A fresh look at testing for asynchronous communication *

Puneet Bhateja[1], Paul Gastin[2], and Madhavan Mukund[1]

[1] Chennai Mathematical Institute, Chennai, India
{puneet,madhavan}@cmi.ac.in
[2] LSV, ENS de Cachan & CNRS, France
Paul.Gastin@lsv.ens-cachan.fr

Abstract. Testing is one of the fundamental techniques for verifying if a computing system conforms to its specification. We take a fresh look at the theory of testing for message-passing systems based on a natural notion of observability in terms of input-output relations. We propose two notions of test equivalence: one which corresponds to presenting all test inputs up front and the other which corresponds to interactively feeding inputs to the system under test. We compare our notions with those studied earlier, notably the equivalence proposed by Tretmans. In Tretmans' framework, asynchrony is modelled using synchronous communication by augmenting the state space of the system with queues. We show that the first equivalence we consider is strictly weaker than Tretmans' equivalence and undecidable, whereas the second notion is incomparable. We also establish (un)decidability results for these equivalences.

1 Introduction

Testing is a fundamental activity in verifying the correctness of systems. In this paper, we focus on testing in the restricted context of reactive systems. A theoretical foundation for testing labelled transition systems was laid in the framework of process algebra, where an operational notion of testing was defined and shown to have an extensional semantic characterization in terms of failures [7, 8]. These ideas were expanded and elaborated in the work of Tretmans [16, 17], in the form of an extensive theory of *conformance testing*—testing when an implementation conforms to its specification. This theory has been used to develop automated tools for testing, such as the TGV system [13].

The initial focus on formalizing testing for labelled transition systems was on synchronous communication, where the send and receive actions for each communication occur simultaneously. However, most communication protocols are based on asynchronous communication, or message-passing via buffers that can be modelled as queues. Many questions remain unanswered about the testing process for such systems. In addition to the usual problem of optimizing the size

* Partially supported by *Timed-DISCOVERI*, a project under the Indo-French Networking Programme.

of test suites without sacrificing coverage, there are also additional issues to be considered, such as the possibility of distributing tests [12].

The first major effort to develop an effective theory of testing for asynchronous communication originated in the thesis of Tretmans [16], in which asynchronous communication is reduced to synchronous communication in a model augmented with infinite queues. A refined version of this theory is presented in [17], in terms of input-output transition systems that interact synchronously.

In parallel, asynchronous communication has also been an active area of study in the field of process algebra. A process algebra with asynchronous communication, whose semantics is given in terms of auxiliary data structures such as queues to store the channel state, has been formulated in [5, 6]. The focus of this work is to identify semantic equivalences that are congruences with respect to process algebraic operators, rather than to formalize testing equivalence per se. Later papers have considered testing equivalence for process algebras with asynchronous communication [4, 2]. In these approaches, there are no explicit channels between processes. Instead, all messages are emitted into a shared pool and can be consumed in any order by receiving processes. This approach towards modelling asynchronous communications is more suitable for name-passing calculi such as the π-calculus, but it is difficult to import any intuition or results to our setting in which processes with a fixed network topology exchange messages through point-to-point queues.

Our approach to testing is to consider a natural notion of observability for systems based on input-output pairs. Using this notion, we propose two notions of test equivalence. The first corresponds to presenting all test inputs up front while the other corresponds to interactively feeding inputs to the system under test. We show that the first equivalence is strictly weaker than Tretmans' equivalence, whereas the second notion is incomparable. Our work is closely related to the queued quiescent trace approach of [14], as explained in Section 4.

We also establish decidability results for these equivalences. We show that the weaker equivalence that we define is undecidable for finite-state systems, as is the equivalence proposed by Tretmans. However, the stronger equivalence is decidable for *well-structured* transition systems. We also show that our weaker notion of equivalence is decidable if we record the input-output behaviour of a system as an unlabelled message sequence chart [11].

The paper is organized as follows. In the next section, we introduce our formal model of asynchronously communicating systems. Three notions of asynchronous testing are introduced in Section 3. We describe the interrelationships between these notions in Section 4 and prove decidability and undecidability results in Section 5. We conclude with a brief discussion on directions for future work.

2 The Model

We work in the setting of labelled transition systems. A *labelled transition system* is a structure $TS = (S, I, \Sigma, \rightarrow)$ where S is a set of states with a subset I of initial states, Σ is an alphabet of actions and $\rightarrow \subseteq S \times \Sigma \times S$ is a labelled transition relation. We will write $s \xrightarrow{a} s'$ to denote that $(s, a, s') \in \rightarrow$.

We are interested in asynchronous systems that interact with their environment by sending and receiving messages. We represent this interaction abstractly by partitioning Σ into two sets: Σ_i, the set of input actions, and Σ_o, the set of output actions. We normally use a, b, c to denote input actions, x, y, z to denote output actions and Greek letters α, β to denote generic actions from Σ.

An action α is said to be *enabled* at a state $s \in S$ if there is some transition $s \xrightarrow{\alpha} s'$. We write $s \xrightarrow{\alpha}$ to denote that α is enabled at s and $s \stackrel{\alpha}{\nrightarrow}$ to denote that α is not enabled at s. We can extend this to sets of actions: for $X \subseteq \Sigma$, $s \xrightarrow{X}$ if $s \xrightarrow{\alpha}$ for *some* $\alpha \in X$ and $s \stackrel{X}{\nrightarrow}$ if $s \stackrel{\alpha}{\nrightarrow}$ for *every* $\alpha \in X$. A state s is said to *refuse* a set $X \subseteq \Sigma$ of actions if $s \stackrel{X}{\nrightarrow}$. A state s is *quiescent* if it refuses Σ_o.

A run of the transition system TS is a sequence of transitions of the form $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_m} s_m$ where $s_0 \in I$. We call this a run of TS over the word $\alpha_1 \alpha_2 \ldots \alpha_m$. Let $L(TS) = \{w \in \Sigma^* \mid TS \text{ admits a run over } w\}$. It is easy to see that $L(TS)$ is a prefix-closed language.

Without loss of generality, we assume that in the transition systems we consider, there is no loop $s_0 \xrightarrow{x_1} s_1 \xrightarrow{x_2} \cdots \xrightarrow{x_m} s_m = s_0$ labelled by a sequence of output labels $x_1 x_2 \ldots x_m \in \Sigma_o^*$. Such a loop would generate an unbounded behaviour of the system that does not require any input from the environment. This kind of spontaneous infinite behaviour is not normally expected from the class of systems we are interested in. In particular, this restriction implies that every transition system we consider has at least one quiescent state.

Asynchronous systems are normally assumed to be *receptive*—at each state s, every input action a should be possible. In practice, a system description will limit itself to providing moves for "useful" input actions at each state. One way to deal with missing inputs is to assume a dead state s_d that refuses Σ_o and has a self loop $s_d \xrightarrow{a} s_d$ for every input a. Whenever a state s refuses an input a, we add a move $s \xrightarrow{a} s_d$. In this interpretation of receptiveness, unexpected inputs cause the system to hang. Our semantics will implicitly capture this version of receptiveness, without requiring the explicit addition of such a dead state. An alternative approach, which we do not consider, is to allow the system to swallow unexpected inputs and continue with normal execution. This can be modelled by adding a self-loop labelled a at any quiescent state that refuses an input a.

