

Unifying Input Output Conformance

Bernhard K. Aichernig¹ and Martin Weiglhofer^{1,2*}

¹ Institute for Software Technology, Graz University of Technology, Austria

² Competence Network Softnet Austria

Abstract. Model-based conformance testing aims to assess the correctness of an implementation with respect to a specification. This raises the question of a proper conformance relation that should be established between implementation and specification. One commonly used conformance relation is the so-called input output conformance (**ioco**), which is defined over labeled transition systems. In this paper we investigate a denotational semantics of the input output conformance relation over reactive processes. We formalize the underlying assumptions of the **ioco** relation in terms of formal healthiness conditions and by adopted operators. Finally, we show that our denotational version of **ioco** can be generalized in the same way as the original relation. Our work aims to provide a unification of input output conformance by lifting the definition from labeled transition systems to reactive processes.

Key words: input output conformance, ioco, unifying theories of programming, UTP, processes, quiescence, fairness, model based testing.

1 Introduction

Software development is a complex and error-prone task. Failures in safety-critical applications may be life-threatening. At least software failures cause incredible costs during and after the software development process. Therefore, software engineers need to be supported by tools, techniques and theories in order to reduce the number of software failures.

Model-based black-box testing techniques aim to assess the correctness of an implementation under test (IUT) with respect to a given specification. The implementation under test is viewed as a black-box with an interface that allows to provide inputs to the IUT and to observe outputs from the implementation under test. The goal of testing is to check whether an IUT conforms to a specification with respect to a particular conformance relation. One of the most popular conformance relations in the area of model-based testing is the input output conformance (**ioco**) relation of Tretmans [1].

Mature research prototypes (e.g. TORX [2], TGV [3]) and successful industrial case studies (e.g. [4, 5]) have shown the usability of this conformance relation in practice. However, the used theory is given by an operational semantics and some of the underlying assumptions have been stated on an informal basis only. Thus, the contributions of this paper can be summarized as follows:

* Authors are listed in alphabetical order.

- We provide a denotational semantics of the input output conformance relation over UTP's [6] reactive processes.
- The underlying assumptions of the **ioco** relation are formalized in terms of healthiness conditions and by adopted operators over reactive processes.
- We provide a unification of the input output conformance relation by lifting the definition from labeled transition systems to reactive processes.

This paper continues as follows: In Section 2, we review the input output conformance relation. Section 3 comprises the formalization of **ioco** in the UTP-framework. Note that we expect some familiarity with UTP. Finally, we discuss our results and further research in Section 4.

2 Input Output Conformance of Labeled Transition Systems

In this section, we briefly discuss the input output conformance relation [1]. This relation expresses the conformance of implementations to their specifications where both are represented by some sort of labeled transition system (LTS).

Definition 1 (Labeled transition system). *A labeled transition system is a tuple $M = (Q^M, A^M \cup \{\tau\}, \rightarrow_M, q_0^M)$, where Q^M is a finite set of states, A^M a finite alphabet and $\tau \notin A^M$ an unobservable action, $\rightarrow_M \subseteq Q^M \times A^M \times Q^M$ is the transition relation, and $q_0^M \in Q^M$ is the initial state.*

An LTS that distinguishes between inputs and outputs is called input output labeled transition system (IOLTS):

Definition 2 (Input output labeled transition system). *An input output labeled transition system (IOLTS) is an LTS $M = (Q^M, A^M \cup \{\tau\}, \rightarrow_M, q_0^M)$ where A^M is partitioned into two disjoint sets $A^M = A_I^M \cup A_O^M$, where A_I^M and A_O^M are input and output alphabets, respectively.*

The class of labeled transition systems with inputs A_I^M and outputs in A_O^M is denoted by $\mathcal{LTS}(A_I^M, A_O^M)$ [1]. We use the following common notations for LTSs and for IOLTSs.

Definition 3. *Given a labeled transition system $M = (Q^M, A^M \cup \{\tau\}, \rightarrow_M, q_0^M)$ and let $q, q', q_i \in Q^M, a_{(i)} \in A_I^M \cup A_O^M$ and $\sigma \in (A_I^M \cup A_O^M)^*$.*

$$\begin{aligned}
q \xrightarrow{a}_M q' &=_{df} (q, a, q') \in \rightarrow_M \\
q \xrightarrow{a}_M &=_{df} \exists q' \bullet (q, a, q') \in \rightarrow_M \\
q \not\xrightarrow{a}_M &=_{df} \nexists q' \bullet (q, a, q') \in \rightarrow_M \\
q \xrightarrow{\tau}_M q' &=_{df} (q = q') \vee \exists q_0, \dots, q_n \bullet (q = q_0 \xrightarrow{\tau}_M q_1 \wedge \dots \wedge q_{n-1} \xrightarrow{\tau}_M q_n = q') \\
q \xrightarrow{a}_M q' &=_{df} \exists q_1, q_2 \bullet q \xrightarrow{\tau}_M q_1 \xrightarrow{a}_M q_2 \xrightarrow{\tau}_M q' \\
q \xrightarrow{a_1 \dots a_n}_M q' &=_{df} \exists q_0, \dots, q_n \bullet q = q_0 \xrightarrow{a_1}_M q_1 \dots q_{n-1} \xrightarrow{a_n}_M q_n = q' \\
q \xrightarrow{\sigma}_M &=_{df} \exists q' \bullet q \xrightarrow{\sigma}_M q'
\end{aligned}$$

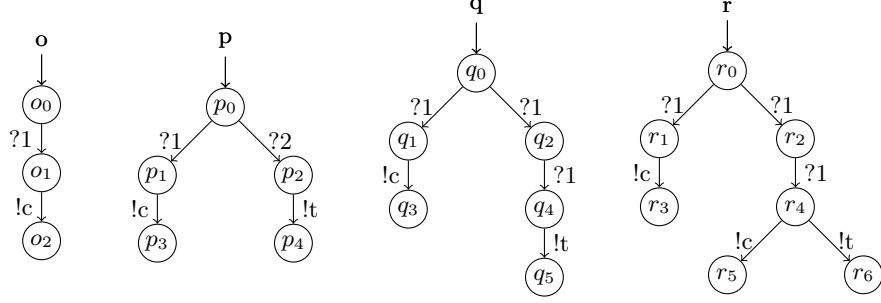


Fig. 1. Examples of input output labeled transition systems.

