Input-Output Conformance Simulation (iocos) for Model Based Testing*

Carlos Gregorio-Rodríguez, Luis Llana, and Rafael Martínez-Torres

Departamento Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
cgr@sip.ucm.es, llana@ucm.es, rmartine@fdi.ucm.es

Abstract. A new model based testing theory built on simulation semantics is presented. At the core of this theory there is an **i**nput-**o**utput **co**nformance **s**imulation relation (iocos). As a branching semantics iocos can naturally distinguish the context of local choices. We show iocos to be a finer relation than the classic ioco conformance relation. It turns out that iocos is a transitive relation and therefore it can be used both as a conformance relation and a refinement preorder. An alternative characterisation of iocos is provided in terms of testing semantics. Finally we present an algorithm that produces a test suite for any specification. The resulting test suite is sound and exhaustive for the given specification with respect to iocos.

Keywords: Model Based Testing, Input Output Conformance Simulation, Formal Methods.

1 Introduction and Related Work

Model-Based Testing (MBT) is an active research area whose goals are to increase correctness and efficiency of the testing process and, at the same time, reducing the costs, in a world of ever increasingly complex systems. From the quite ad-hoc techniques used in seminal papers (see for instance [5]), the use of formal methods in MBT and, in particular, behavioural models to describe the system specification, has made it possible to develop theories, frameworks and tools that automatically produce efficient set of test cases from a model. This allows an effective automation of the testing process to assess whether the system under test behaves as expected by its specification.

A key point at the core of any MBT theory is the implementation or conformance relation, stating whether an implementation is *correct*, in some sense, with respect to a given specification. Tretmans' input-output conformance relation (ioco) [19] is one of the most established ones. A whole MBT framework has been developed around the ioco relation, from theory to tools [20,21,4].

* Research partially supported by the Spanish MEC projects TIN2009-14312-C02-01 and TIN2012-36812-C02-01.

D. Beyer and M. Boreale (Eds.): FMOODS/FORTE 2013, LNCS 7892, pp. 114–129, 2013.

As a behavioural relation over labelled transition systems, ioco can be classified as a linear semantics [9], as most of the relations used in MBT are. That means that they are essentially based on traces and that brings advantages and drawbacks well known in process theory.

One of the disadvantages of linear semantics is the limitation to observe the execution context, that is, the different available choices at a given point. The simple behaviours in Figure 1 highlight this situation. [1] One of the goals guiding our research is precisely to find a MBT theory capable of identifying the implementations conforming not only the trace executions, but also the choices in the specification.

The starting point for our work has been inspired in basic results in process theory: branching semantics [9] form a family of relations that can naturally distinguish the execution context for a process. These semantics are essentially based on simulation [13], they are easily defined by coinduction and thus coalgebraic techniques can be applied in its study. Simulation can be characterised as a game, there exist algorithms to compute it [16,10] and, together with bisimulation [14], is a rather natural and pervasive concept that appears in many different contexts in the literature.

However, there is not much work on MBT and simulation relations. As far as we know the only related work is the one developed by a group at *Microsoft Research* that has several publications on MBT (see for instance [24,23]). Their framework is built on the alternating simulation relation [3,2] defined for interface automata.

This paper is organized as follows: in Section 2, we introduce the formal framework used to model behaviours and some technical definitions and notations used along the paper, including the classic ioco relation. Section 3 presents a reasoned exposition of examples discussing what might and might not be considered a correct implementation of a given specification. This exposition highlights differences and similarities between conformance relations based on linear and branching semantics. These examples settle also the goals that we want to fulfil with the formal definition of the iocos relation. We finish this section by proving iocos to be a refinement of ioco on input-output labelled transition systems. Section 4 focuses on describing and proving the results that show iocos to be a suitable relation for MBT. First, we provide a language for tests and formally define the test execution of a behaviour or implementation. Then we show that iocos can be characterised with a preorder defined in the classic style of testing semantics [15,11]. Finally, we define an algorithm to automatically generate a test suite from a given specification, we prove that the implementations that pass this test suite are exactly those that are in iocos relation with the given specification. Lastly, Section 5 summarises the goals achieved as well as some future lines of research.

[1] In Section 3 we thoroughly get into the details and use further examples to illustrate the limitations of linear semantics.

2 Preliminaries

A common formalism used in MBT to represent not only the models but also the implementations and even the tests are labelled transition systems. In order to deal with input-output behaviours we are going to consider two disjoint finite sets of actions: inputs I and outputs O. Output actions are those initiated by the system, they will be annotated with an exclamation mark, $a!, b!, x!, y! \in O$. Input actions are initiated by the environment and will be annotated with a question mark, $a?, b?, x?, y? \in I$. In many cases we want to name actions in a general sense, inputs and outputs indistinctly. We will consider the set $L = I \cup O$ and we will omit the exclamation or question marks when naming generic actions, $a, b, x, y \in L$.

A state with no output actions cannot autonomously proceed, such a state is called *quiescent*. Quiescence is an essential component of the ioco theory. For the sake of simplicity and without lost of generality (see for instance [20,18]), we directly introduce the event of quiescence as a special action denoted by δ into the definition of our models.

Definition 1. A *labelled transition system with inputs and outputs* is a 4-tuple $(S, I, O, \rightarrow)$ such that

- S is a set of states or behaviours.
- I and O are disjoint sets of input and output actions respectively.We define $L = I \cup O$ and consider a new symbol $\delta \notin L$ for *quiescence*. We will consider also the sets $L_\delta = L \cup \{\delta\}$ and $O_\delta = O \cup \{\delta\}$.
- $\rightarrow \subseteq S \times L_\delta \times S$. As usual we write $p \xrightarrow{a} q$ instead of $(p, a, q) \in \rightarrow$ and $p \xrightarrow{a}$, for $a \in L_\delta$, if there exists $q \in S$ such that $p \xrightarrow{a} q$. Analogously, we will write $p \not\xrightarrow{a}$, for $a \in L_\delta$, if there is no q such that $p \xrightarrow{a} q$. In order to allow only coherent quiescent systems the set of transitions should also satisfy:
  - if $p \xrightarrow{\delta} p'$ then $p = p'$. A quiescent transition is always reflexive.
  - if $p \not\xrightarrow{o!}$ for any $o! \in O$, then $p \xrightarrow{\delta} p$. A state with no outputs is quiescent.
  - if there is $o! \in O$ such that $p \xrightarrow{o!}$, then $p \not\xrightarrow{\delta}$. A quiescent state performs no output actions. □

For the sake of simplicity, we will denote the set of labelled transition systems with inputs and outputs just as LTS. In general we use $p, q, p', q' \ldots$ for states or behaviours, but also i, i', s and s' when we want to emphasise the concrete role of a behaviours as implementation or specification.