In [3], Bourdonov et al study test equivalence for asynchronous systems with forbidden or refused inputs (for instance, an interactive form in which some buttons are disabled). They focus on adapting the testing formalism of [17] to such systems, with specific emphasis on compositionality. Here, on the other hand, we concentrate on expressiveness and decidability, rather than compositionality.

Queue semantics In [16], a *queue semantics* is defined for transition systems with asynchronous communication which is used to transfer notions from the theory of testing for synchronous systems to the asynchronous framework.

Let $TS = (S, I, \Sigma, \rightarrow)$ be a transition system, where $\Sigma = \Sigma_i \uplus \Sigma_o$. A *configuration* of TS is a triple (s, σ_i, σ_o) where s is a state in S and $\sigma_i \in \Sigma_i^*$ and $\sigma_o \in \Sigma_o^*$ are the input and output queues associated with the system.

Initially, the system is in a configuration $(i, \varepsilon, \varepsilon)$, where i is an initial state and both queues are empty. Each input/output move of the original system breaks up into a visible move that alters the input/output queue without changing the internal state and an invisible move in which the input/output action updates the internal state as per the transition relation of the original system.

First, we have two rules describing how the queue based system reads inputs.

Input $\quad (s, \sigma_i, \sigma_o) \xrightarrow{a} (s, \sigma_i a, \sigma_o) \qquad \dfrac{s \xrightarrow{a} s'}{(s, a\sigma_i, \sigma_o) \xrightarrow{\tau} (s', \sigma_i, \sigma_o)}$

External inputs are appended to the input queue, leaving the internal state unchanged. The system can then silently consume the action at the head of the input queue and update its state using a transition of the original system.

Similarly, we have two rules for output actions.

Output $\quad \dfrac{s \xrightarrow{x} s'}{(s, \sigma_i, \sigma_o) \xrightarrow{\tau} (s', \sigma_i, \sigma_o x)} \qquad (s, \sigma_i, x\sigma_o) \xrightarrow{x} (s, \sigma_i, \sigma_o)$

Any output action of the original system results in a silent internal move that changes the state of the system and appends the action to the output queue. The system can then spontaneously emit the action at the head of output queue.

This semantics implies that, at the visible level, output actions can always be postponed. A path of the form $s \xrightarrow{a} s_1 \xrightarrow{x} s_2 \xrightarrow{b} s'$ in the original system may be observed asynchronously as a sequence abx by delaying the output x.

We denote by $Q(TS)$ the transition system whose states are the configurations of TS and whose transitions are governed by the queue semantics.

3 Asynchronous testing equivalence

Our main aim is to formalize what we can observe about the behaviour of an asynchronous system through testing. We define two natural notions of testing for asynchronous systems based on input-output pairs.

3.1 IO Behaviours

If $w \in \Sigma^*$ and $X \subseteq \Sigma$, we denote by $w\!\downarrow_X$ the subword obtained by erasing all letters not in X. We also write $\preceq$ for the prefix relation on words.

As usual, let $TS = (S, I, \Sigma, \rightarrow)$ be a transition system, where $\Sigma = \Sigma_i \uplus \Sigma_o$. A *maximal run* of TS is an execution sequence $i \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n$ such that $i \in I$ and s_n is quiescent. If TS has a maximal run over a word w, we call w a δ-trace (sometimes referred to in the literature as a *quiescent trace*) of TS, written $\delta_{TS}(w)$. Let $\delta_{\text{traces}}(TS)$ denote the δ-traces of TS.

The IO-behaviour of TS corresponds to an operational model of testing where, for each test case, the tester generates a sequence of inputs, supplies them up front, and observes the effect. This corresponds, roughly, to static test generation. Formally, *IOBeh*(TS) is the set of pairs $(u, v) \in \Sigma_i^* \times \Sigma_o^*$ such that, in TS, there is a maximal run $i \xrightarrow{w} s$ labelled w with $w{\downarrow_{\Sigma_o}} = v$, and either $w{\downarrow_{\Sigma_i}} = u$ or $w{\downarrow_{\Sigma_i}} a \preceq u$ and s refuses a for some $a \in \Sigma_i$.

The condition that s refuses a for the case $w{\downarrow_{\Sigma_i}} a \preceq u$ implicitly captures the first notion of receptiveness, where unexpected inputs lead the system to hang. Formally, this means that if we add a dead state s_d to TS as described earlier, the resulting system will have the same IO-behaviours as the original system.

We can provide additional discriminating power to the tester by assuming that inputs are supplied incrementally, instead of being provided up front, analogous to on-the-fly test case generation.

A *block observation* of TS is a sequence $(u_1, v_1) \cdots (u_n, v_n) \in (\Sigma_i^* \times \Sigma_o^*)(\Sigma_i^+ \times \Sigma_o^*)^*$ such that there is a run $s_0 \xrightarrow{w_1} s_1 \cdots \xrightarrow{w_k} s_k$ with $1 \leq k \leq n$ starting from an initial state $s_0 \in I$ and going through quiescent states $s_1, \ldots, s_k$ with:

- $v_j = w_j{\downarrow_{\Sigma_o}}$ for all $1 \leq j \leq n$, and $v_j = \varepsilon$ for all $k < j \leq n$, and
- $u_j = w_j{\downarrow_{\Sigma_i}}$ for all $1 \leq j < k$, and either ($k = n$ and $u_n = w_n{\downarrow_{\Sigma_i}}$) or ($w_k{\downarrow_{\Sigma_i}} a \preceq u_k$ for some $a \in \Sigma_i$ such that s_k refuses a).

A block observation consists of supplying inputs in blocks $u_0 u_1 \ldots u_n$ and observing the incremental output associated with each block. The first input block is permitted to be empty, to account for a spontaneous initial output v_0. Let $IOBlocks(TS)$ denote the set of block observations of TS.

Definition 1. *We define two testing equivalences on asynchronous systems, corresponding to IO-behaviours and block observations.*

$$\begin{aligned} TS \sim_{io} TS' &\stackrel{def}{=} IOBeh(TS) = IOBeh(TS') \\ TS \sim_{ioblock} TS' &\stackrel{def}{=} IOBlocks(TS) = IOBlocks(TS') \end{aligned}$$

3.2 Synchronous testing on queues

In contrast to our direct definition of testing based on the observed input-output behaviour of asynchronous systems, the approach taken in [16] is to reduce asynchronous testing to synchronous testing via the queue semantics. Two systems are said to be testing equivalent in an asynchronous sense if the corresponding interpretations with queues are testing equivalent in a synchronous sense.

Let $\sim_Q$ denote asynchronous testing equivalence under the queue semantics and $\sim_{syn}$ denote the normal synchronous testing equivalence, which coincides with failures semantics [7, 8]. Then,

$$TS \sim_Q TS' \stackrel{\text{def}}{=} Q(TS) \sim_{syn} Q(TS').$$

We do not recall the formal definition of synchronous testing equivalence, because we do not require this branching-time formulation of $\sim_Q$. Instead, it turns out that $\sim_Q$ admits a linear-time characterization (Corollary 5.15 in [16]).

Theorem 2. *$TS \sim_Q TS'$ iff $L(Q(TS)) = L(Q(TS'))$ and $\delta_{traces}(Q(TS)) = \delta_{traces}(Q(TS'))$.*

In the rest of this section, we define some notions related to $L(Q(TS))$ and $\delta_{\text{traces}}(Q(TS))$ that will prove useful in later analysis.

