Formalizing 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 used to verify whether a computing system conforms to its specification. Developing an efficient and exhaustive test suite is a challenging problem, especially in the setting of distributed systems. Formal theories of testing have been proposed and studied for systems that communicate synchronously. However, in the setting of asynchronous systems, little progress has been made since the work of Tretmans, in which asynchrony is modelled using synchronous communication by augmenting the state space of the system with queues.

We develop a theory of testing for asynchronous communication based on a natural notion of input-output observability. 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. We also establish (un)decidability results for these equivalences.

1 Introduction

Testing is a fundamental activity in verifying the correctness of systems. Though formal verification techniques based on model-checking and theorem-proving can provide more precise assertions about correctness, these approaches must necessarily work at the level of formal models of the system to be verified. Formal verification typically cannot detect bugs that are introduced when translating a correct design into an actual implementation.

Testing as a validation process cannot guarantee the absence of bugs, but it can increase confidence. For software, testing may focus on different aspects such as functionality, performance, timing constraints etc. It is typically deployed at different levels, ranging from unit testing to integration of modules to overall system level testing. Though testing has always played a dominant role in software development, much remains to be done to formalize the testing process, including strategies for choosing the nature and extent of test data.

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

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 abstract semantic characterization in terms of failures [1, 2]. These ideas were expanded and elaborated in the work of Tretmans [9], 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 [7].

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. It turns out that the nature of testing changes quite drastically when we model systems with asynchronous communication, in which the receipt of a message occurs separately from and later than the sending of the message. In particular, testing equivalence for systems with synchronous communication is a branching-time equivalence, whereas for systems with asynchronous communication it is a linear-time equivalence.

Most communication protocols are based on asynchronous communication. Notations such as TTCN [4] are common for specifying tests for such systems. However, as in the more general setting of software, many questions remain unanswered about the testing process for such systems. In addition to the usual problem of optimizing the size of test suites without sacrificing coverage, there are also additional issues such as the possibility of distributing tests [6].

Our aim is to develop an effective theory of testing for systems with asynchronous communication. The only major work in this direction so far is from the work of Tretmans [9], in which asynchronous communication is reduced to synchronous communication in a model augmented with infinite queues. This treatment is both unwieldy and, in some senses, not intuitive.

In contrast, we 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.

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 we propose is decidable for "well-structured" transition systems. We also show that our weaker notion of equivalence is decidable in the interesting case where we consider the input-output behaviour of a system to be an unlabelled message sequence chart [5].

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 always 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 $s \not\xrightarrow{\alpha}$ 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 $s \not\xrightarrow{X}$ if $s \not\xrightarrow{\alpha}$ for *every* $\alpha \in X$. A state s is said to *refuse* a set $X \subseteq \Sigma$ of actions if $s \not\xrightarrow{X}$. A state s is *deadlocked* 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 $x_1x_2 \ldots x_m \in \Sigma_o^*$. 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 deadlocked state. Though we implicitly assume these restrictions when presenting our testing framework, our theory can easily be extended to handle the general case.

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. We can deal with missing inputs in two ways. The first is to assume that there is a dead state s_d that refuses Σ_o and has a self loop $s_d \xrightarrow{a} s_d$ for every input action a. Whenever a state s refuses an input action a, we have a move $s \xrightarrow{a} s_d$. This corresponds to an interpretation of receptiveness in which unexpected inputs cause the system to hang. Our semantics will implicitly capture this notion 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 state that refuses an input action a.

Queue semantics

In [9], a *queue semantics* is defined for transition systems with asynchronous communication. This queue semantics 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 the control state i belongs to the set of initial states I and both input and output queues are empty. Each input and output move of the original system is broken up into two moves: a visible move that involves input/output between the input/output queue and the environment 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. This is captured by the following rules.

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

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

The first two rules describe how the queue based system reads inputs. External inputs are always accepted and appended to input queue, leaving the internal state unchanged. The system can then silently consume the action at the head of the input and update its state using a transition of the original system.

The last two rules deal with outputs. 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, leaving the internal state unchanged.

We denote by $Q(TS)$ the transition system corresponding to the configurations of TS using the queue semantics.

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

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 $i \xrightarrow{a_1} s_1 \xrightarrow{a2} \cdots \xrightarrow{a_n} s_n$ such that $i \in I$ and s_n is deadlocked. If TS has a maximal run over a word w, we call w a δ-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. 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.

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.

