Theoretical Computer Science ••• (••••) •••–•••

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

On deciding synchronizability for asynchronously communicating systems ☆

Samik Basu [a,*], Tevfik Bultan [b]

[a] Iowa State University, United States
[b] University of California at Santa Barbara, United States

ARTICLE INFO

Article history:
Received 6 August 2015
Received in revised form 4 August 2016
Accepted 26 September 2016
Available online xxxx
Communicated by P. Aziz Abdulla

Keywords:
Asynchronous systems
Message-passing systems
Synchronizability
Verification

ABSTRACT

Asynchronously communicating systems involve peers or entities that communicate by exchanging messages via buffers. In general, the size of such buffers is not known apriori, i.e., they are considered to be unbounded. As a result, models of asynchronously communicating systems typically exhibit infinite state spaces and it is well-known that reachability and boundedness problems for such models are undecidable. This, in turn, makes automatic verification of asynchronous systems undecidable as well. We discuss a particular class of asynchronous systems over peers for which the interaction behaviors do not change when the peers are made to communicate synchronously. Such systems are referred to as *Synchronizable*. Automatic verification of synchronizable systems is decidable as the verification of the system can be performed using its synchronous counterpart. Recently, we have proved that checking whether or not a system is synchronizable is decidable. In this paper, we consider different types of asynchronous communication, where the type is described in terms of the nature of buffering and the number of buffers, and discuss how/if synchronizability is decidable for each type. The new results subsume the existing ones and present a comprehensive synchronizability study of asynchronous systems.

1. Introduction

With the increasing use of software systems in distributed settings, dependability of distributed systems has remained one of the crucial problems in computing. A distributed system with many components coordinates their executions in order to achieve a desired objective. One emerging paradigm to realize such coordination is based on message-based interaction. In this setting, the components (we refer to them as *peers*) interact by exchanging (sending and receiving) messages and coordinate their activities/execution [1–6].

Due to the distributed nature of the system, the exchange of messages does not occur in a lock-step fashion. That is, a message sent by a peer is not immediately consumed by the receiving peer owing to delays in the communication network. In other words, the sender and the receiver are not always in sync—the system resulting from the communication is called *asynchronous system*. The model of such asynchrony involves buffers—messages sent by the sender are stored in the

☆ This work was supported by the National Science Foundation, under grant CCF1116836.
* Corresponding author.
E-mail addresses: sbasu@iastate.edu (S. Basu), bultan@cs.ucsb.edu (T. Bultan).

http://dx.doi.org/10.1016/j.tcs.2016.09.023
0304-3975/

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023

buffers and messages consumed by the receivers are removed from the buffers. The capacity of the buffer is assumed to be unbounded to capture any and all possible delays that can incur between the sending of the message and its consumption. The unboundedness in the capacity results in models of asynchronous system that exhibit infinite state-space and, in fact, such systems can simulate Turing Machines [7]. This renders their automatic verification, that relies on state-space exploration, undecidable, in general.

As a result, a number of techniques have been developed to identify different subclasses of asynchronous systems for which automatic verification is decidable and tractable (e.g., [8,9]). One such class considered in [10,11] is referred to as the *synchronizable systems*. An asynchronous system over peers is said to be synchronizable if and only if any behavior exhibited by the asynchronous system with unbounded buffers can be exhibited by the same peers when communicating synchronously (i.e., with no buffers). In synchronous systems, the state-space is finitely bounded (by the product of the number of states of each peers) and, therefore, automatic verification of synchronous systems can be performed efficiently. In other words, automatic verification of an asynchronous system, that is synchronizable, can be performed by verifying its synchronous variant.

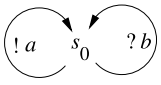
The question remains: "*Is synchronizability checking decidable?*". In [12,13], we proved that synchronizability checking is decidable for asynchronous systems where (a) each receiver peer has a buffer, and (b) each buffer acts as a queue (FIFO).

We consider the behavior of the systems in terms of sequences of sends, which can be viewed as the observable behavior of a system. The consumption of messages (receives) is typically viewed as local to the receiving peers and is not considered observable. The choice for focusing on the sequences of send actions stems from the fact in distributed systems such as Web services, the consumption of messages (receives) is typically viewed as local to the receiving services and is considered unobservable. Furthermore, the desired behaviors of the services are often described in terms of the messages being exchanged (sent) by the participating peers/services [14]. Similar, specifications describe the desired interactions in Singularity OS communication contracts [15] and UBF(B) communication contracts in distributed Erlang programs [16].

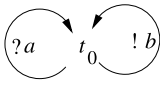
We have proved that an asynchronous system (denoted by $\mathcal{I}$) over a given set of peers described as finite-state machines is synchronizable if and only if the sequences of sends in the corresponding synchronous system (denoted by $\mathcal{I}_0$), where peers communicate synchronously, are identical to that in the 1-bounded asynchronous system (denoted by $\mathcal{I}_1$), where peers communicated asynchronously via buffers of capacity 1. Given that both $\mathcal{I}_0$ and $\mathcal{I}_1$ exhibit behavior with finite-state space, verifying whether or not they have the same set of send-sequences can be performed automatically, which, in turn, makes synchronizability checking decidable.

As a simple example, consider three peers as follows:

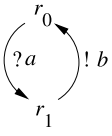
- $\mathcal{P}_1$ has one state s_0 with two loops: one over send action a and other over receive action b.



- $\mathcal{P}_2$ has one state t_0 with two loops: one over receive action a and the other over send action b.



- $\mathcal{P}_3$ has two states r_0 and r_1. It has a transition from r_0 to r_1 on receive action a and a transition from r_1 to r_0 on send action b.



Assume that the start states of each peer are subscripted by 0. Consider that the synchronous communication between $\mathcal{P}_1$ and $\mathcal{P}_2$ results in $\mathcal{I}_0$. In $\mathcal{I}_0$, every time $\mathcal{P}_1$ sends message a by executing $!a$, $\mathcal{P}_2$ is ready to consume it synchronously (in lock-step) by executing $?a$; similarly, every time $\mathcal{P}_2$ sends message b, it can be immediately consumed by $\mathcal{P}_1$. As a result of synchronous interactions, the sequence of sends will be any ordering of a's and b's. The same sequences are also present in their 1-bounded asynchronous system $\mathcal{I}_1$—where the sent messages are not necessarily immediately consumed; instead the messages are stored in a buffer of size 1. Therefore, based on [12], the asynchronous system resulting from $\mathcal{P}_1$ and $\mathcal{P}_2$ is synchronizable.

On the other hand, the synchronous communication between $\mathcal{P}_1$ and $\mathcal{P}_3$ results in sending of a, which is immediately consumed by $\mathcal{P}_3$ and is followed by sending of b by $\mathcal{P}_3$, which, in turn, is consumed synchronously by $\mathcal{P}_1$. The resultant sequence of sends is $ababab\ldots$. However, the 1-bounded asynchronous system involving the interaction between the same

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023

peers $\mathcal{P}_1$ and $\mathcal{P}_3$ includes a sequence of sends: $aabaab\ldots$, as $\mathcal{P}_1$ can send a, which can be consumed immediately by $\mathcal{P}_3$, following which $\mathcal{P}_1$ can again send a, which is placed in the buffer to be consumed when $\mathcal{P}_3$ is ready to perform $?a$ action. As a result, the asynchronous system resulting from $\mathcal{P}_1$ and $\mathcal{P}_3$ is not synchronizable.

In this paper, we build on our previous result and present decidability results for synchronizability for different types of asynchronous systems. These types are defined on the basis of the number of buffers present in the system and their characteristics. For instance, in typical P2P communication, each pair of sender–receiver peers has a dedicated communication buffer; while in shared bus system, all the sender peers share one buffer. The number of buffers present in the system can result in different send-sequences for the same set of communicating peers. Further differences can be exhibited if the buffers allow random or unordered access to the pending messages as opposed to typical FIFO ordered access. We consider these variations of asynchronous systems and discuss their expressivity in terms of the sequences of send actions that can be produced by the systems. We prove that checking synchronizability of these systems is decidable and characterizes their ordering. In summary, the paper includes a comprehensive study of deciding synchronizability for different types of asynchronous systems.

Organization The rest of the paper is organized as follows. In Section 2 we present the necessary formalisms and definitions for describing the different types of asynchronous systems and their synchronizability requirements. In Section 3 we present the decidability results. In Section 4 we discuss the existing work related to our results, and conclude with the summary of the impact of our results.

2. Background

We consider asynchronous systems, where peers participating in the system, communicate via exchange of messages. The asynchrony in the communication comes into effect due to the delay between the time when a message is sent and the time when the message is actually consumed. Such a delay is modeled by simply considering a buffer that holds the messages that are being sent and whenever a peer is ready to consume a message, it looks for the message's presence in the buffer. If the message is present in the buffer, then it is consumed by the peer; otherwise the peer blocks and waits for the message. There can be different types of asynchronous systems where the type is decided by the ordering of messages in the buffer and the number of buffers. In the following, we first describe the types of such systems and then formally define the state-machine models specifying the systems.

2.1. Types of asynchrony

Type:1-1. There exists a buffer in the form of a queue for each pair of peers, where one element of the pair is the sender and the other is the receiver. A message sent by a sender peer to a receiver peer is added to the tail of the queue and messages are consumed by the receiver from the head of the corresponding queue. For a Type:1-1 system containing n peers, there are at most $n \times (n-1)$ queues. This type of systems correspond to P2P communication between entities.

Type:*-1. There exists a buffer for each peer participating in the asynchronous communication. The buffer is in the form of a queue. For a Type:*-1 system containing n peers, there are at most n such queues. Type:*-1 systems typically represent asynchronous communication between entities over the network, e.g., client-server, web services.

Type:*-*. There exists one buffer in the form of a queue for all participating peers. Any message sent to any peer is added to the tail of the queue and messages are consumed by any peer from the head. Peers can only consume messages addressed to them. For a Type:*-* system containing any number of peers, there is 1 queue. The entities communicating via shared bus are represented using Type:*-* systems.

Type:bag. The receive buffer is not in the form of a queue. As a result, the messages can be consumed in an order different from the order in which they were produced/sent. There are three variants of Type:bag asynchronous systems depending on the number of the queues per receivers and senders.

The above types describe the communication models in different distributed algorithms (ranging from P2P, service computing to shared bus applications, which involve FIFO ordering, source ordering or no ordering). [17] presents a detailed survey of different types of multicast and broadcast distributed algorithms along with their communication models and properties of interest.