We use $init(q)$ to denote the actions enabled in state q and $traces(q)$ to denote the traces enabled in state q . Furthermore, we denote the states reachable by a particular trace σ by q **after** σ . More precisely,

Definition 4. Given a labeled transition system $M = (Q^M, A^M \cup \{\tau\}, \rightarrow_M, q_0^M)$ and let $q \in Q^M, Q \subseteq Q^M$ and $\sigma \in (A_I^M \cup A_O^M)^*$.

$$\begin{aligned} init(q) &=_{df} \{a \in A^M \cup \{\tau\} \mid q \xrightarrow{a}_M\} \\ traces(q) &=_{df} \{\sigma \in A^{M*} \mid q \xrightarrow{\sigma}\} \\ q \text{ after}_M \sigma &=_{df} \{q' \mid q \xrightarrow{\sigma}_M q'\} \\ Q \text{ after}_M \sigma &=_{df} \bigcup_{q \in Q} (q \text{ after}_M \sigma) \end{aligned}$$

Note that we will not always distinguish between an IOLTS and its initial state and write $M \Rightarrow_M$ instead of $q_0^M \Rightarrow_M$. We will omit the subscript M (and superscript M) when it is clear from the context.

Example 1. Figure 1 shows four labeled transition systems o , p , q , and r . The input alphabet is given by $A_I = \{1, 2\}$ and the output alphabet is $A_O = \{c, t\}$. We denote input actions by the prefix "?", while output actions have the prefix "!". For example, p_0 **after** $?1 = \{p_1\}$ while q_0 **after** $?1 = \{q_1, q_2\}$.

The **ioco** conformance relation employs the idea of observable quiescence. That is, it is assumed that a special action is enabled in the case where the labeled transition system does not provide any output action. Therefore the input output conformance relation identifies quiescent states: A state q of a labeled transition system is quiescent if neither an output action nor an internal action (τ) is enabled in q .

Definition 5. Let M be a labeled transition system $M = (Q^M, A_O^M \cup A_I^M \cup \{\tau\}, \rightarrow_M, q_0^M)$, then a state $q \in Q^M$ is quiescent, denoted by $\theta(q)$, if $\forall a \in A_O^M \cup \{\tau\} \bullet q \not\xrightarrow{a}_M$.

Usually, δ is used as special action denoting quiescence. Because of a name clash with UTP's deadlock symbol δ [6] we use θ for representing quiescence. By

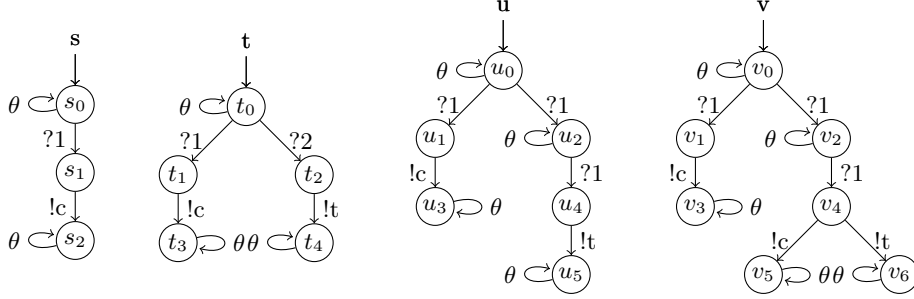


Fig. 2. Examples of suspension automata.

adding θ -labeled transitions to LTSs the quiescence symbol can be used as any other action. By the use of suspension automata θ becomes observable.

Definition 6 (Suspension automata). Let M be a labeled transition system $M = (Q^M, A_I^M \cup A_O^M \cup \{\tau\}, \rightarrow_M, q_0^M)$ then the suspension automaton M_θ is given by $(Q^M, A_I^M \cup A_O^M \cup \{\theta\}, \rightarrow_M \cup \rightarrow_\theta, q_0^M)$ where $\rightarrow_\theta =_{df} \{q \xrightarrow{\theta} q \mid q \in Q^M \wedge \theta(q)\}$. The suspension traces of M are $Straces(M) =_{df} \{\sigma \in (A^M \cup \{\theta\})^* \mid q_0^M \xRightarrow{\sigma}\}$.

Example 2. Fig. 2 shows the suspension automata for the LTSs illustrated in Fig. 1. The transition systems depicted in Fig. 2 are equipped with θ labels for each quiescent state. For example, the states u_0 , u_2 , u_3 , and u_5 are quiescent states since they do not have any outgoing edge labeled with an output action nor any outgoing edge labeled with a τ action.

A major hypothesis of the input output conformance relation is that the implementation can be represented as a labeled transition system. It is not assumed that this LTS is known in advance, but only its existence is required. This is known as a testing hypothesis [7, 8].

The models used for representing implementations are input output transition systems. Since implementations are not allowed to refuse inputs, their models obey to the same restriction. This means that implementations are assumed to be input-enabled and so are their models.

