Form Methods Syst Des
DOI 10.1007/s10703-017-0285-8

Automated verification of automata communicating via FIFO and bag buffers

Lakhdar Akroun[1] · **Gwen Salaün**[1]

Abstract This article presents new results for the automated verification of automata communicating asynchronously via FIFO or bag buffers. The analysis of such systems is possible by comparing bounded asynchronous compositions using equivalence checking. When the composition exhibits the same behavior for a specific buffer bound, the behavior remains the same for larger bounds. This enables one to check temporal properties on the system for that bound and this ensures that the system will preserve them whatever larger bounds are used for buffers. In this article, we present several decidability results and a semi-algorithm for this problem considering FIFO and bag buffers, respectively, as communication model. We also study various equivalence notions used for comparing the bounded asynchronous systems.

Keywords Labeled transition systems · Asynchronous communication · Equivalence checking

1 Introduction

Most software systems are now constructed by reusing and composing existing components or peers. This is the case in many areas such as component-based systems, cloud applications, Web services, or cyber-physical systems. We particularly focus on software entities that are described using behavioral models and exchange messages using asynchronous communication semantics. One of the main problems in this context is to check whether a new system consisting of a set of interacting peers respects certain properties. Verifying asynchronously communicating software has been studied extensively in the last 30 years and is known to be undecidable for FIFO buffers communication in general [1]. A common approach to circumvent this issue is to bound the state space by restricting the cyclic behaviors or by assigning an

✉ Gwen Salaün
Gwen.Salaun@inria.fr

[1] Inria, LIG, CNRS, University of Grenoble Alpes, Grenoble, France

Published online: 20 June 2017

arbitrary bound on buffers. In this article, we do not want to restrict the system by imposing any bound on cyclic behaviors or buffers. Bounding buffers to an arbitrary size during the execution is not a satisfactory solution, because, if at some point buffers' sizes vary (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.

A recent approach [2,3] shows that, when considering FIFO buffers as communication model, in some cases, asynchronous compositions exhibit the same observable behavior from some buffer bound. This property, called *stability*, can be verified in practice using equivalence checking techniques on finite state spaces by comparing bounded asynchronous compositions, although the system consisting of peers interacting asynchronously via unbounded buffers can result in infinite state spaces. This enables one to check temporal properties on the system for that bound (using model checking techniques) and ensures that the system will preserve them whatever larger bounds are used for buffers.

Figure 1 gives an example where peers are modeled using Labeled Transition Systems (LTSs). Transitions are labeled with either send messages (exclamation marks) or receive messages (question marks). Initial states are marked with incoming half-arrows. In the asynchronous composition, each peer is equipped with one input buffer. We can use this approach to detect that when each peer is equipped with a buffer bound fixed to 2, the observable behavior of the system depicted in Fig. 1 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.

This approach is very promising yet suffers a few limitations in its current version because: (i) it does not consider receive messages in the peer behavioral model but only focuses on send messages, (ii) it reasons on a unique communication model where each peer is equipped with one input FIFO buffer, (iii) it relies on specific behavioral semantics when comparing asynchronous compositions, namely trace equivalence in [2] and branching bisimulation in [3].

In this article, we extend these early results in several directions. First, we consider both send and receive messages in models and model comparisons. This is particularly interesting for allowing one to specify and verify properties not only on send messages but also on receive messages. An example of property of interest is to check whether all requests submitted by a client are actually handled by the server as tackled in [4]. Second, we assume that peers can communicate either via FIFO buffers or via bag buffers. Bag buffers are particularly meaningful when the communication network cannot assume proper ordering of messages (a message sent after another one can arrive before). In that case, storing messages in a certain

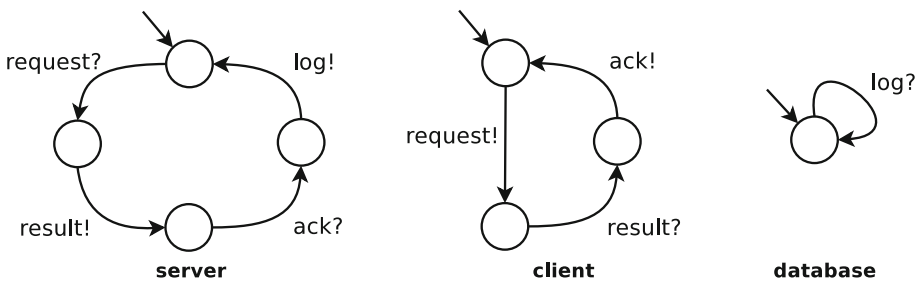

Fig. 1 Motivating example

order does not make sense, and bag buffers are an appropriate communication model. For this reason, many works have been dedicated to the study of systems communicating via bag buffers, see, e.g., [5–8]. Third, we study the stability results varying the equivalence semantics used for asynchronous composition comparison. Each equivalence notion preserves a different temporal logic fragment and enables us to specify temporal properties using those logics. We focus in this paper on the most common classes of equivalence, namely strong, branching, weak, and trace equivalence [9,10].

More precisely, we present in this article how peers can be modeled using LTSs and how they interact via FIFO or bag buffers, assuming the communication model is reliable (no message loss). We prove that the stability property is undecidable for all considered classes of equivalence when assuming FIFO buffers as communication model. We also prove that the stability property is decidable for strong equivalence, undecidable for weak equivalence, and undecidable for trace equivalence when considering bag buffers as communication model. We propose a semi-algorithm for testing stability for branching and strong equivalence when peers communicate via FIFO or bag buffers. We have implemented our semi-algorithm using Python scripts, process algebra encodings, and the equivalence checker available in the CADP toolbox [11]. 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.

The present article extends an early version of this work published in [3] as follows:

- the formal model used to describe peer composition and model comparison consider send messages as well as receive messages;
- we do not focus only on FIFO buffers as asynchronous communication model but also on bag buffers;
- we study and compare asynchronous peer composition varying the equivalence semantics, namely strong, branching, weak, and trace equivalence;
- we carried out new experiments assuming the aforementioned extensions in models, communication and comparison semantics.

The organization of the rest of this article is as follows. Section 2 defines our models for peers and their compositions. Section 3 presents our results on stable systems varying the communication model and the equivalence notion used for comparing asynchronous compositions. Section 4 describes our tool support and experiments we carried out to evaluate our approach. Finally, Sect. 5 reviews related work and Sect. 6 concludes.

2 Communicating systems

We present in this section our model of peers and our definitions of asynchronous compositions considering FIFO and bag buffers, respectively, as communication semantics.