Without losing generality, we will consider implementations and specifications, or, more in general, behaviours under study, as states of the same LTS[2]. This modification simplifies the coinductive definition we are going to present and the reasoning in the proofs.

[2] If we had two different $LTSs$, one for a specification and one for the implementation, we could always consider the larger LTS that is the disjoint union of the original $LTSs$.

Traces play an important role gathering basic information for behaviours. A trace is a finite sequence of symbols of L_δ. We will normally use the symbol σ to denote traces, that is, $\sigma \in L_\delta^*$. The empty trace is denoted by ϵ and we juxtapose, $\sigma_1\sigma_2$, to indicate concatenation of traces. The transition relation of labelled transition systems can naturally be extend using traces instead of single actions.

Definition 2. Let $(S, I, O, \rightarrow) \in LTS$, $p, q \in S$ and $\sigma \in L_\delta^*$. We inductively define $p \xrightarrow{\sigma} q$ as follows:

- $p \xrightarrow{\epsilon} p$
- $p \xrightarrow{a\sigma} q$ for $a \in L_\delta$, $\sigma \in L_\delta^*$ and $p' \in S$ such that $p \xrightarrow{a} p'$ and $p' \xrightarrow{\sigma} q$. □

Next we introduce some definitions and notation that will be frequently used along the paper.

Definition 3. Let $(S, I, O, \rightarrow) \in LTS$, and $p \in S$, $S' \subseteq S$, and $\sigma \in L_\delta^*$, we define:

1. $\mathsf{init}(p) = \{a \mid a \in L_\delta,\ p \xrightarrow{a}\}$, the set of initial actions of p.
2. $\mathsf{traces}(p) = \{\sigma \mid \sigma \in L_\delta^*,\ p \xrightarrow{\sigma}\}$, the set of traces from p.
3. $p\ \mathsf{after}\ \sigma = \{p' \mid p' \in S,\ p \xrightarrow{\sigma} p'\}$, the set of reachable states from p after the execution of trace σ.
4. $\mathsf{outs}(p) = \{x \mid x \in O_\delta,\ p \xrightarrow{x}\}$, the set of outputs of a state p or the quiescent symbol δ.
5. $\mathsf{outs}(S') = \bigcup_{p \in S'} \mathsf{outs}(p)$, the set of outputs of a set of states S'.
6. $\mathsf{ins}(p) = \{x? \mid x? \in I,\ p \xrightarrow{x?}\}$, the set of inputs of a state p. □

A classical requirement for the ioco relation in [20] is that implementations should be *input enabled*, that means that the system is always prepared to perform any input action and therefore all inputs are enabled in all states. Although this assumption maybe natural in some contexts is not so in others. For instance, in a vending machine, a slot, for a credit card or parking ticket, can be only enabled if a card is not inserted; much alike, developers of graphical interfaces do not need to consider any possible event on a window, they just code the response for the interesting events, etc.

Moreover, in the ioco theory, while implementations must be input enabled, specifications do not need to fulfil this requirement. So the original ioco relation is defined between two different domains, general input-output labelled transition systems for specifications and input enabled input-output labelled transition systems for implementations. Hence, the ioco relation is not transitive and cannot be used as a refinement relation: once an implementation conforms a specification, the implementation fixes all the behaviour regarding the input actions; so there is little freedom, if any, to continue the refining process.

In our framework we do not require the implementations to be input enabled. As usual in other testing frameworks, specifications and implementations are expressed in the same formalism, in particular we are going to use *LTSs* for both implementations and specifications.

In order to compare the original ioco relation with the conformance relation we are going to define in next section, we have to adapt the ioco definition to our framework.

Definition 4. Let $(S, I, O, \rightarrow) \in LTS$, the relation $\mathsf{ioco} \subseteq S \times S$ is defined as follows: $i \;\mathsf{ioco}\; s \Leftrightarrow_{def} \forall \sigma \in \mathsf{traces}(s) : \mathsf{outs}(i \;\mathsf{after}\; \sigma) \subseteq \mathsf{outs}(s \;\mathsf{after}\; \sigma)$ □

The ioco relation we use keeps the spirit of the original in [20], but while the original imposed implementations to be input enabled, our definition has been extended to the more general domain of input-output labelled transition systems. Also, the original definition used "suspension traces" (Definition 9 in [20]) while we can consider just traces because the quiescence symbol has already been introduced in the description of the behaviours.

3 Input-Output Conformance Simulation (IOCOS)

In this section we will present the alternative relation that we propose. We have defined this relation according to the following criteria in mind: first, the ioco relation is a well known and accepted relation, we want to find a refinement of the original ioco relation while keeping as close as possible to it. Second, it should be defined as a simulation relation, so we can benefit from the work in this field. Finally, there should be a testing framework, similar to the ones in [1,20]. In order to simplify the reading, we are going to mark the quiescent states as ○, these kind of nodes are shorthands for .

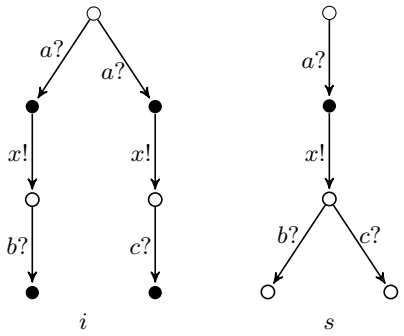

Fig. 1. i ioco s and i iocos s

Next we are going to present some examples that motivate the definition of iocos (Definition 5). The rationale behind the ioco definition is to serve as an *observation methodology* to relate a specification and an implementation. This methodology binds the environment to *traces*, the observations to *output signals*, and comparison to *set inclusion*. Taking traces as environments prevent to distinguish non-determinism. Looking at Figure 1, there are arguments to discard i as a sound implementation of s: after the trace $a?x!$, s can react *always*

to both signals $b?$ and $c?$, while this is not true for i: depending on the selected branch, i can only react to either $b?$ or $c?$ but not both. As we will see, simulation tecniques provide us with necessary insight to solve this problem.