Definition 7 (Input output transition system). An input output transition system (IOTS) is an IOLTS $M = (Q^M, A_I^M \cup A_O^M \cup \{\tau\}, \rightarrow_M, q_0^M)$ where all input actions are enabled (possibly preceded by τ -transitions) in all states:

$$\forall a \in A_I^M, \forall q \in Q^M \bullet q \xRightarrow{a}$$

The class of input output transition systems with inputs A_I^M and outputs in A_O^M is given by $\mathcal{IOTS}(A_I^M, A_O^M) \subseteq \mathcal{LTS}(A_I^M, A_O^M)$ [1].

Example 3. The IOTSs for the suspension automata of Fig. 2 are depicted in Fig. 3. Note that the τ transitions in state z_4 are not because of input-enabledness but due to the restrictions on choices (see Section 3.3).

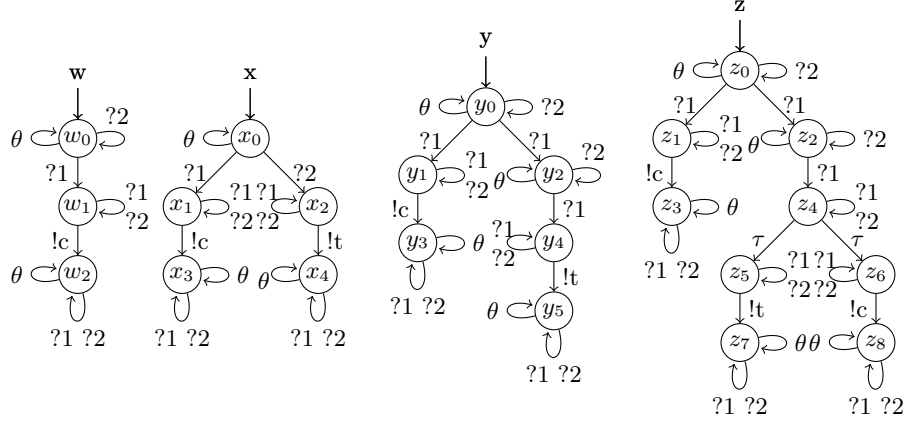


Fig. 3. Examples of input output transition systems (input-enabled by definition).

Before giving the definition of the **io** relation we need to define what are the outputs of a particular state and what are the outputs of a set of states.

Definition 8. Given a labeled transition system $M = (Q^M, A_I^M \cup A_O^M \cup \{\tau\}, \rightarrow_M, q_0^M)$ and let $q \in Q^M$ and $Q \subseteq Q^M$, then

$$\begin{aligned} out_M(q) &=_{df} \{a \in A_O^M \mid q \xrightarrow{a}_M\} \cup \{\theta \mid \theta(q)\} \\ out_M(Q) &=_{df} \bigcup_{q \in Q} (out_M(q)) \end{aligned}$$

Now we are ready to give the definition of the **io** relation. Informally, the input output conformance relation states, that an implementation under test (IUT) conforms to a specification S, iff the outputs of the IUT are outputs of S after an arbitrary suspension trace of S. More formally,

Definition 9 (Input output conformance). Given a set of inputs A_I and a set of outputs A_O then $\mathbf{io} \subseteq \mathcal{IOTS}(A_I, A_O) \times \mathcal{LTS}(A_I, A_O)$ is defined as:

$$IUT \mathbf{io} S =_{df} \forall \sigma \in \text{Straces}(s) \bullet out(IUT \text{ after } \sigma) \subseteq out(S \text{ after } \sigma)$$

Example 4. Consider the LTSs of Figure 2 to be specifications and let the IOTSs of Figure 3 be implementations. Then we have $w \mathbf{io} s$ and $x \mathbf{io} t$. We also have $x \mathbf{io} s$ because ?2 is not a trace of s . Thus, this branch is not relevant with respect to **io**. y does not conform to s , i.e. $\neg(y \mathbf{io} s)$, because $out(y_0 \text{ after}_y ?1) = \{!c, \theta\} \not\subseteq \{!c\} = out(s_0 \text{ after}_s ?1)$. Furthermore $\neg(z \mathbf{io} s)$ because $out(z_0 \text{ after}_z ?1) = \{!c, \theta\} \not\subseteq \{!c\} = out(s_0 \text{ after}_s ?1)$. Due to the use of suspension traces we also have $\neg(z \mathbf{io} u)$ because $out(z_0 \text{ after}_z ?1 \hat{\theta} ?1) = \{!c, !t\} \not\subseteq \{!t\} = out(u_0 \text{ after}_u ?1 \hat{\theta} ?1)$

The **io** definition from above can be lifted to a more general definition where different instantiations correspond to different conformance relations:

Definition 10 (Generic input output conformance). Given a set of inputs A_I and a set of outputs A_O then $\mathbf{ioco}_{\mathcal{F}} \subseteq \mathcal{IOTS}(A_I, A_O) \times \mathcal{LTS}(A_I, A_O)$ is defined as:

$$IUT \mathbf{ioco}_{\mathcal{F}} S =_{df} \forall \sigma \in \mathcal{F} \bullet out(IUT \text{ after } \sigma) \subseteq out(S \text{ after } \sigma)$$

Using $\mathbf{ioco}_{\mathcal{F}}$ we can now express different relations by selecting a proper set of sequences for \mathcal{F} . The input output testing relation (\leq_{iot}) is given by \mathbf{ioco}_{A^*} while the input output refusal relation (\leq_{ior}) can be defined as $\mathbf{ioco}_{(A \cup \{\theta\})^*}$. \mathbf{ioconf} is given by $\mathbf{ioco}_{traces(S)}$.

