systèmes de transitions étiquetées, buffers non bornés, vérification automatique

1 Introduction

Most software systems are now constructed by reusing and composing existing components or peers. This is the case in many different areas such as component-based systems, cloud applications, Web services, or cyber-physical systems. Software entities are often stateful and therefore described using behavioral models. Moreover, asynchronous communication via FIFO buffers is a classic communication model used for such distributed, communicating systems. A crucial problem in this context is to check whether a new system consisting of a set of interacting peers respects certain properties. Analyzing asynchronously communicating software has been studied extensively in the last 30 years and is known to be undecidable in general [10]. A common approach to circumvent this issue is to bound the state space by restricting the cyclic behaviors or imposing an arbitrary bound on buffers. Bounding buffers to an arbitrary size during the execution is not a satisfactory solution: if at some point buffers' sizes change (due to changes in memory requirements for example), it is not possible to know how the system would behave compared to its former version and new unexpected errors can show up.

In this paper, we do not want to impose any arbitrary bounds for buffer sizes. Hence, our focus is on systems that are likely to be unbounded and therefore result in infinite state spaces. It was shown recently that certain properties of distributed systems interacting asynchronously through unbounded buffers can be checked using the *synchronizability* property. [4] relies on this property to check whether a choreography is realizable and [39] uses it for verifying the compatibility of communicating systems. A set of peers is synchronizable if and only if the system generates the same sequences of messages under synchronous and unbounded asynchronous communication, considering only the ordering of the send actions and ignoring the ordering of receive actions. Focusing only on send actions makes sense for verification purposes because: (i) send actions are the actions that transfer messages to the network and are therefore observable, (ii) receive actions correspond to local consumptions by peers from their buffers and can therefore be considered to be local and private information. Synchronizability can be verified by checking the equivalence of the synchronous and 1-bounded asynchronous compositions of the given system. In the 1-bounded asynchronous composition, each peer is equipped with one input buffer bounded to size 1. Thus, this property can be verified using equivalence checking techniques on finite state spaces, although the system consisting of peers interacting asynchronously via unbounded buffers can result in infinite state spaces.

Figure 1 gives an example where peers are modelled using Labelled Transition Systems (LTSs). Transitions are labeled with either emissions (exclamation marks) or receptions (question marks). Initial states are marked with incoming half-arrows. This simple system is synchronizable because the synchronous system consists of sequences of interactions request, result, and ack, and this order is the same in the 1-bounded asynchronous system considering only send actions.

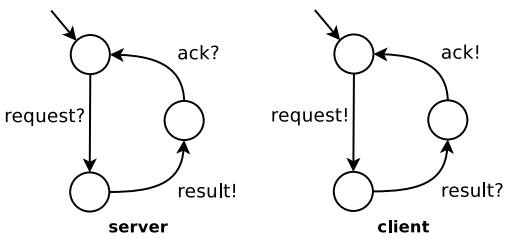

Figure 1: Motivating Example (1)

Let us now focus on a sligthly extended version of this example where in addition the server

stores every request answered in a database (log) as depicted in Figure 2. This system is not synchronizable because the client can submit a second request in the 1-bounded asynchronous system before the server sends the log message to the database, and this behavior is not possible in the synchronous composition. Synchronizability is rather strong and many communicating systems do not satisfy this property. So what can we do for non-synchronizable systems? This second example is typically unbounded, because the database peer has no obligation to consume from its buffer which can infinitely receive log messages. So far, and to the best of our knowledge, existing approaches cannot analyze such infinite-state systems.

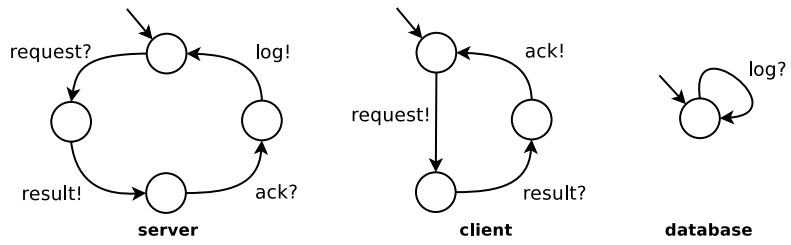

Figure 2: Motivating Example (2)

In this paper, we propose a new approach for analyzing a set of peers described using LTSs, communicating asynchronously via reliable (no loss of messages) and possibly unbounded FIFO buffers. We do not want to restrict the system by imposing any arbitrary bounds on cyclic behaviors or buffers. We introduce a notion of stability for the asynchronous versions of the system. A system is stable if asynchronous compositions exhibit the same observable behavior from some buffer bound. We prove that once the system is stable for a specific buffer bound, it remains stable whatever larger bounds are chosen for buffers. This enables one to check temporal properties on the system for that bound (using model checking techniques for instance) and ensures that the system will preserve them whatever larger bounds are used for buffers. Practically speaking, this allows any developer to check that his/her system satisfies some property for the minimal k satisfying stability, and then (s)he can choose any bound from that k as actual parameter for his/her system. This choice may be guided by memory constraints or performance issues for instance. We also prove that computing this bound is undecidable, but we show how we succeed in computing such bounds in practice for many examples.

Going back to the former example (Fig. 2), we can use our approach to detect that when each peer is equipped with a buffer bound fixed to 2, the observable behavior of this system is stable. This means that we can check properties, such as the absence of deadlocks, on the 2-bounded asynchronous version of the system and the results hold for any asynchronous version of the system where buffer bounds are greater or equal to 2.

We implemented our approach in a tool that first encodes the peer LTSs and their compositions into process algebra, and then uses heuristics, search algorithms, and equivalence checking techniques for verifying whether the system satisfies the stability property. If this is the case, we return the smallest bound respecting this property. Otherwise, when we reach a certain maximal bound, our check returns an inconclusive result. Heuristics and search algorithms aim at guiding the approach towards the smallest bound satisfying stability whereas equivalence checking techniques is used for checking the stability property given a specific bound k. All the steps of our approach are fully automated (no human intervention). We applied our tool support to more than 300 examples of communicating systems, many of them taken from the literature on this topic. These experiments show that a large number of these examples are stable and can therefore be formally analyzed using our approach.

Our contributions with respect to existing results on formal verification of asynchronously communicating systems are the following:

- A general framework for verifying systems interacting asynchronously via reliable FIFO buffers;
- The introduction of the stability property for such systems and a proof showing that stability once acquired for a specific bound k is preserved for upper bounds;
- A proof demonstrating that computing this bound k is undecidable;
- A fully automated tool support that implements the presented approach and was applied to many examples for evaluation purposes.

The organization of the rest of this paper is as follows. Section 2 defines our models for peers and their synchronous/asynchronous compositions. Section 3 presents the stability property and our results on stable systems. Section 4 describes our tool support and experiments we carried out to evaluate our approach. Finally, Section 5 reviews related work and Section 6 concludes.

2 Communicating Systems

We use Labeled Transition Systems (LTSs) for modeling peers. This behavioral model defines the order in which a peer executes the send and receive actions.

Definition 1 *A peer is an LTS $\mathcal{P} = (S, s^0, \Sigma, T)$ where S is a finite set of states, $s^0 \in S$ is the initial state, $\Sigma = \Sigma^! \cup \Sigma^? \cup \{\tau\}$ is a finite alphabet partitioned into a set of send messages, a set of receive messages, and the internal action, and $T \subseteq S \times \Sigma \times S$ is a transition relation.*

We write $m!$ for a send message $m \in \Sigma^!$ and $m?$ for a receive message $m \in \Sigma^?$. We use the symbol τ for representing internal activities. A transition is represented as $s \xrightarrow{l} s' \in T$ where $l \in \Sigma$. This can be directly extended to $s \xrightarrow{\sigma} s'$, $\sigma \in \Sigma^*$, where $\sigma = l_1, ..., l_n$, $s \xrightarrow{l_1} s_1, \ldots, s_i \xrightarrow{l_{i+1}} s_{i+1}, \ldots, s_{n-1} \xrightarrow{l_n} s' \in T$. In the following, for the sake of simplicity, we will denote this by $s \xrightarrow{\sigma} s' \in T^*$.

We assume that peers are deterministic on observable messages meaning that if there are several transitions going out from one peer state, and if all the transition labels are observable, then they are all different from one another. Observable determinism can be easily obtained in peers using standard determinization algorithms [31]. Nondeterminism can also result from internal choices when several transitions (at least two) outgoing from a same state are labeled with τ. It is worth observing that these internal transitions are important when checking the synchronizability property [4, 39], which partly relies on synchronous communication. In such a case, internal transitions outgoing from a same state express an internal choice whereas transitions labeled with observable messages outgoing from a same state stand for an external choice semantics.

Given a set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, we assume that each message has a unique sender and a unique receiver: $\forall i, j \in 1..n$, $i \neq j$, $\Sigma_i^! \cap \Sigma_j^! = \emptyset$ and $\Sigma_i^? \cap \Sigma_j^? = \emptyset$. Furthermore, each message is exchanged between two different peers: $\Sigma_i^! \cap \Sigma_i^? = \emptyset$ for all i. We also assume that each emission has a reception counterpart in another peer (closed systems): $\forall i \in 1..n$, $\forall m \in \Sigma_i^! \implies \exists j \in 1..n$, $i \neq j$, $m \in \Sigma_j^?$.

The synchronous composition of a set of peers corresponds to the system where a communication occurs when one peer can send a message and another peer is in a state in which that message can be received.

Definition 2 (Synchronous Composition) *Given a set of peers* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *with* $\mathcal{P}_i = (S_i, s_i^0, \Sigma_i, T_i)$*, the synchronous composition is the labeled transition system* $LTS_s = (S_s, s_s^0, \Sigma_s, T_s)$ *where:*

- $S_s = S_1 \times \ldots \times S_n$
- $s_s^0 \in S_s$ *such that* $s_s^0 = (s_1^0, \ldots, s_n^0)$
- $\Sigma_s = \cup_i \Sigma_i$
- $T_s \subseteq S_s \times \Sigma_s \times S_s$*, and for* $s = (s_1, \ldots, s_n) \in S_s$ *and* $s' = (s'_1, \ldots, s'_n) \in S_s$

(interact) $s \xrightarrow{m} s' \in T_s$ *if* $\exists i, j \in \{1, \ldots, n\}$ *where* $i \neq j : m \in \Sigma_i^! \cap \Sigma_j^?$ *where* $\exists\ s_i \xrightarrow{m!} s'_i \in T_i$*, and* $s_j \xrightarrow{m?} s'_j \in T_j$ *such that* $\forall k \in \{1, \ldots, n\}, k \neq i \wedge k \neq j \Rightarrow s'_k = s_k$

(internal) $s \xrightarrow{\tau} s' \in T_s$ *if* $\exists i \in \{1, \ldots, n\}$*,* $\exists\ s_i \xrightarrow{\tau} s'_i \in T_i$ *such that* $\forall k \in \{1, \ldots, n\}, k \neq i \Rightarrow s'_k = s_k$

In the asynchronous composition, the peers communicate with each other asynchronously via FIFO buffers. Each peer $\mathcal{P}_i$ is equipped with an unbounded input message buffer Q_i. A peer can either send a message $m \in \Sigma^!$ to the tail of the receiver buffer Q_j at any state where this send message is available, read a message $m \in \Sigma^?$ from its buffer Q_i if the message is available at the buffer head, or evolve independently through an internal transition. We recall that we focus on send actions in this paper. We consider that reading from the buffer is private non-observable information, which is encoded as an internal transition in the asynchronous system.