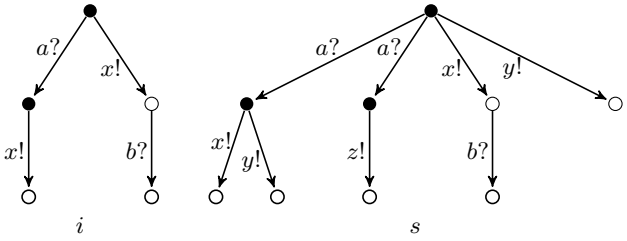

Fig. 2. i ioco s and i iocos s

There are two distinguised trends when dealing with specifications: the *initial* semantics, where a specification sets a minimum that an implementation must fit to be qualified as sound; and the *final* semantics, where specification stands rather like a limit for implementation's behaviour. Both are legitimate, but subset relation $\subseteq$ at Definition 4, unveils ioco clearly in this last category. We also adopt this finality approach: any behaviour of a correct implementation must be considered valid by the specification. Graphically, Figure 2, can summarize this with the next idea: any subtree in the implementation is a subtree in the specification.

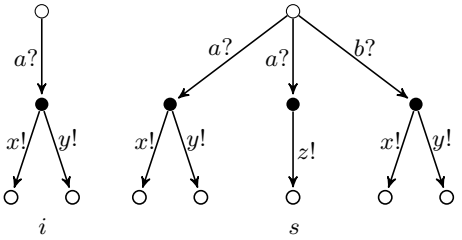

Fig. 3. i ioco s and i iocȯs s

Traditionally, an objection is made to this approach: partial or even empty implementations can be accepted as sound. In this sense we want to reach a trade-off between the previous postulate, specification as limit, while avoid such tricky implementations.

This subttle requirement can be formulated for input actions in the following terms: at least one of each of input actions considered by the specification should be implemented. In this way we can have arguments to discard i as a sound implementation of s in Figure 3: the implementation cannot react to the input action $b?$. However, limits on input actions should be applicable only to those prompted by the specification. Beyond that, implementation should be free to behave. This is the case of the input action $b?$ in Figure 4.

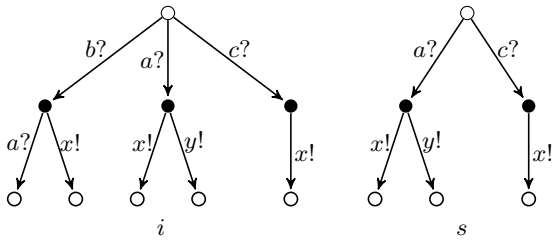

Fig. 4. i ioco s and i iocos s

Now we can give the formal definition of iocos. It reflects the ideas presented in the examples above. Since it is a simulation relation, it cannot be defined directly. So first we give the notion of an iocos-relation. Then the iocos relation would be the union of all iocos-relations.

Definition 5. Let $(S, I, O, \rightarrow) \in LTS$, we say that a relation $R \subseteq S \times S$ is a iocos-relation iff for any $(p, q) \in R$ the following conditions hold

1. $\mathsf{ins}(q) \subseteq \mathsf{ins}(p)$
2. For all $a? \in \mathsf{ins}(q)$ such that $p \xrightarrow{a?} p'$ there exists $q' \in S$ such that $q \xrightarrow{a?} q'$ and $(p', q') \in R$. [3]
3. For all $x \in \mathsf{outs}(p)$ such that $p \xrightarrow{x} p'$ there exists $q' \in S$ such that $q \xrightarrow{x} q'$ and $(p', q') \in R$.

We define the *input-output conformance simulation* as

$$\mathsf{iocos} = \bigcup \{R \mid R \subseteq S \times S,\ R \text{ is a } \mathsf{iocos}\text{-relation}\}$$

and we write p iocos q instead of $(p, q) \in$ iocos. □

Next technical results show that iocos is indeed a well defined preorder relation on LTS, for this purpose we need the following lemma.

Lemma 1. Let $(S, I, O, \rightarrow) \in LTS$, the following properties hold:

- $Id = S \times S$ is a iocos-relation.
- Let $R, R' \subseteq S \times S$ be two iocos-relations, then $R \circ R' = \{(p, r) \mid \exists q \in S : (p, q) \in R \wedge (q, r) \in R'\}$ is a iocos-relation.

□

Corollary 1. Let $(S, I, O, \rightarrow) \in LTS$, then iocos is a preorder. □

The rest of this section is devoted to prove that iocos is a finer relation than ioco. Let us start proving a simple lemma that is a direct consequence of the definition of the function after.

[3] Let us note that the Condition 2 does not imply Condition 1.

Lemma 2. Let $(S, I, O, \rightarrow) \in LTS$, $p \in S$, $a \in L_\delta$ and $\sigma' \in L_\delta^*$, then

$$p \text{ after } a\sigma' = \bigcup\{p' \text{ after } \sigma' | p \xrightarrow{a} p'\}$$

□

Theorem 1. Let $(S, I, O, \rightarrow) \in LTS$ then iocos $\subseteq$ ioco. That is, for any $p, q \in S$, whenever we have p iocos q it is also true that p ioco q.

Proof. Let us consider $p, q \in S$ such that p iocos q. According to Definition 4, it suffices to prove

$$\forall \sigma \in \text{traces}(q) : \text{outs}(p \text{ after } \sigma) \subseteq \text{outs}(q \text{ after } \sigma).$$

We will prove this by induction on length of trace σ.

Basis: $\sigma = \epsilon$. First, $\text{outs}(p \text{ after } \epsilon) = \text{outs}(p)$ and $\text{outs}(q \text{ after } \epsilon) = \text{outs}(q)$, by definition of the after function. Then $\text{outs}(p) \subseteq \text{outs}(q)$, by Definition 5.3 of iocos.