Example 5. The input output transition systems y and z of Figure 3 serve to illustrate the differences between these conformance relations, e.g. $z \leq_{iot} y$, $\neg(z \leq_{ior} y)$, $z \mathbf{ioconf} y$, $\neg(z \mathbf{ioco} y)$, $x \mathbf{ioconf} w$, $x \mathbf{ioco} w$, $\neg(x \leq_{iot} w)$ and $\neg(x \leq_{ior} w)$.

By the use of a particular set of test cases one wants to test if a given implementation conforms to its specification. In the \mathbf{ioco} framework a test case is again a labeled transition system [1]:

Definition 11 (Test case). A test case t is a labeled transition system $t = (Q^t, A_I^t \cup A_O^t \cup \{\theta\}, \rightarrow_t, q_0^t)$ such that

- t is deterministic and has finite behavior
- Q^t contains terminal states pass and fail
- for any state $q \in Q^t$ where $q \neq \text{pass}$ and $q \neq \text{fail}$, either $init(q) = \{a\}$ for some $a \in A_I^t$, or $init(q) = A_O^t \cup \{\theta\}$

Testing is now conducted by running a test case t in parallel with the implementation i . A test run is a trace of the synchronous parallel composition $t || i$ leading to a terminal state of t .

3 Input Output Conformance of Processes

This section presents our denotational semantics of the \mathbf{ioco} conformance relation. We formulate \mathbf{ioco} over UTP's reactive processes.

3.1 Reactive Processes

Basically, the process of testing is modelled as an interaction between two reactive processes, the implementation under test and the test case. A reactive process with respect to the unified theories of programming is defined as follows:

Definition 12 (Reactive process). A reactive process P is one which satisfies the healthiness conditions **R1**, **R2**, **R3** where

$$\begin{aligned} \mathbf{R1}(X) &=_{df} X \wedge (tr \leq tr') \\ \mathbf{R2}(X(tr, tr')) &=_{df} \bigsqcap_s X(s, \hat{s}(tr' - tr)) \\ \mathbf{R3}(X) &=_{df} \mathbb{I} \triangleleft wait \triangleright X \end{aligned}$$

The alphabet of P consists of the following:

- \mathcal{A} , the set of events in which it can potentially engage.
- $tr : \mathcal{A}^*$, the sequence of events which have happened up to the time of observation.
- $wait : Bool$, which distinguishes its waiting states from its terminated states.
- $ref : \mathcal{PA}$, the set of events refused by the process during its wait.

The skip predicate (\mathbb{I}) is defined as in [6]:

$$\mathbb{I} =_{df} \neg ok \wedge (tr \leq tr') \vee ok' \wedge (tr' = tr) \wedge \dots \wedge (wait' = wait)$$

The input output conformance relation distinguishes between inputs and outputs. Outputs are actions that are initiated by and under control of an implementation under test, while input actions are initiated by and under control of the system's environment³. Hence, the alphabet \mathcal{A} of a process consists of two disjoint sets $\mathcal{A} = \mathcal{A}_{in} \cup \mathcal{A}_{out}$. In addition, we will also differentiate between refused inputs $ref_{in} =_{df} ref \cap \mathcal{A}_{in}$ and refused outputs $ref_{out} =_{df} ref \cap \mathcal{A}_{out}$. Thus, also refusals form a partition: $ref_{in} \cap ref_{out} = \emptyset$ and $ref = ref_{in} \cup ref_{out}$. Note that we use ? and ! to indicate inputs and outputs for processes. For example, a process having as input alphabet $\mathcal{A}_{in} = \{1\}$ and as output alphabet $\mathcal{A}_{out} = \{c\}$ is written as $do_{\mathcal{A}}(?1); do_{\mathcal{A}}(!c)$.

A process offering a single event $a \in \mathcal{A}$ for communication is expressed in terms of $do_{\mathcal{A}}(a)$, where

$$\begin{aligned} do_{\mathcal{A}}(a) &=_{df} \Phi(a \notin ref' \triangleleft wait' \triangleright tr' = tr \hat{\ } (a)) \\ \Phi &=_{df} \mathbf{R} \circ and_B = and_B \circ \mathbf{R} \\ B &=_{df} ((tr' = tr) \wedge wait' \vee (tr < tr')) \\ \mathbf{R} &=_{df} \mathbf{R1} \circ \mathbf{R2} \circ \mathbf{R3} \end{aligned}$$

For sequential composition we rely on UTP's standard sequential composition operator: $P(v, v'); Q(v, v') =_{df} \exists v_0 \bullet (P(v, v_0) \wedge Q(v_0, v'))$

3.2 IOCO Specifications

For technical reasons, that is the computability of particular sets during the test case generation, the reactive processes used in the **ioco** framework need to satisfy an additional healthiness condition. The processes need to be strongly responsive⁴, i.e. the processes do not have infinite sequences of internal actions (livelocks). This correspond to CSP's notion of divergence [9]. Hence, the healthiness condition for specifications excludes livelocks:

$$\mathbf{IOCO1} \quad P = P \wedge (ok' \vee wait')$$

Within Tretmans theory [1] quiescence denotes the absence of outputs and the absence of internal actions. Quiescence is encoded by the presence of a particular action θ . Since **ioco** uses traces containing quiescence we need to include

³ 2nd column on page 104 of [1]

⁴ 2nd column on page 105 of [1]

θ in the traces of our processes. Thus, we extend set of events \mathcal{A} by θ for reactive processes. In the sequel we use following abbreviation $\mathcal{A}_\theta =_{df} \mathcal{A} \cup \{\theta\}$. A UTP process is quiescent after a particular trace iff either it has finished its execution or it refuses to do any output action

$$quiescence =_{df} \neg wait' \vee \forall o \in \mathcal{A}_{out} \bullet o \in ref'$$