Definition 3 (Asynchronous Composition) *Given a set of peers* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *with* $\mathcal{P}_i = (S_i, s_i^0, \Sigma_i, T_i)$*, and* Q_i *being its associated buffer, the asynchronous composition is the labeled transition system* $LTS_a = (S_a, s_a^0, \Sigma_a, T_a)$ *where:*

- $S_a \subseteq S_1 \times Q_1 \times \ldots \times S_n \times Q_n$ *where* $\forall i \in \{1, \ldots, n\},\ Q_i \subseteq (\Sigma_i^?)*$
- $s_a^0 \in S_a$ *such that* $s_a^0 = (s_1^0, \epsilon, \ldots, s_n^0, \epsilon)$ *(where* ϵ *denotes an empty buffer)*
- $\Sigma_a = \cup_i \Sigma_i$
- $T_a \subseteq S_a \times \Sigma_a \times S_a$*, and for* $s = (s_1, Q_1, \ldots, s_n, Q_n) \in S_a$ *and* $s' = (s'_1, Q'_1, \ldots s'_n, Q'_n) \in S_a$

(send) $s \xrightarrow{m!} s' \in T_a$ *if* $\exists i, j \in \{1, \ldots, n\}$ *where* $i \neq j : m \in \Sigma_i^! \cap \Sigma_j^?$*, (i)* $s_i \xrightarrow{m!} s'_i \in T_i$*, (ii)* $Q'_j = Q_j m$*, (iii)* $\forall k \in \{1, \ldots, n\} : k \neq j \Rightarrow Q'_k = Q_k$*, and (iv)* $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow s'_k = s_k$

(consume) $s \xrightarrow{\tau} s' \in T_a$ *if* $\exists i \in \{1, \ldots, n\} : m \in \Sigma_i^?$*, (i)* $s_i \xrightarrow{m?} s'_i \in T_i$*, (ii)* $mQ'_i = Q_i$*, (iii)* $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow Q'_k = Q_k$*, and (iv)* $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow s'_k = s_k$

(internal) $s \xrightarrow{\tau} s' \in T_a$ *if* $\exists i \in \{1, \ldots, n\}$*, (i)* $s_i \xrightarrow{\tau} s'_i \in T_i$*, (ii)* $\forall k \in \{1, \ldots, n\} : Q'_k = Q_k$*, and (iii)* $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow s'_k = s_k$

We use $\overline{LTS_a}$ for the same asynchronous composition where the receptions (*consume* rule in Def. 3) are kept in the resulting LTS ($s \xrightarrow{m?} s' \in \overline{T_a}$) instead of being encoded as τ. Furthermore, we use $LTS_a^k = (S_a^k, s_a^0, \Sigma_a^k, T_a^k)$ to define the *bounded* asynchronous composition, where each message buffer is bounded to size k. The definition of LTS_a^k can be obtained from Def. 3 by

allowing send transitions only if the message buffer of the receiving peer has less than k messages in it. Otherwise, the sender is blocked, *i.e.*, we assume reliable communication without message losses. As it is well-known for asynchronously communicating systems, $\forall LTS_a^k$ ($\overline{LTS_a^k}$), we have $LTS_a^k \subseteq LTS_a^q$ ($\forall q \geq k$) ($\overline{LTS_a^k} \subseteq \overline{LTS_a^q}$ ($\forall q \geq k$)).

3 Stability

In this section, we introduce successively the synchronizability property, the stability property, the proof of undecidability for the stability property, and several other results on stable systems.

3.1 Synchronizability

The composition of finite peer LTSs can result into an infinite state system if these peers communicate asynchronously over unbounded buffers. This makes the exhaustive analysis of all executed communication traces impossible and verification tasks in this setting are undecidable [10]. This issue can be avoided for systems that are synchronizable [4]. The synchronizability property checks whether the sequences of send actions generated by the peer composition remain the same under synchronous and asynchronous communication semantics. This enables one to analyze asynchronous systems, even those generating an infinite state space, using the synchronous version of the given system (which has a finite state space). This property was used recently for analyzing properties on communicating systems [39] and is checked using branching equivalence [47], which is the finest equivalence notion in presence of internal behaviors.

Definition 4 (Branching Bisimulation) *Given two LTSs LTS_1 and LTS_2, they are branching bisimilar, denoted by $LTS_1 \equiv_{br} LTS_2$, if there exists a symmetric relation R (called a branching bisimulation) between the states of LTS_1 and LTS_2 satisfying the following two conditions:*

- *The initial states are related by R;*
- *If $R(r, s)$ and $r \xrightarrow{\delta} r\prime$, then either $\delta = \tau$ and $R(r\prime, s)$, or there exists a path $s \xrightarrow{\tau^*} s_1 \xrightarrow{\delta} s\prime$, such that $R(r, s_1)$ and $R(r\prime, s\prime)$.*

For the sake of simplicity and consistency, the relation R is replaced by $\equiv_{br}$ in the rest of this paper, *i.e.*,, $R(r, s)$ is noted $r \equiv_{br} s$.

Definition 5 (Branching Synchronizability) *A set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ is branching synchronizable if $\forall k \geq 1$, $LTS_s \equiv_{br} LTS_a^k$, where $\equiv_{br}$ in that particular case compares synchronizations in the synchronous composition with emissions from peers to peer buffers in the asynchronous composition.*

It was proved that $LTS_s \equiv_{br} LTS_a^1$ is a sufficient and necessary condition for branching synchronizability [39].

The synchronizability property is quite strong and there are many non-synchronizable systems, see *e.g.*, Fig. 2. If the set of peers is not synchronizable, there is no standard approach for formally analyzing those systems. The stability notion we introduce in the next section is a solution to this problem.

3.2 Stability

In this section, we show that systems consisting of a finite set of peers involving cyclic behaviors and communicating over FIFO buffers may stabilize from a specific buffer bound k. We call this property *stability* and we say that the corresponding systems are *stable*. The class of systems which are stable corresponds to systems whose asynchronous compositions remain the same from some buffer bound when we observe send actions only (we ignore receive actions and buffer contents). Stability results do not hold if we consider that receptions are also observable. Focusing only on send actions makes sense for verification purposes because: (i) send actions are the actions that transfer messages to the network and are therefore observable, (ii) receive actions correspond to local consumptions by peers from their buffers and can therefore be considered to be local and private information.

Since stable systems produce the same behavior from a specific bound k, they can be analyzed for that bound to detect for instance the presence of deadlocks or to check whether they satisfy any kind of temporal properties. Stability ensures that these properties will be also satisfied for larger bounds. The stability definition relies on branching equivalence checking, and this equivalence preserves properties written with ACTL\X logic [38].

Contrarily, some systems are not stable. Unstability is due to some particular cycle dependency in the peer behaviors that makes the asynchronous composition exhibit new behaviors every time the buffer bound is increased. In the rest of this section, we will first define the stability property. We will show how stability is established and what can be deduced for such systems. We will also present other results such as the undecidability of the stability problem.

Definition 6 (Stability) *Given a set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, we say that this system is stable if and only if $\exists k$ such that $LTS_a^k \equiv_{br} LTS_a^q$ $(\forall q > k)$.*

Before presenting our main results, we first introduce a few definitions that will be used in the following deduction. First, we define a special state of LTS_a^k as a *border state* that is considered as a boundary between LTS_a^k and LTS_a^{k+1} if the latter has more behaviors than the former due to larger buffers. Precisely, a border state of LTS_a^k should satisfy a condition stating that it has a successor state in LTS_a^{k+1} that is not reachable in LTS_a^k due to limited buffer size. Then the new behavior, which leads to a state whose buffers contain $k+1$ messages that are not contained in LTS_a^k, must begin after this border state.

Definition 7 (Border State) *Given a set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, given a state $s^k = (s_1^k, Q_1^k, ..., s_n^k, Q_n^k) \in S_a^k$, if it satisfies the following condition, we call it a k-border state, simply a border state if there is no ambiguity:*

- *$\exists s^k \xrightarrow{m!} s''^k \in T_a^{k+1}$, where s''^k has one peer whose buffer contains $k+1$ messages.*

A single k-message state is a state where only one buffer contains k messages. Note that such a state cannot be reachable in LTS_a^{k-1} and we must have such a state as a successor state for a $(k-1)$-border state in LTS_a^k according to the condition of Definition 7. Characterizing such states is a key step for demonstrating some of our results, *i.e.,* Lemma 1 and Theorem 1.

Definition 8 (Single k-message State $(SG(s^k))$) *Given a set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ and a state $s^k = (s_1^k, Q_1^k, ..., s_n^k, Q_n^k) \in S_a^k$, if $\exists i \in \{1, ..., n\}, |Q_i^k| = k$, $\forall j \in \{1, ..., n\}, j \neq i, |Q_j^k| < k$, and there exists a border state s^{k-1} such that $s^{k-1} \xrightarrow{m!} s^k \in T_a^k, m? \in \Sigma_i^k$, then s^k is a single k-message state, denoted by $SG(s^k)$.*

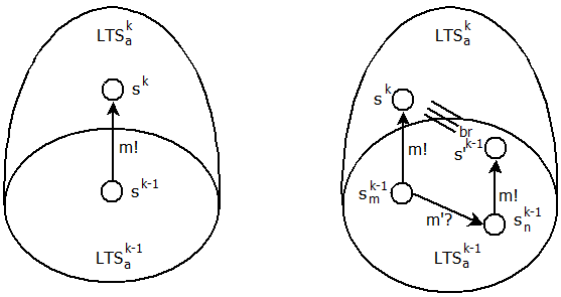

Figure 3: A single k-message state (left part) and a loose k-message state (right part).

In the left part of Figure 3, the small ellipsis represents LTS_a^{k-1} and the bigger one is LTS_a^k. Here, $s^k \in S_a^k$ is a single k-message state and $s^{k-1} \in S_a^{k-1}$ is the corresponding border state.