2.1 Peer model

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

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 $\sigma = l_1, \ldots, l_n$, $s \xrightarrow{l_1} s_1, \ldots, s_i \xrightarrow{l_{i+1}} s_{i+1}, \ldots, s_{n-1} \xrightarrow{l_n} s' \in T$. In the following, for the sake of simplicity, we will denote this by $s \xrightarrow{\sigma} s' \in T^*$.

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 send message has a receive message counterpart in another peer (closed systems): $\forall i \in 1..n, \forall m \in \Sigma_i^! \implies \exists j \in 1..n, i \neq j, m \in \Sigma_j^?$.

2.2 Asynchronous composition with FIFO buffers

The asynchronous composition of a set of peers corresponds to the system where 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.

Definition 2 (*FIFO buffers*) A FIFO buffer is a list of messages (repetition is allowed), where a new message is added at the end of the list and the next message to be consumed is taken from the head of the list.

Definition 3 (*Asynchronous composition with FIFO buffers*) 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 FIFO buffer, the asynchronous composition $(P_1|Q_1)|\ldots|(P_n|Q_n)$ is the labeled transition system $LTS_a = (S_a, s_a^0, \Sigma_a, T_a)$ where:

- $S_a \subseteq S_1 \times Q_1 \times \cdots \times S_n \times Q_n$ 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)
- $\Sigma_a = \sqcup_i \Sigma_i$
- $T_a \subseteq S_a \times \Sigma_a \times S_a$, 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) $s \xrightarrow{m?} s' \in T_a$ if $\exists i \in \{1, \ldots, n\} : m \in \Sigma_i^?$, (i) $s_i \xrightarrow{m?} s_i' \in T_i$, (ii) $m Q_i' = Q_i$, (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) $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow s_k' = s_k$

2.3 Asynchronous composition with bag buffers

When the peers communicate asynchronously with each other via bag buffers (one input buffer per peer), the difference with respect to FIFO buffers lies in the semantics of receive

messages. A peer can read a message m iff it is in a state with an outgoing transition labeled with $m?$ and its buffer contains a message m in the set of awaiting messages (no constraint on the order of the receive messages).

Definition 4 (*Bag buffers*) A bag buffer for a peer $\mathcal{P} = (S, s^0, \Sigma, T)$ is a mapping from the set of messages in $\Sigma^?$ to $\mathbb{N}$.

Given a set of messages $\{a, b\}$, we write equivalently a bag B=(a, b, a) or $B(a) = 2$ and $B(b) = 1$.

Definition 5 (*Asynchronous composition with bag buffers*) 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 B_i being its associated bag buffer, the asynchronous composition $(P_1|B_1)|\ldots|(P_n|B_n)$ is the labeled transition system $LTS_a = (S_a, s_a^0, \Sigma_a, T_a)$ where:

- $S_a \subseteq S_1 \times B_1 \times \cdots \times S_n \times B_n$ where $\forall i \in \{1, \ldots, n\},\ B_i \subseteq (\Sigma_i^?)^*$
- $s_a^0 \in S_a$ such that $s_a^0 = (s_1^0, \epsilon, \ldots, s_n^0, \epsilon)$ (where ϵ denotes an empty buffer)
- $\Sigma_a = \sqcup_i \Sigma_i$
- $T_a \subseteq S_a \times \Sigma_a \times S_a$, and for $s = (s_1, B_1, \ldots, s_n, B_n) \in S_a$ and $s' = (s'_1, B'_1, \ldots s'_n, B'_n) \in S_a$

(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) $B'_j = B_j \cup \{m\}$, (iii) $\forall k \in \{1, \ldots, n\} : k \neq j \Rightarrow B'_k = B_k$, and (iv) $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow s'_k = s_k$

(consume) $s \xrightarrow{m?} s' \in T_a$ if $\exists i \in \{1, \ldots, n\} : m \in \Sigma_i^?$, (i) $s_i \xrightarrow{m?} s'_i \in T_i$, (ii) $m \in B_i$, (iii) $B'_i = B_i - \{m\}$, (iv) $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow B'_k = B_k$, and (v) $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow s'_k = s_k$

(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) $\forall k \in \{1, \ldots, n\} : B'_k = B_k$, and (iii) $\forall k \in \{1, \ldots, n\} : k \neq i \Rightarrow s'_k = s_k$

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 Defs. 3 and 5 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 loss.

The k-bounded asynchronous composition can be noted $(P_1|Q_1^k)|\cdots|(P_n|Q_n^k)$ or $(P_1|(Q_1^1|\cdots|Q_1^1))|\ldots|(P_n|(Q_n^1|\cdots|Q_n^1))$, where each peer is in parallel with k buffers bounded to 1. This pattern can be used for encoding bag buffers, where there is no order among messages, but also for encoding ordered FIFO buffers as originally proposed by R. Milner in [9] (see Sections. 1.2 and 3.3 of this book for details).

3 Stability for FIFO and bag buffers

The class of systems that are stable corresponds to systems whose asynchronous compositions remain the same from some buffer bound k, i.e., the asynchronous composition of the system with buffer bounds greater than k is equivalent ($\equiv$) to the asynchronous system with buffers bounded to k. We will introduce in this section the notions of equivalence we consider in this paper.

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 $LTS_a^k \equiv LTS_a^q$ $(\forall q > k)$.

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 property. Stability ensures that these properties are preserved when buffer bounds are increased.

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

In this section, we study the stability property following two dimensions: the model of communication and the equivalence semantics used for comparing asynchronous compositions ($\equiv$ in Def. 6). We study the stability for the most common classes of equivalence, namely strong, branching, weak, and trace equivalence, denoted $\equiv_{st}$, $\equiv_{br}$, $\equiv_{we}$, and $\equiv_{tr}$, respectively. It is worth observing that the temporal properties that can be verified on a stable system depend on the equivalence semantics used for checking the stability property. As an example, if we use branching bisimulation for analyzing asynchronous systems, one can specify temporal formulas using the ACTL\X logic, because branching bisimulation preserves properties written in ACTL\X [12].

As for communication models, we focus on FIFO and bag buffers, respectively. We focus first on FIFO buffers, which is a classic model of communication in distributed systems. Since this model is Turing powerful, testing stability turns out to be undecidable for all considered classes of equivalence, even for the coarsest (i.e., strong bisimulation). Therefore, in a second step, we study the stability property for a more permissive model of communication where peers interact via bag buffers. Systems communicating via bag buffers can be encoded as labeled Petri nets, hence decidability results existing for labeled Petri nets, and particularly those presented in [13], can be applied to our context. When considering bag buffers as communication model, we prove that the stability property is decidable for strong equivalence, and undecidable for weak and trace equivalence. Table 1 summarizes the decidability results presented in this section.

As far as computability is concerned, we also propose at the end of this section a semi-algorithm for computing the smallest k satisfying the stability property. So far, we are able to analyze systems communicating via FIFO and bag buffers for strong and branching bisimulation. The corresponding tool support is presented with more details in Sect. 4.