Because we exclude livelocks (see healthiness condition **IOCO1**), it is sufficient to use $wait'$ here. Quiescent communication is expressed in terms of $do_{\mathcal{A}}^\theta$, which adds quiescence (θ) to the traces and to the refusal set of a process:

Definition 13 (Quiescent communication). *Let $a \in \mathcal{A}$ be an action of a process' alphabet, then*

$$do_{\mathcal{A}}^\theta(a) =_{df} \begin{cases} \Phi^i(do_{\mathcal{A}}(a)) & \text{if } a \in \mathcal{A}_{out} \\ \Phi^i(\{\theta, a\} \not\subseteq ref' \wedge tr' - tr \in \theta^* \triangleleft wait' \triangleright) & \text{if } a \in \mathcal{A}_{in} \\ tr' - tr \in \theta^* \widehat{\langle a \rangle} & \end{cases}$$

where $\Phi^i =_{df} \mathbf{IOCO1} \circ \mathbf{R} \circ$ and B

Consider the case where a is an input action, i.e., $a \in \mathcal{A}_{in}$: In the case of $wait'$ the process $do_{\mathcal{A}}^\theta(a)$ allows an arbitrary number of θ events (see the θ -loops in Fig. 2). After termination the event a has happened preceded by an arbitrary - possible empty sequence - of quiescence events, i.e. $tr' - tr \in \theta^* \widehat{\langle a \rangle}$. For sake of simplicity we sometimes write θ^* instead of $\{\theta\}^*$. Note that $\theta^* \widehat{\langle a \rangle}$ denotes the set of events where every element of θ^* is concatenated with $\langle a \rangle$.

The following two healthiness conditions describe the possible occurrence of θ events:

$$\begin{aligned} \mathbf{IOCO2} \quad & P = P \wedge (wait' \implies (\theta \notin ref' \iff quiescence)) \\ \mathbf{IOCO3} \quad & P = P \wedge (wait' \implies (\theta \notin ref' \implies \exists s \bullet tr' - tr \in s \widehat{\theta^*})) \end{aligned}$$

Since the quiescence event θ encodes the absence of output events it may occur at any time. This is even true for the deadlock process.

Definition 14 (Quiescent deadlock).

$$\delta^\theta =_{df} \mathbf{R3}(tr' - tr \in \theta^* \wedge wait')$$

Consequently, the classical deadlock process indicating absolute inactivity does not exist within the **ioco** theory.

Although quiescence is preserved by sequential composition, we need to redefine internal and external choices in order to preserve the properties of quiescence. Basically, the composition of processes that start with input actions (i.e. the processes are quiescent initially) is quiescent. If one of the two composed processes is not quiescent initially, the composition is not quiescent as well.

For our quiescence preserving composition operators ($\Pi^\theta, +^\theta$) we use an approach similar to parallel merge of [6]. The idea of parallel by merge is to run two processes independently and merge their results afterwards. In order to express independent execution we need a relabeling function. Given an output alphabet $\{v'_1, v'_2, \dots, v'_n\}$, U_l is defined as follows

Definition 15 (Relabelling).

$$\begin{aligned}\alpha U_l(\{v'_1, v'_2, \dots, v'_n\}) &=_{df} \{v_1, v_2, \dots, v_n, l.v'_1, l.v'_2, \dots, l.v'_n\} \\ U_l(\{v'_1, v'_2, \dots, v'_n\}) &=_{df} (l.v'_1 = v_1) \wedge (l.v'_2 = v_2) \wedge \dots \wedge (l.v'_n = v_n)\end{aligned}$$

Independent execution of P and Q is now expressed by relabeling:

Definition 16 (Independent execution).

$$P \downarrow Q =_{df} P; U_0(out\alpha P) \wedge Q; U_1(out\alpha Q)$$

An internal choice, which takes care of θ within the resulting process, can be defined as follows:

Definition 17 (Quiescence preserving internal choice).

$$\begin{aligned}P \sqcap^\theta Q &=_{df} (P \downarrow Q); M_\sqcap \\ M_\sqcap &=_{df} M_\sqcap^{\delta^\theta} \triangleleft \delta^\theta \triangleright M^{-\delta^\theta}\end{aligned}$$

As this definition illustrates we need two merge relations for the quiescence preserving internal choice: $M_\sqcap^{\delta^\theta}$ and $M^{-\delta^\theta}$. $M_\sqcap^{\delta^\theta}$ merges the very beginning of the two processes P and Q . After that, $M^{-\delta^\theta}$ takes care that $P \sqcap^\theta Q$ behaves like P or Q .

$M_\sqcap^{\delta^\theta}$ and $M^{-\delta^\theta}$ share some common properties, formalized by M^θ : (1) Parts of P and Q are only merged if their *wait'* values are equal, and (2) potentially the initial θ has to be removed from all traces.

Definition 18 (Internal choice - Common merge).

$$\begin{aligned}M^\theta &=_{df} (0.wait \iff 1.wait) \wedge wait' = 0.wait \wedge \\ &\quad ((notInitialQuiet(0.tr - tr) \wedge notInitialQuiet(1.tr - tr)) \\ &\quad \implies notInitialQuiet(tr' - tr))\end{aligned}$$

$$where\ notInitialQuiet(t) =_{df} t \notin \{\widehat{s}u \mid s \in \mathcal{A} \wedge u \in \mathcal{A}_\theta^*\}$$

$M_\sqcap^{\delta^\theta}$ can now be defined by the use of M^θ . In addition to the common merge relation, traces and refusal sets of P and Q are merged into new traces and new refusal sets.