Next we define a special state s^k in LTS_a^k such that it contains only one buffer with k messages, whose precedent state is a border state, denoted by s_b. Furthermore, from s_b, there are two outgoing transitions. One can reach s^k with one emission. The other can reach another state s^{k-1} through a reception followed by the same emission such that s^{k-1} contains at most $k-1$ messages in its buffers. Then we will show $s^k \equiv_{br} s^{k-1}$ by demonstrating that these two traces diverging from s_b will converge in the future, *i.e.,* there will form a confluent diamond.

Definition 9 (Loose k-message State $(LS(s^k))$**)** *Given a set of peers* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *and a state* s^k *such that* $SG(s^k)$, i.e., $\exists s_m^{k-1} \xrightarrow{m!} s^k \in T_a^k$, $m? \in \Sigma_i^?, i \in \{1, ..., n\}$, *where* s_m^{k-1} *is a border state, then* s^k *is called a loose k-message state, denoted by* $LS(s^k)$, *if the following conditions are satisfied*

- $\exists m\prime? \in \Sigma_i^?$, *such that* $s_m^{k-1} \xrightarrow{m\prime?} s_n^{k-1} \xrightarrow{m!} s\prime^{k-1}$, *where* $s_m^{k-1} \xrightarrow{m\prime?} s_n^{k-1}, s_n^{k-1} \xrightarrow{m!} s\prime^{k-1} \in T_a^k$;
- *for* s^k *and* $s\prime^{k-1}$, *we have* $s^k \equiv_{br} s\prime^{k-1}$

In the right part of Figure 3, $s^k \in S_a^k$ is a loose k-message state which is a single k-message state satisfying the above condition. Next we will show that for a given single k-message state, it has a corresponding branching equivalent state in LTS_a^{k-1} iff it is a loose k-message state. The proof is based on the fact that branching equivalence between two states $s^k \in S_a^k$ and $s^{k+1} \in S_a^{k+1} \wedge s^{k+1} \notin S_a^k$ implies a confluent diamond for them in LTS_a^{k+1}.

Lemma 1 *Given a set of peers* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, *the following holds for each single k-message state* $s^k \in S_a^k$.

$$LS(s^k) \Leftrightarrow \exists s^{k-1} \in S_a^{k-1} \text{ such that } s^{k-1} \equiv_{br} s^k$$

Proof .
$\Rightarrow$ Suppose that for a single k-message state $s^k = (s_1^k, Q_1^k, ..., s_n^k, Q_n^k) \in S_a^k, |Q_i^k| = k, i \in \{1, ..., n\}$, $LS(s^k)$ holds. Then in LTS_a^k, $\exists m? \in \Sigma_i^?$, such that $s_0^k \xrightarrow{\sigma} s_m^k \xrightarrow{m!} s^k$, where $s_0^k \xrightarrow{\sigma} s_m^k \in T_a^{k*}, s_m^k \xrightarrow{m!} s^k \in T_a^k$. From Definition 9, we know that s_m^k is a border state and that $\exists m\prime? \in \Sigma_i^?$, $s_0^k \xrightarrow{\sigma} s_m^k \xrightarrow{m\prime?} s_n^k \xrightarrow{m!} s\prime^k$, where $s_m^k \xrightarrow{m\prime?} s_n^k, s_n^k \xrightarrow{m!} s\prime^k \in T_a^k$. This implies that there must be a corresponding part in LTS_a^{k-1}: $s_0^{k-1} \xrightarrow{\sigma} s_m^{k-1} \xrightarrow{m\prime?} s_n^{k-1} \xrightarrow{m!} s\prime^{k-1}$, where $s_0^{k-1} \xrightarrow{\sigma} s_m^{k-1} \in T_a^{k-1*}, s_m^{k-1} \xrightarrow{m\prime?} s_n^{k-1}, s_n^{k-1} \xrightarrow{m!} s\prime^{k-1} \in T_a^{k-1}$, $s\prime^{k-1} = (s_1\prime^{k-1}, Q_1\prime^{k-1}, ..., s_n\prime^{k-1}, Q_n\prime^{k-1}), |Q_j\prime^{k-1}| <$

$k, \forall j \in \{1, ..., n\}$. The reason is that the maximum number of messages in all states between s_0^{k-1} and s'^{k-1} is smaller than k. Now from the second condition of Definition 9, we know that beginning from the state s^k in LTS_a^k and from s'^{k-1} in LTS_a^{k-1}, we have exactly the same branching structure. This implies $s^k \equiv_{br} s'^{k-1}$.
$\Leftarrow$ Now suppose that in LTS_a^k, we have a single k-message state $s^k = (s_1^k, Q_1^k, ..., s_n^k, Q_n^k) \in S_a^k, |Q_i^k| = k, i \in \{1, ..., n\}$, such that $LS(s^k)$ does not hold. From Definition 9, we show that the violation of each one of the two conditions implies $\nexists s^{k-1} \in S_a^{k-1}$, such that $s^k \equiv_{br} s^{k-1}$.

1. If the first condition is not satisfied, *i.e.,* $\nexists m\prime? \in \Sigma_i^?$, such that $s_m^k \xrightarrow{m\prime?} s_n^k \xrightarrow{m!} s'^k$, where $s_m^k \xrightarrow{m\prime?} s_n^k, s_n^k \xrightarrow{m!} s'^k \in T_a^k$. This implies that the reception of $m\prime$ that is contained in the buffer Q_i^k cannot be executed between the border state s_m^k and the state juste before $m!$. If $m\prime?$ is executed after $m!$ preceding the border state, then there is no equivalent state in LTS_a^{k-1} with respect to s^k since the part after the border state cannot be reached in LTS_a^{k-1}. Otherwise, if $m\prime?$ is before the border state, *e.g.,* suppose $s_h^k \xrightarrow{l!/?} s_m^k \xrightarrow{m!} s^k$, where $s_h^k \xrightarrow{l!/?} s_m^k, s_m^k \xrightarrow{m!} s^k \in T_a^k$, we have $s_h^k \xrightarrow{m\prime?} s_p^k \xrightarrow{l!/?} s_q^k \xrightarrow{m!} s_r^k$, where $s_h^k \xrightarrow{m\prime?} s_p^k, s_p^k \xrightarrow{l!/?} s_q^k, s_q^k \xrightarrow{m!} s_r^k \in T_a^k$. In this case, we can deduce that $m\prime?$ should execute before $l!/?$. In other words, $\nexists s^k \xrightarrow{m\prime?} s \in T_a^k$. This means that we cannot reach s_r^k from s^k, *i.e.,* there is no confluent diamond. Thus, there is no branching equivalent state in LTS_a^{k-1} for s^k.

2. $\exists m\prime? \in \Sigma_g^?$, $s_m^k \xrightarrow{m\prime?} s_n^k \xrightarrow{m!} s'^k$, where $s_m^k \xrightarrow{m\prime?} s_n^k, s_n^k \xrightarrow{m!} s'^k \in T_a^k$, $s^k \not\equiv_{br} s'^k$. It is clear that we have also $s^k \not\equiv_{br} s_n^k$ since $m!$ is before s^k but after s_n^k. Thus, there is no branching equivalent state in LTS_a^{k-1} for the state s^k.

From above, if a single k-message state is not a loose one, then $\nexists s^{k-1} \in S_a^{k-1}$ such that $s^{k-1} \equiv_{br} s^k$.

Thus, we have proved this lemma for both directions.

■

Here is one of the main results of this paper. If we are able to find a bound k such that the k-bounded and the $(k+1)$-bounded asynchronous systems are branching equivalent, then we prove that the system remains stable, meaning that the observable behavior is always the same for any bound greater than k. The main interest of the stability property is that the k-bounded version of the system can be analyzed using existing model checking tools and this result ensures that these properties are preserved when buffer bounds are increased or if buffers are unbounded.

Theorem 1 *Given a set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, if $\exists k \in \mathbb{N}$, such that $LTS_a^k \equiv_{br} LTS_a^{k+1}$, then we have $LTS_a^k \equiv_{br} LTS_a^q, \forall q > k$.*

Proof .
We prove this theorem by induction. The idea is to show $LTS_a^k \equiv_{br} LTS_a^q, \forall q > k$ from $LTS_a^k \equiv_{br} LTS_a^{k+1}$. We rewrite $q > k$ by $q = k + m, m \geq 1$. Now the base case $m = 1$ is true from the hypothesis, *i.e.,* $LTS_a^k \equiv_{br} LTS_a^{k+1}$. To prove that this is also true for $\forall m > 1$, we suppose $LTS_a^k \equiv_{br} LTS_a^{k+1} \equiv_{br} ... \equiv_{br} LTS_a^{k+m}, m \geq 1$ and we show $LTS_a^k \equiv_{br} LTS_a^{k+m+1}$. To do this, we consider two cases according to whether the maximum number of messages in all buffers is smaller than $k + m + 1$ or not in the asynchronous composition, which are formally represented as follows:

1. In LTS_a^{k+m+1}, $\forall s^{k+m+1} = (s_1^{k+m+1}, Q_1^{k+m+1}, ..., s_n^{k+m+1}, Q_n^{k+m+1}) \in S_a^{k+m+1}$, we have $\forall i \in \{1, ..., n\}$, $|Q_i^{k+m+1}| \leq k + m$;

2. In LTS_a^{k+m+1}, $\exists s^{k+m+1} = (s_1^{k+m+1}, Q_1^{k+m+1}, ..., s_n^{k+m+1}, Q_n^{k+m+1}) \in S_a^{k+m+1}$, such that $\exists i \in \{1, ..., n\}$, $|Q_i^{k+m+1}| = k + m + 1$.

Now we show that in each of the two cases, we can deduce $LTS_a^k \equiv_{br} LTS_a^{k+m+1}$.

- The first case implies that in the asynchronous composition of this set of peers LTS_a, the buffer of any peer contains at most $k+m$ messages in all states of LTS_a. Hence, LTS_a^{k+m+1} is exactly the same LTS as LTS_a^{k+m}. This implies $LTS_a^{k+m+1} \equiv_{br} LTS_a^{k+m}$. From the inductive hypothesis $LTS_a^k \equiv_{br} LTS_a^{k+m}$, we have $LTS_a^k \equiv_{br} LTS_a^{k+m+1}$.