Let us recall the formal definitions of the classes of equivalences we consider in this work. These definitions are taken from [9,10].

Definition 7 (*Strong bisimulation*) Two LTSs LTS_1 and LTS_2 are strongly bisimilar, denoted by $LTS_1 \equiv_{st} LTS_2$, if there exists a symmetric relation R (called a strong 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 $r \xrightarrow{a} r\prime$, then there exists a transition $s \xrightarrow{a} s\prime$, such that $R(r\prime, s\prime)$.

Table 1 Summary of decidability results

	Strong	Branching	Weak	Trace
FIFO buffers	Undecidable			
Bag buffers	Decidable	?	Undecidable	

Definition 8 (*Branching bisimulation*) Two LTSs LTS_1 and LTS_2 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 $R(r\prime, s\prime)$.

Definition 9 (*Weak bisimulation*) Two LTSs LTS_1 and LTS_2 are weakly bisimilar, denoted by $LTS_1 \equiv_{we} LTS_2$, if there exists a symmetric relation R (called a weak 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 $\delta = a$ and there exists a path $s \xrightarrow{\tau^*} s_1 \xrightarrow{a} s_2 \xrightarrow{\tau^*} s\prime$, such that $R(r\prime, s\prime)$.

Definition 10 (*Trace equivalence*) Given the asynchronous composition LTS_a of 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 set of traces of the asynchronous composition, denoted $\mathcal{T}(LTS_a)$, is the set of action sequences starting from s_a^0. Two asynchronous compositions LTS_a and $LTS_a^{'}$ are trace equivalent, denoted $LTS_a \equiv_{tr} LTS_a^{'}$, iff $\mathcal{T}(LTS_a) = \mathcal{T}(LTS_a^{'})$.

3.1 Systems communicating via FIFO buffers

We start our study with systems communicating via FIFO buffers. We prove in this section that the stability property is undecidable for all classes of equivalence considered.

Theorem 1 *Given a set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ communicating asynchronously via FIFO buffers, it is undecidable whether the corresponding asynchronous system is stable for all classes of equivalence.*

To prove that testing the stability is an undecidable problem, we reduce the halting problem of a Turing machine to the test of stability of a set of peers communicating asynchronously. We prove this undecidability result for strong bisimulation. Since strong bisimulation is a particular case of branching and weak bisimulation (two transition systems without silent transitions are strongly bisimilar iff they are branching and weakly bisimilar), testing stability is undecidable for branching and weak bisimulation. Our reduction uses a deterministic Turing machine and the corresponding asynchronous system is deterministic. Since two deterministic transition systems are strongly equivalent iff they are trace equivalent, then testing stability is undecidable for trace equivalence too.

We start the proof with some preliminaries and notation, then we give an overview of the proof. Afterwards, we detail the construction of a system of two peers simulating the Turing machine and finally we prove the undecidability result.

Proof Preliminaries and notation the turing machine used is a deterministic one-way-infinite single tape model. A Turing machine is defined as $M = (Q_M, \Sigma_M, \Gamma_M, q_0, q_{halt}, B, \delta_M)$ where Q_M is the set of states, Σ_M is the input alphabet, Γ_M is the tape alphabet, $q_0 \in Q_M$ is the initial state and q_{halt} is the accepting state. $B \in \Gamma_M$ is the blank symbol and $\delta_M : Q_M \times \Gamma_M \rightarrow Q_M \times \Gamma_M \times \{left, right\}$ is the transition function. The machine M accepts an input word $w= a_1, \ldots, a_m$ iff M halts on w. If M does not halt on w and the word is not

accepted at a state q, then M initiates a loop. This loop reads any symbol and moves to the right. Hence, if the word w is not accepted, then the machine executes an infinite loop by reading symbols and moving to the right. This looping behavior is not usual in classic Turing machines and acceptance semantics, but this simplifies the reduction without modifying the expressiveness of the Turing machine as shown in [14]. A *configuration* of the Turing machine M is a word $uqv\#$ where uv is a word from the tape alphabet, q is a state of M (meaning that M is in the state q and the head pointing on the first symbol of v), and # is a fixed symbol that is not in Γ_M (used to indicate the end of the word on the tape).

Overview starting from a Turing machine M and an input word w, we construct a pair of peers P_1 and P_2, such that the machine M halts on w if and only if the corresponding system $\{P_1, P_2\}$ is stable, i.e., there exists a k such that $LTS_a^k \equiv_{st} LTS_a$, where LTS_a is the asynchronous composition of the system $\{P_1, P_2\}$.

Construction the peer P_1 simulates the execution of the machine M on w while P_2 is used to receive and re-send messages to P_1. A configuration of M of the form $uqv\#$ is encoded in the buffer of P_1 with the content $uheadv\#$. The peer P_1 is defined as $(S_{P_1}, s_{q_0}, \Sigma_{P_1}, T_{P_1})$ where S_{P_1} is the set of states, s_{q_0} is the initial state where q_0 is the initial state of M. The alphabet $\Sigma_{P_1} = \Sigma^!_{P_1} \cup \Sigma^?_{P_1}$ is defined as follows:

- $\Sigma^!_{P_1} = \Sigma_M \cup \Gamma_M \cup \{head\} \cup \{\#\}$ where all messages sent from P_1 to P_2 are indexed with 2 (e.g., P_1 sends B^2 instead of sending the blank symbol, inversely P_2 sends B^1 instead of sending B to P_1).
- $\Sigma^?_{P_1} = \Sigma_M \cup \Gamma_M \cup \{head\} \cup \{\#\}$ where all messages received from P_2 are indexed with 1.

Now we present how each action of the machine M is encoded.

1. For each transition of M of the form $\delta_M(q, a) = (q', a', right)$ we have the following transitions in T_{P_1}: $s_q \xrightarrow{head^1?} s_1 \xrightarrow{a^1?} s_2 \xrightarrow{a'^2!} s_3 \xrightarrow{head^2!} s_{q'}$.
   If the peer is in the state q and the buffer starts with $head^1a^1$ then the two messages are read and the peer P_1 sends the next configuration to P_2 as depicted in Fig. 2a. s_i's are fresh intermediary states.
2. For each transition of M of the form $\delta_M(q, a) = (q', a', left)$ and for each $x \in \Gamma_M$ we have the following transitions in T_{P_1}: $s_q \xrightarrow{x^1?} s_1 \xrightarrow{head^1?} s_2 \xrightarrow{a^1?} s_3 \xrightarrow{head^2!} s_4 \xrightarrow{x^2!} s_5 \xrightarrow{a'^2!} s_{q'}$. P_1 starts by reading the letter before $head$, then it reads $head$, the next letter, and sends the new configuration to P_2 as depicted in Fig. 2b.
3. For each state s_q where q is a state of M we have the following cycle in T_{P_1}: $s_q \xrightarrow{head^1?} s_1 \xrightarrow{\#^1?} s_2 \xrightarrow{head^2!} s_3 \xrightarrow{B^2!} s_4 \xrightarrow{\#^2!} s_q$. As depicted in Fig. 2c, the configuration of M is extended to the right with a blank symbol. P_1 starts by reading the current configuration of the machine, then sends the next configuration of M to P_2 (P_1 adds a blank symbol before #).
4. For each letter $x \in \Gamma_M \cup \{\#\}$ and each s_q where q is a state of M, we have the following cycle: $s_q \xrightarrow{x^1?} s_1 \xrightarrow{x^2!} s_q$ where P_1 reads x indexed with 1, then sends x indexed with 2.