Induction: $\sigma = a\sigma', a \in L_\delta$. If $p \xrightarrow{a}\!\!\!\!\!/$ then $\sigma \notin \text{traces}(p)$ and $\text{outs}(p\text{after}\sigma) = \varnothing$. So let us consider p' such that $p \xrightarrow{a} p'$. Since $\sigma \in \text{traces}(q)$, $q \xrightarrow{a}$. If $a \in I$ then $a \in \text{ins}(q)$ and by Definition 5.2 there is $q' \in S$ such that $q \xrightarrow{a} q'$ and p' iocos q'. If $a \in O_\delta$, then by Definition 5.3 there is $q' \in S$ such that $q \xrightarrow{a} q'$. and p' iocos q'. Hence, in any case there is $q' \in S$ such that $q \xrightarrow{a} q'$ and p' iocos q'. By induction $\text{outs}(p' \text{ after } \sigma') \subseteq \text{outs}(q' \text{ after } \sigma')$. Therefore:

$$\text{outs}(p \text{ after } \sigma) = \text{outs}(p \text{ after } a\sigma') = \bigcup\{\text{outs}(p' \text{ after } \sigma') \mid p \xrightarrow{a} p'\} \subseteq$$
$$\bigcup\{\text{outs}(q' \text{ after } \sigma') \mid p \xrightarrow{a} p', q \xrightarrow{a} q', p' \text{ iocos } q'\} \subseteq$$
$$\bigcup\{\text{outs}(q' \text{ after } \sigma') \mid q \xrightarrow{a} q'\} = \text{outs}(q \text{ after } a\sigma') = \text{outs}(q \text{ after } \sigma)$$ □

Complementing the previous result, examples in Figures 1 and 3 show that iocos is a strict refinement of ioco.

4 Testing Framework

After presenting the ideas behind the iocos relation and its essential basic properties, in this section, we show that iocos can indeed be a suitable relation for MBT: for any specification a test suite can be automatically derived such that an implementation would be correct (wrt iocos) if and only if it passes all the tests in the test suite.

The technical approach we use to achieve these results is slightly different to the one used by Tretmans for ioco. We first show iocos to have a characterisation as a testing semantics, Definition 9, in the classic sense of De Nicola and Hennessy. Then we define an algorithm, Definition 10, that for a given behaviour generates a set of tests (test suite) that can discriminate the suitable implementations or refinements of that behaviour, Theorem 3. We use the testing characterisation to prove this result.

4.1 Tests Definition and Execution

The kind of experiments we have to conduct on behaviours are called tests. Tests would play the role of environments for the implementations.

There are two kind of choices in the tests: the one corresponding to the $+$ operator and the one corresponding to the $\oplus$ operator. The former is the usual choice operator as in [20]. The latter, borrowed from [1], corresponds to an *or* operator: in order p to pass $T_1 \oplus T_2$ it is enough that P passes either T_1 or T_2. As in [1], the presence of these choice operators implies the ability of make copies of the machine at intermediate points. Then it is necessary to perform the tests on the copies, and finally to combine the results to obtain the outcome of the overall test.

Definition 6. A *test* is a syntactical term defined by the following Backus-Naur Form:

$$T = \text{✗} \mid \text{✓} \mid T_1 \oplus T_2 \mid T_1 + T_2 \mid a;T \qquad \text{where } a \in L_\delta$$

We denote the set of tests as $\mathcal{T}$. □

As usual in MBT, the environments we want to model with tests have some added particularities that we need to consider. First, tests should be able to respond at any moment to any possible output of the implementation under test. That is, tests like $a?;T_{a?}$ with $a? \in I$, will not be accepted as valid tests. These test should be completed into tests like $a?;T_{a?} + \sum_{x \in O_\delta} x;T_x$. For the sake of simplicity we use $\sum_{i \in \{1,\ldots,n\}} T_i$ as a shortcut for $T_1 + \cdots + T_n$. Analogously, $o!;T_{o!}$ is not an acceptable test, instead we need to consider the test $o!;T_{o!} + \sum_{x \in O_\delta,\ x \neq o!} x;T_x$. All these conditions are reflected in the following definition.

Definition 7. Let $T \in \mathcal{T}$ be a test, T is *valid* iff it has one of the following forms:

1. $T = \text{✗}$ or $T = \text{✓}$.
2. $T = a?;T_{a?} + \sum_{x \in O_\delta} x;T_x$ where $x \in O_\delta$, $a? \in I$, and $T_{a?}$, T_x are valid tests.
3. $T = \sum_{x \in O_\delta} x;T_x$ where T_x is a valid test for $x \in O_\delta$.
4. $T = T_1 \oplus T_2$ where T_1 and T_2 are valid tests.

We denote the set of valid tests as $\mathcal{T}_v$. □

So far we have a language for tests, and we have now to define how these tests interacts with a behaviour and what will the result of the execution of that experiment be. Following the ideas of Abramsky in [1] we use a predicate to define the outcomes of the interaction between a test and the behaviour or implementation being tested.

Definition 8. Let $(S, I, O, \rightarrow) \in LTS$, we inductively define the predicate $\mathsf{pass} \subseteq S \times \mathcal{T}$ as follows (let us assume that $s \in S$, $a? \in I$, and $x \in O_\delta$)

$$
\begin{array}{l}
s \;\mathsf{pass}\; \bigstar = false \\
s \;\mathsf{pass}\; \checkmark = true \\
s \;\mathsf{pass}\; x; T_x = \begin{cases} true & \text{if } x \notin \mathsf{outs}(s) \\ \bigwedge\{s' \;\mathsf{pass}\; T_x | s \xrightarrow{x} s'\} & \text{otherwise} \end{cases} \\
s \;\mathsf{pass}\; a?; T_{a?} = \begin{cases} false & \text{if } a? \notin \mathsf{ins}(s) \\ \bigwedge\{s' \; pass \; T_{a?} | s \xrightarrow{a?} s'\} & \text{otherwise} \end{cases} \\
s \;\mathsf{pass}\; T_1 + T_2 = s \;\mathsf{pass}\; T_1 \wedge s \;\mathsf{pass}\; T_2 \\
s \;\mathsf{pass}\; T_1 \oplus T_2 = s \;\mathsf{pass}\; T_1 \vee s \;\mathsf{pass}\; T_2
\end{array}
$$

□

Let us note that for the sake of convenience the predicate pass is defined over the whole set of tests, with a simpler structural formulation, while at the end we will only be interested in valid tests.