Fig. 1 illustrates role (and type) of buffers in different types of asynchronous communication. As noted before, we focus primarily on sequences of send actions—the send sequences. Given a set of peers, their asynchronous communication results in different sets of send-sequences depending on the type of buffering. This is because, if a peer needs to consume some set of messages in a particular order before sending a message m, then the peer can potentially block if the messages to be consumed are not delivered or are not present in the buffer in the order they are expected to be consumed; as a result, the peer will not be able to send m.

For instance, sequences of sends in a Type:*-1 system are a subset of that in a Type:1-1 system involving the same set of peers; while the sequences of sends in a Type:*-* system are a subset of that in a Type *-1 system. This relationship follows from the restrictions the buffer-queues impose on the peers that are waiting to consume messages from the buffer-queue(s);

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023

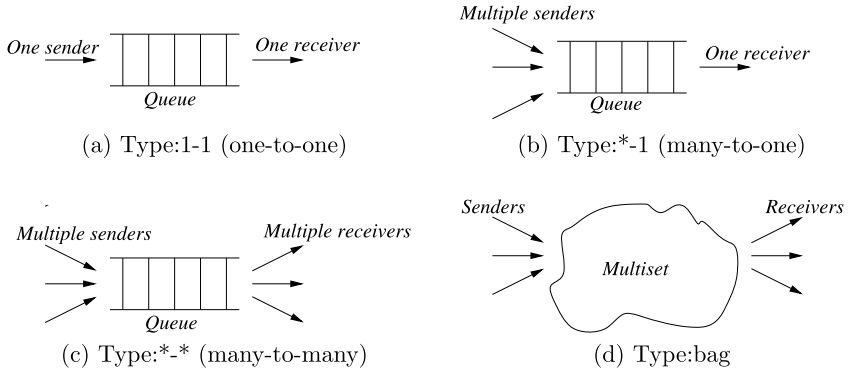


Fig. 1. Types of asynchronous communication.

$\mathcal{P}_1$	$\mathcal{P}_2$	$\mathcal{P}_3$	$\mathcal{P}_4$	Systems
s_0 $\downarrow !a_1$ s_1 $\downarrow !a_2$ s_2	t_0 $\downarrow !a_3$ t_1 $\downarrow ?a_1$ t_2	r_0 $\downarrow ?a_2$ r_1 $\downarrow !a_4$ r_2	u_0 $\downarrow ?a_3$ u_1 $\downarrow ?a_4$ u_2	Type:1-1/*-1/bag includes send sequences: $a_1a_2a_4a_3$ Type:*-* does not include the above send sequence
s_0 $\downarrow !a_1$ s_1 $\downarrow ?a_3$ s_2	t_0 $\downarrow !a_2$ t_1	r_0 $\downarrow ?a_1$ r_1 $\downarrow ?a_2$ r_2 $\downarrow !a_3$ r_3		Type:1-1/bag includes the send sequence $a_2a_1a_3$ Type:*-1/*-* does not include the above send sequence
s_0 $\downarrow !a_1$ s_1 $\downarrow !a_2$ s_2 $\downarrow ?a_3$ s_3	t_0 $\downarrow ?a_2$ t_1 $\downarrow ?a_1$ t_2 $\downarrow !a_3$ t_3			Type:bag includes the send sequence $a_1a_2a_3$ Type:1-1/*-1/*-* does not include the above send sequence

Fig. 2. Differences between systems in terms of sequences of sends due to different types of buffering mechanism.

Type:1-1 is the least restrictive as each sender–receiver pair has its own queue, Type:*-1 is moderately restrictive as each receiver has its own queue and Type:*-* is the most restrictive as all peers share one queue.

The situation becomes interesting when we consider Type:bag system. As the ordering in which messages are sent (added to the buffer(s)) is not important, the behavior of the receivers in Type:bag system does not depend on the number of buffers. As a result, the buffering in Type:bag asynchronous communication can be represented as a multiset to and from which all the senders and receivers deliver and consume messages, respectively (Fig. 1(d)). Furthermore, the behavior of the receiver is not as restrictive as any of the Types 1-1, *-1 and *-* systems. We will discuss this formally in the following sections.

Fig. 2 illustrates the differences between the systems in terms of the send sequences. For instance, in the first row, there are 4 peers. Each peer's behavior is described using finite state machine where transitions describe evolution using either a send action (denoted by $!m$) or a receive action (denoted by $?m$). As per the first row peers, consider an interaction scenario as follows: $\mathcal{P}_1$ sends a_1 to $\mathcal{P}_2$ and then it sends a_2 to $\mathcal{P}_3$. If the interaction is following Type:1-1, Type:*-1 or Type:bag asynchrony, then $\mathcal{P}_3$ can consume the message a_2 and proceed to send a_4. However, if the interaction is following Type:*-* asynchrony, where there is only one buffer-queue, then $\mathcal{P}_3$ cannot consume a_2 from the queue, as the message at the head of the queue is a_1. That is, $\mathcal{P}_3$ needs to wait till $\mathcal{P}_2$ sends a_3 and then consumes a_1 before it can consume a_2 and send a_4. The Type:*-* asynchrony imposes an ordering between the sends a_3 and a_4, which is not present in other types of asynchrony.

In row 2 of Fig. 2, there are three peers. Their interactions illustrate the restrictions in interaction imposed by both Type:*-1 and Type:*-* asynchrony. Consider the scenario, where $\mathcal{P}_2$ sends a_2 to $\mathcal{P}_3$ before $\mathcal{P}_1$ sends a_1. If we consider Type:1-1 or Type:bag asynchrony, the ordering in which the messages are sent to $\mathcal{P}_3$ does not matter as it either has buffer-queue for each sender (Type:1-1) or it can consume messages out of order (Type:bag). In contrast, the ordering is important for Type:*-1 and Type:*-* asynchrony, in which case $\mathcal{P}_3$ will fail to consume if a_2 is sent before a_1.

Finally, the last row in Fig. 2 presents two peers and illustrates the difference between Type:bag and other types of asynchrony. The example shows that out of order consumption of messages is possible in Type:bag asynchrony.

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023

2.2. Formal description of peers & systems

Next, we present the formal models for asynchronously communicating peers and their interactions [12,13] taking into consideration the types of asynchrony described above.

Definition 1 *(Peer behavior).* A peer behavior (or simply a peer), denoted by $\mathcal{P}$, is a Finite State Machine (M, T, s_0, δ) where M is the union of input (M^{in}) and output (M^{out}) message sets, T is the finite set of states, $s_0 \in T$ is the initial state, and $\delta \subseteq T \times (M \cup \{\epsilon\}) \times T$ is the transition relation.

A transition $\tau \in \delta$ can be one of the following three types: (1) a send-transition of the form $(t_1, !m_1, t_2)$ which sends out a message $m_1 \in M^{\text{out}}$, (2) a receive-transition of the form $(t_1, ?m_2, t_2)$ which consumes a message $m_2 \in M^{\text{in}}$ from its input queue, and (3) an ϵ-transition of the form (t_1, ϵ, t_2). We write $t \xrightarrow{a} t'$ to denote that $(t, a, t') \in \delta$. □

We will focus on deterministic peer behaviors, where $\forall t_1, t_2 : t \xrightarrow{a} t_1 \ \wedge \ t \xrightarrow{a} t_2 \Rightarrow (t_1 = t_2)$. Peer behaviors can be determinized following standard methods for translation of non-deterministic state machines to deterministic ones. Fig. 2 illustrates some example peers. The initial states are subscripted with 0. Each transition is represented by send or receive actions.

The behavior of the system resulting from the asynchronous communication between the peers depends on the type of asynchrony. As Type:*-1 asynchrony is the most common form of asynchrony and as our prior results [12,13] are on such systems, we will first describe the system corresponding to the Type:*-1 asynchrony and refer to it as the Type:*-1 system. Other types will be described as variants of the Type:*-1 system definition.

Definition 2 *(Type:*-1 system behavior).* A Type:*-1 system behavior (or simply a Type:*-1 system) over a set of peers $\langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$, where $\mathcal{P}_i = (M_i, T_i, s_{0i}, \delta_i)$ with $M_i = M_i^{\text{in}} \cup M_i^{\text{out}}$ and $\forall i, j : i \neq j, M_i^{\text{in}} \cap M_j^{\text{in}} = M_i^{\text{out}} \cap M_j^{\text{out}} = \emptyset$, is denoted by a state machine (possibly infinite state) ${}^{*-1}\mathcal{I} = (M, C, c_0, \Delta)$, where M is the set of messages, C is the set of states, c_0 is the initial state, and Δ is the transition relation defined as:

1. $M = \cup_i M_i$
2. $C \subseteq T_1 \times \mathcal{Q}_1 \times T_2 \times \mathcal{Q}_2 \times \ldots \times T_n \times \mathcal{Q}_n$ such that $\forall i \in [1..n] : \mathcal{Q}_i \subseteq (M_i^{\text{in}})^*$
3. $c_0 \in C$ such that $c_0 = (s_{-1}\epsilon, s_{02}, \epsilon, s_{03} \ldots, s_{0n}\epsilon)$; and
4. $\Delta \subseteq C \times M \times C$,
   for $c = (t_1, Q_1, t_2, Q_2, \ldots t_n, Q_n)$ and $c' = (t_1', Q_1', t_2', Q_2', \ldots t_n', Q_n')$
   (a) $c \xrightarrow{!m} c' \in \Delta$ if $\exists i, j \in [1..n] : m \in M_i^{\text{out}} \cap M_j^{\text{in}}$,
      i. $t_i \xrightarrow{!m} t_i' \in \delta_i$,
      ii. $Q_j' = Q_j m$,
      iii. $\forall k \in [1..n] : k \neq j \Rightarrow Q_k = Q_k'$ and
      iv. $\forall k \in [1..n] : k \neq i \Rightarrow t_k' = t_k$
   (b) $c \xrightarrow{?m} c' \in \Delta$ if $\exists i \in [1..n] : m \in M_i^{\text{in}}$
      i. $t_i \xrightarrow{?m} t_i' \in \delta_i$,
      ii. $Q_i = m Q_i'$,
      iii. $\forall k \in [1..n] : k \neq i \Rightarrow Q_k = Q_k'$ and
      iv. $\forall k \in [1..n] : k \neq i \Rightarrow t_k' = t_k$
   (c) $c \xrightarrow{\epsilon} c' \in \Delta$ if $\exists i \in [1..n]$
      i. $t_i \xrightarrow{\epsilon} t_i' \in \delta_i$,
      ii. $\forall k \in [1..n] : Q_k = Q_k'$ and
      iii. $\forall k \in [1..n] : k \neq i \Rightarrow t_k' = t_k$ □

In the above, we have assumed that for each message m, there is a unique sender and a receiver. This can be easily achieved by associating with each message a sender and a receiver. A Type:*-1 system state is described in terms of local states of the participating peers and their respective receive queues. The transitions describe the evolution of the system from one state to another via send, receive or internal (ϵ) actions. The send action (item 4a) is non-blocking and as a result of the send action, the message sent is added to the tail of the receive queue of the receiver (4a-ii). The receive (item 4b) is blocking because a receiver can only make a move on a receive action if message to be consumed is present at the head of the receive queue (4b-ii). The epsilon-labeled transition (item 4c) is presented to allow for internal actions in the peers; internal actions simply change the local state of the peer executing it and do not directly affect any other peers.