Note that at a state s_q representing a state q of the machine M, there is only one outgoing transition labeled with $head^1?$. The same remark applies for the first transition of actions of type 2 and 4, hence P_1 is deterministic.

The peer P_2 is defined as $(S_{P_2}, s_{init}, \Sigma_{P_2}, T_{P_2})$ where S_{P_2} is the set of states and s_{init} the initial state. P_2 starts by sending the initial configuration of M to P_1, then reaches the state

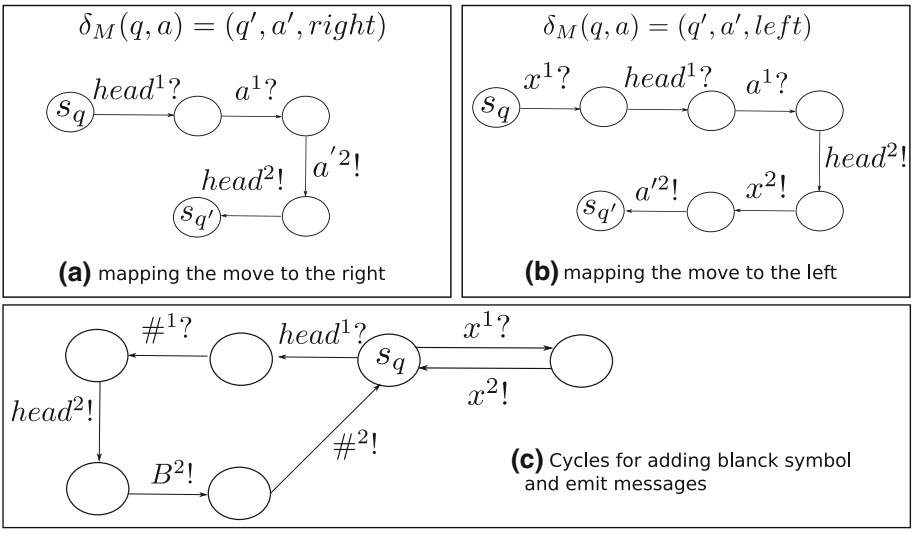

Fig. 2 Mapping the instructions of the machine M to transitions of the peer P_1

s_{univ}, which contains a set of cycles used to receive any message from P_1 and re-send them. Σ_{P_2}=$\Sigma^!_{P_2} \cup \Sigma^?_{P_2}$ is defined as follows:

- $\Sigma^!_{P_2}$= $\Sigma_M \cup \Gamma_M \cup \{head\} \cup \{\#\}$ where all messages sent from P_2 to P_1 are indexed with 1.
- $\Sigma^?_{P_2}$= $\Sigma_M \cup \Gamma_M \cup \{head\} \cup \{\#\}$ where all messages received from P_1 are indexed with 2.

P_2 contains the following transitions:

- $s_{init} \xrightarrow{head^1!} s_1 \xrightarrow{a_1^1!} \ldots \xrightarrow{a_n^1!} s_n \xrightarrow{\#^1!} s_{univ} \in T_{P_2}$ where $w = a_1 a_2 \ldots a_n$.
- $s_{univ} \xrightarrow{x^2?} s_1 \xrightarrow{x^1!} s_{univ} \in T_{P_2}$ where x is any symbol in $\Sigma^?_{P_2}$.

Now we will prove that, given a Turing machine M with an input word w and the corresponding communicating system $\{P_1, P_2\}$ contructed as above, M halts on w iff $\{P_1, P_2\}$ is stable. Suppose that M halts on w. Then, the number of configurations of the machine is finite. Hence, from our construction, the asynchronous composition of $\{P_1, P_2\}$ is finite, so there exists a k such that $LTS^k_a \equiv_{st} LTS_a$.

Now suppose the machine does not halt on w. Then, the corresponding communicating system executes infinitely two cycles: (1) one adding a blank symbol, (2) another reading blank symbols and moving to the right. Hence, for a given bound k, the behavior of the system resulting from the execution of one of the two cycles in LTS^{k+1}_a may not be reproduced in LTS^k_a, due to the buffer bound, then $LTS^k_a \not\equiv_{st} LTS^{k+1}_a$. We will prove that, with our construction, $\forall k$, $LTS^k_a \not\equiv_{st} LTS^{k+1}_a$ (hence, the system is not stable) when the machine M does not halt on w. First, we need to define a *border state*. A border state s^k in LTS^k_a is a state where at least one buffer contains k messages and there is an outgoing transition from s^k labeled with a send message that reaches a state in LTS^{k+1}_a. Now we will detail the proof for cycles of type (1), the cycles of type (2) do not increase the buffers size. Suppose M does not halt. Lets s^k be the configuration of LTS^k_a that represents the configuration of the machine when the infinite loop starts. At s^k the system can execute the first cycle adding a blank

symbol. In s^k the buffer of P_1 is full (size equal to k) and the buffer of P_2 is empty. More precisely, the buffer of P_1 contains the following word: $head^1 \#^1 a_1 \ldots a_m$, where $m = k-2$. At s^k, the system executes the cycle that adds a blank symbol: $s^k \xrightarrow{head^1?} s_1 \xrightarrow{\#^1?} s_2 \xrightarrow{head^2!} s_3 \xrightarrow{B^2!} s_4 \xrightarrow{\#^2!} s^{k'}$.

At $s^{k'}$ the buffer of P_1 contains $k-2$ messages and the buffer of P_2 contains three messages. The sum of the two buffers is $k+1$ messages, due to the addition of the blank symbol, but $s^{k'}$ is still in LTS_a^k. From our construction, at the configuration $s^{k'}$, P_1 sends a_1,..., a_{m-1}: $s^{k'} \xrightarrow{a_1^2!} s_1 \xrightarrow{a_2^2!} \ldots \xrightarrow{a_{m-1}^2!} s^{k''}$. $s^{k''}$ is a border state for LTS_a^k, where, the buffer of P_2 contains k messages and the buffer of P_1 contains one message. At $s^{k''}$, P_1 can send the message a_m, then the system reaches a configuration s^{k+1}, which is in LTS_a^{k+1} but not in LTS_a^k.