Next let us show the tests that discriminate i and s in the previous examples when $i \,\mathsf{ioco}\!\!\not\,\, s$. The test $T = a?; x!; b?; \checkmark$ differentiates the behaviours in Figure 1. It is passed by the specification s but not by the implementation i. Let us note that this test is not valid. But its related valid test

$$T^v = a?; (x!; (b?; \checkmark + x!; \bigstar + \delta; \checkmark) + \delta; \bigstar) + x!; \bigstar + \delta; \checkmark$$

also differenciates i and s.

The next test is related to Figure 2. Since $i \,\mathsf{ioco}\, s$, because of Theorem 2, we cannot find a test that distinguishes i and s. Since the number of potential test is infinite, it is not feasible to check all of them. In this case there is a *maximal test* according to the algorithm in Definition 10. This test is the following one:

$$
\begin{array}{l}
\mathit{let} \\
\quad T_{a_1?} = (x!\checkmark + y!; \checkmark + z!; \bigstar + \delta; \bigstar) \\
\quad T_{a_2?} = (z!; \checkmark + x!; \bigstar + y!; \bigstar + \delta; \bigstar) \\
\quad T_{b?} = (b?\checkmark + x!; \bigstar + y!; \bigstar + z!; \bigstar + \delta; \checkmark) \\
\mathit{in} \\
\quad T = a?; (T_{a_1?} \oplus T_{a_2?}) + x!; T_{b?} + y!; \checkmark + z!; \bigstar + \delta; \bigstar
\end{array}
$$

It is easy to check that both $i \;\mathsf{pass}\; T$ and $s \;\mathsf{pass}\; T$. All other tests generated according to Defintion 10 can be built from the previous test by pruning branches.

For the specification s in Figure 3 there are two maximal tests according to Defintion 10:

$$
\begin{array}{l}
\mathit{let} \\
\quad T_{a_1?} = (x!\checkmark + y!; \checkmark + z!; \bigstar + \delta; \bigstar) \\
\quad T_{a_2?} = (z!; \checkmark + x!; \bigstar + y!; \bigstar + \delta; \bigstar) \\
\quad T_{b?} = (x!; \checkmark + y!; \checkmark + z!; \bigstar + \delta; \bigstar) \\
\mathit{in} \\
\quad T_1 = a?; (T_{a_1?} \oplus T_{a_2?}) + x!; \bigstar + y!; \bigstar + z!; \bigstar + \delta; \checkmark \\
\quad T_2 = b?; T_{b?} + x!; \bigstar + y!; \bigstar + z!; \bigstar + \delta; \checkmark
\end{array}
$$

It is easy to check that T_1 is passed by s and i, but T_2 is only passed by s.

4.2 Testing Characterisation of iocos

Now we are going to prove that the iocos relation can be characterise in terms of testing. With the pass predicate we have defined a notion of test execution. Upon this notion it is easy to define a testing preorder ($\sqsubseteq_T$) in terms of how many tests are passed: a behaviour will be *better* than other if the former passes more tests than the latter. This section is devoted to prove that this testing preorder is precisely the inverse of the iocos relation: $\mathsf{iocos} = {\sqsubseteq_T}^{-1}$, (Theorem 2).

Definition 9. Let $(S, I, O, \rightarrow) \in LTS$ and $p, q \in S$, we define the preorder

$$p \sqsubseteq_T q \text{ iff } \forall T \in \mathcal{T}_v : \ p \,\mathsf{pass}\, T \implies q \,\mathsf{pass}\, T$$

□

To improve the readability, the proof of Theorem 2 ($\mathsf{iocos} = {\sqsubseteq_T}^{-1}$) has been split into Proposition 1 (${\sqsubseteq_T}^{-1} \subseteq \mathsf{iocos}$) and Proposition 2 ($\mathsf{iocos} \subseteq {\sqsubseteq_T}^{-1}$).

Proposition 1. Let $(S, I, O, \rightarrow)$ and $p, q \in S$, if $q \sqsubseteq_T p$ then $p \,\mathsf{iocos}\, q$.

Proof. In order to prove $p \,\mathsf{iocos}\, q$ we must find an iocos-relation R such that $(p, q) \in R$. Let us define

$$R = \{(p_1, p_2) \mid p_1, p_2 \in S,\ p_2 \sqsubseteq_T p_1\}$$

It is clear that $(p, q) \in R$. We have to prove that R is an iocos-relation. We are going to prove it by contradiction, that is, if there is $(p_1, p_2) \in R$ that does not satisfy one of the conditions of the definition of an ioco-relation (Definition 5), then $p_2 \not\sqsubseteq_T p_1$. So we must find a test $T \in \mathcal{T}_v$ such that $p_2 \,\mathsf{pass}\, T$ but $p_1 \,\mathsf{pa\not{s}s}\, T$. Let us distinguish the cases according to the condition that the pair (p_1, p_2) does not hold:

(p_1, p_2) *does not hold 1 in Definition 5.* So there is $a? \in \mathsf{ins}(p_2)$ such that $a? \notin \mathsf{ins}(p_1)$. Let us consider the test:

$$T = a?;\checkmark + \sum_{x \in O_\delta} x;\checkmark$$

It is clear that $p_1 \,\mathsf{pa\not{s}s}\, T$ but $p_2 \,\mathsf{pass}\, T$.

(p_1, p_2) *does not hold 2 in Definition 5.* We can assume that Definition 5.1 holds. Then, there is $a? \in \mathsf{ins}(p_1)$ and $p_1' \in S$ such that $p_1 \xrightarrow{a?} p_1'$ and $(p_1', p_2') \notin R$ for any p_2' such that $p_2 \xrightarrow{a?} p_2'$. Let us consider the set $P = \{p_2' \mid p_2 \xrightarrow{a?} p_2',\ (p_1', p_2') \notin R\}$. Since Definition5.1 holds, $P \neq \varnothing$. For any $r \in P$ there is a test T_r such that $r \,\mathsf{pass}\, T$ and $p_1' \,\mathsf{pa\not{s}s}\, T_r$. Then let us consider the test

$$T = a?;\bigoplus_{r \in P} T_r + \sum_{x \in O_\delta} x;\checkmark$$

Then $p_2 \,\mathsf{pass}\, T$ but $p_1 \,\mathsf{pa\not{s}s}\, T$.