- In the second case, since there exists a buffer containing $k + m + 1$ messages, there are new states in LTS_a^{k+m+1} that are not in LTS_a^{k+m}. Now from the inductive hypothesis, we have $LTS_a^{k+m-1} \equiv_{br} LTS_a^{k+m}$. Then from Lemma 1, we can deduce that for each single $(k + m)$-message state s^{k+m} in LTS_a^{k+m}, we have $LS(s^{k+m})$, *i.e.,* $\exists s^{k+m-1} \in S_a^{k+m-1}$, such that $s^{k+m-1} \equiv_{br} s^{k+m}$. From Definition 9, we know that the only difference between the path from s_a^0 to s^{k+m-1} and that from s_a^0 to s^{k+m} is that the former has one more reception that has however no impact on the execution after s^{k+m-1} since $s^{k+m-1} \equiv_{br} s^{k+m}$. This implies that we have exactly the same structure with respect to emissions from the states s^{k+m-1} and s^{k+m}. Now suppose that $LTS_a^{k+m} \not\equiv_{br} LTS_a^{k+m+1}$, *i.e.,* $\exists s^{k+m+1} \in S_a^{k+m+1}$, for which $\nexists s^{k+m} \in S_a^{k+m}$, such that $s^{k+m+1} \equiv_{br} s^{k+m}$. In other words, we have $SG(s^{k+m+1})$ and $\neg LS(s^{k+m+1})$. Now let s^{k+m} be the border state for s^{k+m+1}, we can deduce that if $\neg LS(s^{k+m+1})$, for its border state s^{k+m}, $\nexists s^{k+m-1} \in S_a^{k+m-1}$, such that $s^{k+m} \equiv_{br} s^{k+m-1}$. In other words, $\neg LS(s^{k+m+1})$ implies $\neg LS(s^{k+m})$. It follows that $LTS_a^{k+m-1} \not\equiv_{br} LTS_a^{k+m}$, which contradicts our inductive hypothesis. Thus we must have $LTS_a^{k+m} \equiv_{br} LTS_a^{k+m+1}$. From above, based on $LTS_a^k \equiv_{br} ... \equiv_{br} LTS^{k+m}$, we prove $LTS_a^{k+m} \equiv_{br} LTS_a^{k+m+1}$.

This proves the theorem.

■

Given this result, one can check on the minimal k-bounded system satisfying stability, any temporal property written with ACTL\X logic involving emissions, and this property will hold for any system with larger bounds.

Proposition 1 *Given a set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, if $\exists k$ s.t. $LTS_a^k \equiv_{br} LTS_a^q$ $(\forall q > k)$, and for some property P, $LTS_a^k \models P$, then $LTS_a^q \models P$ $(\forall q > k)$.*

Proof .
This trivially follows from Theorem 1.

■

3.3 Undecidability

Before showing the undecidability of checking the stability of a system, we first present a sufficient and necessary condition for a system that is stable.

Lemma 2 *Given a set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, this system is stable iff the following condition is satisfied, denoted by $\mathcal{C}$:*

$$\exists k \in \mathbb{N}, \forall s^k \in S_a^k.[\neg SG(s^k) \vee LS(s^k)].$$

Proof .
$\Rightarrow$ Suppose that the given set of peers is stable. Next we show that the stability of the set of peers implies that the condition $\mathcal{C}$ is satisfied. Since this set of peers is stable, from Theorem 1, we deduce $\exists k \in \mathbb{N}$, such that $LTS_a^k \equiv_{br} LTS_a^{k-1}$. There are two cases for this:

1. $LTS_a^k = (S_a^k, s_a^0, \Sigma_a^k, T_a^k)$ and $LTS_a^{k-1} = (S_a^{k-1}, s_a^0, \Sigma_a^{k-1}, T_a^{k-1})$ have exactly the same structure, *i.e.,* $S_a^k = S_a^{k-1}, \Sigma_a^k = \Sigma_a^{k-1}, T_a^k = T_a^{k-1}$;
2. LTS_a^k and LTS_a^{k-1} do not have the same structure but are branching equivalent, *i.e.,* $S_a^k \neq S_a^{k-1}$ and $T_a^k \neq T_a^{k-1}$.

We demonstrate that any one of the two cases must satisfy the condition $\mathcal{C}$.

- In the first case, we can deduce that in LTS_a^k, for each peer, its buffer contains at most $k-1$ messages in all states, *i.e.,* $\forall s^k = (s_1^k, Q_1^k, ..., s_n^k, Q_n^k) \in S_a^k, \forall i \in \{1, ..., n\}$, $|Q_i^k| < k$. This implies that there is no single k-message state in LTS_a^k, *i.e.,* $\forall s^k \in S_a^k, \neg SG(s^k)$. Hence, the condition $\mathcal{C}$ is satisfied.
- Now consider the second case, where LTS_a^k and LTS_a^{k-1} are not exactly the same one. Hence, we can deduce $\exists s^k = (s_1^k, Q_1^k, ..., s_n^k, Q_n^k) \in S_a^k, \exists i \in \{1, ..., n\}$, $|Q_i^k| = k$ in LTS_a^k. In other words, there exists at least one single k-message state. Furthermore, since $LTS_a^k \equiv_{br} LTS_a^{k-1}$, for all states in LTS_a^k, there must be a corresponding branching equivalent state in LTS_a^{k-1}, including all single k-message states. Thus, from Lemma 1, we can deduce that $\forall s^k \in S_a^k$, we have either $\neg SG(s^k)$ or $LS(s^k)$. This means that the condition $\mathcal{C}$ is satisfied.

So if the set of peers is stable, then the condition $\mathcal{C}$ must be satisfied.
$\Rightarrow$ Now suppose that the condition $\mathcal{C}$ is satisfied. Thus, we consider three cases: all states are not single k-message states, formally described by the first case in the following; all states are loose k-message states, which is not possible since the initial state has no message in all buffers; some of the states are single k-messages states that should be loose ones, formally presented by the second case below:

1. $\exists k \in \mathbb{N}, \forall s^k \in S_a^k$, we have $\neg SG(s^k)$;
2. $\exists k \in \mathbb{N}, \forall s^k \in S_a^k$ such that $SG(s^k)$, we have $LS(s^k)$.

Now we show that any one of these two conditions implies that the system is stable.

- The first condition means that in LTS_a^k, there is no single k-message state, *i.e.,* the maximum number of messages in buffers is smaller than k for any state. It can thus be deduced that LTS_a^k and LTS_a^{k-1} have exactly the same structure, which implies $LTS_a^k \equiv_{br} LTS_a^{k-1}$. From Theorem 1, this is a stable set of peers.
- The second condition means that all single k-message state in LTS_a^k are loose ones. From Lemma 1, it can be deduced that $\forall s^k \in S_a^k$, there is a corresponding state $s^{k-1} \in S_a^{k-1}$, such that $s^k \equiv_{br} s^{k-1}$. It is clear that $\forall s^k \in S_a^{k-1}$, we have at least one corresponding branching equivalent state in LTS_a^k. This implies that $LTS_a^k \equiv_{br} LTS_a^{k-1}$. Hence, this set of peers is stable from Theorem 1.

Hence, if the condition $\mathcal{C}$ is satisfied, then the system is stable.

■

Next we show that checking the stability of the system is undecidable by showing that verifying the condition $\mathcal{C}$ is an undecidable problem. Yet there are many cases in which stability is satisfied and the corresponding bound can be computed in those cases using heuristics, search algorithms, and equivalence checking (see Section 4).

Theorem 2 *Given a set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, it is undecidable whether the corresponding asynchronous system is stable.*

Proof .
From Lemma 2, we know that $\mathcal{C}$ is a sufficient and necessary condition to check that a system is stable. We show that the problem of checking this condition is undecidable. The idea is to demonstrate that the boundedness problem [10] is its subproblem. The communication of a set of peers with FIFO buffers is said to be bounded iff $\exists k \in \mathbb{N}$ such that for any reachable state $s = (s_1, Q_1, ..., s_n, Q_n) \in S_a$, we have $\forall i \in \{1, ..., n\}, |Q_i| < k$. Now recall that to check whether the condition $\mathcal{C}$ is violated, the following two conditions should be both checked one after another:

- $\forall k \in \mathbb{N}, \exists s^k \in S_a^k$, such that $SG(s^k)$;
- $\neg LS(s^k)$.

Now we show that the first condition is equivalent to the boundedness problem. If this condition is satisfied, it can be deduced that $\nexists k \in \mathbb{N}$, such that $\forall s = (s_1, Q_1, ..., s_n, Q_n) \in S_a$, we have $\forall i \in \{1, ..., n\}, |Q_i| < k$. So this system is not bounded. On the other hand, if the condition is not satisfied, *i.e.,* $\exists k \in \mathbb{N}$, there does not exist a reachable state such that it is a single k-message state. This means that $\forall s^k = (s_1^k, Q_1^k, ..., s_n^k, Q_n^k) \in S_a^k$, we have $\neg SG(s^k)$. From Definition 8, to have $\neg SG(s^k)$, consider three cases for s^k: 1) at least two buffers contain k messages, *i.e.,* $\exists i_1, ..., i_m \in \{1, ..., n\}, |Q_{i_j}^k| = k, j \in \{1, ..., m\}, m \geq 2$; 2) no buffer contains k messages, *i.e.,* $\forall i \in \{1, ..., n\}, |Q_i^k| < k$; 3) there does not exist a border state s^{k-1} such that $s^{k-1} \xrightarrow{m!} s^k \in T_a^k$, where $SG(s^k)$. For the first case, if this is true, there must be at least one s^k such that $SG(s^k)$ since to reach a state where more than one buffer has k messages, we have to first reach such a state s^k, $SG(s^k)$. This implies that the first case is not possible with the assumption without single k-message states. Then the second case means that the maximum number of messages in buffers is smaller than k. And for the third case, even in s^k, there exists a buffer containing k messages, this state cannot be reached in LTS_a^k since there is no border state before it. From above, we analyze all three cases for a state s^k such that $\neg SG(s^k)$. We can see that $\neg SG(s^k)$ implies that s^k is bounded by k. This implies that if $\forall s^k \in S_a^k$, we have $\neg SG(s^k)$, then the system is bounded. So we have shown that this first condition is equivalent to the boundedness problem, which is thus a subproblem of checking $\mathcal{C}$: consequently, if the former is undecidable, which was proved in [10], then so is the latter.

■

3.4 Other Results

In the rest of this section, we present several additional results on the stability of communicating systems.

Proposition 2 *A set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ is stable if it is synchronizable.*

Proof .
From Definition 5, a system is synchronizable meaning that all its asynchronous compositions are branching equivalent whatever is the bound for buffers. This means that $\forall k, q \in \mathbb{N}$, we have $LTS_a^k \equiv_{br} LTS_a^q$. Now from Definition 6, it follows that this system is stable.

■

The opposite is false because stability ensures branching equivalence of the asynchronous systems from a specific bound k, which means that systems for lower bounds and for its synchronous version behave differently.

The next result concerns computability of the smallest buffer bound for stable systems.

Proposition 3 *Given a stable set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, the minimal bound k satisfying the stability property is computable.*

Proof .
Since this set of peers is stable, from Theorem 1, we know that $\exists k \in \mathbb{N}, LTS_a^k \equiv_{br} LTS_a^{k+1}$. Since k is finite, the construction of LTS_a^k, LTS_a^{k+1} and their equivalence checking is computable. Hence, the minimal bound k for the stability property is also computable.