Fig. 3 illustrates the Type:*-1 system. Each configuration in the system is described using the local states of the peers along with the contents of the corresponding queue. The empty queue is denoted by $\emptyset$ and non-empty queue is denoted by $[m_1 m_2 \ldots m_k]$, where m_1 is at the head of the queue and m_k is at the tail of the queue. Initially, all queues are empty. In

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023

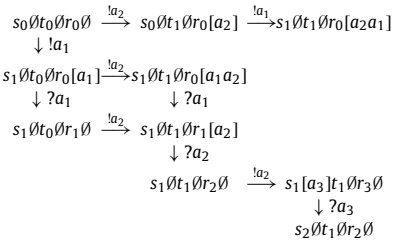


Fig. 3. Type:*-1 system corresponding to peers in row 2 of Fig. 2.

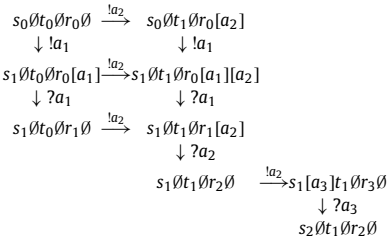


Fig. 4. Type:1-1 system corresponding to peers in row 2 of Fig. 2.

each configuration, the local state of a peer is followed by the queue contents for that peer. For instance, the configuration $s_1[a_3]t_1\emptyset r_3\emptyset$ corresponds to the case where the peer $\mathcal{P}_1$ is at state s_1 and its queue contains is a_3, and the peers $\mathcal{P}_2$ and $\mathcal{P}_3$, each with empty queues, are at states t_1 and r_3, respectively. The configurations evolve as the peers send messages (appended to the tail of the appropriate buffer-queue) or receive messages (consumed from the head of the receiver's buffer-queue).

Note that a Type:*-1 system describes the FIFO n-1 communication in distributed asynchronous systems [18,19]. The communication model ensures that

If a message m_1 is sent before m_2 to the same receiver, then m_1 is consumed before m_2.

The presence of one buffer for each receiver ensures that the messages are consumed by a peer in the order in which they are delivered to the receiver's buffer.

Definition 3 *(Type:1-1 system behavior).* A Type:1-1 system over n peers, denoted by ${}^{1-1}\mathcal{I} = (M, C, c_0, \Delta)$, consists of configurations described as a tuple of local states of n peers and $n \times (n-1)$ queues. In addition to the configuration setup, for the same set of peers, a Type:1-1 system differs from a Type:*-1 system in terms of the impact of send and receive actions. A send transition between configurations results in addition of the sent message to the tail of a queue Q_{ij}, where i and j are the indices of the sender and receiver peers, respectively. A receive transition, on the other hand, results in removal of message being consumed from the head of the queue Q_{ij}, where the receiver peer j consumes a message sent by the sender peer i. □

Going back to the example in Fig. 3, in the Type:*-1 asynchrony, the system is blocked in configuration $s_1\emptyset t_1\emptyset r_0[a_2a_1]$ as the peer $\mathcal{P}_3$ cannot consume a_2 before a_1. However, if Type:1-1 asynchrony is considered, then $\mathcal{P}_3$ would have two buffer-queues—one for the messages sent by $\mathcal{P}_1$ and the other for the messages sent by $\mathcal{P}_2$. In other words, a_1 and a_2 will be in two different buffer-queues, and therefore, $\mathcal{P}_3$ will be able to consume them (first a_1 and then a_2) even when a_2 is sent by $\mathcal{P}_2$ before a_1 is sent by $\mathcal{P}_1$. Fig. 4 illustrates this Type:1-1 system. In this case, the configurations are denoted by the local states of the peers—each such state is followed by the queues corresponding to the peer presenting the queue-contents. For instance, at the configuration $s_1\emptyset t_1\emptyset r_0[a_1][a_2]$ in Fig. 4, the peer $\mathcal{P}_3$ is at the local state r_0 and it has two buffer-queues; in one of them a_1 is pending to be consumed and in the other, a_2 is pending to be consumed. They are in two different queues as the messages are sent from two different sending peers.

As per [18,19], Type:1-1 system corresponds to the behavior of FIFO 1-1 communication in distributed system, where the following condition holds

If m_1 and m_2 are sent by a sender to a receiver, then the receiver consumes the messages in the order in which they are sent.

The above is referred to as FIFO 1-1 communication model in [19].

This condition is *weaker* than the causal ordering condition as the causal ordering spans across multiple senders. The difference between the strengths of the conditions between causal ordering communication and simple FIFO communication is captured explicitly in our models in terms of the number of buffer-queues involved in the communication. As has been mentioned before, the behavior of the Type-:1-1 system is less restrictive than that of the Type:*-1 system. Messages from *two different senders* can be consumed out of order by a receiver in a Type:1-1 system. In contrast, in Type:*-1 systems, the messages are consumed in the order in which they are sent by different senders.

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023

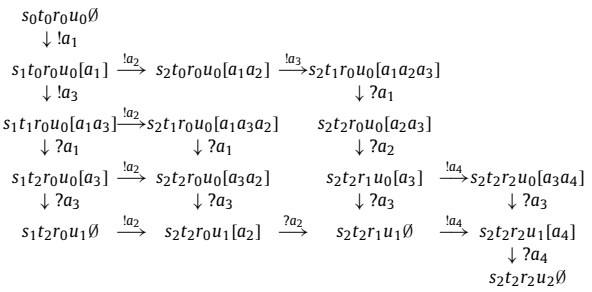


Fig. 5. Type:*-* system corresponding to peers in row 1 in Fig. 2.

Definition 4 *(Type:*-* system behavior).* A Type:*-* system over n peers, denoted by ${}^{*-*}\mathcal{I} = (M, C, c_0, \Delta)$, consists of configurations described as a tuple of local states of n peers and 1 queue. In addition to the configuration setup, for the same set of peers, a Type:*-* system differs from a Type:*-1 system in terms of the impact of send and receive actions. A send transition between configurations results in addition of the sent message to the tail of the only queue Q. A receive transition, on the other hand, results in removal of message being consumed from the head of the queue Q. □

Fig. 5 presents Type:*-* system from the peers in row 1 of Fig. 2. Each configuration in the system is denoted by the local states of the participating peers followed by the contents of the only buffer-queue present in Type:*-* system. Note that in such a system, a_2 cannot be consumed before consuming a_1 (unlike in other types of asynchrony)—therefore, the sequence of actions $!a_1\ !a_2\ ?a_2\ !a_4\ldots$ resulting in the sequence of send actions $a_1a_2a_4\ldots$ is not possible in this Type:*-* system. The Type:*-* communication is same as the FIFO n-n communication model discussed in [19], where the messages are globally ordered and are consumed in the order they are sent.

Finally consider the Type:bag systems. Recall that, Type:bag systems have three variants depending on the number of buffers involved in the asynchronous communication between peers. We proceed with the definition of the Type:bag system containing just one buffer.

Definition 5 *(Type:bag system behavior with one buffer).* A Type:bag system over n peers, denoted by ${}^{\mathtt{bag}}\mathcal{I} = (M, C, c_0, \Delta)$, consists of configurations described as a tuple of local states of n peers and 1 buffer. In addition to the configuration setup, the Type:bag system differs from Type:*-1 system in terms of the impact of send and receive actions. A send transition between configurations results in addition of the sent message to the only buffer Q. A receive transition, on the other hand, results in removal of message being consumed from the buffer Q. □

The Type:bag system corresponds to the Fully Asynchronous Communication model discussed in [19], where no order on the sending of message is imposed.

Other variants of a Type:bag system can be described in a similar fashion. Recall the two important aspects of asynchronous systems: the send actions are never blocked as the buffers are considered to have unbounded capacity; and the receive actions depend on whether or not the message to be consumed is present in an accessible position in the buffer (for instance at the head of the buffer for FIFO ordered consumption). However, for Type:bag systems, the consumption of messages does not depend on the order in which messages are sent—as messages are buffered in a multiset instead of a FIFO queue, they can be consumed in any order. Therefore, the send and receive actions in a Type:bag system are not dependent on the number of buffers involved in asynchronous communication. In other words, the behavior of a Type:bag system, as defined in Definition 5 captures the behavior of all other variants (in terms of number of buffers) of Type:bag systems.

While the Definition 5 for Type:bag systems is straightforward, we will consider an **alternate but equivalent** definition that naturally captures out-of-order consumption of send-messages. We will also use this definition to present the results in this paper.

Definition 6 *(Alternate definition of Type:bag system behavior).* A Type:bag system over n peers, denoted by ${}^{\mathtt{bag}}\mathcal{I} = (M, C, c_0, \Delta)$, consists of configurations described as a tuple of local states of n peers and $|Mout|$ buffers, where $|Mout|$ denotes the number of distinct messages that can be sent by any peers (recall that, each message has a unique sender and a unique receiver). Whenever a message m is sent, it is added to the buffer Q_m corresponding to the message m. A $?m$ action results in consumption of message from the buffer Q_m. □

It is immediate that Definitions 5 and 6 describe the same behavior of asynchronous systems where the ordering of distinct messages being sent is not considered. This is because, the send actions remain unblocked as the buffer capacities are considered to be unbounded. If a peer starts its behavior by performing $!m$ in a system as per Definition 5, then the

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023

same peer can initiate its behavior by performing the same send action in the system as per Definition 6. If a peer starts by performing $!m_1$ followed by consumption of n_1 (due to action $?n_1$) followed by the consumption n_2 (due to action $?n_2$) in a system behaving as per the Definition 5, then there exists some other peer(s) that must have delivered the messages n_1 and n_2 in the system buffer before they are consumed. Note that, the messages need not be sent in the order in which they are consumed, as the buffer can be accessed out of order for consumption of messages. Therefore, the possible sequences of send actions include any order between m_1, n_1 and n_2. The same sequences are also possible in the system behavior as per the Definition 6; this is because, each message is delivered in buffers corresponding to it, and therefore, out of order consumption of messages is permitted. Proceeding further, if a sequence of actions is possible in the system behaving as per the Definition 5, then the same sequence is also possible in a system behaving as per the Definition 6. In the rest of the paper, we will use the Definition 6 for Type:bag systems, which facilitates in presenting our results in an uniform fashion for the different types of asynchronous systems discussed in this paper.