(p_1, p_2) *does not hold 3 in Definition 5.* There is $x \in \mathsf{outs}(p_1) \cup \{\delta\}$ such that $p_1 \xrightarrow{x} p_1'$ but $(p_1', p_2') \notin R$ for any p_2 such that $p_2 \xrightarrow{x} p_2'$. Let us consider the set $P = \{p_2' \mid p_2 \xrightarrow{x} p_2',\ (p_1', p_2') \notin R\}$. If $P = \varnothing$, let us consider the test

$$T = x; \text{✗} + \sum_{x \neq y, y \in O_\delta} y; \text{✓}$$

Then p_2 pass T and p_1 pas̸s T.
So let us suppose $P \neq \varnothing$. Then for any $r \in P$ there is T_r such that r pass T and p_1' pas̸s T_r. So let us consider the test

$$T = x; \bigoplus_{r \in P} T_r + \sum_{x \neq y, y \in O_\delta} y; \text{✓}$$

Then p_2 pass T and p_1 pas̸s T.

□

Proposition 2. Le $(S, I, O, \rightarrow) \in LTS$ and $p, q \in S$. If p iocos q then $q \sqsubseteq_T p$.

Proof. Let us consider a test $T \in \mathcal{T}_v$ such that q pass T. We have to prove that p pass T. Let us prove by structural induction on T.

$T =$ ✓ or $T =$ ✗. If $T =$ ✗ then q pas̸s T that is a contradiction. The test ✓ is passed for any $p \in S$.

$T = T_1 \oplus T_2$. By definition of q pass T then either q pass T_1 or q pass T_2. Let us assume q pass T_1, the other case is symmetric. By induction p pass T_1, therefore p pass $T_1 \oplus T_2 = T$.

$T = \sum_{x \in O_\delta} x; T_x$. Let us consider the set $O' = \{x \mid x \in O_\delta, \exists p' : p \xrightarrow{x} p'\}$; $O' \neq \varnothing$, by definition of LTS. In order to prove that p pass T we have to prove that p_x pass T_x for any $p_x \in S$ such that $p \xrightarrow{x} p_x$. So let us consider any of these p_x. By Definition 5.3, there is q_x such that $q \xrightarrow{x} q_x$ and p_x iocos q_x. Since q pass T, q_x pass T_x. Then by induction, p_x pass T_x. Since this is true for any $x \in O'$ and $O' \neq \varnothing$, p pass T.

$T = \sum_{x \in O_\delta} x; T_x + a?T_{a?}$. First $a? \in \mathsf{ins}(q)$ since q pass T. By Definition 5.1, $\mathsf{ins}(q) \subseteq \mathsf{ins}(p)$, therefore $a? \in \mathsf{ins}(p)$. Let us consider $p_{a?} \in S$ such that $p \xrightarrow{a?} p_{a?}$. By Definition 5.2, there is $q_{a?} \in S$ such that $q \xrightarrow{a?} q_{a?}$ and $p_{a?}$ iocos $q_{a?}$. Since q pass T, $q_{a?}$ pass $T_{a?}$ and then, by induction, p_a pass $T_{a?}$. Like in the previous case let us consider the set $O' = \{x \mid x \in O_\delta, \exists p' : p \xrightarrow{x} p'\}$. Reasoning like in the previous case we obtain that p_x pass T_x for any $p_x \in S$ and $x \in O'$ such that $p \xrightarrow{x} p_x$.
So, we obtain:

- $p \xrightarrow{a?}$.
- For any $p_{a?} \in S$ such that $p \xrightarrow{a?} p_{a?}$ we obtain $p_{a?}$ pass $T_{a?}$.
- For any $x \in O_\delta$ such that $p \xrightarrow{x} p_x$ we obtain p_x pass T_x.

Therefore p pass T.

□

Theorem 2. (iocos $= \sqsubseteq_T{}^{-1}$) Let $(S, I, O, \rightarrow) \in LTS$ and $i, s \in S$, then i iocos s iff $s \sqsubseteq_T i$. □

4.3 Test Generation

In Section 4.2 we have showed that $i \,\mathsf{iocos}\, s$ iff $\forall T \in \mathcal{T}_v : \; s \,\mathsf{pass}\, T \implies i \,\mathsf{pass}\, T$. This is a classic testing characterisation result that opens the door to testing implementations for iocos-correctness. However, if we wanted to test $s \sqsubseteq_T i$ we would have to try all possible tests, since we do not know which are the tests that s passes.

An essential characteristic for a MBT framework is to be able to automatically produce a test suite from a model specification. For iocos we present this algorithm in Definition 10 which is a variation from the algorithm shown in [20]. The main difference in our algorithm is the inclusion of the $\oplus$ operator in tests. Let us note that if we have a deterministic specification[4]then $\oplus$ operator is applied to a singleton and therefore the resulting test does not really use such operator.

Definition 10. Let $(S, I, O, \rightarrow) \in LTS$ and $p \in S$. We denote with $\mathcal{T}(p)$ the set of valid tests from p by applying a finite number of recursive applications of one of the following non-deterministic choices:

1. $T = \checkmark \in \mathcal{T}(p)$.
2. If $a? \in \mathsf{ins}(p)$, then $T \in \mathcal{T}(p)$ where

$$T = a?; \bigoplus \{T_{p_{a?}} \mid p \xrightarrow{a?} p_{a?}\} + \sum_{\substack{x \in \mathsf{outs}(p) \\ x \neq \delta}} x; \bigoplus \{T_{p_x} \mid p \xrightarrow{x} p_x\} + \sum_{\substack{x \in O, x \neq \delta \\ x \notin \mathsf{outs}(p)}} x; \text{✗} \; + \; \delta; T_\delta(p)$$

3. If $\mathsf{ins}(p) = \varnothing$ then $T \in \mathcal{T}(p)$ where

$$T = \sum_{\substack{x \in \mathsf{outs}(p) \\ x \neq \delta}} x; \bigoplus \{T_{p_x} \mid p \xrightarrow{x} p_x\} + \sum_{\substack{x \in O, x \neq \delta \\ x \notin \mathsf{outs}(p)}} x; \text{✗} + \delta; T_\delta(p)$$

In all cases the tests T_p are chosen non-deterministically from the set $\mathcal{T}(p)$, $T_\delta(p) = \checkmark$ if $p \xrightarrow{\delta}$, and $T_\delta(p) = \text{✗}$ otherwise. □

The essential goal of the algorithm is to produce a test suite as tight as possible to the given specification. The rest of this section is devoted to prove the completeness of the algorithm for the iocos relation. There are two basic properties for the test suite: soundness (Proposition 3) and exhaustiveness (Proposition 4). The most basic property is soundness, meaning all tests from the set $\mathcal{T}(p)$ to be correct with respect p: p passes all tests in the set $\mathcal{T}(p)$.