Tracks We begin by defining an ordering @ on words. Intuitively, w @ w' (read as "w is *aped* by w'") if w can be observed as w' by postponing some outputs. In the process, w' could accept additional inputs. Formally, w @ w' if:

- $w{\downarrow}_{\Sigma_i} \preceq w'{\downarrow}_{\Sigma_i}$.
- $w{\downarrow}_{\Sigma_o} = w'{\downarrow}_{\Sigma_o}$.
- For every pair of prefixes w_j, w'_j of w, w' of length j, $w'_j{\downarrow}_{\Sigma_o} \preceq w_j{\downarrow}_{\Sigma_o}$.

The relation @ is a partial order on Σ^*. It is easy to see that $L(Q(TS))$, the prefix closed language of TS under the queue semantics, is upward-closed with respect to @: if $w \in L(Q(TS))$ and w @ w' then $w' \in L(Q(TS))$.

A *track* is an @-minimal word in $L(Q(TS))$. It is shown in [16] that every track is actually a word in $L(TS)$, the original transition system interpreted without the queue semantics. Moreover, since $L(Q(TS))$ is upward-closed with respect to @, the set of tracks completely determines the set of traces. Note that not every word in $L(TS)$ is a track: for instance, TS could explicitly have execution sequences $axby$ and $abxy$. Since $axby$ @ $abxy$, $abxy$ is not a track. Let Tracks(TS) denote the set of tracks of TS.

Empty and blocked deadlocks We can classify quiescent traces into two groups. Recall that we have assumed a receptive model of asynchronous communication in which input actions are always enabled but unexpected inputs cause the system to hang. This gives rise to two possible scenarios when a system deadlocks. In the first scenario, the system is waiting for input with an empty input queue and can potentially make progress if a suitable input arrives. In the second scenario, the system has received an unexpected input and can never recover. We refer to these as empty and blocked deadlocks, respectively.

To define empty and blocked deadlocks formally, we need a new relation. We say that $w \in \Sigma^*$ is strictly aped by $w' \in \Sigma^*$, denoted $w \;|@|\; w'$, if w @ w' and $|w| = |w'|$. We can then define the empty and blocked deadlocks of $Q(TS)$.

$$\begin{aligned}\delta_{\text{empty}}(Q(TS)) = \{w \in \Sigma^* \mid \exists\, i \xrightarrow{w'} s \text{ in } TS \text{ with } i \in I,\\ s \text{ quiescent and } w' \;|@|\; w\}.\end{aligned}$$

$$\begin{aligned}\delta_{\text{block}}(Q(TS)) = \{w \in \Sigma^* \mid \exists\, i \xrightarrow{w'} s \text{ in } TS \text{ with } i \in I, \exists\, a \in \Sigma_i \text{ such that}\\ s \text{ refuses } \Sigma_o \cup \{a\} \text{ and } w'a \;@\; w\}.\end{aligned}$$

Observe that $\delta_{\text{empty}}(Q(TS))$ is $|@|$-upward closed and consists of traces w such that $(i, \varepsilon, \varepsilon) \xrightarrow{w} (s, \varepsilon, \varepsilon)$ in $Q(TS)$ with s quiescent. Similarly, $\delta_{\text{block}}(Q(TS))$ is @-upward closed and consists of traces w such that $(i, \varepsilon, \varepsilon) \xrightarrow{w} (s, a\sigma_i, \varepsilon)$ in $Q(TS)$ where s refuses $\Sigma_o \cup \{a\}$. It is not difficult to see that

$$\delta_{\text{traces}}(Q(TS)) = \delta_{\text{empty}}(Q(TS)) \cup \delta_{\text{block}}(Q(TS)).$$

However, note that the sets $\delta_{\text{empty}}(Q(TS))$ and $\delta_{\text{block}}(Q(TS))$ may overlap. In fact, it is even possible $TS_1 \sim_Q TS_2$ but $\delta_{\text{empty}}(Q(TS_1)) \neq \delta_{\text{empty}}(Q(TS_2))$ or $\delta_{\text{block}}(Q(TS_1)) \neq \delta_{\text{block}}(Q(TS_2))$ [16]. Despite these shortcomings, we will find these notions very useful.

4 Comparing the three equivalences

Our first set of results compare the three testing equivalences we have introduced earlier. We show that $\sim_{io}$ is strictly weaker than $\sim_Q$ and $\sim_{ioblock}$, but $\sim_Q$ and $\sim_{ioblock}$ are incomparable.

Proposition 3. *If $TS_1 \sim_{ioblock} TS_2$, then $TS_1 \sim_{io} TS_2$.*

Proof. This follows from the fact that $\mathit{IOBeh}(TS) = \mathit{IOBlocks}(TS) \cap (\Sigma_i^* \times \Sigma_o^*)$ for any transition system TS. $\square$

Proposition 4. *If $TS_1 \sim_Q TS_2$, then $TS_1 \sim_{io} TS_2$.*

Proof. Let TS_1 and TS_2 be two transition systems such that $TS_1 \sim_Q TS_2$. We show that $TS_1 \sim_{io} TS_2$. Let $(u, v) \in \mathit{IOBeh}(TS_1)$ and let $i \xrightarrow{w} s$ be a maximal run in TS_1 labelled w, with $w{\downarrow_{\Sigma_o}} = v$, and either $w{\downarrow_{\Sigma_i}} = u$ or $w{\downarrow_{\Sigma_i}} a \preceq u$ and s refuses a for some $a \in \Sigma_i$.

Case 1: Suppose $w{\downarrow_{\Sigma_i}} = u$. By definition of the empty deadlocks, we obtain $w \in \delta_{\text{empty}}(Q(TS_1))$. Since $TS_1 \sim_Q TS_2$, we have $w \in \delta_{\text{traces}}(Q(TS_2))$.

If $w \in \delta_{\text{empty}}(Q(TS_2))$ then, in TS_2, there is a maximal run $i' \xrightarrow{w'} s'$ with $w'\ |@|\ w$. Since $w'{\downarrow_{\Sigma_i}} = w{\downarrow_{\Sigma_i}} = u$ and $w'{\downarrow_{\Sigma_o}} = w{\downarrow_{\Sigma_o}} = v$, we have $(u, v) \in \mathit{IOBeh}(TS_2)$.

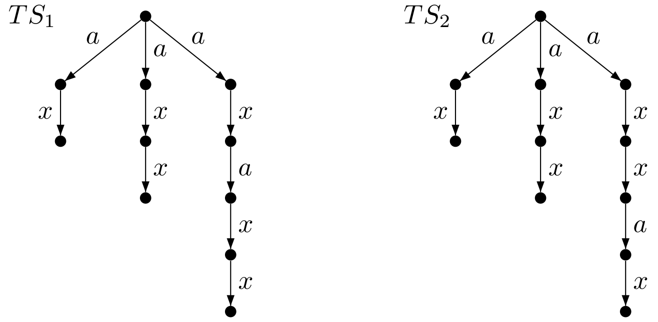
If $w \in \delta_{\text{block}}(Q(TS_2))$ then, in TS_2, there is a maximal run $i' \xrightarrow{w'} s'$ where s' refuses $\Sigma_o \cup \{b\}$ and $w'b\ @\ w$ for some $b \in \Sigma_i$. Since $(w'b){\downarrow_{\Sigma_i}} \preceq w{\downarrow_{\Sigma_i}} = u$ and $w'{\downarrow_{\Sigma_o}} = w{\downarrow_{\Sigma_o}} = v$, we have $(u, v) \in \mathit{IOBeh}(TS_2)$.