Definition 7 *(k-Bounded Type:*-1 system).* A k-bounded Type:*-1 system (denoted by ${}^{*-1}\mathcal{I}_k$) is a system where the receive queue size for any peer is k. The k-bounded system behavior is, therefore, defined by augmenting condition 4(a) in Definition 2 to include the condition $|Q_j| < k$, where $|Q_j|$ denotes the length of the queue for peer j.

The k-bounded variant of other types is defined in similar fashion by constraining the capacity of the queue(s) or buffer(s). □

In a k-bounded system the send actions are blocked when the corresponding receive queue/buffer, where the sent message is supposed to be buffered, is full (i.e., it already contains k messages pending to be consumed by the receiver). Therefore, k-bounded systems have a finite state-space.

The behavior of any system resulting from synchronous composition of the participating peers is denoted by $\mathcal{I}_0$. Formally, we describe the behavior of a synchronous system as follows.

Definition 8 *(Synchronous system behavior).* A synchronous system over a set of peers $\langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$, where $\mathcal{P}_i = (M_i, T_i, s_{0i}, \delta_i)$ with $M_i = M_i^{\text{in}} \cup M_i^{\text{out}}$ and $\forall i, j : i \neq j, M_i^{\text{in}} \cap M_j^{\text{in}} = M_i^{\text{out}} \cap M_j^{\text{out}} = \emptyset$, is denoted by a finite state machine $\mathcal{I}_0 = (M, C, c_0, \Delta)$, where M is the set of messages, C is the set of states, c_0 is the initial state, and Δ is the transition relation defined as

1. $M = \cup_i M_i$
2. $C \subseteq T_1 \times T_2 \times \ldots \times T_n$
3. $c_0 \in C$ such that $c_0 = (s_{01}, s_{02}, \ldots, s_{0n})$
4. $\Delta \subseteq C \times M \times C$

   for $c = (t_1, t_2, \ldots t_n)$ and $c' = (t'_1, t'_2, \ldots t'_n)$ $c \xrightarrow{!m} c' \in \Delta$ if

   $\exists i, j \in [1..n] : m \in M_i^{\text{out}} \cap M_j^{\text{in}}$, such that $t_i \xrightarrow{!m} t'_i \in \delta_i$ and $t_j \xrightarrow{?m} t'_j \in \delta_j$; and $\forall k \in [1..n] : k \notin i, j \Rightarrow t'_k = t_k$.

In a synchronous system, the sender and receiver of messages move in lock-step, i.e., a sender can only send a message when the receiver is ready to consume the message, and the act of sending and receiving messages is synchronized.

Notation. We will use the notation $\mathcal{I}$, $\mathcal{I}_k$ $(k \geq 1)$ and $\mathcal{I}_0$, respectively, to denote unbounded buffer, k-bounded asynchronous system of any type, and synchronous system. We use the notation $\overset{!m}{\rightsquigarrow}$ to denote a sequence of transitions in a system containing zero or more transitions over actions in $\{\epsilon\} \cup M^{\text{in}}$ and a single transition over $!m$.

2.3. Send sequences & languages

The following concepts form the basis for describing the interaction behavior of the system, where the interaction is viewed as messages sent from one peer to another [12].

Definition 9 *(Language equivalence).* Given a system $\mathcal{I} = (M, C, c_0, F, \Delta)$ over a set of peers $\langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$, a path π is a finite or infinite sequence of configurations of the form

$$c_0 \xrightarrow{a_1} c_1 \xrightarrow{a_2} c_2 \xrightarrow{a_3} \ldots$$

where $a_i \in M$. We say that m is a send action if $\exists k : !m = a_k$. Furthermore, m_j is the j-th send in the path π if $!m_j = a_i$ and there are $j-1$ sends in π's prefix: $c_0 \xrightarrow{a_1} c_1 \xrightarrow{a_2} c_2 \ldots \xrightarrow{a_{i-1}} c_{i-1}$.

Given a path π in a system $\mathcal{I} = (M, C, c_0, F, \Delta)$, a send sequence in π is $m_1 m_2 m_3 \ldots$ such that m_j is the j-th send in the π.

The language of $\mathcal{I} = (M, C, c_0, F, \Delta)$, denoted by $\mathcal{L}(\mathcal{I})$, is the set of sequences of send actions on any (finite or infinite) path in $\mathcal{I}$. Systems $\mathcal{I}$ and $\mathcal{I}'$ are language equivalent if and only if $\mathcal{L}(\mathcal{I}) = \mathcal{L}(\mathcal{I}')$. □

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023

Proposition 1 *(Language ordering). The following holds:*

$$\begin{aligned} & \mathcal{L}({}^{*-*}\mathcal{I}) \subseteq \mathcal{L}({}^{*-1}\mathcal{I}) \subseteq \mathcal{L}({}^{1-1}\mathcal{I}) \subseteq \mathcal{L}({}^{\mathtt{bag}}\mathcal{I}) \text{ and} \\ \forall k > 0: \quad & \mathcal{L}({}^{*-*}\mathcal{I}_k) \subseteq \mathcal{L}({}^{*-1}\mathcal{I}_k) \subseteq \mathcal{L}({}^{1-1}\mathcal{I}_k) \subseteq \mathcal{L}({}^{\mathtt{bag}}\mathcal{I}_k) \end{aligned}$$

Proof. The ordering in terms of execution (and therefore, the language) for the different types of systems has been proved in [19]. In the following, for the sake of completeness we present an overview of the proof in the context of language described in terms of send-sequences.

The proof follows directly from the number of buffers and type of buffering mechanism. For the unbounded buffer system, recall that the send actions are never blocked. A receive action is only allowed when the message to be consumed due to the receive action are present in an accessible position in the buffer (e.g., head of the queue for FIFO buffers). The statements below follow from the definition of the behavior of the systems:

- A receiver peer in ${}^{\mathtt{bag}}\mathcal{I}$ can consume a message if it has been sent and is present in buffer corresponding to the message.
- A receiver peer in ${}^{1-1}\mathcal{I}$ can consume a message if it has been sent and is present at the head of the buffer corresponding to the sender and the receiver of the message.
- A receiver peer in ${}^{*-1}\mathcal{I}$ can consume a message if it has been sent and is present at the head of the buffer corresponding to the receiver peer.
- A receiver peer in ${}^{*-*}\mathcal{I}$ can consume a message if it has been sent and is present at the head of the buffer shared by all the peers.

Therefore, if a peer $\mathcal{P}$ needs to consume a sequence of messages σ in a particular order before producing/sending a message m and if the messages in the sequence σ have been already sent, the systems that have more buffers will impose less restriction on the order in which the messages in σ are sent to allow $\mathcal{P}$ to sent its own message m. It is immediate that ${}^{\mathtt{bag}}\mathcal{I}$ is the least restrictive as each message has its own buffer. ${}^{1-1}\mathcal{I}$ is less restrictive than ${}^{*-1}\mathcal{I}$ system as the former has pair-wise communication buffer-queues as opposed to one buffer-queue per receiver in the latter. Finally, ${}^{*-*}\mathcal{I}$ system is the most restrictive as there is just one buffer-queue, which forces the ordering among the receivers based on the ordering in which messages (to be consumed by them) are sent.

In k-bounded systems, on the other hand, the number of messages pending to be consumed in any buffer is bounded by k. The above constraints on the consumption of messages from the corresponding buffers hold for the k-bounded systems as well. Therefore, if there exists a send sequence in ${}^{1-1}\mathcal{I}_k$, the same send sequence must be present in ${}^{\mathtt{bag}}\mathcal{I}_k$, because the latter has one buffer of size k per message as opposed to one buffer of size k per sender–receiver pair in the former. The reverse, however, is not true as ${}^{\mathtt{bag}}\mathcal{I}_k$ system has more buffers than the corresponding ${}^{1-1}\mathcal{I}_k$ system. Similarly, ${}^{*-1}\mathcal{I}_k$ send sequences are also present in ${}^{1-1}\mathcal{I}_k$ and not the other way around; and ${}^{*-*}\mathcal{I}_k$ send sequences are present in ${}^{*-1}\mathcal{I}_k$. □

Proposition 2 *(Ordering based on buffer-size). The following holds for any types of asynchronous system and for all k:* $\mathcal{L}(\mathcal{I}_k) \subseteq \mathcal{L}(\mathcal{I}_{k+1}) \subseteq \mathcal{L}(\mathcal{I})$.

Proof. This is due to the fact that buffer capacity restricts the possible behavior of the peers participating in the communication. □

Temporal properties. We consider temporal ordering of sends expressed in the logic of Linear Temporal Logic (LTL). The syntax of LTL property φ over the sequences of messages can be described as follows:

$$\varphi \rightarrow \text{true} \mid M^{\text{out}} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathtt{X}\varphi \mid \varphi \,\mathtt{U}\, \varphi$$

In the current context, the semantics of the LTL properties are given in terms of the infinite send-sequence π. We will use π^i to denote the i-th suffix of the sequence π. The property true is satisfied by any sequence. The property $m \in M^{\text{out}}$ is satisfied if the first message sent in the sequence π is m. The negation and conjunctive properties have the natural meaning (following the semantics of boolean operations). The property $\mathtt{X}\varphi$ is satisfied by the sequence π if and only if φ is satisfied by π^1. The property $\varphi_1 \,\mathtt{U}\, \varphi_2$ is satisfied by π if and only if there exists a π^j that satisfies φ_2 and for all $i < j : \pi^i$ satisfies φ_1. A system $\mathcal{I}$ satisfies an LTL property if and only if all send-sequences starting from the start state of $\mathcal{I}$ satisfies the LTL property.

For details of LTL, refer to [20].

Observation 1. *Given two systems* $\mathcal{I}$ *and* $\mathcal{I}'$*, if* $\mathcal{L}(\mathcal{I}) = \mathcal{L}(\mathcal{I}')$ *then for any LTL property* φ *over the send actions,* $\mathcal{I}$ *satisfies* ϕ *if and only if* $\mathcal{I}'$ *satisfies* φ.