Proposition 3. (***Soundness***)
Let $(S, I, O, \rightarrow) \in LTS$ and $p \in S$. Then $p \,\mathsf{pass}\, T$ for any $T \in \mathcal{T}(p)$.

[4] A behaviour is deterministic when for any $x \in L_\delta$, if $p \xrightarrow{x} p_1$ and $p \xrightarrow{x} p_2$ then $p_1 = p_2$.

Proof. by structural induction on the set of terms $\mathcal{T}(p)$. In base case, it trivially holds, since p pass $\checkmark$ for $T = \checkmark$. For recursive cases we have:

$T = a?; \bigoplus T_{p_{a?}} + \sum_{\substack{x \in \mathsf{outs}(p) \\ x \neq \delta}} x; \bigoplus T_{p_x} + \sum_{\substack{x \in O, x \neq \delta \\ x \notin \mathsf{outs}(p)}} x; \boldsymbol{\times} + \delta; T_\delta(p)$**.** First let us note that p pass $\delta; T_\delta(p)$ trivially ($T_\delta(p) = \checkmark$ if $p \xrightarrow{\delta}$). By definition of the generator algorithm (Definition 10), $a? \in \mathsf{ins}(p)$ and $T_{p_{a?}} \in \mathcal{T}(p_{a?})$ for any $p_{a?}$ such that $p \xrightarrow{a?} p_{a?}$. By induction hipothesis $p_{a?}$ pass $T_{p_{a?}}$, therefore p pass $a?; \bigoplus T_{p_{a?}}$.
Similarly, if $x \in \mathsf{outs}(P)$, $T_{p_x} \in \mathcal{T}(p_x)$ for any p_x such that $p \xrightarrow{x} p_x$. By induction hipothesis p_x pass T_{p_x}, therefore p pass $\sum_{x \in \mathsf{outs}(p)} x; \bigoplus T_{p_x}$. If $x \notin \mathsf{outs}(p)$ then $p \xrightarrow{x}$, so p pass $\sum_{\substack{x \in O_\delta \\ x \notin \mathsf{outs}(p)}} x; \boldsymbol{\times}$
Finally, since p passes all addends from test T, P pass T.

$T = \sum_{\substack{x \in \mathsf{outs}(p) \\ x \neq \delta}} x; \bigoplus T_{p_x} + \sum_{\substack{x \in O, x \neq \delta \\ x \notin \mathsf{outs}(p)}} x; \boldsymbol{\times} + \delta; T_\delta(p)$**.** This case is similar to the previous one. We only have to take into account that $\mathsf{outs}(p) \neq \varnothing$. □

Proposition 4. (***Exhaustiveness***) Let $(S, I, O, \rightarrow) \in LTS$ and $p, q \in S$. If $\forall T \in \mathcal{T}(p) : q$ pass T then $p \sqsubseteq_T q$.

Proof. Let us prove the theorem by contradiction. Let us suppose $p \not\sqsubseteq_T q$, then there is a test $T \in \mathcal{T}_v$ such that p pass T and q pa$\not$ss T. Let us prove that if there is a test T such that p pass T and q pa$\not$ss T there is a test $T_p \in \mathcal{T}(p)$ such that q pa$\not$ss T_p by induction on T. The base case is when $T = \checkmark$ or $T = \boldsymbol{\times}$; but in these cases there is nothing to prove since q pass $\checkmark$ and p pa$\not$ss $\boldsymbol{\times}$.

So let us consider the recursive cases.

$T = T_1 \oplus T_2$**.** Then p pass T_1 or p pass T_2. In both cases we obtain the result by induction.

$T = \sum_{x \in O_\delta} x; T_x$**.** For any $x \in O_\delta$ let us consider a test T'_x as follows:
$x = \delta$**.** If $q \xrightarrow{\delta}$ then $T_x = \checkmark$, otherwise $T_x = \boldsymbol{\times}$.
$x \in \mathsf{outs}(q) \cap \mathsf{outs}(p)$**.** If q_x pass T_x for any q_x such that $q \xrightarrow{x} q_x$, let us consider $T_x = \checkmark$. Otherwise there is q_x such that $q \xrightarrow{x} q_x$ and q_x pa$\not$ss T_x. However, p_x pass T_x for any p_x such that $p \xrightarrow{x} p_x$. So by induction, for any of those p_x there is a test $T_{p_x} \in \mathcal{T}(p_x)$ such that q_x pa$\not$ss T_{p_x}. Then let us consider $T'_x = \bigoplus \{T_{p_x} \mid p \xrightarrow{x} p_x\}$.
$x \in \mathsf{outs}(q), x \notin \mathsf{outs}(p)$**.** In this case $T'_x = \boldsymbol{\times}$.
$x \notin \mathsf{outs}(q), x \in \mathsf{outs}(p)$**.** In this case $T'_x = \checkmark$.
So the test $T_p = \sum_{x \in O_\delta} x; T'_x$ satisfies that $T_p \in \mathcal{T}(p)$ and q pa$\not$ss T_p.

$T = a?; T_{a?} + \sum_{x \in O_\delta} x; T_x$**.** For $x \in O_\delta$ let us consider the test T'_x as in the previous case. Let us note that, since p pass T, $a? \in \mathsf{ins}(p)$. Let us consider the test $T'_{a?}$ as follows:
$a? \in \mathsf{ins}(q)$**.** $T'_{a?}$ is build in a similar way as in the previous case when $x \in \mathsf{outs}(q) \cap \mathsf{outs}(p)$.
$a? \notin \mathsf{ins}(q)$**.** $T'_{a?} = \checkmark$.
So the test $T_p = a?; T'_{a?} + \sum_{x \in O_\delta} x; T'_x$ satisfies that $T_p \in \mathcal{T}(p)$ and q pa$\not$ss T_p. □

Theorem 3. (***Completeness***) Let $(S, I, O, \rightarrow) \in LTS$ and $p, q \in S$, $\forall T \in \mathcal{T}(p) : q$ pass T iff p iocos q. □

5 Conclusions and Future Work

MBT aims to offer a real and practical solution to effectively check the correctness of a system implementation against the model provided by the specification. At the core of any concrete MBT theory, framework or tool, it lies a conformance relation to decide if a given implementation is correct for the proposed specification.