■

The computability of the stability property will be illustrated on real-world examples in Section 4 by using heuristics, search algorithms, and equivalence checking.

Beyond these results on stability, a common approach is to identify classes of systems where stability is guaranteed. Synchronizable systems (Prop. 3) and *finite systems* lie in this category. By finite systems, we mean systems where peers interact over unbounded FIFO buffers and whose asynchronous compositions are finite. This is the case for instance for acyclic peers, for systems with a single cyclic peer where cycles do not involve only emissions, or for systems with peers exhibiting *independent* cycles, *i.e.*, cycles that are not included one from another [34]. Apart from these simple cases, two candidates are (i) systems where peers exhibit cycles of emissions, and (ii) *well-formed systems* (Definition 10).

Let us start with systems where peers can infinitely send messages. We establish that the existence of a cycle of emissions in one peer does not determine whether whether the system is stable or not, which is however a sufficient condition for unboundedness [10].

Proposition 4 *Given a set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, if a peer $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ exhibits a cycle of emissions $\phi = s_i^k \xrightarrow{m_1!} s_i^{k+1} \ldots \xrightarrow{m_n!} s_i^k$, this does not imply that the system is unstable.*

Proof .
We prove this result by demonstrating that a system with a cycle of emissions in one peer that is able to infinitely send messages may be stable or not. Suppose first that for this system, in LTS_a, $\forall \sigma \in \Sigma_a^*, s_a^0 \xrightarrow{\sigma} s^{n-1} \xrightarrow{m_i!} s^n, s_a^0 \xrightarrow{\sigma} s^{n-1} \in T_a^*, s^{n-1} \xrightarrow{m_i!} s^n \in T_a$, where $m_i!$ is one send message in ϕ, then $\exists \sigma\prime \in \Sigma_a^*, s_a^0 \xrightarrow{\sigma\prime} s^{m-1} \xrightarrow{m_i!} s^m \xrightarrow{m_i?} s^{m+1}, s_a^0 \xrightarrow{\sigma\prime} s^{m-1} \in T_a^*$, $s^{m-1} \xrightarrow{m_i!} s^m, s^m \xrightarrow{m_i?} s^{m+1} \in T_a$, where $P_!(\sigma) = P_!(\sigma\prime)$. Here we denote the projection of σ on send messages by $P_!(\sigma)$. This means that whenever a message in this cycle is emitted, there exists at least one execution with the same sequence of emissions, where the corresponding peer is ready to receive it. In this case, the system may be stable.
Now suppose that in this system, $\exists \sigma \in \Sigma_a^*, s_a^0 \xrightarrow{\sigma} s^{n-1} \xrightarrow{m_i!} s^n, s_a^0 \xrightarrow{\sigma} s^{n-1} \in T_a^*, s^{n-1} \xrightarrow{m_i!} s^n \in T_a$, where $m_i!$ is one send message in ϕ, for which $\nexists \sigma\prime \in \Sigma_a^*, s_a^0 \xrightarrow{\sigma\prime} s^{m-1} \xrightarrow{m_i!} s^m \xrightarrow{m_i?} s^{m+1}$, such that $s_a^0 \xrightarrow{\sigma\prime} s^{m-1} \in T_a^*, s^{m-1} \xrightarrow{m_i!} s^m, s^m \xrightarrow{m_i?} s^{m+1} \in T_a$, where $P_!(\sigma) = P_!(\sigma\prime)$. This means that $\exists \sigma\prime\prime \in \Sigma_a^*$, $s_a^0 \xrightarrow{\sigma\prime} s^{m-1} \xrightarrow{m_i!} s^m \xrightarrow{\sigma\prime\prime} s^t \xrightarrow{m_i?} s^{t+1}$, where $s_a^0 \xrightarrow{\sigma\prime} s^{m-1}, s^m \xrightarrow{\sigma\prime\prime} s^t \in T_a^*$ and $s^{m-1} \xrightarrow{m_i!} s^m, s^t \xrightarrow{m_i?} s^{t+1} \in T_a$. Next we show that there is at least one send message in the sequence $\sigma\prime\prime$. This can be proved by assuming that all messages in $\sigma\prime\prime$ are receive messages, which in turn implies $\exists s_a^0 \xrightarrow{\sigma\prime} s^m \xrightarrow{m_i!} s^n \xrightarrow{m_i?} s^{n+1}$ and thus a contradiction. So it is clear that in $\sigma\prime\prime$, we must have at least one send message. This implies that $\forall k \in \mathbb{N}$, in LTS_a^{k+1}, $\exists s^0 \xrightarrow{\sigma} s^m \xrightarrow{m_i!} s^n \xrightarrow{m_{(i+1) \ mod \ n}!} s^{n+1} \ldots \xrightarrow{m_{(i+j) \ mod \ n}!} s^{n+j}$, whose corresponding part in LTS_a^k is $s^0 \xrightarrow{\sigma} s^m \xrightarrow{m_i!} s^n \xrightarrow{m_{(i+1) \ mod \ n}!} s^{n+1} \ldots \xrightarrow{m_{(i+j-1) \ mod \ n}!} s^{n+j-1}$. This implies $LTS_a^k \not\equiv_{br} LTS_a^{k+1}$ and the system is thus not stable.

■

The example shown in Figure 4 has a cycle of emissions. In its synchronous composition, before entering this cycle, $d!$ should be synchronized with $d?$ only after $c!$ while $c!$ can only be synchronized with $c?$ after $d!$, which leads to a deadlock. It follows that this system is not synchronizable due to this deadlock, which is however not the case for its asynchronous composition. This means that there is no branching equivalence between the synchronous and asynchronous compositions. However, it is stable from bound 1, *i.e.,* $LTS_a^1 \equiv_{br} LTS_a^k, k > 1$.

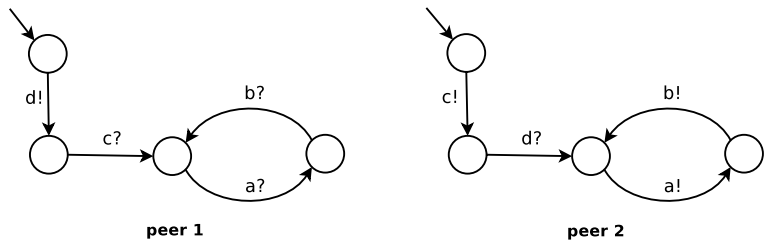

Figure 4: A Stable System Exhibiting a Cycle of Emissions

Figure 5 depicts another example with a cycle of emissions that is not stable. For this example, if we first execute $a!$ and $b!$ before $c!$, then to read the receive message from the buffer of peer 2, we have to wait for $c!$. Hence, consider the execution where c is still not sent. We have that $\forall k$, LTS_a^{k+1} always exhibits a new observable behavior compared to LTS_a^k. The reason is that the cycle of emissions in peer 1 can infinitely send a and b, which is only constrained by the size of buffer in peer 2. This implies that this system is not stable.

Another result concerns well-formed systems [4]. A system consisting of a set of peers is well-formed iff whenever the size of the buffer, Q_i, of the i-th peer is non-empty, the system can move to a state where Q_i is empty. In other words, well-formedness concerns the ability of a system to eventually consume all messages in any of its buffers. In order to check this property, we have to keep receive messages and thus analyze it on $\overline{LTS_a}$ instead of LTS_a .

Definition 10 (Well-formed System) *Given a set of peers* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$*, it is well-formed, denoted by* $WF(\overline{LTS_a})$*, if* $\forall s = (s_1, Q_1, ..., s_n, Q_n) \in \overline{S_a}, \forall Q_i$*, such that* $|Q_i| > 0$*, then* $\exists s \xrightarrow{\sigma} s\prime \in \overline{T_a}^*$*, where* $s\prime = (s_1\prime, Q_1\prime, ..., s_n\prime, Q_n\prime) \in \overline{S_a}, |Q_i\prime| = 0$.

Proposition 5 *The well-formedness of a set of peers* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *can be checked with the following CTL temporal formula on its asynchronous composition* $\overline{LTS_a}$*:*

$$AG(|Q_i| > 0) \Rightarrow EF(|Q_i| = 0)$$

This formula means that whenever the size of any buffer is non-empty, *i.e.,* $AG(|Q_i| > 0)$, then there is at least one execution in the system to move to a state where Q_i is empty, *i.e.,* $EF(|Q_i| = 0)$. We show that a well-formed system is not mandatorily stable, *i.e.*, well-formedness is not a sufficient condition for stability.

Proposition 6 *Given a set of peers* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$*, its well-formedness does not imply that this set of peers is stable.*

Proof .
This can be proved by demonstrating that a well-formed system can be unstable. We illustrate this through the example depicted in Figure 5. Suppose that one peer in the system contains a cycle $\phi = s_i^k \xrightarrow{\sigma} s_i^k$, where $\sigma \in \Sigma_a^+$ and σ contains at least one send message. In this example, one

such cycle is $\sigma = a!.b!$ in peer 1. Furthermore, there is at least one receive message corresponding to one send message in ϕ, denoted by $m_k?$, in another peer with a preceding send message $m_t!$, such as $a?$ or $b?$ in peer 2, that is the reception of the emission $a!$ or $b!$ in peer 1, with a preceding emission $c!$. Suppose further that the execution of the cycle ϕ is totally independent of the send message $m_t!$, *i.e.*, ϕ can continue to execute regardless of the emission of m_t, and that there is no constraint on $m_t!$, *i.e.*, m_t can be emitted at any moment. This is the case for our example, where the cycle $a!.b!$ is independent of $c!$ whose execution is however possible at any moment. For such a system, we can deduce that it is not stable due to the cycle of emissions in peer 1 that can infinitely send messages, which generates new observable behaviors in LTS_a^{k+1} compared to LTS_a^k, $\forall k \in \mathbb{N}$, when $c!$ is still not executed. Furthermore, this system is well-formed because the execution of $c!$ is possible at any moment in peer 2, then all messages in its buffer can be eventually read (cycle of receptions) until the buffer is empty.

■

Although well-formedness does not imply stability, one can check whether a stable system is well-formed for the smallest k satisfying stability for instance. Then, we show below that if a system is both stable and well-formed for some k, it remains well-formed for larger bound q greater than k.