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023

The above observation is key to the verifiability of LTL properties over send-actions in asynchronous systems with unbounded buffers. We will prove that under certain conditions, an asynchronous system with unbounded buffers exhibit the same behavior with respect to send actions (i.e., has the same language) as its synchronous counterpart (one where the participating peers communicate synchronously). When such conditions are satisfied, then the LTL properties over send actions are satisfied by the asynchronous system with unbounded buffers if and only if it is also satisfied by the corresponding synchronous system. As the synchronous system has finite state-space, it is automatically verifiable using standard LTL model checking techniques as realized in model checking tools such as Spin [21].

3. Deciding synchronizability

In the subsequent sections, we identify the condition under which one can guarantee that the system is language equivalent to its synchronous counterpart. We refer to such systems as **Synchronizable** systems. Verification of LTL properties (over sequences of messages sent) for synchronizable systems is decidable as the verification can be performed on the synchronous system which exhibits finite-state behavior. Formally, synchronizability is defined as follows.

Definition 10 *(Synchronizability).* An asynchronous systems $\mathcal{I}$ (of any type) is synchronizable with respect to the sequences of sends if and only if $\mathcal{L}(\mathcal{I}_0) = \mathcal{L}(\mathcal{I})$.

The contribution of this paper is to show that checking synchronizability is decidable for different types of asynchronous systems.

3.1. Hierarchy of synchronizability

We first show the ordering between the types of systems in terms of their synchronizability.

Theorem 1 *(Hierarchy of synchronizability). Let* ${}^{1\text{-}1}\mathcal{I}$, ${}^{*\text{-}1}\mathcal{I}$, ${}^{*\text{-}*}\mathcal{I}$ *and* ${}^{\mathtt{bag}}\mathcal{I}$ *be the Type:1-1, Type:*-1, Type:*-* and Type:bag systems, respectively, resulting from the corresponding types of asynchronous communications involving the same set of peers. Then,*

1. ${}^{\mathtt{bag}}\mathcal{I}$ *is synchronizable* $\Rightarrow$ ${}^{1\text{-}1}\mathcal{I}$ *is synchronizable*
2. ${}^{1\text{-}1}\mathcal{I}$ *is synchronizable* $\Rightarrow$ ${}^{*\text{-}1}\mathcal{I}$ *is synchronizable*
3. ${}^{*\text{-}1}\mathcal{I}$ *is synchronizable* $\Rightarrow$ ${}^{*\text{-}*}\mathcal{I}$ *is synchronizable*

Proof. Recall the ordering of the types in terms of the send sequences: Proposition 1. The proof of the first item is as follows:

	${}^{\mathtt{bag}}\mathcal{I}$ is synchronizable	
$\Leftrightarrow$	$\mathcal{L}({}^{\mathtt{bag}}\mathcal{I}_0) = \mathcal{L}({}^{\mathtt{bag}}\mathcal{I})$	From Definition 10
$\Leftrightarrow$	$\mathcal{L}({}^{1\text{-}1}\mathcal{I}_0) = \mathcal{L}({}^{\mathtt{bag}}\mathcal{I})$	As ${}^{1\text{-}1}\mathcal{I}_0$ is identical to ${}^{\mathtt{bag}}\mathcal{I}_0$
		Buffers do not have any role in synchronous communication
$\Rightarrow$	$\mathcal{L}({}^{1\text{-}1}\mathcal{I}_0) \supseteq \mathcal{L}({}^{1\text{-}1}\mathcal{I})$	From Proposition 1 : $\mathcal{L}({}^{1\text{-}1}\mathcal{I}) \subseteq \mathcal{L}({}^{\mathtt{bag}}\mathcal{I})$
$\Rightarrow$	$\mathcal{L}({}^{1\text{-}1}\mathcal{I}_0) = \mathcal{L}({}^{1\text{-}1}\mathcal{I})$	From Proposition 2 : $\mathcal{L}({}^{1\text{-}1}\mathcal{I}_0) \subseteq \mathcal{L}({}^{1\text{-}1}\mathcal{I})$
$\Rightarrow$	${}^{1\text{-}1}\mathcal{I}$ is synchronizable	From Definition 10

The proof for the other implications follows similar arguments. □

3.2. Synchronizability of Type:*-1 systems

In [12], we proved that synchronizability checking for Type:*-1 systems is decidable and presented an efficient way to perform the checking. We present the main result from [12] for the completeness of presentation of the decidability results in this paper. We first present the following Lemma which is central to proving the synchronizability for Type:*-1 systems.

Lemma 1. *If* $\mathcal{L}({}^{*\text{-}1}\mathcal{I}_0) = \mathcal{L}({}^{*\text{-}1}\mathcal{I}_1)$ *then for every sequence of sends* $\sigma = m_1 m_2 \dots m_l$ *in* ${}^{*\text{-}1}\mathcal{I}_1$ *resulting from a path*

$$t_0^1 \overset{!m_1}{\rightsquigarrow} t_1^1 \overset{!m_2}{\rightsquigarrow} \dots \overset{!m_l}{\rightsquigarrow} t_l^1$$

where some send action m is blocked at the state t_l^1 *(as the buffer of the receiver peer is full), there exists some other path in* ${}^{*\text{-}1}\mathcal{I}_1$ *resulting in the same send sequence* σ *followed by the send action m.*

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023

Proof. Given the path

$$t_0^1 \overset{!m_1}{\rightsquigarrow} t_1^1 \overset{!m_2}{\rightsquigarrow} \dots \overset{!m_l}{\rightsquigarrow} t_l^1 \text{ in } {}^{*-1}\mathcal{I}_1 \tag{1}$$

assume that the send action m is blocked because the receiver peer $\mathcal{P}$ cannot consume the pending message in its buffer at the configuration t_l^1.

As $\mathcal{L}({}^{*-1}\mathcal{I}_1) = \mathcal{L}({}^{*-1}\mathcal{I}_0)$, there exists a path,

$$t_0^0 \overset{!m_1}{\rightsquigarrow} t_1^0 \overset{!m_2}{\rightsquigarrow} \dots \overset{!m_l}{\rightsquigarrow} t_l^0 \text{ in } {}^{*-1}\mathcal{I}_0 \tag{2}$$

We prove by induction on the length of σ that there exists a path over the same sequence σ in ${}^{*-1}\mathcal{I}_1$ where every message sent to $\mathcal{P}$ is consumed immediately by the peer $\mathcal{P}$, and therefore, message m sent to $\mathcal{P}$ is not blocked. Let such a path be

$$t_0'^1 \overset{!m_1}{\rightsquigarrow} t_1'^1 \overset{!m_2}{\rightsquigarrow} \dots \overset{!m_l}{\rightsquigarrow} t_l'^1 \tag{3}$$

We will use the paths in Equations (1) and (2) to construct the above path.

We use $t_i^j{\downarrow_{\mathcal{P}}}$ and $t_i^j{\downarrow_{\mathtt{E}}}$ to denote the local states of the peer $\mathcal{P}$ and the local states of the peers ($\in \mathtt{E}$) other than $\mathcal{P}$ in state t_i^j, respectively. Note that, as t_0^1, t_0^0, $t_0'^1$ are start states in the system, the local states of all peers at these states are identical.

Base case: i=1. If $!m_1$ is a send action from some peer in $\mathtt{E}$ to another peer in $\mathtt{E}$, then the resulting next states of t_0^1 and $t_0'^1$ are identical.

If $!m_1$ is an action from some peer in $\mathtt{E}$ to the peer $\mathcal{P}$, then there exists a receive action $?m_1$ at the local state of peer $\mathcal{P}$ (as t_0^0 in ${}^{*-1}\mathcal{I}_0$ can also start with $!m_1$ send in the send-sequence). We construct $t_1'^1$ as follows: $t_1'^1{\downarrow_{\mathcal{P}}} = t_1^0{\downarrow_{\mathcal{P}}}$ and $t_1'^1{\downarrow_{\mathtt{E}}} = t_1^1{\downarrow_{\mathtt{E}}}$. Therefore, at $t_1'^1$, the message m_1 sent to $\mathcal{P}$ is already consumed by $\mathcal{P}$ and its buffer is empty.

If $!m_1$ is an action from peer $\mathcal{P}$ to some peer in $\mathtt{E}$, then the resulting next states of t_0^1 and $t_0'^1$ are also identical.

We can, therefore, construct a path of length 1 from $t_1'^0$ to $t_1'^1$ such that $t_1'^1{\downarrow_{\mathcal{P}}} = t_1^0{\downarrow_{\mathcal{P}}}$ and $t_1'^1{\downarrow_{\mathtt{E}}} = t_1^1{\downarrow_{\mathtt{E}}}$.

Induction Step. Let $\forall i \in [0, n < l] : t_i'^1{\downarrow_{\mathcal{P}}} = t_i^0{\downarrow_{\mathcal{P}}}$ and $t_i'^1{\downarrow_{\mathtt{E}}} = t_i^1{\downarrow_{\mathtt{E}}}$, i.e., in all these states, $t_i'^1$, the message queue of $\mathcal{P}$ is empty.

Using the same arguments as above, we can prove that $t_{i+1}'^1{\downarrow_{\mathcal{P}}} = t_{i+1}^0{\downarrow_{\mathcal{P}}}$ and $t_{i+1}'^1{\downarrow_{\mathtt{E}}} = t_{i+1}^1{\downarrow_{\mathtt{E}}}$.

Therefore, paths (1) and (3) are over exactly the same sequence of send actions. Observe that the following holds:

- at state $t_l'^1$ peer $\mathcal{P}$ has an empty message queue
- at states t_l^1 and $t_l'^1$, the local states of the peers $\in E$ are identical.

As a result, at the configuration $t_l'^1$, the send action m is not blocked. □

We are now ready to prove the following theorem, which states that if the send sequences in ${}^{*-1}\mathcal{I}_0$ and ${}^{*-1}\mathcal{I}_1$ are identical, then increasing the size of the buffer does not exhibit new behavior in terms of sequences of sends.

Theorem 2. $\mathcal{L}({}^{*-1}\mathcal{I}_0) = \mathcal{L}({}^{*-1}\mathcal{I}_1) \Rightarrow \forall k \geq 1 : \mathcal{L}({}^{*-1}\mathcal{I}_k) = \mathcal{L}({}^{*-1}\mathcal{I}_{k+1})$.

Proof. We present here a proof-by-contradiction.