Definition 19 (Internal choice - Initial merge).

$$\begin{aligned}M_\sqcap^{\delta^\theta} &=_{df} M^\theta \wedge (ref' = (0.ref \setminus \{\theta\}) \cup (\{\theta\} \cap (0.ref \cup 1.ref)) \vee \\ &\quad ref' = (1.ref \setminus \{\theta\}) \cup (\{\theta\} \cap (0.ref \cup 1.ref)))\end{aligned}$$

By adding $\{\theta\} \cap (0.ref \cup 1.ref)$ to the set of refused actions the new process refuses θ only if one of the two processes refuse to exhibit a θ event. In other words, only if both processes do not refuse θ , i.e., $\theta \notin 0.ref \wedge \theta \notin 1.ref$, the resulting process does not refuse θ as well, i.e., $\theta \notin ref'$.

$M^{-\delta^\theta}$ takes care that $P \sqcap^\theta Q$ behaves like P or Q . Additionally, M^θ is applied in order to potentially remove θ from the traces.

Definition 20 (Internal choice - Terminal merge).

$$M^{-\delta^\theta} =_{df} M^\theta \wedge ((tr' = 0.tr \wedge ref' = 0.ref) \vee (tr' = 1.tr \wedge ref' = 1.ref))$$

A quiescence preserving external choice operator is given by

Definition 21 (Quiescence preserving external choice).

$$\begin{aligned} P +^\theta Q &=_{df} (P \triangleleft Q); M_+ \\ M_+ &=_{df} M_+^{\delta^\theta} \triangleleft \delta^\theta \triangleright; M^{-\delta^\theta} \end{aligned}$$

Except the merge relation $M_+^{\delta^\theta}$ the external choice is equivalent to the internal choice. The difference is how the very beginning of P and Q is combined to form the new process $P +^\theta Q$.

Definition 22 (External choice merge relation).

$$M_+^{\delta^\theta} =_{df} M^\theta \wedge ref' = ((0.ref \cap 1.ref) \setminus \{\theta\}) \cup (\{\theta\} \cap (0.ref \cup 1.ref))$$

We can now define what type of processes can be used for modelling particular systems within the **ioco** testing framework.

Definition 23 (ioco specification). *An ioco specification is a reactive process satisfying the healthiness conditions **IOCO1**, **IOCO2** and **IOCO3**. In addition its set of possible events is partitioned into the quiescent event, input events, and output events: $\mathcal{A} = \mathcal{A}_{out} \cup \mathcal{A}_{in} \cup \{\theta\}$ where $\mathcal{A}_{out} \cap \mathcal{A}_{in} = \emptyset$*

Processes expressed in terms of $do_{\mathcal{A}}^\theta$, $;$, $+^\theta$ and \sqcap^θ are **ioco** specifications.

Remark 1. The class of labeled transition systems (LTS) used for the **ioco** relation is restricted to image finite LTSs [10]. Image finite LTSs are limited in their possible non-deterministic choices, i.e. image finite LTSs are bounded in terms of non-determinism. This requirement is only due to the properties of Tretmans' test case generation algorithm. Since we are interested in a predicative semantics we do not face the problem of image-finiteness

Remark 2. The TGV tool [3], which claims to generate test cases with respect to **ioco**, uses a different notion of quiescence: $quiescence_{TGV} =_{df} quiescence \vee (\neg ok' \wedge \neg wait')$. Note that we rely on *quiescence* rather than on *quiescence_{TGV}*.

3.3 IOCO Implementations

The input output conformance relation uses labeled transition systems to represent implementations. As mentioned in Section 2, it is not assumed that this LTS is known in advance, but only its existence is required. Our formalization requires something similar: implementations can be expressed as processes.

Processes for representing implementations in terms of the **ioco** relation need to satisfy the properties of specifications plus three additional properties: some restrictions on allowed choices⁵, input-enabledness⁶, and fairness⁷.

⁵ 2nd column on page 104 of [1]

⁶ 1st column on page 107 of [1]

⁷ 2nd column on page 115 of [1]

Restrictions on choices. An implementation is not allowed to freely choose between the actions enabled in a particular state. The **io** relation distinguishes between inputs and outputs not only by partitioning a process' alphabet, but also by assigning responsibilities to these two alphabets: “Output actions are initiated by and under control of the system under test while input actions are initiated by and under control of the system's environment” [1].

In terms of choices this means that choices between outputs are internal choices with respect to the implementation. Since internal choices are represented by disjunctions over the refused actions we need to restrict the choices between outputs to such disjunctions:

$$\mathbf{IOCO4} \quad P = P \wedge (wait' \implies (|\mathcal{A}_{out}| - 1) \leq |ref'_{out}|)$$

Because of this healthiness condition the τ transitions in state z_4 of Figure 3 are required. If these two transitions are absent the IOTS z would allow the environment to choose between output $!t$ and output $!c$.

In difference to that, input actions are under control of the system's environment. Thus, the choice of the input is up to the environment, i.e., choices between inputs are external choices. This restriction is enforced by requiring input-enabledness (see **IOCO5**). Because of $ref'_{in} = \emptyset$ an internal choice between inputs is impossible.

In addition, if there are inputs and outputs enabled in a particular state of an implementation the choice between input and output is up to the environment. That is, choices between inputs and outputs are external choices. Again, this restriction is covered by having input-enabled implementations, i.e. $ref'_{in} = \emptyset$ (see **IOCO5**). However, as identified in [11], an external choice between input and output allows the environment to prevent the system from providing an output. That is, whenever the implementation can choose between providing an output or waiting for a stimulus, the choice is triggered by the environment.