A *block observation* of TS is a sequence $(u_0, v_0)(u_1, v_1) \cdots (u_n, v_n)$ where $u_0 \in \Sigma_i^*$, $u_j \in \Sigma_i^+$ for $1 \leq j \leq n$, $v_k \in \Sigma_o^*$ for $0 \leq k \leq n$ and there is a maximal run $i \xrightarrow{w_0} s_0 \xrightarrow{w_1} \cdots \xrightarrow{w_n} s_n$ with $i \in I$ such that:

- The states s_0, s_1, s_2, …, s_n are the only deadlocked states along this run.
- $\forall 0 \leq j \leq n$, $w_j{\downarrow_{\Sigma_o}} = v_j$.
- $\forall 0 \leq j < n$, $w_j{\downarrow_{\Sigma_i}} = u_j$.
- Either $w_n{\downarrow_{\Sigma_i}} = u_n$ or $w_n{\downarrow_{\Sigma_i}} a \preceq u_n$ and s_n 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. Once again, the conditions on $w_n{\downarrow_{\Sigma_i}}$ implicitly capture receptiveness. 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.*

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

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 [9] 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 [1, 2]. Then,

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

We do not recall the formal definition of synchronous testing equivalence, because we do not require this branching-time formulation of the equivalance $\sim_Q$. Instead, it turns out that $\sim_Q$ admits a linear-time characterization (see Corollary 5.15 in [9]).

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, denoted @. Intuitively, u @ v if u can be observed as v by postponing some outputs. In the process, v could accept additional inputs.[3] Formally, u @ v if the following conditions hold.

- $u{\downarrow_{\Sigma_i}} \preceq v{\downarrow_{\Sigma_i}}$.
- $u{\downarrow_{\Sigma_o}} = v{\downarrow_{\Sigma_o}}$.
- For every pair of prefixes u_j, v_j of u, v of length j, $v_j{\downarrow_{\Sigma_o}} \preceq u_j{\downarrow_{\Sigma_o}}$.

By the first condition, the input actions of v can extend those of u. However, by the second condition both u and v must have the same sequence of outputs. Finally, the third condition asserts that each output letter in v appears no earlier than the corresponding output letter in u.

The relation @ is a partial order on Σ^*. It is not difficult 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 [9] 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 deadlocked 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

[3] The symbol @ should be pronounced "ape", in the sense "copy" or "mimic". Thus w @ w' is to be read as w is aped by w'.

recover. The first kind of deadlock is called an empty deadlock while the second kind of deadlock is called a blocked deadlock.

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 $w \mathbin{@} w'$ and $|w| = |w'|$. We can then define the empty and blocked deadlocks of $Q(TS)$.

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

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

Observe that $\delta_{\text{empty}}(Q(TS))$ is $|@|$-upward closed and $\delta_{\text{block}}(Q(TS))$ is @-upward closed. 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 $\delta_{\text{block}}(Q(TS_1)) \neq \delta_{\text{block}}(Q(TS_2))$ [9]. 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. For any transition system TS, we have $\mathit{IOBeh}(TS) = \{(u_0u_1 \ldots u_n, v_0v_1 \ldots v_n) \mid (u_0, v_0)(u_1, v_1) \ldots (u_n, v_n) \in \mathit{IOBlocks}(TS)\}$. From this, it follows that if $\mathit{IOBlocks}(TS_1) = \mathit{IOBlocks}(TS_2)$, then $\mathit{IOBeh}(TS_1) = \mathit{IOBeh}(TS_2)$. □

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.

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 $(u, v) \in \mathit{IOBeh}(TS_2)$.

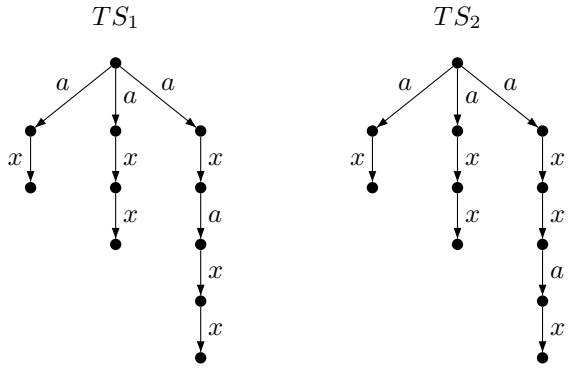
If $w \in \delta_{\text{block}}(Q(TS_2))$ then, in TS_2, there is a maximal run $i' \xrightarrow{w'} s'$ where s' refuses $\Sigma_o \cup \{b\}$ and $w'b \mathbin{@} w$ for some $b \in \Sigma_i$. Since $(w'b)\downarrow_{\Sigma_i} \preceq w\downarrow_{\Sigma_i} = u$ and $w'\downarrow_{\Sigma_o} = w\downarrow_{\Sigma_o}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