Case 2: Suppose $w{\downarrow_{\Sigma_i}} a \preceq u$ and s refuses $a \in \Sigma_i$. As above, by definition of the blocked deadlocks we get $wa \in \delta_{\text{block}}(Q(TS_1))$. Let u' be such that $u = w{\downarrow_{\Sigma_i}} au'$. We have $wa\ @\ wau'$ and we obtain $wau' \in \delta_{\text{block}}(Q(TS_1))$ since this set is @-upward closed. Since $TS_1 \sim_Q TS_2$ we deduce $wau' \in \delta_{\text{traces}}(Q(TS_2))$.

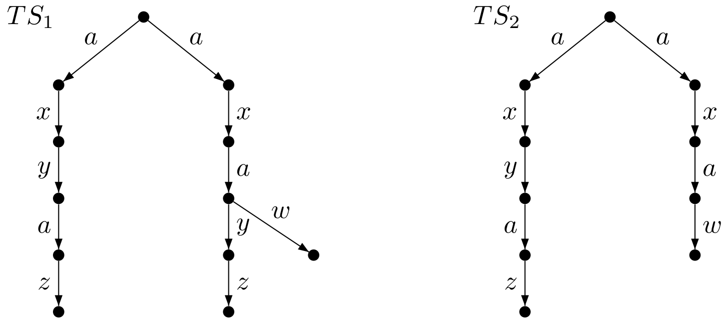
If $wau' \in \delta_{\text{empty}}(Q(TS_2))$ then, in TS_2, there is a maximal run $i' \xrightarrow{w'} s'$ with $w'\ |@|\ wau'$. Since $w'{\downarrow_{\Sigma_i}} = w{\downarrow_{\Sigma_i}} au' = u$ and $w'{\downarrow_{\Sigma_o}} = w{\downarrow_{\Sigma_o}} = v$, we have $(u, v) \in \mathit{IOBeh}(TS_2)$.

If $wau' \in \delta_{\text{block}}(Q(TS_2))$ then, in TS_2, there is a maximal run $i' \xrightarrow{w'} s'$ where s' refuses $\Sigma_o \cup \{b\}$ and $w'b\ @\ wau'$ for some $b \in \Sigma_i$. Since $w'{\downarrow_{\Sigma_i}} b \preceq w{\downarrow_{\Sigma_i}} au' = u$ and $w'{\downarrow_{\Sigma_o}} = w{\downarrow_{\Sigma_o}} = v$, we have $(u, v) \in \mathit{IOBeh}(TS_2)$. $\square$

The implications we have proved are strict. Below, we show two systems that are related by $\sim_{io}$ but not by $\sim_Q$. Here $\Sigma_i = \{a\}$ and $\Sigma_o = \{x\}$. For both systems, the IO-behaviours are given by $\{(\varepsilon, \varepsilon), (a, x), (a, xx)\} \cup \{(a^n, x), (a^n, x^2), (a^n, x^3) \mid n > 1\}$, so $TS_1 \sim_{io} TS_2$. However, notice that $axaxx \in \text{Tracks}(TS_1) \setminus \text{Tracks}(TS_2)$ because $axxax \in L(TS_2)$ and $axxax\ @\ axaxx$. Hence, $TS_1 \not\sim_Q TS_2$. This example also establishes that $\sim_{io}$ is strictly weaker than $\sim_{ioblock}$ since $(a, x)(a, xx) \in \mathit{IOBlocks}(TS_1) \setminus \mathit{IOBlocks}(TS_2)$.

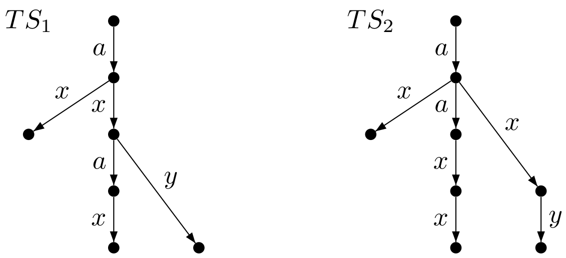


The equivalences $\sim_Q$ and $\sim_{ioblock}$ are incomparable. Below, we show two systems that are related by $\sim_Q$ but not by $\sim_{ioblock}$. Here, $\Sigma_i = \{a\}$ and $\Sigma_o = \{w, x, y, z\}$. We have $\text{Tracks}(TS_1) = \text{Tracks}(TS_2) = \{\varepsilon, ax, axy, axyaz, axaw\}$. Also, the set of empty deadlocks for both systems is the $|@|$-upper closure of $\{\varepsilon, ax, axy, axyaz, axaw\}$. Finally, the set of blocked deadlocks for both systems is the @-upper closure of $\{axyaza, axawa\}$. Hence $TS_1 \sim_Q TS_2$. However, $(a, x)(a, yz)$ is in $\mathit{IOBlocks}(TS_1) \setminus \mathit{IOBlocks}(TS_2)$, so $TS_1 \not\sim_{ioblock} TS_2$.



Similarly, we give below two systems that are related by $\sim_{ioblock}$ but not by $\sim_Q$. We have $axax \in \delta_{\text{traces}}(Q(TS_1)) \setminus \delta_{\text{traces}}(Q(TS_2))$, so $TS_1 \not\sim_Q TS_2$. On the other hand, $TS_1 \sim_{ioblock} TS_2$ since the block observations of TS_1 and TS_2 are

$$\begin{aligned}\{(\varepsilon, \varepsilon)\} &\cup \{(a^n, x), (a^n, xy), (aa^n, x^2) \mid n \geq 1\} \cdot (a^+ \times \{\varepsilon\}) \\ &\cup \{(\varepsilon, \varepsilon)\} \cdot \{(a^n, x), (a^n, xy), (aa^n, x^2) \mid n \geq 1\} \cdot (a^+ \times \{\varepsilon\})\end{aligned}$$



Queued quiescent traces Our equivalences $\sim_{io}$ and $\sim_{ioblock}$ correspond to the notions queued quiescent trace equivalence and queued suspension trace equivalence, respectively, defined in [14]. While we directly provide extensional characterizations of these equivalences, the corresponding notions are developed in [14] via an intensional definition of testing that uses a variant of IO-automata with queues, from which an extensional definition is derived.

In [14], queued quiescent trace equivalence is compared with an equivalence called ioco, defined by Tretmans in [17], which differs slightly from the queue equivalence $\sim_Q$ that we consider here. It is shown, by examples, that some systems distinguished by ioco are equated by queued quiescent trace equivalence and that some systems equated by queued quiescent trace equivalence are distinguished by queued suspension trace equivalence. However, there is no formal characterization of the relative expressive powers of these three equivalences.

5 Decidability of asynchronous test equivalence

We now examine the decidability of test equivalence for finite-state systems.

5.1 Undecidability of $\sim_{io}$

We prove this result using a reduction from the equivalence problem for rational relations [1, 15]. We start by recalling some definitions. Let A, B be two finite alphabets. With componentwise concatenation, the set $A^* \times B^*$ is a monoid. A rational relation over A and B is a rational subset R of $A^* \times B^*$. Equivalently, R is a mapping from A^* to $\mathcal{P}(B^*)$ where $u \in A^* \mapsto R(u) = \{v \in B^* \mid (u, v) \in R\}$.