Assume that $\mathcal{L}({}^{*-1}\mathcal{I}_0) = \mathcal{L}({}^{*-1}\mathcal{I}_1)$ and $\exists k > 1 : \mathcal{L}({}^{*-1}\mathcal{I}_1) \neq \mathcal{L}({}^{*-1}\mathcal{I}_k)$. The second conjunct in the assumption along with Proposition 2 leads to

$$\exists k > 1 : \mathcal{L}({}^{*-1}\mathcal{I}_1) \subset \mathcal{L}({}^{*-1}\mathcal{I}_k)$$

That is, there exists a (finite) path (*witness*) in ${}^{*-1}\mathcal{I}_k$ distinguishing it from ${}^{*-1}\mathcal{I}_1$. In other words, both ${}^{*-1}\mathcal{I}_k$ and ${}^{*-1}\mathcal{I}_1$ have a path over the same sequence of send actions such that the path eventually leads to a state from where ${}^{*-1}\mathcal{I}_k$ can perform a send action which is not possible in ${}^{*-1}\mathcal{I}_1$. As we are only interested in the send-sequences, consider that such a path with l send actions is

$$t_0^k \overset{!m_1}{\rightsquigarrow} t_1^k \overset{!m_2}{\rightsquigarrow} \dots \overset{!m_l}{\rightsquigarrow} t_l^k \text{ in } {}^{*-1}\mathcal{I}_k \tag{4}$$

and the corresponding path in ${}^{*-1}\mathcal{I}_1$ that mimics the above till the l send actions is

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023

$$t_0^1 \overset{!m_1}{\rightsquigarrow} t_1^1 \overset{!m_2}{\rightsquigarrow} \dots \overset{!m_l}{\rightsquigarrow} t_l^1 \text{ in } {}^{*-1}\mathcal{I}_1 \tag{5}$$

such that $\forall j \in [0..l] : t_j^k = t_j^1$.

To allow for the difference between ${}^{*-1}\mathcal{I}_1$ and ${}^{*-1}\mathcal{I}_k$, assume that t_l^k is capable of realizing $\overset{!m}{\rightsquigarrow}$ which is not possible from t_l^1.

Now, we can use the Lemma 1, which ensures that there exists some path in ${}^{*-1}\mathcal{I}_1$ other than the one in Equation (5) over the same sequence of sends $m_1 m_2 \dots m_l$ following which m is possible. This contradicts our assumption that path (4) is a witness distinguishing ${}^{*-1}\mathcal{I}_k$ and ${}^{*-1}\mathcal{I}_1$. □

Theorem 3 *(Deciding Type:*-1 synchronizability). $\mathcal{L}({}^{*-1}\mathcal{I}_0) = \mathcal{L}({}^{*-1}\mathcal{I}_1)$ if and only if ${}^{*-1}\mathcal{I}$ is (language) synchronizable.*

Proof. We know from Theorem 2, $\mathcal{L}({}^{*-1}\mathcal{I}_0) = \mathcal{L}({}^{*-1}\mathcal{I}_1) \Rightarrow \forall k \geq 1 : \mathcal{L}({}^{*-1}\mathcal{I}_k) = \mathcal{L}({}^{*-1}\mathcal{I}_{k+1})$. This further leads to

$$\mathcal{L}({}^{*-1}\mathcal{I}_0) = \mathcal{L}({}^{*-1}\mathcal{I}_1) \Rightarrow \mathcal{L}({}^{*-1}\mathcal{I}_0) = \mathcal{L}({}^{*-1}\mathcal{I}) \tag{6}$$

Next,

$$\begin{aligned} \mathcal{L}({}^{*-1}\mathcal{I}_0) = \mathcal{L}({}^{*-1}\mathcal{I}) &\Rightarrow \mathcal{L}({}^{*-1}\mathcal{I}_0) = \mathcal{L}({}^{*-1}\mathcal{I}_k)\ \forall k \geq 1 \text{ see Proposition 2} \\ &\Rightarrow \mathcal{L}({}^{*-1}\mathcal{I}_0) = \mathcal{L}({}^{*-1}\mathcal{I}_1) \end{aligned} \tag{7}$$

From Equations (6) and (7), it follows that $\mathcal{L}({}^{*-1}\mathcal{I}_0) = \mathcal{L}({}^{*-1}\mathcal{I}_1) \Leftrightarrow \mathcal{L}({}^{*-1}\mathcal{I}_0) = \mathcal{L}({}^{*-1}\mathcal{I})$. □

3.3. Synchronizability of Type:1-1 systems

We now focus on the other types of asynchronous systems. The proof methodology follows the same strategy as in Section 3.2. The following lemma, which holds for Type:1-1 communication as well (proof by construction is similar to Lemma 1), forms the basis of the synchronizability condition.

Lemma 2. *If $\mathcal{L}({}^{1-1}\mathcal{I}_0) = \mathcal{L}({}^{1-1}\mathcal{I}_1)$ then for every sequence of sends $\sigma = m_1 m_2 \dots m_l$ in ${}^{1-1}\mathcal{I}_1$ resulting from a path*

$$t_0^1 \overset{!m_1}{\rightsquigarrow} t_1^1 \overset{!m_2}{\rightsquigarrow} \dots \overset{!m_l}{\rightsquigarrow} t_l^1$$

where some send action m is blocked at the state t_l^1 (as the buffer of the sender–receiver pair is full), there exists some other path resulting in the same send sequence σ followed by the send action m.

Theorem 4 *(Deciding Type:1-1 synchronizability). $\mathcal{L}({}^{1-1}\mathcal{I}_0) = \mathcal{L}({}^{1-1}\mathcal{I}_1)$ if and only if ${}^{1-1}\mathcal{I}$ is (language) synchronizable.*

Proof. The proof proceeds in the same fashion as in Theorem 3. The primary objective is to show that

$$\mathcal{L}({}^{1-1}\mathcal{I}_0) = \mathcal{L}({}^{1-1}\mathcal{I}_1) \Rightarrow \forall k \geq 1 : \mathcal{L}({}^{1-1}\mathcal{I}_k) = \mathcal{L}({}^{1-1}\mathcal{I}_{k+1})$$

which can be proved using the same arguments as in Theorem 2. The only difference stems from the fact that instead of one buffer per receiver as in Type:*-1 system, Type:1-1 system has one buffer per sender–receiver pair.

There is a path $t_0^k \overset{!m_1}{\rightsquigarrow} t_1^k \overset{!m_2}{\rightsquigarrow} \dots \overset{!m_l}{\rightsquigarrow} t_l^k$ in ${}^{1-1}\mathcal{I}_k$, where send action m is possible at t_l^k and all paths matching the send sequence $t_0^1 \overset{!m_1}{\rightsquigarrow} t_1^1 \overset{!m_2}{\rightsquigarrow} \dots \overset{!m_l}{\rightsquigarrow} t_l^1$ in ${}^{1-1}\mathcal{I}_1$ ends in a configuration (of the form t_l^1) from where send action m is not possible. The peer $\mathcal{P}$, which responsible for consuming m from a sender peer $\mathcal{P}'$, is not ready to consume the previous message sent to it by $\mathcal{P}'$ and, therefore, the message queue, corresponding to sender being $\mathcal{P}'$ and receiver $\mathcal{P}$, is full (contains 1 message to be consumed by $\mathcal{P}$).

The rest of the proof uses the Lemma 2, and is identical to the one presented in Theorems 2 and 3. □

3.4. Synchronizability of Type:*-* systems

The behavior of Type:*-* system is the most restrictive among the types of the asynchronous systems considered in this paper. As noted before, this is because the number of the buffer-queues in the Type:*-* system is one; which imposes an ordering among the receiver-peers (even when messages being consumed are not sent by the same sender-peer). As a result of such restriction, the property of the Type:*-* system, in terms of synchronizability, is different from the other types that we have considered. The following proposition discusses that synchronizability of Type:*-* system cannot be verified by checking the language equivalence between ${}^{*-*}\mathcal{I}_0$ and ${}^{*-*}\mathcal{I}_1$.

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023

Proposition 3. *$\mathcal{L}({}^{*-*}\mathcal{I}_0) = \mathcal{L}({}^{*-*}\mathcal{I}_1)$ does not imply $\mathcal{L}({}^{*-*}\mathcal{I}_0) = \mathcal{L}({}^{*-*}\mathcal{I})$ (i.e., does not imply that ${}^{*-*}\mathcal{I}$ is synchronizable).*

Proof. Unlike other types of systems, in Type:*-* system, a peer can block its own send actions if it cannot consume the messages sent to it. This is because there is just one buffer-queue. The above proposition can be proved using a simple witness example. Let there be two peers $\mathcal{P}_1$ and $\mathcal{P}_2$ with the following behavior: $\mathcal{P}_1$ can either consume m_2 or produce m_1, while $\mathcal{P}_2$ can either consume m_1 or produce m_2. Therefore, ${}^{*-*}\mathcal{I}_0$ and ${}^{*-*}\mathcal{I}_1$, resulting from (Type:*-*) asynchronous composition of these peers produce the send-sequences: m_1 or m_2. However, ${}^{*-*}\mathcal{I}_2$ (in general for all $k > 1$, ${}^{*-*}\mathcal{I}_k$) can produce the send-sequences m_1, m_2, m_1m_2 and m_2m_1. □

Intuitively, the Proposition 3 shows that the ${}^{*-*}\mathcal{I}_1$ may not have enough buffer-queue capacity (which is 1) to show that the system ${}^{*-*}\mathcal{I}$ is *not* synchronizable. This results in the following condition for synchronizability checking for the Type:*-* systems.

Theorem 5 *(Deciding Type:*-* synchronizability). Given n peers, the following holds for their interaction resulting from Type:*-* asynchrony: $\mathcal{L}({}^{*-*}\mathcal{I}_0) = \mathcal{L}({}^{*-*}\mathcal{I}_n)$ if and only if ${}^{*-*}\mathcal{I}$ is (language) synchronizable.*

Proof. From Proposition 2, it follows that

$$\mathcal{L}({}^{*-*}\mathcal{I}_0) = \mathcal{L}({}^{*-*}\mathcal{I}) \Rightarrow \forall k \geq 1 : \mathcal{L}({}^{*-*}\mathcal{I}_0) = \mathcal{L}({}^{*-*}\mathcal{I}_k) \Rightarrow \mathcal{L}({}^{*-*}\mathcal{I}_0) = \mathcal{L}({}^{*-*}\mathcal{I}_n)$$

To prove the reverse, we need to prove

$$\mathcal{L}({}^{*-*}\mathcal{I}_0) = \mathcal{L}({}^{*-*}\mathcal{I}_n) \Rightarrow \forall k \geq n : \mathcal{L}({}^{*-*}\mathcal{I}_k) = \mathcal{L}({}^{*-*}\mathcal{I}_{k+1})$$

where n is the number of peers.