Case 2: Suppose $w\downarrow_{\Sigma_i} a \preceq u$ and s refuses a. 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 $wa \mathbin{@} wau'$ 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 $wau' \in \delta_{\text{traces}}(Q(TS_2))$.

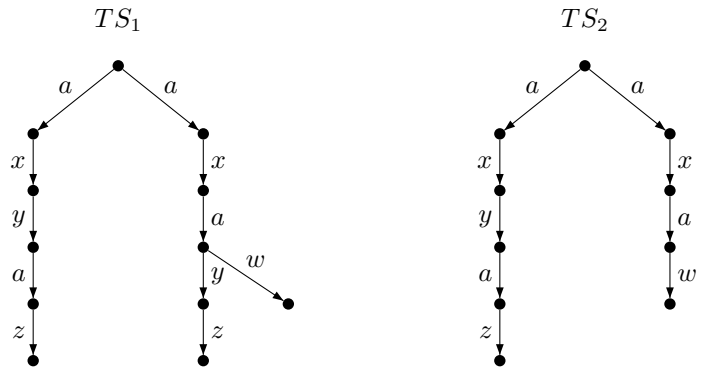
If $wau' \in \delta_{\text{empty}}(Q(TS_2))$ then, in TS_2, there is a maximal run $i' \xrightarrow{w'} s'$ with $w' \mid@\mid wau'$. Since $w'\downarrow_{\Sigma_i} = w\downarrow_{\Sigma_i} au' = u$ and $w'\downarrow_{\Sigma_o} = w\downarrow_{\Sigma_o} = v$, we have $(u, v) \in \mathit{IOBeh}(TS_2)$.

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 $w'b \mathbin{@} wau'$ 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 $(u, v) \in \mathit{IOBeh}(TS_2)$. □

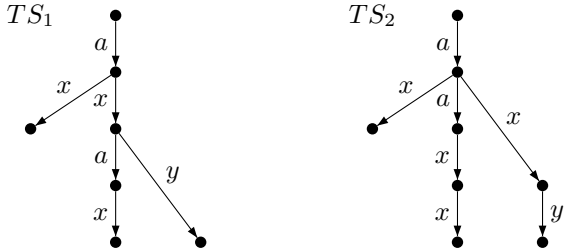
The implication we have proved is strict. Below, we give two transition systems that are related by $\sim_{io}$ but not by $\sim_Q$. Here $\Sigma_i = \{a\}$ and $\Sigma_o = \{x\}$. For both systems, the IO-behaviours are given by $\{(\varepsilon, \varepsilon), (a, x), (a, xx)\} \cup \{(a^n, x), (a^n, x^2), (a^n, x^3) \mid n > 1\}$, 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 $axxax \mathbin{@} axaxx$. Hence, $TS_1 \not\sim_Q TS_2$.



The equivalences $\sim_Q$ and $\sim_{\mathit{ioblock}}$ are incomparable. Below, we give two transition systems that are related by $\sim_Q$ but not by $\sim_{\mathit{ioblock}}$. Here, $\Sigma_i = \{a\}$ and $\Sigma_o = \{w, x, y, z\}$. Now, $\text{Tracks}(TS_1) = \text{Tracks}(TS_2) = \{\varepsilon, ax, axy, axyaz, axaw\}$. 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 $\{\varepsilon, axyaza, axawa\}$. Hence $TS_1 \sim_Q TS_2$. However, the incremental behaviour $(a, x)(a, yz)$ is in $\mathit{IOBlocks}(TS_1) \setminus \mathit{IOBlocks}(TS_2)$, so $TS_1 \not\sim_{\mathit{ioblock}} TS_2$. This example also establishes that $\sim_{io}$ is strictly weaker than $\sim_{\mathit{ioblock}}$ because the IO-behaviours of TS_1 and TS_2 are $\{(\varepsilon, \varepsilon), (a, x), (a, xy)\} \cup \{(a^n, xyz), (a^n, xw) \mid n > 1\}$.