Since the system $\{P_1, P_2\}$ is deterministic, the sequence of messages belonging to the path starting at the initial state to $s^{k''}$ is unique and the same in LTS_a^{k+1} and LTS_a^k. Hence, LTS_a^{k+1} and LTS_a^k are strongly bisimilar only if $s^{k''}$ in LTS_a^{k+1} is strongly bisimilar to $s^{k''}$ in LTS_a^k. In LTS_a^{k+1}, at state $s^{k''}$, P_1 can send the message a_m. But in LTS_a^k, P_1 cannot send the message a_m at $s^{k''}$, because the buffer of P_2 contains k messages. Hence, $LTS_a^k \not\equiv_{st} LTS_a^{k+1}$. When the Turing machine M does not halt on w and executes infinitely a cycle that adds a blank symbol, then the corresponding communicating system increases the size of the buffers by one. We have proved that $LTS_a^k \not\equiv_{st} LTS_a^{k+1}$ for an arbitrary k, hence, $\forall k$, $LTS_a^k \not\equiv_{st} LTS_a^{k+1}$, then there is no k where $LTS_a^k \equiv_{st} LTS_a$.

With this reduction we prove that the machine M halts on w iff $\exists k$ such that $LTS_a^k \equiv_{st} LTS_a$, and this shows that checking the stability is undecidable for FIFO buffers and strong bisimulation. □

In [1], the authors study the boundedness of communicating systems where each pair of peers communicates through a full-duplex FIFO channel, which means that each peer has a specific FIFO buffer for each partner peer. For example, if the system consists of three peers, then each peer has two buffers. The FIFO model that we consider in this section is a particular case of the communication model used in [1]. Therefore, as a side effect of Theorem 1, we can conclude that testing stability for the model proposed in [1] is undecidable as well.

Corollary 1 *Given a set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ communicating asynchronously via FIFO buffers, where each pair of peers communicates through full-duplex FIFO channels [1], it is undecidable whether the corresponding asynchronous system is stable for all classes of equivalence.*

3.2 Systems communicating via bag buffers

A bag buffer can be seen as a set of messages, and a peer can read from a buffer if the message is in the bag. When the peers communicate asynchronously with each other via bag buffers, it is known that the communicating system can be encoded as a labeled Petri net, where each bag is modeled as a set of places, one place for each message type [8]. In that case, we prove that the stability problem is decidable for strong bisimulation by reusing equivalence results from [13].

Theorem 2 *Given a set of peers $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ communicating asynchronously via bag buffers, it is decidable whether the corresponding asynchronous system is stable for strong bisimulation.*

Proof In [13] (Theorem 4.8), the authors prove that checking if a labeled Petri net is strongly bisimilar to some unspecified finite automaton is a decidable problem. A communicating system with bag buffers can be encoded as a labeled Petri net, where each message type is represented with a special place in the labeled Petri net. When a peer of the communicating system reads a message m, the corresponding labeled Petri net removes a token from the place representing the message m. Inversely, when the communicating system sends a message m, this action is encoded by incrementing the place of m, with one token. Hence, we can reduce the test of stability to the test of existence of a finite automaton, which is strongly bisimilar to a labeled Petri net such that if there exists a finite state machine strongly bisimilar to the Petri net, then there exists a bounded asynchronous composition strongly bisimilar to the unbounded asynchronous composition. On the other hand, if there is no finite state machine bisimilar to the Petri net, there is no bounded asynchronous composition bisimilar to the unbounded asynchronous system. Hence, testing whether a system communicating via bag buffers is stable with respect to strong bisimulation is decidable. □

Typed channels is another communication model used in communicating systems, where each channel stores messages of a specific type only. Typed channels can be encoded using a bag buffer (one bag can represent several typed channels since order does not matter). Therefrom, as a direct consequence of Theorem 2, we can conclude that checking stability for systems with typed channels is also decidable.

Corollary 2 *Given a set of peers* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *communicating asynchronously via typed buffers, it is decidable whether the corresponding asynchronous system is stable for strong bisimulation.*

When considering weaker equivalences (weak and trace), the stability problem becomes undecidable.

Theorem 3 *Given a set of peers* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *communicating asynchronously with bag buffers, it is undecidable whether the corresponding asynchronous system is stable for weak bisimulation and trace equivalence.*

Proof In [13] (Theorem 4.12), the authors prove that checking if a labeled Petri net is weakly bisimilar to some unspecified finite automaton is an undecidable problem. We can construct a system communicating via bag buffers from a labeled Petri net. Each place of the Petri net is encoded with a peer (a transition system) and its associated bag buffer. When the Petri net is in a place p and can fire a transition (there is at least one token in p), in the corresponding communicating system, the bag buffer of the peer p contains at least one message. Firing a transition (removing a token from the place p and adding a token in the output place p') is encoded by reading a message from the buffer of the peer p and sending a message to the buffer of the peer p'. If there exists a finite state machine weakly bisimilar to a labeled Petri net, then the corresponding communicating system is stable for weak bisimulation. If there is no finite state machine weakly bisimilar to a Petri net, then the corresponding system is not stable. Suppose that testing stability with bags buffers and weak bisimulation is decidable. Hence, since we can encode a labeled Petri net with a communicating system, testing if there exists a finite state automaton weakly bisimilar to a labeled Petri net is decidable. This is a contradiction. Hence, testing if a communicating system with bag buffers is stable with weak bisimulation is undecidable. The same argument is used for trace equivalence, because we can encode a labeled Petri net with a communicating system with bag buffers as shown before, and in [13] (Theorem 3.5), the authors prove that it is undecidable to check whether a labeled Petri net is trace equivalent to some unspecified finite state automaton. □

As typed channels can be encoded using a bag, this undecidability result also stands when we consider typed channels as communication model.

Corollary 3 *Given a set of peers* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *communicating asynchronously via typed buffers, it is undecidable whether the corresponding asynchronous system is stable for weak bisimulation and trace equivalence.*

3.3 Semi-algorithm for testing stability

We present in this subsection a semi-algorithm for testing the stability property for systems communicating via FIFO or bag buffers with strong and branching bisimulation. The sufficient condition ensuring stability is checked as follows: if there exists a bound k such that the k-bounded and the $(k+1)$-bounded asynchronous systems are equivalent, then we prove that the system remains stable, meaning that the observable behavior is always the same for any bound greater than k. In the rest of this section we use $\equiv$ for both strong or branching bisimulation.