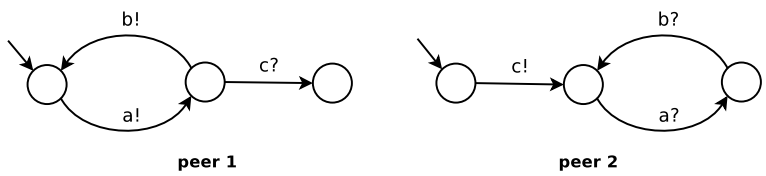

Figure 5: A Well-formed yet Unstable System

Theorem 3 *Given a set of peers* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$*, if* $\exists k$ *s.t.* $LTS_a^k \equiv_{br} LTS_a^q$ $(\forall q > k)$ *and* $WF(\overline{LTS_a^k})$*, then we have* $WF(\overline{LTS_a^q})$ $(\forall q > k)$*.*

Proof .
Suppose that for the given system, we have $LTS_a^k \equiv_{br} LTS_a^{k+1} \wedge WF(\overline{LTS_a^k})$. Its stability ensures that $\forall s^k \in S_a^k$, $\exists s^{k+1} \in S_a^{k+1}$ such that $s^k \equiv_{br} s^{k+1}$, and the inverse is also true. Furthermore, the well-formedness of $\overline{LTS_a^k}$ means that $\forall s^k \in \overline{S_a^k}$, $\forall Q_i^k \in s^k$, $|Q_i^k| > 0$, there must exist another state $s\prime^k$ reachable from s^k in $\overline{LTS_a^k}$ such that all messages in this buffer are consumed, *i.e.,* $Q_i^k\prime \in s\prime^k$, $|Q_i^k\prime| = 0$. Now there are two possible cases for $\forall s^{k+1} \in \overline{S_a^{k+1}}$: 1) $\nexists Q_i^{k+1} \in s^{k+1}$, $|Q_i^{k+1}| = k+1$; 2) $\exists Q_i^{k+1} \in s^{k+1}$, $|Q_i^{k+1}| = k+1$. For the first case, s^{k+1} must also exist in LTS_a^k. Hence, from $WF(\overline{LTS_a^k})$, for each non-empty buffer in s^{k+1}, there must be another reachable state where this buffer becomes empty. Consider the second case, where we have at least one buffer containing $k+1$ messages. Since $LTS_a^k \equiv_{br} LTS_a^{k+1}$, then $\exists s^k \in S_a^k$, such that $s^k \equiv_{br} s^{k+1}$. From Lemma 1, we have $LS(s^{k+1})$, *i.e.*, it satisfies the two conditions described in Definition 9. From the first condition meaning the existence of a confluent diamond, it can be deduced that from such a s^{k+1}, we can reach a state that is also contained in $\overline{LTS_a^k}$. Hence, with $WF(\overline{LTS_a^k})$, it follows that for s^{k+1} in the second case, for each non-empty buffer, we can find another reachable state where this buffer is empty. Thus, we have $WF(\overline{LTS_a^{k+1}})$. Now from $LTS_a^k \equiv_{br} LTS_a^q$ $(\forall q > k)$, we have $LTS_a^{k+1} \equiv_{br} LTS_a^{k+2}$. Furthermore, we have proved $WF(\overline{LTS_a^{k+1}})$. Thus, similarly, we can prove $WF(\overline{LTS_a^{k+2}})$ and also $WF(\overline{LTS_a^q})$, $(\forall q > k)$.

■

Finally, we show that the stability results cannot be used for checking properties involving not only emissions but also receptions.

Proposition 7 *Given a set of peers* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$*, if this system is stable, i.e.,* $\exists k$ *s.t.* $\overline{LTS_a^k} \equiv_{br} \overline{LTS_a^q}$ $(\forall q > k)$*, and for some property* P*,* $\overline{LTS_a^k} \models P$*, then we do not have necessarily* $\overline{LTS_a^q} \models P$ $(\forall q > k)$*.*

Proof .
Consider the example in Figure 2. As explained in Section 1, this system is stable from $k = 2$. Suppose now that the property under consideration is that it is forbidden to have three successive receptions of *log*. This property is verified on $\overline{LTS_a^2}$ but not on $\overline{LTS_a^3}$, because to have three successive receptions of *log*, this peer requires a buffer with size at least three. Therefore, the property is satisfied for $\overline{LTS_a^2}$, but is violated for $\overline{LTS_a^3}$.

■

4 Tool Support

In this section, we first present our method for checking whether a set of LTSs is stable. Our approach relies on an encoding into process algebra on the one hand, which is used for computing peer synchronous and asynchronous compositions, and on heuristics and search algorithms for computing the smallest bound satisfying the stability property on the other hand. These different aspects will be introduced in this section as well. Finally, we will comment on experiments we made either on real-world examples taken from the literature or on hand-crafted examples to see how our approach scales.

4.1 Method

First of all, we introduce some methodological aspects that put in practice the results on systems stability we have presented in the former section. Figure 6 overviews the main steps of the overall approach. Given a set of peer LTSs, we first check whether this system is branching synchronizable [39]. If this is the case, it means that the observable behavior for the synchronous and asynchronous composition always remains the same whatever buffer size is chosen. Therefore, in such a case, the synchronous product can be used for analysis purposes.

If the set of peers is not synchronizable, we compute an initial bound k using one of the heuristics we will introduce below (Section 4.3). For that bound, we verify whether the k-bounded asynchronous system is branching equivalent to the $(k+1)$-bounded system. If it is the case, the system is stable for bound k. Then, we decrement k for searching the smallest k satisfying the stability property. Thus, we can analyze properties of interest for that system (buffers bounded to the smallest k) and, if they are satisfied, we proved that these properties are preserved for any upper bound.

If for a specific k, the equivalence check between the k-bounded asynchronous system and the $(k+1)$-bounded system returns false, we modify k (using, *e.g.*, binary search or incremental update of the former bound) and apply the check again. We repeat the process up to a certain arbitrary bound *kmax* that makes the approach abort if attained. If we reach that bound, the test is inconclusive. The only solution in that situation is to bound arbitrarily the system and analyze the system for that specific bound.

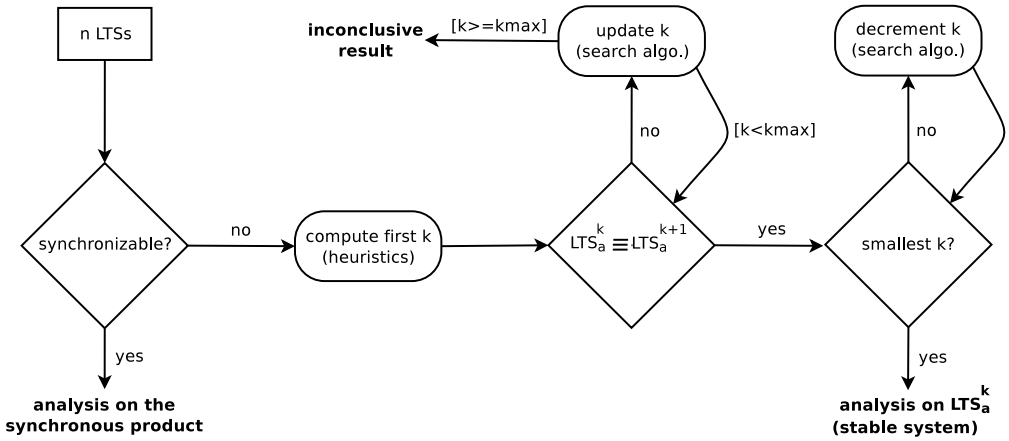

Figure 6: Methodological Aspects

4.2 Process Algebra Encoding

In order to automate these different checks, we use an encoding into process algebra and equivalence checking. In particular, we encode the set of peer LTSs into the LNT process algebra [18], one of the input languages of the CADP toolbox [26]. Each peer LTS is represented in LNT using several processes. Each state in the peer LTS is encoded by one process, whose body corresponds to a choice between all the possible transitions outgoing from that state. Each branch of this choice first executes the action labelling the corresponding transition (message emission or reception, or internal action) followed by a call to the process encoding the target state of that transition. We also define in LNT FIFO buffers and classic operations to manipulate them. Finally, synchronous and bounded asynchronous compositions are generated using parallel composition and hiding operators. More precisely, in the asynchronous composition case, all peers and buffers are composed in parallel. Each peer reads from its buffer and sends messages to other peer buffers. Receptions corresponding to message consumptions for peers from their buffers are hidden.

In a second step, from this encoding, we use CADP compilers and exploration tools to generate synchronous and asynchronous composition LTSs. These LTSs are finally compared with CADP equivalence checking tools [7] for checking the stability property. The whole process is fully automated thanks to some Python scripts we wrote and SVL verification scripts [25] that are automatically generated. Python scripts are particularly used for generating the LNT code and for searching the smallest bound k satisfying stability. SVL scripts automate the compilation from LNT to LTSs and call the equivalence checking tools.

4.3 Heuristics and Search Algorithms

In our experimental results, we use five strategies for searching the smallest bound k from which the set of peers interacting asynchronously become stable. Since the calculation of this bound is undecidable, all computations are bound by an arbitrary upper bound *kmax*. Each strategy consists of the computation of an initial bound k and an algorithm calculating the next bound to attempt.

- Our first strategy (#1) is *brute force*, *i.e.*, we start from bound k equal to one and we increment it by one until obtaining a positive result for the equivalence check or reaching

kmax.

- A second strategy (#2) computes the longest sequence of emissions in all peer LTSs, then starts from this number and uses a binary search algorithm. For example, if the stability check for the first k is true (false, resp.), the next k to explore is midway between 1 and k (between k and *kmax*, resp.).

  The intuition behind the longest sequence of emissions is that in that case all peers can at least send all their messages even if no peer consumes any message from its buffer. Experiments show that in most cases, the searched bound k is greater than or equal to this computed size.

- A third strategy (#3) uses again the longest sequence of emissions for the initial k, but then progresses by incrementing or decrementing the bound till reaching *kmax* or the smallest k satisfying stability. Precisely, if the stability check for the first k is true (false, resp.), the next bound to explore is $k - 1$ ($k + 1$, resp.).

  The problem of the longest sequence of emissions is when several peers have the same partner as recipient of their messages. In such a case, the partner buffer can get full even if the sending peers still want to emit other messages to it.

- A fourth strategy (#4) computes the maximum between the longest sequence of emissions in all peers and the highest number of emissions destinated to a same peer, and then uses the binary search algorithm (as for #2) for computing the next bounds. The second part of the maximum function corresponds to the case in which the buffer size prevents some peers to send messages because that buffer is already full. This case is somehow the counterpart of the longest sequence of emissions but from the receiver point of view.

- A fifth strategy (#5) uses the same initial k computation as presented for strategy #4, and then increments or decrements the bound till completion of the process as in strategy #3.

In the rest of this section, for the sake of evaluation of our approach, we will present experimental results and use them for comparing the strategies introduced above.