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, $\mathit{IOBlocks}(TS_1) = \mathit{IOBlocks}(TS_2) = \{(\varepsilon, \varepsilon)\} \cup \{(a^n, xx) \mid n \geq 2\} \cup \{(a^n, x), (a^n, xy) \mid n \geq 1\}$, so $TS_1 \sim_{ioblock} TS_2$.



5 Decidability of asynchronous test equivalence

We now examine the decidability of asynchronous test equivalence for finite-state systems. We show that $\sim_{io}$ and $\sim_Q$ are both undecidable. For $\sim_{ioblock}$, we can establish decidability for the case where states are split as input and output states. Finally, we show that $\sim_{io}$ is decidable for test specifications corresponding to unlabelled message-sequence charts.

5.1 Undecidability of $\sim_{io}$

We prove this result using a reduction from the equivalence problem for rational relations. 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, we can view R as a mapping from A^* to $\mathcal{P}(B^*)$ defined for $u \in A^*$ by $R(u) = \{v \in B^* \mid (u, v) \in R\}$.

In the following, we let $K = \text{Rat}(B^*)$. 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 we are done reading the input word and we exit the automaton in state j. The relation $\mathcal{R}(\mathcal{A})$ realized by $\mathcal{A}$ is defined for $u = a_1 \cdots a_k \in A^*$ by $(u, v) \in \mathcal{R}(\mathcal{A})$ iff there exists $i_0, \ldots, i_k \in S$, with $v \in \lambda_{i_0} \mu(a_1)_{i_0,i_1} \cdots \mu(a_k)_{i_{k-1},i_k} \gamma_{i_k}$. With union as addition and concatenation as multiplication, the set K is a semiring with $\emptyset$ as zero element and $\{\varepsilon\}$ as unit. Hence the set of matrices $K^{S \times S}$ equipped with matrix multiplication is a monoid and we can extend μ to a monoid morphism $\mu : A^* \to K^{n \times n}$. Viewing λ as a row vector and γ as a column vector, we have $\mathcal{R}(\mathcal{A})(u) = \lambda \mu(u) \gamma$ for each $u \in A^*$.

A relation $R \subseteq A^* \times B^*$ is rational if and only if it can be realized by some K-automaton. Without loss of generality, we may assume that $\lambda_i \neq \emptyset$ implies $\lambda_i = \{\varepsilon\}$ for each state $i \in S$.

The equivalence problem for rational relations given by finite K-automata is undecidable [8]. This undecidability holds even for rational relation for which $|B| = 1$ and given by a K-automaton where K is the set $\mathcal{P}_{\text{fin}}(B^*)$ of *finite* subsets of B^*. So in the following we assume that $B = \{b\}$ is a singleton and that $K = \mathcal{P}_{\text{fin}}(B^*)$.

In the construction below, 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. The reduction is easy. Let $\mathcal{A} = (S, \lambda, \mu, \gamma)$ be a K-automaton and 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 identify all states (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 $\mathcal{A}' = (S', I, \Sigma, \to)$ 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 should be clear for those familiar with rational relations and K-automata. Its proof is included in the Appendix, for completeness.

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

Proof. First, the maximal paths in $\mathcal{A}'$ are of the form $p \xrightarrow{w} q$ for $p \in I$ and $q \in S$. They 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, $IOBeh(\mathcal{A}') \subseteq IOBeh(\mathcal{A}'')$. 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 $IOBeh(\mathcal{A}'') \cap A^* \times B^* = IOBeh(\mathcal{A}')$.

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 $v' \in \mu(u)$ 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 $v\#^{1+|x|} = v'v''\#^{1+k} = (wv''\#w'){\downarrow_{\Sigma_o}}$. Since f_q is a deadlocked state we deduce that $(ux, v\#^{1+|x|}) \in IOBeh(\mathcal{A}'')$.