Theorem 4 *Given a set of peers* $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ *communicating via FIFO or bags buffers, if* $\exists k \in \mathbb{N}$*, such that* $LTS_a^k \equiv LTS_a^{k+1}$*, then we have* $LTS_a^k \equiv LTS_a^q, \forall q > k$ *for strong and branching bisimulation.*

Proof We will prove the theorem by induction for bag buffers, starting with the following base case: If $LTS_a^k \equiv LTS_a^{k+1}$ then $LTS_a^k \equiv LTS_a^{k+2}$. Let us recall that the strong and branching bisimulations are congruences with respect to the operators of process algebras [15], that is, if P and P' are strongly/branching bisimilar, then for every Q the processes $P|Q$ and $P'|Q$ are strongly/branching bisimilar too.

Suppose that $\exists k \in \mathbb{N}$, such that $LTS_a^k \equiv LTS_a^{k+1}$, then:

$$(P_1|B_1^k)|\cdots|(P_n|B_n^k) \equiv (P_1|B_1^{k+1})|\cdots|(P_n|B_n^{k+1}) \tag{1}$$

A bag buffer of size k can be written as a parallel composition of k buffers of size 1, hence:

$$(P_1|B_1^{k+2})|\cdots|(P_n|B_n^{k+2}) \equiv (P_1|B_1^{k+1}|B_1^1)|\cdots|(P_n|B_n^{k+1}|B_n^1) \tag{2}$$

Then, by congruence and using equation (1) we have:

$$(P_1|B_1^{k+2})|\cdots|(P_n|B_n^{k+2}) \equiv (P_1|B_1^k|B_1^1)|\cdots|(P_n|B_n^k|B_n^1) \tag{3}$$

$$(P_1|B_1^{k+2})|\ldots|(P_n|B_n^{k+2}) \equiv (P_1|B_1^{k+1})|\cdots|(P_n|B_n^{k+1}) \tag{4}$$

$$(P_1|B_1^{k+2})|\cdots|(P_n|B_n^{k+2}) \equiv (P_1|B_1^k)|\cdots|(P_n|B_n^k) \tag{5}$$

The same argument can be used to prove the induction case, i.e., we suppose that $LTS_a^k \equiv LTS_a^{k+i}$ and we demonstrate that $LTS_a^k \equiv LTS_a^{k+i+1}$. This proves that if $LTS_a^k \equiv LTS_a^{k+1}$, then we have $LTS_a^k \equiv LTS_a^q, \forall q > k$. □

The same reasoning can be used for proving that this theorem holds with other communication models, e.g., FIFO buffers. The only difference is in the buffer encoding, but for all these variants, one buffer of size k can be modeled using a parallel composition of k buffers of size 1. This is described in [9] (Sections. 1.2 and 3.3) for FIFO buffers for instance. As far as typed channels are concerned, they can be encoded using bag buffers in a straightforward way.

It is worth emphasizing that Theorem 4 does not hold for weak bisimulation and trace equivalence, because these relations are not congruences for the process algebra operators and in particular for the parallel composition.

4 Tool support

We have implemented the approach presented beforehand in this article and our tool is available online[1] (source code, dataset of examples). It is easily usable provided that the user installs Python 2.7.x and the CADP toolbox [11]. Our tool implements the semi-algorithm presented in Sect. 3.3 and works as follows. Given a set of peer LTSs, we compute an initial bound k. For that bound, we verify whether the k-bounded asynchronous system is equivalent to the $(k+1)$-bounded system. If this is the case, the system is stable for bound k, and properties can be analyzed using that bound. If the equivalence check returns false, we modify k and apply the check again. We repeat the process up to a certain arbitrary bound *kmax* that makes the approach abort inconclusively if attained. All these checks are achieved automatically using compilers, exploration tools, and equivalence checking tools available in CADP.

Several heuristics and search algorithms can be used for computing an initial bound k and the next bound to attempt, resp. In this section, the initial k is computed as the maximum between the longest sequence of send messages in all peers and the highest number of send messages destinated to a same peer. The intuition behind the longest sequence of send messages is that all peers can at least send all their messages even if no peer consumes any message from its buffer. The second part of the maximum function corresponds to the case in which the buffer size prevents some peer to send messages because that buffer is already full. Next values of k are computed by decrementing / incrementing the bound till reaching *kmax* or the smallest k satisfying stability. Other heuristics and search algorithms have been implemented (e.g., binary search) but turned out to be less efficient.

Experimental results we used a Mac OS laptop running on a 2.3 GHz Intel Core i7 processor with 16 GB of memory. Our database of examples contains more than 300 examples on which we carried out these experiments. Many of these examples come from the literature on the subject. Table 2 presents some of these examples, where we use bag buffers as communication model and strong bisimulation as comparison notion when computing stability. During our experiments, we used a bound *kmax* arbitrarily fixed to 10.

We first comment on the results presented in Table 2. Most of these examples are stable (23 out of 29) and can thus be analyzed using the verification approach proposed in this article. It does not take long (a few minutes) to check the stability property for most examples and the corresponding state spaces are quite small (but for examples where *kmax* is reached or when the number of peers is higher). Computation times increase mainly due to the number of peers (see, e.g., examples 28 and 29) and due to the size of buffer bounds, which induces the computation of larger intermediate state spaces (see, e.g., examples 18 and 28). Note that the strategy we use for these experiments can be quite costly in time if the initial computed k is high: this is the case of example 18, which starts computing compositions for k=6 and k=7, resulting in large intermediate state spaces, whereas the actual k respecting stability is smaller.

Now, beyond Table 2, let us comment on experiments we achieved on all the examples of our dataset. More than half of the examples are stable and can be analyzed using our approach. It is worth noting that the numbers increase to about 70% when only send messages (instead of both send and receive messages) are considered in models and model comparisons. These experiments also confirm that more examples satisfy stability when considering a weaker then more permissive equivalence, i.e., branching bisimulation instead of strong bisimulation. The number of stable systems is slightly higher when using FIFO buffers, which is a more

[1] http://convecs.inria.fr/people/Gwen.Salaun/Tools/stabc.zip.