We proceed by considering any send sequence σ of the form $m_0m_1 \ldots m_l$ from the start state of the system (i.e., from the start states of the peers participating in the system) such that

- $l \leq n$
- each m_i is from a distinct peer, i.e., no two messages are sent by the same peer.

This sequence is present in ${}^{*-*}\mathcal{I}_n$ and therefore, from the premise, it is also present in ${}^{*-*}\mathcal{I}_0$. Note that, in ${}^{*-*}\mathcal{I}_0$, the messages are consumed immediately after they are sent.

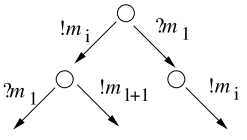
Assumption. Now assume that there is an extension (by one message m_{l+1} sent by $\mathcal{P}$) of the sequence σ such that the extension is present in ${}^{*-*}\mathcal{I}_k$ (for a $k > n$) but not present in ${}^{*-*}\mathcal{I}_n$.

There are two possible cases. In the first case, m_1 is consumed by some peer $\mathcal{P}'$ ($\neq \mathcal{P}$). As the sequence σ is present in ${}^{*-*}\mathcal{I}_0$, $\mathcal{P}'$ must be able to consume to m_1 too, which makes the contents in the buffer-queue $< n$. This will allow the sequence $m_1m_2 \ldots m_lm_{l+1}$ in the ${}^{*-*}\mathcal{I}_n$ and contradicts our assumption for the first case.

In the second case, m_1 is consumed by the peer $\mathcal{P}$. If the assumption holds, then we can infer that

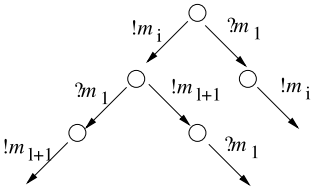
- $?m_1$ (consumption of m_1) and $!m_{l+1}$ (sending of m_{l+1}) are in two different behavioral path in $\mathcal{P}$, and
- $?m_1$ is not followed by $!m_{l+1}$.

If $\mathcal{P}$ participates in the sequence σ by sending m_i, then, the behavior of $\mathcal{P}$ must include the following structure: $!m_i$ is followed by a branch point, from where there is one branch where $?m_1$ and another branch where is $!m_{l+1}$. The scenario is illustrated below (note that the behavior must include $!m_i$ followed by $?m_1$ and the other way around because the send sequences containing m_im_1 and m_1m_i are allowed in ${}^{*-*}\mathcal{I}_n$ and therefore, in ${}^{*-*}\mathcal{I}_0$ as well).



Therefore, the peer $\mathcal{P}$ can send m_i, which can be immediately consumed by some peer other than $\mathcal{P}$ (as such a sequence is present in ${}^{*-*}\mathcal{I}_0$). At this point the buffer-queue is empty. After m_i is sent and consumed, it can be followed by m_1 sent by some peer other than $\mathcal{P}$, which in turn can be followed by m_{l+1} sent by $\mathcal{P}$. This sequence: $m_im_1m_{l+1}$ is present in the system ${}^{*-*}\mathcal{I}_n$ (note that the minimum n is 2). As per the premise, this sequence must be also present in ${}^{*-*}\mathcal{I}_0$. Therefore, $?m_1$ must be followed by $!m_{l+1}$. This contradicts the assumption in the second case. (On similar note, the sequence $m_im_{l+1}m_1$ is present in the ${}^{*-*}\mathcal{I}_n$ and ${}^{*-*}\mathcal{I}_0$ as well.)

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023



In short, there exists no extension of σ of length 1, that can distinguish between ${}^{*-*}\mathcal{I}_n$ and ${}^{*-*}\mathcal{I}_k$, for any $k > n$, when $\mathcal{L}({}^{*-*}\mathcal{I}_n) = \mathcal{L}({}^{*-*}\mathcal{I}_0)$.

Proceeding further, the sequence σ' equal to $m_1 m_2 \dots m_l m_{l+1}$ is present in both ${}^{*-*}\mathcal{I}_n$ and ${}^{*-*}\mathcal{I}_0$. We have also established that m_1 can be consumed by some receiver-peer. Therefore, the number of pending messages in the buffer-queue in the sequence σ' in the system ${}^{*-*}\mathcal{I}_n$ is $\leq n$. Using the arguments above, we can prove that if ${}^{*-*}\mathcal{I}_k$ $(k > n)$ contains any extension of σ' using one additional message, then the extension is also present in ${}^{*-*}\mathcal{I}_n$ and in ${}^{*-*}\mathcal{I}_0$ as well.

Thus, we conclude $\mathcal{L}({}^{*-*}\mathcal{I}_0) = \mathcal{L}({}^{*-*}\mathcal{I}_n)$ implies $\forall k \geq n : \mathcal{L}({}^{*-*}\mathcal{I}_n) = \mathcal{L}({}^{*-*}\mathcal{I}_k)$, where n is the total number of peers participating in the communication. □

3.5. Synchronizability of Type:bag systems

Finally, we consider Type:bag systems. Recall that, all variants of Type:bag systems with respect to the number of buffers are equivalent. Furthermore, we have defined the Type:bag systems such that every message has a buffer of its own (Definition 6). We first present the lemma, which is provable in the same fashion as Lemma 1.

Lemma 3. *If* $\mathcal{L}({}^{\mathtt{bag}}\mathcal{I}_0) = \mathcal{L}({}^{\mathtt{bag}}\mathcal{I}_1)$ *then for every sequence of sends* $\sigma = m_1 m_2 \dots m_l$ *in* ${}^{\mathtt{bag}}\mathcal{I}_1$ *resulting from a path*

$$t_0^1 \overset{!m_1}{\rightsquigarrow} t_1^1 \overset{!m_2}{\rightsquigarrow} \dots \overset{!m_l}{\rightsquigarrow} t_l^1$$

where some send action m is blocked at the state t_l^1 *(as the buffer holding the messages m is full), there exists some other path resulting in the same send sequence* σ *followed by the send action m.*

Theorem 6 *(Deciding Type:bag synchronizability).* $\mathcal{L}({}^{\mathtt{bag}}\mathcal{I}_0) = \mathcal{L}({}^{\mathtt{bag}}\mathcal{I}_1)$ *if and only if* ${}^{\mathtt{bag}}\mathcal{I}$ *is (language) synchronizable.*

Proof. Recall that, we are using Definition 6 to describe the behavior of Type:bag system. Each message has a buffer of its own, which can be trivially viewed as a queue.

The proof for $\mathcal{L}({}^{\mathtt{bag}}\mathcal{I}_0) = \mathcal{L}({}^{\mathtt{bag}}\mathcal{I}_1) \Rightarrow \forall k \geq 1 : \mathcal{L}({}^{\mathtt{bag}}\mathcal{I}_k) = \mathcal{L}({}^{\mathtt{bag}}\mathcal{I}_{k+1})$ proceeds with the similar arguments as in Theorem 2. There is a path $t_0^k \overset{!m_1}{\rightsquigarrow} t_1^k \overset{!m_2}{\rightsquigarrow} \dots \overset{!m_l}{\rightsquigarrow} t_l^k$ in ${}^{\mathtt{bag}}\mathcal{I}_k$, where send action m is possible at t_l^k and all paths matching the send sequence $t_0^1 \overset{!m_1}{\rightsquigarrow} t_1^1 \overset{!m_2}{\rightsquigarrow} \dots \overset{!m_l}{\rightsquigarrow} t_l^1$ in ${}^{\mathtt{bag}}\mathcal{I}_1$ ends in a configuration (of the form t_l^1) from where send action m is not possible. The peer $\mathcal{P}$, which responsible for consuming m not ready to consume m that was sent in the past, and therefore, the queue that holds all messages of type m is full (contains 1 message to be consumed by $\mathcal{P}$).

The rest of the proof uses the Lemma 3, and is identical to the one presented in Theorems 2 and 3. □

4. Related work & discussion

Automatic verification and analysis of asynchronous systems, modeled as communicating finite state machines, is undecidable, in general. This follows from the results by Brand and Zaphiropulo [7], where the authors prove that computing reachable state-space and deciding boundedness of the queues in a communicating finite state machine is undecidable. Therefore, a number of decidable subclasses have been proposed and analyzed. One such subclass, synchronizable systems, was introduced and investigated by Fu et al. in [10,11] in the context of Web service interactions. The primary objective is to analyze and verify the interactions in Web services against certain desired properties. The subclass of synchronizable systems differs from other subclasses in the sense that synchronizability requirement does not restrict the type of communication (e.g., lossy-communication [22,23], half-duplex communication [8] or local behavioral conformance in MPI programs [24–26]) or the topology of interaction (e.g., tree-based communication in well-queuing push-down systems [9, 27]). However, Fu et al. did not address the question of whether synchronizability checking can be done automatically and efficiently. The authors in [10,11] developed the sufficient condition for synchronizability and showed that a large number of asynchronous systems are synchronizable, and therefore, verifiable. As per the sufficient condition peers at any state can either send or receive but not both. This condition is referred to as the *autonomous condition*. Similar sufficient conditions have been developed in the work on session type [28–30], where the authors focus on reducing the problem of verifying correct interaction between communicating peers into a typing problem.

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023

The significance of this subclass of synchronizable systems depends on the conjecture that many asynchronous systems fall into this class, which makes their automatic analysis possible and which automates the verification of their correctness. In other words, even if the peers in the system communicate asynchronously with no fixed and known capacity for the buffers, the behavior resulting from their communication can be equivalently represented by a system with synchronous communication. Preliminary experiments on synchronizability are conducted using 86 channel contracts in Singularity OS [15]—an experimental OS developed by Microsoft with the primary design principle being process isolation. The contracts are protocols that communicating processes must follow in order to achieve a desired functionality. The main objective is to improve dependability of software systems. All inter-process communication occurs via message-passing over channels (or buffers), where the messages are consumed in the order in which they are sent. Therefore, their behavioral model corresponds to Type:*-1 system described in this paper. The experiments conducted in [13] have shown that all but two of the contracts are synchronizable; the contracts that are not synchronizable have bugs resulting in deadlocking communication [31].

The question of deciding synchronizability has been an open problem since its introduction in [10]. In [12], we have proved that synchronizability with respect to send-sequences for asynchronous systems communicating with one buffer-queue per receiver peer (Type:*-1 systems) is decidable. We have also proved in [13] synchronizability for Type:*-1 systems are decidable when one considers the reachability of synchronous states (states with no pending messages to be consumed) in addition to sequences of sends. In this paper, we build on our prior work and present a comprehensive study of different types of models of asynchronous systems and prove whether or not synchronizability of these systems is decidable.