4.4 Experimental Results

We used a Mac OS laptop running on a 2.3 GHz Intel Core i7 processor with 16 GB of Memory and carried out experiments on more than 300 examples of communicating systems. Table 1 presents experimental results for some real-world examples. We also present at the end of the table a few hand-crafted examples for showing how our approach scales (the examples from the literature are quite small). The table gives for each example the number of peers (P), the total number of states (S) and transitions (T) involved in these peers, the bound k if the system is stable (0 if the system is synchronizable and *kmax* if this upper bound is reached during the analysis process), the size of the k-bounded asynchronous system (minimized modulo branching reduction), and the time for applying the whole process for each strategy introduced beforehand. In the LTS_a^k column, we give the size of the 1-bounded asynchronous composition for synchronizable systems and the size of the *kmax*-bounded asynchronous composition for non-stable systems. During our experiments, we used a bound *kmax* arbitrarily fixed to 10, which enabled us to keep computation times within a few hours during our experiments.

All the examples included in the table involve cyclic behaviors. The examples from the literature correspond to quite simple systems (up to 5 peers). 10 examples are synchronizable, 13 examples are stable, and 4 examples are not stable. To sum up, out of the 27 real-world

Id	Description	|P|	$\|S\|/\|T\|$	k	LTS_a^k $\|S\|/\|T\|$	Time (in seconds)				
						#1	#2	#3	#4	#5
(1)	Estelle specification [32]	2	7/9	5	28/54	126	216	149	209	120
(2)	News server [39]	2	9/9	3	14/22	89	180	65	173	85
(3)	Client/server [10]	2	6/10	0	3/4	34				
(4)	CFSM system [32]	2	6/7	*kmax*	393/802	222	107	213	103	212
(5)	Promela program (1) [33]	2	6/6	1	3/4	52	71	67	68	66
(6)	Promela program (2) [34]	2	8/8	*kmax*	275/616	219	107	231	103	228
(7)	Web Services [23]	3	13/12	0	7/7	44				
(8)	Trade system [22]	3	12/12	0	30/46	44				
(9)	Online stock broker [24]	3	13/16	*kmax*	197/452	>1h	222	>1h	223	>1h
(10)	FTP Transfer [9]	3	20/17	2	15/19	91	224	155	215	155
(11)	Client/Server [15]	3	14/13	0	8/7	44				
(12)	Mars Explorer [12]	3	34/34	2	21/25	93	176	142	170	140
(13)	Online Computer Sale [19]	3	26/26	0	11/12	69				
(14)	E-museum [16]	3	33/40	3	27/46	146	>1h	138	243	182
(15)	Client/Supplier [14]	3	31/33	0	17/19	44				
(16)	Restaurant Service [46]	3	15/16	1	10/12	68				
(17)	Travel Agency [44]	3	32/38	0	18/21	44				
(18)	Vending Machine [27]	3	15/14	0	8/8	44				
(19)	Travel Agency [5]	3	42/57	3	29/42	118	>1h	113	>1h	112
(20)	Train station [43]	4	18/18	2	19/26	114	195	137	197	165
(21)	Factory job manager [13]	4	20/20	0	12/15	54				
(22)	Bug report repository [28]	4	12/12	1	7/8	85	221	137	227	136
(23)	Cloud application [30]	4	8/10	*kmax*	26,754/83,200	352	208	339	208	337
(24)	Sanitary agency [42]	4	35/41	3	44/71	144	196	137	196	137
(25)	SQL Server [41]	4	32/38	2	22/31	165	195	137	199	170
(26)	SSH Protocol [36]	4	26/28	0	16/18	97				
(27)	Booking system [37]	5	45/53	1	27/35	179	285	165	>1h	>1h
(28)	Hand-crafted example	5	396/801	4	17,376/86,345	227	>1h	184	313	189
(29)	——	6	16/18	5	202/559	278	641	188	641	188
(30)	——	7	38/38	6	1,716/6,468	363	763	391	767	393
(31)	——	10	48/47	8	14,904/57,600	624	800	294	804	294
(32)	——	14	85/80	4	19,840/113,520	506	1,449	483	1,442	485
(33)	——	16	106/102	3	22,400/132,400	478	1,620	454	1,621	453
(34)	——	20	128/116	4	80,640/522,480	728	2,194	698	2,183	699

Table 1: Experimental results

examples presented in the top part of the table, 23 can be now analyzed using existing verification techniques thanks to the approach proposed in this report. For these real-world examples, LTSs are quite small and computation times reasonable (up to a few minutes). Computation times may be less satisfactory for non-stable systems whose intermediate state spaces grow gradually with the size of the buffer bounds, until we stop once *kmax* is reached, see online stock broker for instance (9), which takes time because intermediate state spaces for certain couples *(peer,buffer)* contains millions of states and transitions. However, in other cases the computation time remains reasonable, see, *e.g.*, CFSM system (4), Promela program (6), or cloud application (23). It is worth observing that some examples that are presented as unbounded and then impossible to analyze in the literature are actually stable, see, *e.g.*, Estelle specification (1), which means that they can be analyzed for the minimal k satisfying stability and these properties still hold for larger buffer bounds.

As far as hand-crafted examples are concerned, we observe that the LTS sizes and computation times increase mainly with the size of the peer LTSs, the number of peers, and the size of buffer bounds. Yet we are able to check stable systems involving a large number of peers in a reasonable

time (slightly more than 10 minutes for the last example with 20 peers using #1, #3, or #5). This time remains quite low for peers exhibiting many states and transitions (about 3 minutes for the first hand-crafted example (28) using #1, #3, or #5).

Now let us compare the five strategies we used in our experiments. It turns out that strategy #2 and #4 are not very efficient in terms of performance for two main reasons. First, if the equivalence check for the initial k returns false, the binary search algorithm computes a second value for k, which can be quite high. This results in calculating asynchronous systems with large buffer bounds, which requires more computation time than for lower k values as computed with the other strategies. Second, in some cases, binary search takes time before converging to the result, and this causes additional equivalence computations. The advantage of binary search is that for non-stable systems, k increases quite fast and quickly reaches $kmax$, see CFSM system (4) for instance.

Strategies #3 and #5 are much better than strategies #2 and #4 but also better in most cases than strategy #1. This gain is not clear for small examples and examples requiring a small buffer bound, but it becomes obvious for examples involving more peers and for those requiring larger buffer bounds. These two strategies (#3 and #5) are particularly satisfactory when the initial k finely approximates the expected result. In such cases, we can have up to a factor 2 gain in terms of computation time, see the fourth hand crafted example (31) for illustration purposes.

5 Related Work

One of the first results on analyzing communicating state machines was proposed by Brand and Zafiropulo [10]. They defined the unspecified receptions compatibility notion for interaction protocols described using Communicating Finite State Machines (CFSMs). This work focuses on the verification of n interacting processes executed in parallel and exchanging messages via FIFO buffers. When considering unbounded buffers, the authors showed that the resulting state spaces may be infinite, and the problem becomes undecidable. Gouda *et al.* [29] presents sufficient conditions to compute a bound k from which two finite state machines communicating through 1-directional channels are guaranteed to progress indefinitely. Jeron and Jard [32] propose a sufficient condition for testing unboundedness, which can be used as a decision procedure in order to check reachability for CFSMs.

Abdulla *et al.* [1] propose some verification techniques for CFSMs. They present a method for performing symbolic forward analysis of unbounded *lossy* channel systems. In [33], the authors present an incomplete boundedness test for communication channels in Promela and UML RT models. They also provide a method to derive *upper bound* estimates for the maximal occupancy of each individual message buffer. Cécé and Finkel [17] focus on the analysis of infinite half-duplex systems and present several (un)decidability results. For instance, they prove that a symbolic representation of the reachability set is computable in polynomial time and show how to use this result to solve several verification problems. Recently, Beohar and Cuijpers [6] studied sufficient and necessary conditions for desynchronizability modulo branching bisimulation. This property ensures that the asynchronous version of the system where processes interact via half-duplex buffers do not introduce spurious interleavings compared to its synchronous version. Poizat *et al.* [40] propose the notions of bounded analysis and bounded decomposition for Symbolic Transition Systems, which can be used to test boundedness of systems and to generate finite simulations for them so that standard model-checking techniques may be applied for verification purposes.

Darondeau *et al.* [20] identify a decidable class of systems consisting of non-deterministic

communicating processes that can be scheduled while ensuring boundedness of buffers. [21] proposed a causal chain analysis to determine upper bounds on buffer sizes for multi-party sessions with asynchronous communication. Bouajjani and Emmi [8] consider a bounded analysis for message-passing programs, which does not limit the number of communicating processes nor the buffers' size. However, they limit the number of communication cycles. They propose a decision procedure for reachability analysis when programs can be sequentialized. By doing so, program analysis can easily scale while previous related techniques quickly explode. Recently, Ouederni *et al.* [39] presented a sufficient condition for checking compatibility of a set of asynchronously communicating components with unbounded message buffers. They exploit the synchronizability property introduced in [4], but if systems are non-synchronizable, they cannot be analyzed with this approach.

Communicating systems have been intensively studied in the context of choreographies and realizability/conformance checking in the last 10 years. In [11], the authors tackle the choreography conformance issue from a theoretical point of view, and propose notions of contract refinement and choreography conformance for services that communicate through message queues. [28] proposes techniques to check whether a set of peers interacting asynchronously can realize a choreography with finite buffers, and if so, for what buffer sizes. [35] studies several notions of realizability and investigates decidability results for choreographies involving services interacting via buffers, which do not assume that messages arrive in the same order in which they have been sent. As mentioned earlier in this paper, Basu *et al.* proposed to check choreography conformance and realizability verifying the synchronizability property [4]. The realizability problem for Message Sequence Charts (MSCs) has also been studied, see, *e.g.*, [45, 2]. [2] for instance presents some decidability results on bounded MSC graphs, which are graphs obtained from MSCs using bounded buffers.

Compared to all these results, we do not impose any bound on the number of peers, cycles, or buffer bounds. Another main difference is that we do not want to ensure or check boundedness of the systems under analysis. Contrarily, we are particularly interested in unbounded (yet possibly stable) systems. We also go one step farther than synchronizable systems, since we propose verification techniques for systems that do not satisfy this property.

In [3], the authors work on a similar problem and propose an algorithm for identifying if the interactions of a given system with unbounded queues can be mimicked by the same system where queues are bounded. This paper assumes that determining the existence of a k verifying this property is computable, whereas we proved in this paper that this problem is undecidable. Moreover, [3] relies on trace equivalence and LTL logic. Trace equivalence is not adequate for analyzing communicating systems since this equivalence does not preserve important properties such as deadlock-freeness. We consider a finest notion of equivalence in this paper (branching), which allows one to check properties written with ACTL\X logic [38]. Last, the tool support provided in [3] does not provide any result (infinite loop, inconclusive result, or error) for more than half of the examples presented in Table 1.