Conversely, let $(u', v') \in IOBeh(\mathcal{A}'') \setminus A^* \times B^*$. 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. 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 $w = w_1w_2$. 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. By Lemma 6, we have

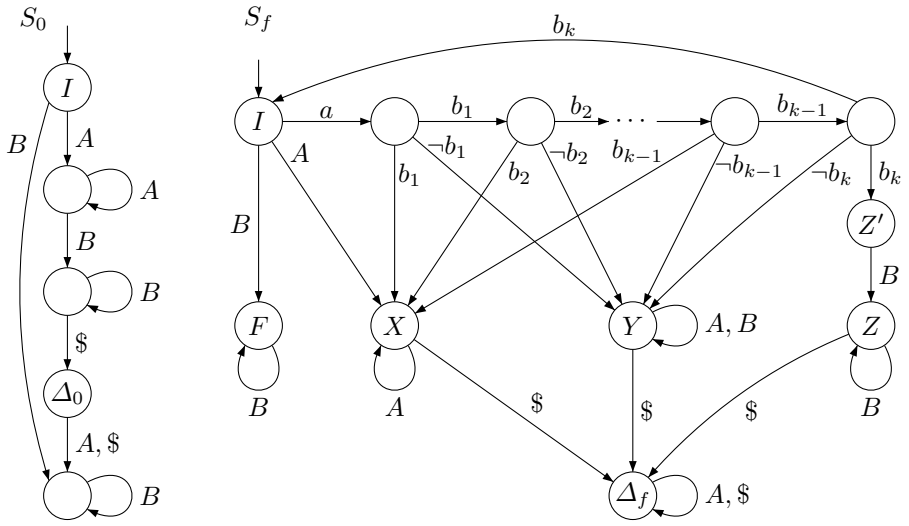
$$IOBeh(\mathcal{A}' \uplus \mathcal{B}'') = IOBeh(\mathcal{A}') \cup IOBeh(\mathcal{B}') \cup \mathcal{R}(\mathcal{B})\{(x, \#^{1+|x|}) \mid x \in A^*\}$$
$$IOBeh(\mathcal{A}'' \uplus \mathcal{B}') = IOBeh(\mathcal{A}') \cup IOBeh(\mathcal{B}') \cup \mathcal{R}(\mathcal{A})\{(x, \#^{1+|x|}) \mid x \in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

The result follows. □

5.2 Undecidability of $\sim_Q$

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

We consider a new symbol \$ we 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 below:



As usual, the transition system S_f corresponds to the morphism f and has one loop $ab_1b_2 \dots b_k$ for each $a \in A$ such that $f(a) = b_1b_2 \dots b_k$. Formally the set of states of S_f is $Q_f = \{I, F, X, Y, Z, Z', \Delta_f\} \cup \{(a, i) \mid a \in A, 0 < i < |f(a)|\}$ and its initial state in 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 $f(a) = b_1b_2 \cdots b_k$, 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:

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

Here, $S_i + S_j$ denotes the disjoint union of the two systems with multiple initial states.

The broad idea is that the deadlocks of $Q(S_0)$ will be contained in those of M_2 if and only if the PCP has no solution.

The only deadlocked state in S_0 is Δ_0 and this state does not refuse any input letter. Therefore, $\delta_{\text{block}}(S_0) = \emptyset$. Similarly, the only deadlocked states in S_f are X and Δ_f and none of them refuses some input letter so that we also get $\delta_{\text{block}}(S_f) = \emptyset$. Therefore,

$$\delta_{\text{traces}}(M_1) = \delta_{\text{empty}}(S_0) \cup \delta_{\text{empty}}(M_2) \quad \text{and} \quad \delta_{\text{traces}}(M_2) = \delta_{\text{empty}}(M_2)$$

and $M_1 \sim_Q M_2$ if and only if

$$\text{Tracks}(M_1) = \text{Tracks}(M_2) \quad \text{and} \quad \delta_{\text{empty}}(S_0) \subseteq \delta_{\text{empty}}(M_2).$$

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 $\text{Tracks}(S_f) = B^*$. □

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 $B^+A^+\$$. □

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

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 distinghish 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 $w \mid@\mid uv$ 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 $w \mid@\mid uv$ 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 $w \mid@\mid uv$ and $I \xrightarrow{w} Z$ in S_f.

Hence, there is a run $I \xrightarrow{w\$} \Delta_f$ in S_f and since $w\$ \mid@\mid uv\$$ we obtain $uv\$ \in \delta_{\text{empty}}(S_f)$.

Conversely, let $I \xrightarrow{w'} s$ be a run in S_f with s deadlocked and $w' \mid@\mid uv\$$. Since $\$$ must occur in w' we deduce that $s = \Delta_f$ and $w' = w\$$ with $w \mid@\mid uv$. 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 $v \neq f(u)$. □

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, $M_1 \sim_Q M_2$. □

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

We will show that $\sim_{ioblock}$ is decidable when the states are partitioned as input and output states. An input state is a state that refuses Σ_o and an output state is one that refuses Σ_i. A *well structured* transition system is one in which every state is either an input state or an output state.