Le $(K, +, \times, 0, 1)$ be a semiring. A K-automaton over A is a tuple $\mathcal{A} = (S, \lambda, \mu, \gamma)$ with S a finite set of states, $\lambda, \gamma \in K^S$ and $\mu(a) \in K^{S\times S}$ for each $a \in A$. Intuitively, the automaton outputs λ_i when it is entered in state i, then it outputs $\mu(a)_{i,j}$ whenever a transition labelled a from i to j is taken and finally, it outputs γ_j when the input word has been completely read and we exit the automaton in state j. The value $(\mathcal{A}, u)$ computed by $\mathcal{A}$ on the input word $u = a_1 \cdots a_k \in A^*$ is the sum over all paths $i_0, \ldots, i_k \in S$ of the products $\lambda_{i_0}\mu(a_1)_{i_0,i_1} \cdots \mu(a_k)_{i_{k-1},i_k}\gamma_{i_k}$. Since K is a semiring, the set of matrices $K^{S\times S}$ equipped with matrix multiplication is a monoid and we can extend μ to a monoid morphism $\mu : A^* \to K^{S\times S}$. Viewing λ as a row vector and γ as a column vector, we have $(\mathcal{A}, u) = \lambda\mu(u)\gamma$ for each $u \in A^*$. Without loss of generality, we may assume that $\lambda_i \neq 0$ implies $\lambda_i = 1$ for each state $i \in S$.

The set $K = \mathrm{Rat}(B^*)$ equipped with union as addition and concatenation as multiplication is a semiring with $\emptyset$ as zero element and $\{\varepsilon\}$ as unit. A relation $R \subseteq A^* \times B^*$ is rational if and only if it can be realized by some $\mathrm{Rat}(B^*)$-automaton. We denote by $\mathcal{R}(\mathcal{A})$ the rational relation realized by $\mathcal{A}$ and for $u = a_1 \cdots a_k \in A^*$ we have $(u, v) \in \mathcal{R}(\mathcal{A})$ iff $v \in \lambda_{i_0}\mu(a_1)_{i_0,i_1} \cdots \mu(a_k)_{i_{k-1},i_k}\gamma_{i_k}$ for some $i_0, \ldots, i_k \in S$. The equivalence problem for rational relations given by $\mathrm{Rat}(B^*)$-automata is undecidable [1, 15]. This undecidability holds even for rational relation for which $|B| = 1$ and given by a K-automaton where K is the semiring $\mathcal{P}_{\mathrm{fin}}(B^*)$ of *finite* subsets of B^*. So in the following we assume that $B = \{b\}$ is a singleton and that $K = \mathcal{P}_{\mathrm{fin}}(B^*)$.

We prefer to avoid ε-transitions. We call a K-automaton $\mathcal{A} = (S, \lambda, \mu, \gamma)$ *strict* if none of the sets $\mu(a)_{p,q}$ and γ_q contain the empty word ε. We show that the undecidability still holds for rational relations given by strict K-automata. Let $\mathcal{A} = (S, \lambda, \mu, \gamma)$ be a K-automaton. Define $\mathcal{A}^s = (S, \lambda, \mu^s, \gamma^s)$ by $\mu^s(a)_{p,q} = b\mu(a)_{p,q}$ and $\gamma^s_q = b\gamma_q$. Then, $\mathcal{A}^s$ is strict and for each $u \in A^*$ we have $\lambda\mu^s(u)\gamma^s = b^{|u|+1}\lambda\mu(u)\gamma$ (recall that $B = \{b\}$ so the semiring K is commutative). Then, $\mathcal{R}(\mathcal{A}) = \mathcal{R}(\mathcal{B})$ if and only if $\mathcal{R}(\mathcal{A}^s) = \mathcal{R}(\mathcal{B}^s)$. Therefore, equivalence is undecidable for rational relations given by strict K-automata.

We now associate to a strict K-automaton $\mathcal{A} = (S, \lambda, \mu, \gamma)$ a transition system $\mathcal{A}'$ over Σ with $\Sigma_i = A$ and $\Sigma_o = B \uplus \{\#\}$ where $\#$ is a new output letter. For each $(p, a, q) \in S \times A \times S$ we consider an automaton $\mathcal{A}_{p,a,q}$ recognizing $\mu(a)_{p,q}$ and such that $\mathcal{A}_{p,a,q}$ has a unique initial state $i_{p,a,q}$ with no ingoing transition, a unique final state $f_{p,a,q}$ with no outgoing transition and all other states have outgoing transitions. To construct $\mathcal{A}'$, we first take the disjoint union of the automata $\mathcal{A}_{p,a,q}$ for $(p, a, q) \in S \times A \times S$. Then, for each $q \in S$, we merge all states $f_{p,a,q}$ with $(p, a) \in S \times A$ into a single state denoted simply by q. Finally, for each $(p, a, q) \in S \times A \times S$, we add the transition $p \xrightarrow{a} i_{p,a,q}$. Thus we obtain the transition system $\mathcal{A}' = (S', I, \Sigma, \rightarrow)$ with $I = \{i \in S \mid \lambda_i \neq \emptyset\}$. Note that in $\mathcal{A}'$, all transitions leaving the states in S are labelled with input letters and all transitions leaving states in $S' \setminus S$ are labelled with output letters. Hence, the deadlocked states in $\mathcal{A}'$ are exactly those in S.

For each pair of states $p, q \in S$ we consider the relation

$$T_{p,q} = \{(w{\downarrow_A}, w{\downarrow_B}) \in A^* \times B^* \mid p \xrightarrow{w} q \text{ in } \mathcal{A}'\}.$$

The following lemma is a standard result from the theory of rational relations and K-automata.

Lemma 5. *For each $p, q \in S$, we have*

$$T_{p,q} = \{(u, v) \in A^* \times B^* \mid v \in \mu(u)_{p,q}\}.$$

For each $q \in S$ we consider an automaton $\mathcal{A}_q$ recognizing $\gamma_q\#$ and such that $\mathcal{A}_q$ has a unique initial state i_q with no ingoing transition, a unique final state f_q with no outgoing transition and all other states have outgoing transitions. We let $\mathcal{A}^+_q$ be $\mathcal{A}_q$ with the additional transitions $f_q \xrightarrow{a} f'_q$ for $a \in A$ and $f'_q \xrightarrow{\#} f_q$ so that f_q does not refuse any input letter. Finally, we let $\mathcal{A}''$ be the disjoint union of $\mathcal{A}'$ together with the automata $\mathcal{A}^+_q$ for $q \in S$ and the additional transitions $x \xrightarrow{b} i_q$ for each transition $x \xrightarrow{b} q$ of $\mathcal{A}'$. Note that the deadlocked states of $\mathcal{A}''$ are $S \cup \{f_q \mid q \in S\}$.

Lemma 6. $\mathit{IOBeh}(\mathcal{A}'') = \mathit{IOBeh}(\mathcal{A}') \cup \mathcal{R}(\mathcal{A}) \cdot \{(x, \#^{1+|x|}) \mid x \in A^*\}$.