Table 2 Experimental results

Id	Description	$\|P\|$	$\|S\|/\|T\|$	k	LTS_a^k $\|S\|/\|T\|$	Time (in sec)
(1)	Estelle specification [16]	2	7/9	*kmax*	47,818/106,474	198
(2)	News server [17]	2	9/9	3	21/34	55
(3)	Client/server [1]	2	6/10	1	7/8	35
(4)	CFSM system [16]	2	6/7	*kmax*	5,299/10,959	190
(5)	Promela program (1) [18]	2	6/6	2	8/12	54
(6)	Promela program (2) [19]	2	8/8	*kmax*	726/1,585	207
(8)	Web services [20]	3	13/12	2	22/33	99
(9)	Trade system [21]	3	12/12	1	13/15	48
(10)	Online stock broker [22]	3	13/16	*kmax*	3,823/8,214	>1h
(11)	FTP transfer [23]	3	20/17	4	51/90	73
(12)	Client/server [24]	3	14/13	2	16/17	72
(13)	Mars explorer [25]	3	34/34	3	52/74	78
(14)	Online computer sale [26]	3	26/26	2	21/23	73
(15)	E-museum [27]	3	33/40	3	94/183	138
(16)	Client/supplier [28]	3	31/33	2	37/47	100
(17)	Restaurant service [29]	3	15/16	2	25/37	73
(18)	Travel agency [30]	3	32/38	4	60/99	>1h
(19)	Vending machine [31]	3	15/14	2	17/19	73
(20)	Travel agency [32]	3	42/57	3	82/143	74
(21)	Train station [33]	4	18/18	2	39/66	124
(22)	Factory job manager [34]	4	20/20	2	30/45	92
(23)	Bug report repository [35]	4	12/12	*kmax*	124/254	>1h
(24)	Cloud application [36]	4	8/10	*kmax*	47,916/211,750	321
(25)	Sanitary agency [37]	4	35/41	3	154/310	92
(26)	SQL server [38]	4	32/38	3	49/74	94
(27)	SSH protocol [39]	4	26/28	2	28/30	123
(28)	Booking system [40]	5	45/53	2	62/85	>1h
(29)	Hand-crafted example	10	63/58	2	243,540/1,448,577	602

restrictive model than bag buffers. When using bag buffers, the corresponding asynchronous compositions are larger in terms of number of states and transitions, which induces more possible causes of violating the equivalence notions on which relies the stability analysis.

5 Related work

Brand and Zafiropulo show in [1] that the verification problem for FSMs interacting via (unbounded) FIFO buffers is undecidable. Gouda et al. [41] present sufficient conditions to compute a bound k from which two finite state machines communicating through 1-directional buffers are guaranteed to progress indefinitely. Jeron and Jard [16] 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. [42] propose some verification

techniques for CFSMs. They present a method for performing symbolic forward analysis of unbounded *lossy* buffers systems. In [18], the authors present an incomplete boundedness test for communication buffers 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 [43] 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.

A notion of existential-boundedness was introduced in [44] for communicating automata. The idea is to assume unbounded buffers, but to consider only executions that can be rescheduled on bounded ones. Darondeau et al. [45] identify a decidable class of systems consisting of non-deterministic communicating processes that can be scheduled while ensuring boundedness of buffers. [46] proposed a causal chain analysis to determine upper bounds on buffer sizes for multi-party sessions with asynchronous communication. Bouajjani and Emmi [47] 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.

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 (universal) boundedness of the systems under analysis. Contrarily, we are particularly interested in unbounded (yet possibly stable) systems. Existential boundedness in turn assumes structural hypothesis on models, e.g., at most one sending transition and no mix of send/receive messages outgoing from a same state in [44,45], whereas we do not impose any restriction on our LTS models.

The stability property was originally introduced in [2,3,48], but these early results only focus on send messages when comparing asynchronous compositions using equivalence checking. Moreover, both papers rely on FIFO buffers as communication model and assume specific equivalence notions as comparison criterion (trace equivalence in [2] and branching bisimulation in [3]). The current article goes much further by keeping send and receive messages when checking stability, by considering several equivalence notions when comparing asynchronous compositions, and by studying other communication models.

6 Conclusion

We have presented in this article a framework for formally analyzing systems communicating via FIFO or bag buffers. This work focuses on cyclic systems modeled using finite state machines. We have studied the stability property, which shows that the asynchronous composition of a set of peers may exhibit the same observational behavior (send and receive messages) from a specific bound. In this article, we have varied the communication model (FIFO buffers, bag buffers) as well as the equivalence relations used for comparing the asynchronous compositions (strong, branching, weak, and trace equivalences). We have presented several decidability results and a semi-algorithm for determining the smallest k satisfying the stability property for stable systems. In such a case, model checking techniques can then be used on the asynchronous version of the system with FIFO/bag 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. We implemented the semi-algorithm presented in this article for automatically checking the stability property. Our experiments have showed that many examples satisfy the stability property and this can be computed in a reasonable time.

As far as future works are concerned, a first open issue is to determine whether the stability problem is decidable or not for systems communicating via bag buffers and compared using branching bisimulation. Algorithms for checking the stability property in practice also deserve to be studied more thoroughly. So far, we have proposed a semi-algorithm for asynchronous systems analyzed with strong and branching bisimulation for both FIFO and bag buffers. Algorithms or semi-algorithms for weak and trace equivalences are still to be defined.