For every concrete case study and industrial application, to select a suitable conformance relation is a decision that may depend on many ingredients: costs of implementation, security considerations, performance, context of application... We think it would be desirable to have a theory with the capacity to express conformance at different levels.

The research we present in this paper is a humble step in that direction. Instead of the classic approach based on linear semantics, we have used a conformance relation based on simulation semantics. The reason for this decision is that some recent research on process theory has shown [8,7] that the family of simulation semantics forms a backbone on the spectrum of semantics from which a hierarchy of layers of linear semantics can be derived in a systematic way. To further follow the applicability of this theoretical results, to the particular case of MBT with input-output transition systems, is one of lines of research we are currently working on.

Along the paper we have settled the basics results for a MBT theory based on a conformance simulation relation: The definition of iocos as a conformance relation has been motivated through examples; we have showed iocos to be an strict refinement of classic ioco relation for input-output labelled transition systems; a testing characterisation of iocos has been provided and also a test suite generation algorithm from specifications.

However, there are still well known issues in MBT that we need to address in our proposal. Regarding applicability we are specially interested in test selection and on-the-fly, or on-line, testing [22] which does not need to generate a priori test suites, but instead try to check dynamically the implementation under test.

As for test selection we are interested in the use metrics [6], but we think we can benefit from the new insights of recent works in the area (see for instance [17]). Moreover, the coinductive definition of iocos and the well known characterisations of simulations as games make our approach very suitable to further research the use of on-line testing with iocos.

References

1. Abramsky, S.: Observational equivalence as a testing equivalence. Theoretical Computer Science 53(3), 225–241 (1987)
2. de Alfaro, L.: Game models for open systems. In: Dershowitz, N. (ed.) Verification: Theory and Practice. LNCS, vol. 2772, pp. 269–289. Springer, Heidelberg (2004)
3. Alur, R., Henzinger, T.A., Kupferman, O., Vardi, M.Y.: Alternating refinement relations. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 163–178. Springer, Heidelberg (1998)

4. Belinfante, A.: Jtorx: A tool for on-line model-driven test derivation and execution. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 266–270. Springer, Heidelberg (2010)
5. Chow, T.S.: Testing software design modeled by finite-state machines. IEEE Trans. Software Eng. 4(3), 178–187 (1978)
6. Feijs, L.M.G., Goga, N., Mauw, S., Tretmans, J.: Test selection, trace distance and heuristics. In: Schieferdecker, I., König, H., Wolisz, A. (eds.) TestCom. IFIP Conference Proceedings, vol. 210, pp. 267–282. Kluwer (2002)
7. de Frutos-Escrig, D., Gregorio-Rodríguez, C., Palomino, M.: On the unification of process semantics: Equational semantics. Electronic Notes in Theoretical Computer Science 249, 243–267 (2009)
8. de Frutos-Escrig, D., Gregorio-Rodríguez, C.: (Bi)simulations up-to characterise process semantics. Information and Computation 207(2), 146–170 (2009)
9. van Glabbeek, R.J.: The Linear Time – Branching Time Spectrum I. In: Handbook of Process Algebra, pp. 3–99. Elsevier (2001)
10. van Glabbeek, R.J., Ploeger, B.: Correcting a space-efficient simulation algorithm. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 517–529. Springer, Heidelberg (2008)
11. Hennessy, M.: Algebraic Theory of Processes. MIT Press (1988)
12. Hierons, R.M., Bowen, J.P., Harman, M. (eds.): Formal Methods and Testing, An Outcome of the FORTEST Network, Revised Selected Papers. LNCS, vol. 4949. Springer, Heidelberg (2008)
13. Milner, R.: An algebraic definition of simulation between programs. In: Proceedings 2nd Joint Conference on Artificial Intelligence, pp. 481–489. BCS (1971)
14. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
15. Nicola, R.D., Hennessy, M.: Testing equivalences for processes. Theoretical Computer Science 34(1-2), 83–133 (1984), http://www.sciencedirect.com/science/article/pii/0304397584901130
16. Ranzato, F., Tapparo, F.: A new efficient simulation equivalence algorithm. In: LICS, pp. 171–180. IEEE Computer Society (2007)
17. Romero Hernández, D., de Frutos Escrig, D.: Defining distances for all process semantics. In: Giese, H., Rosu, G. (eds.) FORTE 2012 and FMOODS 2012. LNCS, vol. 7273, pp. 169–185. Springer, Heidelberg (2012)
18. Stokkink, G., Timmer, M., Stoelinga, M.: Talking quiescence: a rigorous theory that supports parallel composition, action hiding and determinisation. In: Petrenko, A.K., Schlingloff, H. (eds.) MBT. EPTCS, vol. 80, pp. 73–87 (2012)
19. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. Software - Concepts and Tools 17(3), 103–120 (1996)
20. Tretmans, J.: Model based testing with labelled transition systems. In: Hierons, et al. (eds.) [12], pp. 1–38
21. Tretmans, J., Brinksma, E.: Torx: Automated model-based testing. In: Hartman, A., Dussa-Ziegler, K. (eds.) First European Conference on Model-Driven Software Engineering, pp. 31–43 (December 2003), http://doc.utwente.nl/66990/
22. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approaches. Softw. Test., Verif. Reliab. 22(5), 297–312 (2012)
23. Veanes, M., Bjørner, N.: Alternating simulation and ioco. In: Petrenko, A., Simão, A., Maldonado, J.C. (eds.) ICTSS 2010. LNCS, vol. 6435, pp. 47–62. Springer, Heidelberg (2010)
24. Veanes, M., Campbell, C., Grieskamp, W., Schulte, W., Tillmann, N., Nachmanson, L.: Model-based testing of object-oriented reactive systems with spec explorer. In: Hierons et al, [12], pp. 39–76