Proof. First, maximal paths in $\mathcal{A}'$ are of the form $p \xrightarrow{w} q$ for $p \in I$ and $q \in S$. These are also maximal paths in $\mathcal{A}''$. Moreover, a state $q \in S$ refuses exactly the same input letters in $\mathcal{A}'$ and in $\mathcal{A}''$. Hence, $\mathit{IOBeh}(\mathcal{A}') \subseteq \mathit{IOBeh}(\mathcal{A}'')$. Conversely, the maximal paths in $\mathcal{A}''$ which do not use the letter # cannot enter one of the automata $\mathcal{A}_q$. Hence, they are also maximal paths in $\mathcal{A}'$ and we deduce that $\mathit{IOBeh}(\mathcal{A}'') \cap A^* \times B^* = \mathit{IOBeh}(\mathcal{A}')$.

Second, let $(u, v) \in \mathcal{R}(\mathcal{A})$. We have $v \in \lambda\mu(u)\gamma$ hence we find $p, q \in S$ with $v \in \lambda_p \mu(u)_{p,q}\gamma_q$. It follows that $\lambda_p \neq \emptyset$ (i.e., $p \in I$), which implies $\lambda_p = \{\varepsilon\}$ by our assumption on K-automata. Hence we can write $v = v'v''$ with $v' \in \mu(u)_{p,q}$ and $v'' \in \gamma_q$. By Lemma 5 we find a path $p \xrightarrow{w} q$ in $\mathcal{A}'$ with $u = w{\downarrow_A}$ and $v' = w{\downarrow_B}$. Replacing the last transition $x \xrightarrow{b} q$ of this path by $x \xrightarrow{b} i_q$ we find a path $p \xrightarrow{wv''\#} f_q$ in $\mathcal{A}''$. For $x = a_1 \cdots a_k$, this path can be extended with $f_q \xrightarrow{w'} f_q$ where $w' = a_1\# \cdots a_k\#$. We have $ux = (wv''\#w'){\downarrow_{\Sigma_i}}$ and $v\#^{1+|x|} = (wv''\#w'){\downarrow_{\Sigma_o}}$. Since f_q is a deadlocked state we deduce that $(ux, v\#^{1+|x|}) \in \mathit{IOBeh}(\mathcal{A}'')$.

Conversely, let $(u', v') \in \mathit{IOBeh}(\mathcal{A}'') \setminus A^* \times B^*$. Let $p \xrightarrow{w'} s$ be a run in $\mathcal{A}''$ with $p \in I$, s deadlocked, $w'{\downarrow_{\Sigma_o}} = v'$ and either $w'{\downarrow_{\Sigma_i}} = u'$ or $w'{\downarrow_{\Sigma_i}} a \preceq u'$ and s refuses $a \in \Sigma_i$. Since $v' \notin B^*$, we must have $s = f_q$ for some $q \in S$ and $w' = w\#a_1\# \cdots a_k\#$ with $w \in (A \cup B)^*$ and $x = a_1 \cdots a_k \in A^*$. Since $s = f_q$ does not refuse any input letter, we get $w'{\downarrow_{\Sigma_i}} = u'$. With $u = w{\downarrow_{\Sigma_i}}$ and $v = w{\downarrow_{\Sigma_o}}$ we have $v' = v\#^{1+k}$ and $u' = ux$. The path $p \xrightarrow{w'} f_q$ can be split in $p \xrightarrow{w_1} i_q \xrightarrow{w_2\#} f_q \xrightarrow{a_1\#\cdots a_k\#} f_q$ so that $p \xrightarrow{w_1} q$ is a path in $\mathcal{A}'$ and $i_q \xrightarrow{w_2\#} f_q$ is a path in $\mathcal{A}_q$ and $w = w_1 w_2$. We deduce that $w_2 \in \gamma_q$, $u = w_1{\downarrow_A}$ and $v = (w_1{\downarrow_B})w_2$. By Lemma 5 we have $w_1{\downarrow_B} \in \mu(u)_{p,q}$. Therefore, $v \in \mu(u)_{p,q}\gamma_q$. Since $p \in I$ we have $\lambda_p = \{\varepsilon\}$ and we obtain $v \in \lambda\mu(u)\gamma = \mathcal{R}(\mathcal{A})(u)$. □

If we have another rational relation defined by a strict K-automaton $\mathcal{B}$ then we define similarly $\mathcal{B}'$ and $\mathcal{B}''$.

Theorem 7. *$\mathcal{A}' \uplus \mathcal{B}'' \sim_{io} \mathcal{A}'' \uplus \mathcal{B}'$ if and only if $\mathcal{R}(\mathcal{A}) = \mathcal{R}(\mathcal{B})$. Therefore, the $\sim_{io}$ equivalence is undecidable.*

Proof. The result follows from the following equations obtained from Lemma 6.

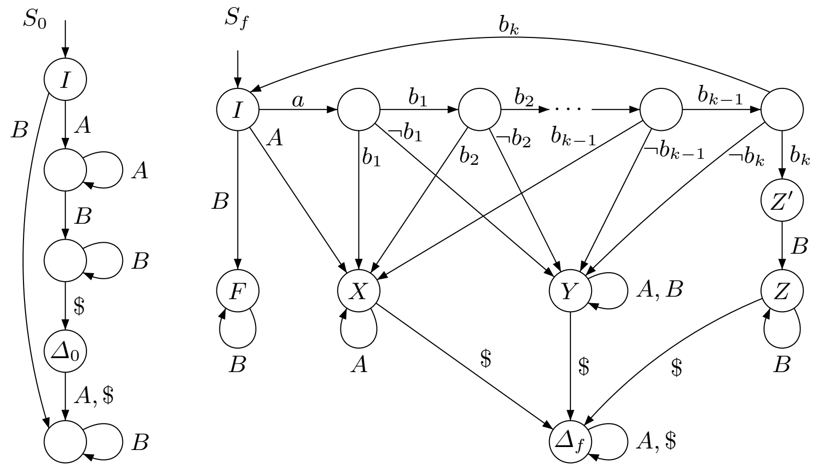
$$\mathit{IOBeh}(\mathcal{A}' \uplus \mathcal{B}'') = \mathit{IOBeh}(\mathcal{A}') \cup \mathit{IOBeh}(\mathcal{B}') \cup \mathcal{R}(\mathcal{B})\{(x, \#^{1+|x|}) \mid x \in A^*\}$$
$$\mathit{IOBeh}(\mathcal{A}'' \uplus \mathcal{B}') = \mathit{IOBeh}(\mathcal{A}') \cup \mathit{IOBeh}(\mathcal{B}') \cup \mathcal{R}(\mathcal{A})\{(x, \#^{1+|x|}) \mid x \in A^*\} \quad \square$$

5.2 Undecidability of $\sim_Q$

Let A and B be two finite alphabets and let $f, g : A^+ \to B^+$ be two morphisms corresponding to an instance of Post's Correspondence Problem (PCP). The PCP instance has a solution if and only if we have $f(u) = g(u)$ for some $u \in A^+$.

We consider a new symbol \$ and define the input and output alphabets as $\Sigma_i = A \cup \{\$\}$ and $\Sigma_o = B$. We then construct two transition systems from the ingredients shown in the figure on the next page.