References

1. Brand D, Zafiropulo P (1983) On communicating finite-state machines. J ACM 30(2):323–342
2. Basu S, Bultan T (2014) Automatic verification of interactions in asynchronous systems with unbounded buffers. In: Proceedings of ASE'14, pp 743–754
3. Akroun L, Salaün G, Ye L (2016) Automated analysis of asynchronously communicating systems. In: Proceedings of SPIN'16, Vol. 9641 of LNCS, Springer, pp 1–18
4. Barbanera F, van Bakel S, de Liguoro U (2017) Orchestrated session compliance. J Log Algebraic Meth Progr 86(1):30–76
5. Beauxis R, Palamidessi C, Valencia FD (2008) On the asynchronous nature of the asynchronous pi-calculus. In: Concurrency, Graphs and Models, Vol. 5065 of LNCS, Springer, pp 473–492
6. Garavel H, Thivolle D (2009) Verification of GALS Systems by Combining Synchronous Languages and Process Calculi. In: Proceedings of SPIN'09, Vol. 5578 of LNCS, Springer, pp 241–260
7. Ravn AP, Srba J, Vighio S (2011) Modelling and verification of web services business activity protocol. In: Proceedings of TACAS'11, Vol. 6605 of LNCS, Springer, pp 357–371
8. Clemente L, Herbreteau F, Sutre G (2014) Decidable topologies for communicating automata with FIFO and bag channels. In: Proceedings of CONCUR'14, Vol. 8704 of LNCS, Springer, pp 281–296
9. Milner R (1989) Communication and concurrency. Prentice-Hall, Inc., Upper Saddle River
10. van Glabbeek RJ (2001) The linear time–branching time spectrum I, vol 1. Handbook of process algebra. Elsevier, Amsterdam
11. Garavel H, Lang F, Mateescu R, Serwe W (2011) CADP 2010: A toolbox for the construction and analysis of distributed processes. In: Proceedings of TACAS'11, Vol. 6605 of LNCS, Springer, pp 372–387
12. Nicola RD, Vaandrager FW (1990) Action versus state based logics for transition systems. In: Semantics of concurrency, Vol. 469 of LNCS, Springer, pp 407–419
13. Jancar P, Esparza J, Moller F (1999) Petri nets and regular processes. J Comput Syst Sci 59(3):476–503
14. Finkel A, McKenzie P (1997) Verifying identical communicating processes is undecidable. Theor Comput Sci 174(1–2):217–230
15. Fokkink W (2000) Introduction to process algebra, texts in theoretical computer science. An EATCS series. Springer, Berlin
16. Jéron T, Jard C (1993) Testing for unboundedness of FIFO channels. Theor Comput Sci 113(1):93–117
17. Ouederni M, Salaün G, Bultan T (2013) Compatibility checking for asynchronously communicating software. In: Proceedings of FACS'13, Vol. 8348 of LNCS, Springer, pp 310–328
18. Leue S, Mayr R, Wei W (2004) A scalable incomplete test for message buffer overflow in promela models. In: Proceedings SPIN'04, Vol. 2989 of LNCS, Springer, pp 216–233
19. Leue S, Stefanescu A, Wei W (2008) dependency analysis for control flow cycles in reactive communicating processes. In: Proceedings of SPIN'08, Vol. 5156 of LNCS, Springer, pp 176–195
20. Fu X, Bultan T, Su J (2004) Analysis of interacting BPEL web services. In: Proceedings of WWW'04, ACM Press, pp 621–630
21. Deniélou PM, Yoshida N (2012) Multiparty session types meet communicating automata. In: Proceedings of ESOP'12, Vol. 7211 of LNCS, Springer, pp 194–213
22. Fu X, Bultan T, Su J (2004) Conversation protocols: a formalism for specification and verification of reactive electronic services. Theor Comput Sci 328(1–2):19–37
23. Bracciali A, Brogi A, Canal C (2005) A formal approach to component adaptation. J Softw Syst 74(1):45–54

24. Canal C, Poizat P, Salaün G (2006) Synchronizing behavioural mismatch in software composition. In: Proceedings of FMOODS'06, Vol. 4037 of LNCS, Springer, pp 63–77
25. Brogi A, Popescu R (2006) Automated generation of BPEL adapters. In: Proceedings of ICSOC'06, Vol. 4294 of LNCS, Springer, pp 27–39
26. Cubo J, Salaün G, Canal C, Pimentel E, Poizat P (2007) A model-based approach to the verification and adaptation of WF/.NET components, In: Proceedings of FACS'07, Vol. 215 of ENTCS, Elsevier, pp 39–55
27. Canal C, Poizat P, Salaün G (2008) Model-based adaptation of behavioural mismatching components. IEEE Trans Softw Eng 34(4):546–563
28. Cámara J, Martín JA, Salaün G, Canal C, Pimentel E (2010) Semi-automatic specification of behavioural service adaptation contracts. Electr Notes Theor Comput Sci 264(1):19–34
29. van der Aalst WMP, Mooij AJ, Stahl C, Wolf K (2009) Service interaction: patterns, formalization, and analysis. In: Proceedings of SFM'09, Vol. 5569 of LNCS, Springer, pp 42–88
30. Seguel R, Eshuis R, Grefen PWPJ (2010) Generating minimal protocol adaptors for loosely coupled services. In: Proceedings of ICWS'10, IEEE computer society, pp 417–424
31. Gierds C, Mooij AJ, Wolf K (2012) Reducing adapter synthesis to controller synthesis. IEEE T Serv Comput 5(1):72–85
32. Bennaceur A, Chilton C, Isberner M, Jonsson B (2013) Automated mediator synthesis: combining behavioural and ontological reasoning. In: Proceedings of SEFM'13, Vol. 8137 of LNCS, Springer, pp 274–288
33. Salaün G, Bultan T, Roohi N (2012) Realizability of choreographies using process algebra encodings. IEEE Trans Serv Comput 5(3):290–304
34. Bultan T, Ferguson C, Fu X (2009) A tool for choreography analysis using collaboration diagrams. In: Proceedings of ICWS'09, IEEE, pp 856–863
35. Gössler G, Salaün G (2011) Realizability of choreographies for services interacting asynchronously. In: Proceedings of FACS'11, Vol. 7253 of LNCS, Springer, pp 151–167
36. Güdemann M, Salaün G, Ouederni M (2012) Counterexample Guided Synthesis of Monitors for Realizability Enforcement. In: Proceedings of ATVA'12, Vol. 7561 of LNCS, Springer, pp 238–253
37. Salaün G, Bordeaux L, Schaerf M (2004) Describing and reasoning on web services using process algebra. In: Proceedings of ICWS'04, IEEE Computer Society, pp 43–50
38. Poizat P, Salaün G (2007) Adaptation of open component-based systems. In: Proceedings of FMOODS'07, Vol. 4468 of LNCS, Springer, pp 141–156
39. Martín JA, Pimentel E (2011) Contracts for security adaptation. J Log Algebr Progr 80(3–5):154–179
40. Mateescu R, Poizat P, Salaün G (2008) Adaptation of service protocols using process algebra and on-the-fly reduction techniques. In: Proceedings of ICSOC'08, Vol. 5364 of LNCS, Springer, pp 84–99
41. Gouda MG, Manning EG, Yu Y-T (1984) On the progress of communications between two finite state machines. Inf Control 63(3):200–216
42. Abdulla PA, Bouajjani A, Jonsson B (1998) On-the-fly analysis of systems with unbounded, lossy FIFO channels. In: Proceedings CAV'98, Vol. 1427 of LNCS, Springer, pp 305–318
43. Cécé G, Finkel A (2005) Verification of programs with half-duplex communication. Inf Comput 202(2):166–190
44. Genest B, Muscholl A, Seidl H, Zeitoun M (2006) Infinite-state high-level MSCs: model-checking and realizability. J Comput Syst Sci 72(4):617–647
45. Darondeau P, Genest B, Thiagarajan PS, Yang S (2010) Quasi-static scheduling of communicating tasks. Inf Comput 208(10):1154–1168
46. Deniélou PM, Yoshida N (2010) Buffered communication analysis in distributed multiparty sessions. In: Proceedings CONCUR'10, Vol. 6269 of LNCS, Springer, pp 343–357
47. Bouajjani A, Emmi M (2012) Bounded phase analysis of message-passing programs. In: Proceedings of TACAS'12, Vol. 7214 of LNCS, Springer, pp 451–465
48. Canal C, Salaün G (2016) Stability-based adaptation of asynchronously communicating software. In: Proceedings of SEFM'16, Vol. 9763 of LNCS, Springer, pp 321–336