In a well-structured transition system TS, it is not difficult to establish that every member of $\mathit{IOBlocks}(TS)$ is of the form $(\varepsilon, v_0)(a_1, v_1) \ldots (a_n, v_n)$, with each $a_j \in \Sigma_i$ and each $v_j \in \Sigma_o^*$, corresponding to a maximal run of the form $i \xrightarrow{v_0} s_0 \xrightarrow{a_1 v_1} s_1 \cdots s_{n-1} \xrightarrow{a_n v_n} s_n$.

From this, it follows that $TS_1 \sim_{ioblock} TS_2$ iff $L_\delta(TS_1) = L_\delta(TS_2)$, where $L_\delta(TS)$ is the language obtained by regarding TS as a finite-state automaton with all deadlocked states as final states. Thus, we have the following result.

Theorem 12. *The relation $\sim_{ioblock}$ is decidable for well structured transition systems.*

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

A message sequence chart, or MSC, is a diagram that visually represents a sequence of communications between a set of agents [5]. 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 [3]. 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 result.

Theorem 13. *The relation $\sim_{io}$ is decidable for tests described using unlabelled MSCs.*

6 Future work

We have presented two intuitive notions of asynchronous testing and compared their expressive power with the definition due to Tretmans. For one of these, $\sim_{ioblock}$, we have a positive decidability result for a large class of interesting systems. 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 problem. Another interesting issue is to see how testing can be done in a distributed manner.

References

1. R. de Nicola and M. Hennessy: Testing equivalences for processes, *Theoretical Computer Science*, **34** (1984) 83–133.
2. 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.
3. J.G. Henriksen, M. Mukund, K. Narayan Kumar, M. Sohoni and P.S. Thiagarajan: A Theory of Regular MSC Languages. *Information and Computation*, **202**(1) (2005) 1–38.
4. ISO (1992) Information Technology — Open Systems Interconnection Conformance Testing Methodology and Framework. ISO/IEC 9646-1/2/3. Part 1: General concept – Part 2: Abstract test Suite Specification — Part 3: The Tree and Tabular Combined Notation (TTCN).
5. ITU-TS Recommendation Z.120: *Message Sequence Chart (MSC)*. ITU-TS, Geneva (1997).
6. C. Jard: Synthesis of distributed testers from true-concurrency models of reactive systems, *Information & Software Technology*, **45**(12) (2003) 805–814.
7. C. Jard and T. Jéron: TGV: theory, principles and algorithms. *Software Tools for Technology Transfer*, **7**(4)(2005) 297–315.
8. J. Sakarovitch: *Eléments de théorie des automates*, Vuibert (2003).
9. J. Tretmans: *A formal approach to conformance testing*, PhD Thesis, University of Twente, The Netherlands (1992).

A Appendix

Here we supply the proof of

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

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

Proof. Let $v \in \mu(u)_{p,q}$ with $u = a_1 \cdots a_k \in A^k$. Since $\mu(u) = \mu(a_1) \cdots \mu(a_k)$ we find a sequence of states $p = p_0, p_1, \cdots, p_{k-1}, p_k = q$ such that we have $v \in \mu(a_1)_{p_0,p_1} \cdots \mu(a_k)_{p_{k-1},p_k}$. Hence, $v = v_1 \cdots v_k$ with $v_j \in \mu(a_j)_{p_{j-1},p_j}$ for each $0 < j \leq k$. By definition of $\mathcal{A}_{p_{j-1},a_j,p_j}$ we find in this automaton a path $i_{p_{j-1},a_j,p_j} \xrightarrow{v_j} f_{p_{j-1},a_j,p_j}$. Therefore, we have in $\mathcal{A}'$ the path

$$p_0 \xrightarrow{a_1} i_{p_0,a_1,p_1} \xrightarrow{v_1} p_1 \quad \cdots \quad p_{k-1} \xrightarrow{a_k} i_{p_{k-1},a_k,p_k} \xrightarrow{v_k} p_k,$$

that is, a path $p \xrightarrow{w} q$ with $w = a_1 v_1 \cdots a_k v_k$. Since $u = w{\downarrow_A}$ and $v = w{\downarrow_B}$ we deduce that $(u, v) \in T_{p,q}$.

The converse can be shown similarly. □