Remark 3. Recently, the constraints on the semantics of choices within implementations have been relaxed by [10]. By changing the properties of test cases (see Remark 4 in Section 3.5), choices between inputs and outputs are now choices of the implementation. Our work focuses on the original definition of **io**.

Input-enabledness. Input-enabledness requires that an implementation accepts every input in every (waiting) state. More precisely, an implementation cannot prevent the environment from providing an input, while running.

$$\mathbf{IOCO5} \quad P = P \wedge (wait' \implies (ref'_{in} = \emptyset))$$

While specifications are expressed in terms of $do_{\mathcal{A}}^{\theta}$, implementations use $\iota_{\mathcal{A}}$. More precisely, we express implementations by the use of $\iota_{\mathcal{A}}^{\theta}$. But let us start with $\iota_{\mathcal{A}}$ first. $\iota_{\mathcal{A}}$ takes care of the input-enabledness of processes.

For sake of simplicity we use following abbreviation to denote a sequence of inputs without a particular action: $\mathcal{A}_{in \setminus a}^* =_{df} (\mathcal{A}_{in} \setminus \{a\})^*$.

Definition 24 (Input-enabled communication). *Let $a \in \mathcal{A}$ be an action of a process' alphabet, then*

$$\iota_{\mathcal{A}}(a) =_{af} \Phi^i(\text{ref}'_{in} = \emptyset \wedge a \notin \text{ref}' \wedge \text{tr}' - \text{tr} \in \mathcal{A}_{in \setminus a}^* \triangleleft \text{wait}' \triangleright \text{tr}' - \text{tr} \in \mathcal{A}_{in \setminus a}^* \widehat{\langle a \rangle})$$

$\iota_{\mathcal{A}}(a)$ is similar to $do_{\mathcal{A}}(a)$. It denotes that the process $\iota_{\mathcal{A}}(a)$ cannot refuse to perform an a -action. Furthermore, $\iota_{\mathcal{A}}(a)$ cannot refuse to perform any input action. After executing any input action sequence ended by an a action the process $\iota_{\mathcal{A}}(a)$ terminates successfully.

Input enabledness also affects the representation of a deadlock. An input-enabled process needs to accept an input action at any time. That is an input-enabled process can only deadlock on outputs. Therefore, the deadlock process δ_l , which substitutes δ in the case of input-enabled processes, is given by:

Definition 25 (Output deadlock).

$$\delta_l =_{af} \mathbf{R}\mathfrak{Z}(\text{tr}' - \text{tr} \in \mathcal{A}_{in}^* \wedge \text{wait}')$$

Again, as for the non-input-enabled case, we need a quiescent version of $\iota_{\mathcal{A}}$. $\iota_{\mathcal{A}}^\theta(a)$ has θ events within its traces if a is an input event.

Definition 26 (Input-enabled quiescent communication). *Let $a \in \mathcal{A}$ be an action of a process' alphabet, then*

$$\iota_{\mathcal{A}}^\theta(a) =_{af} \begin{cases} \iota_{\mathcal{A}}(a) & \text{if } a \in \mathcal{A}_{out} \\ \Phi^i(\text{ref}'_{in} = \emptyset \wedge \theta \notin \text{ref}' \wedge \text{tr}' - \text{tr} \in (\mathcal{A}_{in \setminus a} \cup \theta)^* \triangleleft \text{wait}' \triangleright \text{tr}' - \text{tr} \in (\mathcal{A}_{in \setminus a} \cup \theta)^* \widehat{\langle a \rangle}) & \text{if } a \in \mathcal{A}_{in} \end{cases}$$

Combining input-enabledness with quiescence requires again a slight modification of the deadlock process. This leads to the output quiescent deadlock:

Definition 27 (Quiescent output deadlock).

$$\delta_l^\theta =_{af} \mathbf{R}\mathfrak{Z}(\text{tr}' - \text{tr} \in (\mathcal{A}_{in} \cup \theta)^* \wedge \text{wait}')$$

Fairness. Fairness is especially important for allowing theoretical exhaustive test case generation algorithms. Without assuming fairness of implementations there is not even a theoretical possibility of generating a failing test case for any non-conforming implementation. An unfair implementation may always lead a test case away from its errors. The fairness assumption for the input output conformance relation requires that an implementation shows all its possible non-deterministic behaviors when it is re-executed with a particular set of inputs. To express this fairness on an implementation we use a probabilistic choice operator similar to He and Sanders [12]. According to He and Sanders a probabilistic choice between two processes A and B can be expressed with $A \text{ }_p\oplus B$ where $0 \leq p \leq 1$. This expression equals A with probability p and B with probability $1 - p$. For example, $A \text{ }_{0.9}\oplus B$ denotes that the expression A will be chosen in 90% of the cases when executing the implementation.