The transition system S_f corresponds to the morphism f and has one loop $ab_1b_2 \ldots b_k$ for each $a \in A$ such that $f(a) = b_1b_2 \ldots b_k$. Formally the set of states



of S_f is $Q_f = \{I, F, X, Y, Z, Z', \Delta_f\} \cup \{(a, i) \mid a \in A, 0 < i \leq |f(a)|\}$ and its initial state is I. The transitions between states in $\{I, F, X, Y, Z, Z', \Delta_f\}$ are precisely given in the picture above, which also contains the intuition for the other transitions defined, for each $a \in A$ with $f(a) = b_1 b_2 \cdots b_k$, by:

- $I \xrightarrow{a} (a,1) \xrightarrow{b_1} (a,2) \xrightarrow{b_2} (a,3) \quad \cdots \quad (a, k-1) \xrightarrow{b_{k-1}} (a,k) \xrightarrow{b_k} I$,
- $(a,i) \xrightarrow{b} Y$ if $1 \leq i \leq k$ and $b \in B \setminus \{b_i\}$,
- $(a,i) \xrightarrow{b_i} X$ if $1 \leq i < k$, and $(a,k) \xrightarrow{b_k} Z'$.

For the morphism g, we construct an analogous system S_g. We want to compare the following two systems, where $S_i + S_j$ denotes the disjoint union of the two systems with multiple initial states.

- $M_1 = S_0 + S_f + S_g$
- $M_2 = S_f + S_g$

The only deadlocked state in S_0 is Δ_0. Since this state does not refuse any input letter, $\delta_{\text{block}}(S_0) = \emptyset$. Similarly, the only deadlocked states in S_f are X and Δ_f and neither refuses any input letter, so $\delta_{\text{block}}(S_f) = \emptyset$. Therefore, $\delta_{\text{traces}}(M_1) = \delta_{\text{empty}}(S_0) \cup \delta_{\text{empty}}(M_2)$ and $\delta_{\text{traces}}(M_2) = \delta_{\text{empty}}(M_2)$ and $M_1 \sim_Q M_2$ if and only if $\text{Tracks}(M_1) = \text{Tracks}(M_2)$ and $\delta_{\text{empty}}(S_0) \subseteq \delta_{\text{empty}}(M_2)$.

Lemma 8. $\text{Tracks}(M_1) = \text{Tracks}(M_2) = \text{Tracks}(S_f) = B^*$.

Proof. First, let $v \in B^+$. Then v is @-minimal and $I \xrightarrow{v} F$ in S_f. Therefore, $B^* \subseteq \text{Tracks}(S_f)$. Since any word $w \in \Sigma^*$ apes its projection on the output alphabet B, we deduce that $\text{Tracks}(S_f) = B^*$. $\square$

Lemma 9. *$\delta_{empty}(S_0)$ is the $|@|$-upper closure of $A^+B^+\$$.*

Proof. Follows from the definition of δ_{empty} and the fact that the set of words $w' \in \Sigma^*$ having a run $I \xrightarrow{w'} \Delta_0$ in S_0 is $A^+B^+\$$. $\square$

Lemma 10. *Let $u \in A^+$ and $v \in B^+$. Then, $uv\$ \in \delta_{empty}(S_f)$ iff $v \neq f(u)$.*

Proof. If $v \neq f(u)$, the construction of S_f guarantees that there is some witnessing interleaving w of u and v that leads to one of the states X, Y or Z. Formally, assuming that $v \neq f(u)$ with $u = a_1 \cdots a_p$, we distinguish three cases:

1. If $v \prec f(u)$, let j be such that $f(a_1 \cdots a_{j-1}) \preceq v \prec f(a_1 \cdots a_j)$. Consider $w = a_1 f(a_1) \cdots a_{j-1} f(a_{j-1}) a_j (f(a_1 \cdots a_{j-1})^{-1} v) a_{j+1} \cdots a_p$. Then $w \; |@| \; uv$ and $I \xrightarrow{w} X$ in S_f.
2. If $v = f(a_1 \cdots a_{j-1}) v' b v''$ with $v' \prec f(a_j)$, $b \in B$ and $v'b \not\preceq f(a_j)$. Consider $w = a_1 f(a_1) \cdots a_{j-1} f(a_{j-1}) a_j v' b v'' a_{j+1} \cdots a_p$. Then $w \; |@| \; uv$ and $I \xrightarrow{w} Y$ in S_f.
3. If $f(u) \prec v$. Consider $w = a_1 f(a_1) \cdots a_p f(a_p)(f(u)^{-1} v)$. Then $w \; |@| \; uv$ and $I \xrightarrow{w} Z$ in S_f.

Hence, there is a run $I \xrightarrow{w\$} \Delta_f$ in S_f. Since $w\$ \; |@| \; uv\$$, $uv\$ \in \delta_{\text{empty}}(S_f)$.

Conversely, let $I \xrightarrow{w'} s$ be a run in S_f with s deadlocked and $w' \; |@| \; uv\$$. Since \$ must occur in w' we deduce that $s = \Delta_f$ and $w' = w\$$ with $w \; |@| \; uv$. Moreover, there is a run in S_f labelled w going from I to one of the states X, Y or Z. Let $u = a_1 \cdots a_p$.

1. If $I \xrightarrow{w} X$ then we have $w = a_1 f(a_1) \cdots a_{j-1} f(a_{j-1}) a_j (f(a_1 \cdots a_{j-1})^{-1} v) a_{j+1} \cdots a_p$ for some j such that $f(a_1 \cdots a_{j-1}) \preceq v \prec f(a_1 \cdots a_j)$ and we deduce that $v \neq f(u)$.
2. If $I \xrightarrow{w} Y$ then we have $w = a_1 f(a_1) \cdots a_{j-1} f(a_{j-1}) a_j v' b w''$ for some j such that $v' \prec f(a_j)$, $b \in B$ and $v'b \not\preceq f(a_j)$. We deduce that $v \neq f(u)$.
3. If $I \xrightarrow{w} Z$ then we have $w = a_1 f(a_1) \cdots a_p f(a_p) v'$ with $v' \in B^+$ and we deduce that $v \neq f(u)$. $\square$

Theorem 11. *$M_1 \sim_Q M_2$ iff the PCP instance (f, g) has no solution.*

Proof. First, assume that the PCP instance (f, g) has a solution and let $u \in A^+$ be such that $v = f(u) = g(u)$. Then, $uv\$ \in \delta_{\text{empty}}(S_0) \setminus \delta_{\text{empty}}(M_2)$ by Lemmas 9 and 10. Therefore, $M_1 \not\sim_Q M_2$.

Conversely, if the PCP instance (f, g) has no solution, then for every $u \in A^+$ and $v \in B^+$ we have either $v \neq f(u)$ or $v \neq g(u)$. Hence, $uv\$ \in \delta_{\text{empty}}(M_2)$ by Lemma 10. Using Lemma 9 we deduce that $\delta_{\text{empty}}(S_0) \subseteq \delta_{\text{empty}}(M_2)$ since these sets are $|@|$-upward closed. Therefore, $M_1 \sim_Q M_2$. $\square$

5.3 Decidability of $\sim_{ioblock}$ for well structured systems

Let α and β be block-observations. We say that α is *finer than* β, denoted $\alpha \preceq \beta$, if β can be obtained from α by merging consecutive blocks. More precisely, if $\alpha = (u_1, v_1) \cdots (u_n, v_n)$ and $0 < j_1 < \cdots < j_p = n$ $(p \geq 1)$ then α is finer than $\beta = (u_1 \cdots u_{j_1}, v_1 \cdots v_{j_1}) \cdots (u_{1+j_{p-1}} \cdots u_{j_p}, v_{1+j_{p-1}} \cdots v_{j_p})$. Clearly, if $\alpha \in \mathit{IOBlocks}(TS)$ and $\alpha \preceq \beta$ then $\beta \in \mathit{IOBlocks}(TS)$.