There have been other studies on characterization of different asynchronous communication models [19]. We characterized asynchronous systems based on the number of buffers and the type of buffering. We present the ordering among these systems in terms of the send-sequences. We prove that synchronizability checking for these systems is decidable and can be performed effectively by checking for (language) equivalence between finite-buffer behavior of the systems.

In addition to deciding synchronizability, the significance of the results is two-fold. The first is immediate. Any asynchronous system exhibiting infinite-state behavior, if synchronizable, can be automatically verified by verifying its synchronous counter-part. This broadens the scope of application of automatic verification techniques.

Secondly, these decidability results also lead to deciding *realizability* of distributed systems. The question is whether one can develop/implement peers such that when they communicate in a distributed environment (i.e., asynchronously), they generate the sequences of sends that match the specification of the distributed system. This problem is referred to as *realizability*. The problem for MSC-graphs (an extension of Message-sequence charts) has been shown to be undecidable [32]. However, in [33], we proved that realizability checking with respect to *send-sequences* is decidable when the system is specified as a finite state machine. Given a specification $\mathcal{C}$ over messages exchanged between peers using finite state machines, $\mathcal{C}$ is realizable if and only if

- $\mathcal{I}^{\mathcal{C}}$ is synchronizable, and
- $\mathcal{C}$ is equivalent to $\mathcal{I}^{\mathcal{C}}$,

where $\mathcal{I}^{\mathcal{C}}$ is the behavior of the asynchronous system where peers' behaviors are obtained by projecting $\mathcal{C}$ on each participating peer. Note that the second item can be verified using $\mathcal{I}_0^{\mathcal{C}}$ as $\mathcal{I}^{\mathcal{C}}$ is synchronizable. This result considers Type:*-1 systems and uses the synchronizability of Type:*-1 systems from [12]. In other words, in [33], we have proved Type:*-1 realizability of specifications. Based on the results in this paper, we can immediately prove Type:1-1, Type:*-* and Type:bag realizability of specifications. The hierarchy of realizability also follows the same ordering as the synchronizability (see Section 3.1), i.e., a specification is Type:bag realizable implies it is Type:1-1 realizable, which implies, it is Type:*-1 realizable, which, in turn, implies it is Type:*-* realizable. In summary, the results cover a wide range of asynchronous systems and prove that synchronizability checking (and, therefore, realizability checking) for such systems is decidable. Perhaps the closest to the realizability problem is the problem of synthesis. In [34–36], the authors consider synthesizing distributed systems that conform to a regular language specification. The authors state that the algorithm for construction can be applied to synthesize communicating finite state machines (CFSM) with *bounded* communication channels. In contrast, realizability (whether or not there exist CFSMs whose interaction conforms to FSM specification) problem considers asynchronous communication over unbounded channels.

A different variety of synchronizability problem [37–39] involves determining if the heads of an n-tape automaton can be synchronized while reading the input (i.e., determining if, for each accepting run of an n-tape automaton, there exists an accepting run where the heads of the tapes are synchronized). Generalizations of this problem have also been studied for multitape pushdown automata and multitape Turing machines [40]. Note that, the input tapes studied in these earlier results are read-only input tapes unlike the communication channels that we study in this paper which support both read and write operations. We are not aware of any results that connect the synchronizability of asynchronous communication we study in this paper to the synchronizability of multitape automata, however, we believe that, this would be an interesting future research direction.

References

[1] A. Desai, V. Gupta, E. Jackson, S. Qadeer, S. Rajamani, D. Zufferey, P: safe asynchronous event-driven programming, in: Programming Languages Design and Implementation, PLDI, 2013, ACM, 2013, pp. 321–332.

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023

[2] Java message service, http://java.sun.com/products/jms/, 2015.
[3] Microsoft message queuing service, https://msdn.microsoft.com/en-us/library/aa967729%28v=vs.110%29.aspx, 2015.
[4] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D.F. Ferguson, Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More, Prentice Hall, 2005.
[5] G. Banavar, T.D. Chandra, R.E. Strom, D.C. Sturman, A case for message oriented middleware, in: 13th International Symposium on Distributed Computing, DISC, 1999, pp. 1–18.
[6] D.A. Menascé, MOM vs. RPC: communication models for distributed applications, IEEE Internet Computing 9 (2) (2005) 90–93.
[7] D. Brand, P. Zafiropulo, On communicating finite-state machines, J. ACM 30 (2) (1983) 323–342.
[8] G. Cécé, A. Finkel, Verification of programs with half-duplex communication, Inform. and Comput. 202 (2005) 166–190.
[9] S.L. Torre, P. Madhusudan, G. Parlato, Context-bounded analysis of concurrent queue systems, in: 14th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems, TACAS, 2008, pp. 299–314.
[10] X. Fu, T. Bultan, J. Su, Analysis of interacting BPEL web services, in: Proc. 13th Int. World Wide Web Conf., 2004, pp. 621–630.
[11] X. Fu, T. Bultan, J. Su, Synchronizability of conversations among web services, IEEE Trans. Softw. Eng. 31 (12) (2005) 1042–1055.
[12] S. Basu, T. Bultan, Choreography conformance via synchronizability, in: 20th International World Wide Web Conference, 2011, pp. 795–804.
[13] S. Basu, T. Bultan, M. Ouederni, Synchronizability for verification of asynchronously communicating systems, in: 13th International Conference on Verification, Model Checking, and Abstract Interpretation, 2012, pp. 56–71.
[14] Web service choregraphy description language, https://www.w3.org/TR/ws-cdl-10-primer/, 2006.
[15] Singularity design note 5: channel contracts. Singularity rdk documentation (v1.1), http://www.codeplex.com/singularity, 2004.
[16] J. Armstrong, Getting Erlang to talk to the outside world, in: Proceedings of the 2002 ACM SIGPLAN Workshop on Erlang, 2002, pp. 64–72.
[17] X. Défago, A. Schiper, P. Urbán, Total order broadcast and multicast algorithms: taxonomy and survey, ACM Comput. Surv. 36 (4) (2004) 372–421.
[18] B. Charron-Bost, F. Mattern, G. Tel, Synchronous, asynchronous, and causally ordered communication, Distrib. Comput. 9 (4) (1996) 173–191.
[19] F. Chevrou, A. Hurault, P. Quéinnec, On the Diversity of Asynchronous Communication, Tech. rep., Institut de Recherche en Informatique de Toulouse, France, May 2015.
[20] E. Clarke, O. Grumberg, D.A. Peled, Model Checking, The MIT Press, Cambridge, Massachusetts, 1999.
[21] G.J. Holzmann, The model checker spin, IEEE Trans. Softw. Eng. 23 (1997) 279–295.
[22] P.A. Abdulla, B. Jonsson, Verifying programs with unreliable channels, Inform. and Comput. 127 (2) (1996) 91–101.
[23] B. Masson, P. Schnoebelen, On verifying fair lossy channel system, in: Proceedings, Mathematical Foundations of Computer Science 2002: 27th International Symposium, MFCS 2002 Warsaw, Poland, August 26–30, 2002, Springer, Berlin, Heidelberg, 2002, pp. 543–555.
[24] R. Manohar, A.J. Martin, Slack elasticity in concurrent computing, in: Mathematics of Program Construction, MPC, 1998, pp. 272–285.
[25] S.F. Siegel, Efficient verification of halting properties for MPI programs with wildcard receives, in: 6th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI, 2005, pp. 413–429.
[26] S. Vakkalanka, A. Vo, G. Gopalakrishnan, R.M. Kirby, Precise dynamic analysis for slack elasticity: adding buffering without adding bugs, in: 17th Euro. MPI Conf. Advances in Message Passing Interface, 2010, pp. 152–159.
[27] A. Heußner, J. Leroux, A. Muscholl, G. Sutre, Reachability analysis of communicating pushdown systems, in: 13th Int. Conf. on Foundations of Software Science and Computational Structures, FOSSACS, 2010, pp. 267–281.
[28] K. Honda, V.T. Vasconcelos, M. Kubo, Language primitives and type discipline for structured communication-based programming, in: 7th European Symp. on Programming Languages and Systems, ESOP, 1998, pp. 122–138.
[29] K. Honda, N. Yoshida, M. Carbone, Multiparty asynchronous session types, in: Proceedings of Symposium Principles of Programming Languages, 2008, pp. 273–284.
[30] P. Deniélou, N. Yoshida, Multiparty session types meet communicating automata, in: Programming Languages and Systems – 21st European Symposium on Programming, ESOP, 2012, 2012, pp. 194–213.
[31] Z. Stengel, T. Bultan, Analyzing singularity channel contracts, in: Proc. 18th Int. Symp. on Software Testing and Analysis, ISSTA, 2009, pp. 13–24.
[32] R. Alur, K. Etessami, M. Yannakakis, Realizability and verification of {MSC} graphs, Theoret. Comput. Sci. 331 (1) (2005) 97–114.
[33] S. Basu, T. Bultan, M. Ouederni, Deciding choreography realizability, in: 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL, 2012, pp. 191–202.
[34] B. Genest, On implementation of global concurrent systems with local asynchronous controllers, in: 16th International Conference on Concurrency Theory, Springer, Berlin, Heidelberg, 2005, pp. 443–457.
[35] B. Genest, H. Gimbert, A. Muscholl, I. Walukiewicz, Optimal Zielonka-type construction of deterministic asynchronous automata, in: 37th International Colloquium on Automata, Languages and Programming, Springer, Berlin, Heidelberg, 2010, pp. 52–63.
[36] S. Akshay, I. Dinca, B. Genest, A. Stefanescu, Implementing realistic asynchronous automata, in: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 24, Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 2013, pp. 213–224.
[37] Ö. Egecioglu, O.H. Ibarra, N.Q. Trân, Multitape NFA: weak synchronization of the input heads, in: Proceedings of the 38th Conference on Current Trends in Theory and Practice of Computer Science, 2012, pp. 238–250.
[38] O.H. Ibarra, N.Q. Trân, On synchronized multi-tape and multi-head automata, Theoret. Comput. Sci. 449 (2012) 74–84.
[39] O.H. Ibarra, N.Q. Trân, How to synchronize the heads of a multitape automaton, Internat. J. Found. Comput. Sci. 24 (6) (2013) 799–814.
[40] O.H. Ibarra, N.Q. Trân, Weak synchronization and synchronizability of multi-tape pushdown automata and Turing machines, J. Autom. Lang. Comb. 19 (1–4) (2014) 119–132.

Please cite this article in press as: S. Basu, T. Bultan, On deciding synchronizability for asynchronously communicating systems, Theoret. Comput. Sci. (2016), http://dx.doi.org/10.1016/j.tcs.2016.09.023