6 Conclusion

We have presented in this paper a framework for formally analyzing systems communicating via (possibly unbounded) FIFO buffers. This work focuses on cyclic systems modelled using finite state machines. We have introduced the stability property, which shows that several systems become stable from a specific buffer bound k when focusing on send messages. The stability problem is undecidable in the general case, but for many systems we can determine whether those systems are stable using heuristics, search algorithms, and branching equivalence checking.

Experiments showed that many real-world examples satisfy this stability property and this can be identified in a reasonable time. Model checking techniques can then be used on the asynchronous version of the system with buffers bound to the smallest k satisfying stability. If a stable system satisfies a specific property for that k, the property will be satisfied too if buffer bounds are increased or if buffers are unbounded.

Acknowledgements. This work has been supported by the OpenCloudware project (2012-2015), which is funded by the French *Fonds national pour la Société Numérique* (FSN), and is supported by *Pôles* Minalogic, Systematic, and SCS. We would like to thank Radu Mateescu for his valuable suggestions to improve the paper.

References

[1] P. A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-Fly Analysis of Systems with Unbounded, Lossy FIFO Channels. In *Proc. CAV'98*, volume 1427 of *LNCS*, pages 305–318. Springer, 1998.

[2] R. Alur, K. Etessami, and M. Yannakakis. Realizability and Verification of MSC Graphs. *Theoretical Computer Science*, 331(1):97–114, 2005.

[3] S. Basu and T. Bultan. Automatic Verification of Interactions in Asynchronous Systems with Unbounded Buffers. In *Proc. of ASE'14*, pages 743–754, 2014.

[4] S. Basu, T. Bultan, and M. Ouederni. Deciding Choreography Realizability. In *Proc. of POPL'12*, pages 191–202. ACM, 2012.

[5] A. Bennaceur, C. Chilton, M. Isberner, and B. Jonsson. Automated Mediator Synthesis: Combining Behavioural and Ontological Reasoning. In *Proc. of SEFM'13*, volume 8137 of *LNCS*, pages 274–288. Springer, 2013.

[6] Harsh Beohar and Pieter J. L. Cuijpers. Avoiding Diamonds in Desynchronisation. *Sci. Comput. Program.*, 91:45–69, 2014.

[7] D. Bergamini, N. Descoubes, C. Joubert, and R. Mateescu. BISIMULATOR: A Modular Tool for On-the-Fly Equivalence Checking. In *Proc. of TACAS'05*, volume 3440 of *LNCS*, pages 581–585. Springer, 2005.

[8] A. Bouajjani and M. Emmi. Bounded Phase Analysis of Message-Passing Programs. In *Proc. of TACAS'12*, volume 7214 of *LNCS*, pages 451–465. Springer, 2012.

[9] A. Bracciali, A. Brogi, and C. Canal. A Formal Approach to Component Adaptation. *Journal of Software Systems*, 74(1):45–54, 2005.

[10] D. Brand and P. Zafiropulo. On Communicating Finite-State Machines. *J. ACM*, 30(2):323–342, 1983.

[11] M. Bravetti and G. Zavattaro. Contract Compliance and Choreography Conformance in the Presence of Message Queues. In *Proc. of WS-FM'08*, LNCS, pages 37–54, 2009.

[12] A. Brogi and R. Popescu. Automated Generation of BPEL Adapters. In *Proc. of ICSOC'06*, volume 4294 of *LNCS*, pages 27–39. Springer, 2006.

[13] T. Bultan, C. Ferguson, and X. Fu. A Tool for Choreography Analysis Using Collaboration Diagrams. In *Proc. of ICWS'09*, pages 856–863. IEEE, 2009.

[14] J. Cámara, J. Antonio Martín, G. Salaün, C. Canal, and E. Pimentel. Semi-Automatic Specification of Behavioural Service Adaptation Contracts. *Electr. Notes Theor. Comput. Sci.*, 264(1):19–34, 2010.

[15] C. Canal, P. Poizat, and G. Salaün. Synchronizing Behavioural Mismatch in Software Composition. In *Proc. of FMOODS'06*, volume 4037 of *LNCS*, pages 63–77. Springer, 2006.

[16] C. Canal, P. Poizat, and G. Salaün. Model-Based Adaptation of Behavioural Mismatching Components. *IEEE Transactions on Software Engineering*, 34(4):546–563, 2008.

[17] G. Cécé and A. Finkel. Verification of Programs with Half-duplex Communication. *Inf. Comput.*, 202(2):166–190, 2005.

[18] D. Champelovier, X. Clerc, H. Garavel, Y. Guerte, V. Powazny, F. Lang, W. Serwe, and G. Smeding. Reference Manual of the LOTOS NT to LOTOS Translator (Version 5.4). INRIA/VASY, 149 pages, 2011.

[19] J. Cubo, G. Salaün, C. Canal, E. Pimentel, and P. Poizat. A Model-Based Approach to the Verification and Adaptation of WF/.NET Components. In *Proc. of FACS'07*, volume 215 of *ENTCS*, pages 39–55. Elsevier, 2007.

[20] P. Darondeau, B. Genest, P. S. Thiagarajan, and S. Yang. Quasi-Static Scheduling of Communicating Tasks. In *Proc. of CONCUR'08*, volume 5201 of *LNCS*, pages 310–324. Springer, 2008.

[21] P.-M. Deniélou and N. Yoshida. Buffered Communication Analysis in Distributed Multiparty Sessions. In *Proc. CONCUR'10*, volume 6269 of *LNCS*, pages 343–357. Springer, 2010.

[22] P.-M. Deniélou and N. Yoshida. Multiparty Session Types Meet Communicating Automata. In *Proc. of ESOP'12*, volume 7211 of *LNCS*, pages 194–213. Springer, 2012.

[23] X. Fu, T. Bultan, and J. Su. Analysis of Interacting BPEL Web Services. In *Proc. of WWW'04*, pages 621–630. ACM Press, 2004.

[24] X. Fu, T. Bultan, and J. Su. Conversation Protocols: A Formalism for Specification and Verification of Reactive Electronic Services. *Theoretical Computer Science*, 328(1-2):19–37, 2004.

[25] H. Garavel and F. Lang. SVL: A Scripting Language for Compositional Verification. In *Proc. FORTE'01*, pages 377–394. Kluwer, 2001.

[26] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2010: A Toolbox for the Construction and Analysis of Distributed Processes. In *Proc. of TACAS'11*, volume 6605 of *LNCS*, pages 372–387. Springer, 2011.

[27] C. Gierds, A. J. Mooij, and K. Wolf. Reducing Adapter Synthesis to Controller Synthesis. *IEEE T. Services Computing*, 5(1):72–85, 2012.

[28] G. Gössler and G. Salaün. Realizability of Choreographies for Services Interacting Asynchronously. In *Proc. of FACS'11*, volume 7253 of *LNCS*, pages 151–167. Springer, 2011.

[29] M. G. Gouda, E. G. Manning, and Y.-T. Yu. On the Progress of Communications between Two Finite State Machines. *Information and Control*, 63(3):200–216, 1984.

[30] M. Güdemann, G. Salaün, and M. Ouederni. Counterexample Guided Synthesis of Monitors for Realizability Enforcement. In *Proc. of ATVA'12*, volume 7561 of *LNCS*, pages 238–253. Springer, 2012.

[31] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979.

[32] T. Jéron and C. Jard. Testing for Unboundedness of FIFO Channels. *Theor. Comput. Sci.*, 113(1):93–117, 1993.

[33] S. Leue, R. Mayr, and W. Wei. A Scalable Incomplete Test for Message Buffer Overflow in Promela Models. In *Proc. SPIN'04*, volume 2989 of *LNCS*, pages 216–233. Springer, 2004.

[34] S. Leue, A. Stefanescu, and W. Wei. Dependency Analysis for Control Flow Cycles in Reactive Communicating Processes. In *Proc. of SPIN'08*, volume 5156 of *LNCS*, pages 176–195. Springer, 2008.

[35] N. Lohmann and K. Wolf. Decidability Results for Choreography Realization. In *Proc. of ICSOC'11*, volume 7084 of *LNCS*, pages 92–107. Springer, 2011.

[36] J. A. Martín and E. Pimentel. Contracts for Security Adaptation. *J. Log. Algebr. Program.*, 80(3-5):154–179, 2011.

[37] R. Mateescu, P. Poizat, and G. Salaün. Adaptation of Service Protocols Using Process Algebra and On-the-Fly Reduction Techniques. In *Proc. of ICSOC'08*, volume 5364 of *LNCS*, pages 84–99. Springer, 2008.

[38] R. De Nicola and F. W. Vaandrager. Action versus State Based Logics for Transition Systems. In *Semantics of Concurrency*, volume 469 of *LNCS*, pages 407–419. Springer, 1990.

[39] M. Ouederni, G. Salaün, and T. Bultan. Compatibility Checking for Asynchronously Communicating Software. In *Proc. of FACS'13*, volume 8348 of *LNCS*, pages 310–328. Springer, 2013.

[40] P. Poizat, J.-C. Royer, and G. Salaün. Bounded Analysis and Decomposition for Behavioural Descriptions of Components. In *Proc. of FMOODS'06*, volume 4037 of *LNCS*, pages 33–47. Springer, 2006.

[41] P. Poizat and G. Salaün. Adaptation of Open Component-based Systems. In *Proc. of FMOODS'07*, volume 4468 of *LNCS*, pages 141–156. Springer, 2007.

[42] G. Salaün, L. Bordeaux, and M. Schaerf. Describing and Reasoning on Web Services using Process Algebra. In *Proc. of ICWS'04*, pages 43–50. IEEE Computer Society, 2004.

[43] G. Salaün, T. Bultan, and N. Roohi. Realizability of Choreographies Using Process Algebra Encodings. *IEEE Transactions on Services Computing*, 5(3):290–304, 2012.

[44] R. Seguel, R. Eshuis, and P. W. P. J. Grefen. Generating Minimal Protocol Adaptors for Loosely Coupled Services. In *Proc. of ICWS'10*, pages 417–424. IEEE Computer Society, 2010.

[45] S. Uchitel, J. Kramer, and J. Magee. Incremental Elaboration of Scenario-based Specifications and Behavior Models using Implied Scenarios. *ACM Transactions on Software Engineering and Methodology*, 13(1):37–85, 2004.

[46] W. M. P. van der Aalst, A. J. Mooij, C. Stahl, and K. Wolf. Service Interaction: Patterns, Formalization, and Analysis. In *Proc. of SFM'09*, volume 5569 of *LNCS*, pages 42–88. Springer, 2009.

[47] R. J. van Glabbeek and W. P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *J. ACM*, 43(3):555–600, 1996.

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399