We say that a block observation $\alpha = (u_1, v_1) \cdots (u_n, v_n)$ is *reduced* if $u_1 = \varepsilon$ and $u_j \in \Sigma_i$ for $1 < j \leq n$. A transition system is *well structured* (WS) if each state either refuses Σ_i or refuses Σ_o. A transition system is *receptive* if no quiescent state s refuses an input: $s \xrightarrow{a}$ for all $a \in \Sigma_i$.

Lemma 12. *Assume that TS is WS. If $\beta \in \mathit{IOBlocks}(TS)$ then there exists $\alpha \in \mathit{IOBlocks}(TS)$* reduced *with $\alpha \preceq \beta$. Therefore, $\mathit{IOBlocks}(TS)$ is characterized by its* reduced *block-observations.*

Let $L_\delta(TS)$ be the language accepted by TS with quiescent states as final.

Lemma 13.

1. *Assume that TS is WS. If $w = v_1 a_2 v_2 \cdots a_n v_n \in L_\delta(TS)$ with $v_j \in \Sigma_o^*$ and $a_j \in \Sigma_i$ then $(\varepsilon, v_1)(a_2, v_2) \cdots (a_n, v_n) \in \mathit{IOBlocks}(TS)$.*
2. *Let TS be WS and receptive. If $(\varepsilon, v_1)(a_2, v_2) \cdots (a_n, v_n) \in \mathit{IOBlocks}(TS)$ with $v_j \in \Sigma_o^*$ and $a_j \in \Sigma_i$ then $w = v_1 a_2 v_2 \cdots a_n v_n \in L_\delta(TS)$.*

We deduce from the lemmata above that $\sim_{ioblock}$ is decidable for WS and receptive transition systems since for these systems $\sim_{ioblock}$ amounts to language equivalence: $TS_1 \sim_{ioblock} TS_2$ iff $L_\delta(TS_1) = L_\delta(TS_2)$.

For $a \in \Sigma_i$, we define $L_{\delta,a}(TS)$ as the language accepted by TS when the final states are all the quiescent states that refuse a.

Lemma 14. *Assume that TS is WS. If $w = v_1 a_2 v_2 \cdots a_k v_k \in L_{\delta,a_{k+1}}(TS)$ with $v_j \in \Sigma_o^*$ for $1 \leq j \leq k$ and $a_j \in \Sigma_i$ for $2 \leq j \leq n$ $(k < n)$ then $(\varepsilon, v_1)(a_2, v_2) \cdots (a_n, v_n) \in \mathit{IOBlocks}(TS)$.*

From this we derive a sufficient condition for $\sim_{ioblock}$.

Lemma 15. *Assume that TS_1 and TS_2 are well-structured and that $L_\delta(TS_1) = L_\delta(TS_2)$ and $L_{\delta,a}(TS_1) = L_{\delta,a}(TS_2)$ for all $a \in \Sigma_i$. Then, $TS_1 \sim_{ioblock} TS_2$.*

5.4 Decidability of $\sim_{io}$ for unlabelled MSC tests

A message sequence chart, or MSC, visually represents a sequence of communications between a set of agents [11]. In an MSC, processes are represented by vertical lines, with time flowing downward, and messages are drawn as arrows connecting the vertical lines. One way of characterizing patterns of communications is in terms of the MSCs they generate. For these characterizations, message labels are often omitted, as in the treatment of regular MSC languages in [10]. When restricted to the communications between the tester and the system under test, this corresponds to a setting in which the input and output alphabets are both singletons, since all messages to and from the system under test are unlabelled. The reduction used to prove Theorem 7 allows us to model $\sim_{io}$ using rational relations. It is known that equality is decidable for rational relations over a pair of unary alphabets. Hence, we have the following.

Theorem 16. *For tests described using unlabelled MSCs, $\sim_{io}$ is decidable.*

6 Future work

We have presented two intuitive notions of asynchronous testing and compared their expressive power with the definition due to Tretmans. Much work remains to be done to apply these new notions to make testing more effective. As mentioned in the introduction, the key problem remains that of identifying efficient yet exhaustive test sets for a given system. There is also the question of how to efficiently represent a family of such tests—see for instance [9]. Another interesting issue is to see how testing can be done in a distributed manner, extending the work reported in [12].

References

1. J. Berstel: *Transductions and Context-Free Languages*, Teubner Studienbücher, Informatik (1979).
2. M. Boreale, R. de Nicola and R. Pugliese: Trace and Testing Equivalence in Asynchronous Processes, *Inf. and Comput.*, **172** (2002), 139–164.
3. I.B. Bourdonov, A.S. Kossatchev and V.V. Kuliamin, : Formal Conformance Testing of Systems with Refused Inputs and Forbidden Actions, *MBT 2006*, Vienna, Austria, *ENTCS*, Elsevier (2006).
4. I. Castellani and M Hennessy: Testing Theories for Asynchronous Languages, *Proc. FSTTCS '98*, Springer LNCS **1530** (1998) 90–101.
5. F.S. de Boer, J.W. Klop and C. Palamidessi: Asynchronous communication in process algebra, *Proc. 7th IEEE Logics in Computer Science (LICS)*, IEEE Computer Society Press (1992) 137–147.
6. F.S. de Boer, J.N. Kok, C. Palamidessi and J.J.M.M. Rutten: The failure of failures: Towards a paradigm for asynchronous communication, *Proc. CONCUR 91*, Springer LNCS **527** (1991) 111–126.
7. R. de Nicola and M. Hennessy: Testing equivalences for processes, *Theor. Comput. Sci.*, **34** (1984) 83–133.
8. R.J. van Glabbeek: The linear time-branching time spectrum I: The semantics of concrete, sequential processes, in *Handbook of Process Algebra*, J.A. Bergstra, A. Ponse and S.A. Smolka, eds., Elsevier (2001) 3–99.
9. O. Henniger: On test case generation from asynchronously communicating state machines, *Proc. IWTCS'97* Cheju Island, South Korea, (1997).
10. J.G. Henriksen, M. Mukund, K. Narayan Kumar, M. Sohoni and P.S. Thiagarajan: A Theory of Regular MSC Languages, *Inf. and Comput.*, **202**(1) (2005) 1–38.
11. ITU-TS Recommendation Z.120: *Message Sequence Chart (MSC).* ITU-TS, Geneva (1997).
12. C. Jard: Synthesis of distributed testers from true-concurrency models of reactive systems, *Information & Software Technology*, **45**(12) (2003) 805–814.
13. C. Jard and T. Jéron: TGV: theory, principles and algorithms, *Software Tools for Technology Transfer*, **7**(4)(2005) 297–315.
14. A. Petrenko, N. Yevtushenko and J.L. Huo: Testing Transition Systems with Input and Output Testers, *Proc TestCom 2003*, Sophia Antipolis, France, (2003) 129–145.
15. J. Sakarovitch: *Eléments de théorie des automates*, Vuibert (2003).
16. J. Tretmans: *A formal approach to conformance testing*, PhD Thesis, University of Twente, The Netherlands (1992).
17. J. Tretmans: Test Generation with Inputs, Outputs and Repetitive Quiescence, *Software—Concepts and Tools*, **17**(3) (1996) 103–120.