The probabilistic version of our quiescence preserving internal choice, i.e. \sqcap^θ , is given by $P \sqcap^\theta Q$. The laws for $P \sqcap^\theta Q$ are similar to the laws for $P \oplus Q$, i.e.,

$$\begin{aligned}
 P \sqcap^\theta Q &= P \\
 P \sqcap^\theta Q &= Q \sqcap^\theta P \\
 P \sqcap^\theta P &= P \\
 P \sqcap^\theta (Q \sqcap^\theta R) &= (P \sqcap^\theta Q) \sqcap^\theta R, & r = ((1-p)q)/(1-(pq)) \\
 (P \sqcap^\theta Q); R &= (P; R) \sqcap^\theta (Q; R)
 \end{aligned}$$

A normal quiescence preserving internal choice is given by the non-deterministic choice of all possible probabilistic choices, i.e.

$$P \sqcap^\theta Q = \sqcap \{P \sqcap^\theta Q \mid 0 \leq p \leq 1\} \sqsubseteq P \sqcap^\theta Q$$

Using probabilistic choices basically means, that when one implements a choice it refines the specification by choosing a certain probability for this choice. Fairness is now simply expressed by restricting the probabilities p the implementer can choose to $0 < p < 1$:

Definition 28 (Internal fair (quiescence preserving) choice).

$$P \sqcap_f^\theta Q =_{df} \sqcap \{P \sqcap^\theta Q \mid 0 < p < 1\}$$

Thus, when executing a test case on the implementation the implementation will eventually exhibit all its possible behavior. As stated by the following lemma fair internal quiescence preserving choices are valid implementations of internal quiescence preserving choices. This guarantees that our internal quiescence preserving choice can be safely implemented by its fair version.

Lemma 1. $P \sqcap^\theta Q \sqsubseteq P \sqcap_f^\theta Q$

Proof.

$$\begin{aligned}
 P \sqcap_f^\theta Q &= && \{\text{definition of } \sqcap_f^\theta\} \\
 &= \sqcap \{P \sqcap^\theta Q \mid 0 < p < 1\} && \{\text{definition of } \sqcap\} \\
 &\Rightarrow \sqcap \{P \sqcap^\theta Q \mid 0 < p < 1\} \sqcap P \sqcap Q && \{\text{laws for } \sqcap^\theta\} \\
 &= \sqcap \{P \sqcap^\theta Q \mid 0 < p < 1\} \sqcap P \sqcap^\theta Q \sqcap P \sqcap^\theta Q && \{\text{definition of } \sqcap\} \\
 &= \sqcap \{P \sqcap^\theta Q \mid 0 \leq p \leq 1\} && \{\text{laws for } \sqcap^\theta\} \\
 &= P \sqcap^\theta Q
 \end{aligned}$$

Given the notion of input-enabledness and fairness we can now define which processes serve to represent **ioco** testable implementations:

Definition 29 (ioco testable implementation). *An ioco testable implementation is a reactive process satisfying the healthiness conditions **IOCO1-IOCO5**. In addition, an ioco testable implementation must be fair.*

Processes expressed in terms of $\iota_{\mathcal{A}}^\theta$, $;$, $+^\theta$, and Π_f^θ are **io** testable implementations if their choices obey to the following rules: (1) Choices between outputs are fair internal choices (Π_f^θ); (2) Choices between inputs and choices between inputs and outputs are external choices ($+^\theta$).

3.4 Predicative Input Output Conformance Relation

Recall that informally an IUT conforms to a specification S, iff the outputs of the IUT are outputs of S after an arbitrary suspension trace of S.

Thus, we need the (suspension) traces of a process, which are obtained by hiding all observations except the traces

Definition 30 (Traces of a process).

$$Trace(P) =_{df} \exists ref, ref', wait, wait', ok, ok' \bullet P$$

In addition to all traces of a particular process we need the traces after which a process is quiescent. Due to the chosen representation of quiescence (see Section 3.2) we use the following predicate in order to obtain the traces after which a process is quiescent

Definition 31 (Quiet traces of a process).

$$Quiet(P) =_{df} \exists ref'_{in} \bullet (P[false/wait'] \vee P[A_{out}/ref'_{out}])$$

Using these two predicates the input output conformance relation between implementation processes (see Definition 29) and specification processes (see Definition 23) can be defined as follows

Definition 32 (\sqsubseteq_{ioco}). *Given an implementation process I and a specification process S, then*

$$\begin{aligned} S \sqsubseteq_{ioco} I =_{df} [& \forall t \in \mathcal{A}_\theta^*, \forall o \in \mathcal{A}_{out} \bullet \\ & ((Trace(S)[t/tr'] \wedge Trace(I)[\widehat{t} o/tr']) \implies Trace(S)[\widehat{t} o/tr']) \wedge \\ & ((Trace(S)[t/tr'] \wedge Quiet(I)[t/tr']) \implies Quiet(S)[t/tr']] \end{aligned}$$

In order to distinguish the input output conformance given in denotational semantics from its operational semantics version we use different symbols. Note that because \sqsubseteq_{ioco} is related to refinement I **io** S is given by $S \sqsubseteq_{ioco} I$.

io relates the outputs (including quiescence) of I and S for all suspension traces of S . In contrast to that, our \sqsubseteq_{ioco} definition comprises two different parts. The first part considers only outputs while the second part deals with quiescence.

Using the predicative definition \sqsubseteq_{ioco} we can now show the relation between the input output conformance relation and refinement.

Theorem 1. $\sqsubseteq \subseteq \sqsubseteq_{ioco}$

Proof. In order to proof $\sqsubseteq \subseteq \sqsubseteq_{ioco}$, we have to show that $S \sqsubseteq I \implies S \sqsubseteq_{ioco} I$

$$\begin{aligned}
 & S \sqsubseteq I \\
 & \{\text{definition of } \sqsubseteq\} \\
 & = [I \implies S] \\
 & \{\text{propositional calculus}\} \\
 & = [I \implies S] \wedge [(I \vee I) \implies (S \vee S)] \\
 & \{\text{definition of } []\} \\
 & = [(I \implies S) \wedge ((I \vee I) \implies (S \vee S))] \\
 & \{\text{propositional calculus}\} \\
 & \Rightarrow [(I \implies S) \wedge ((\exists ref'_{in} \bullet (I[false/wait'] \vee I[A_{out}/ref'_{out}])) \implies \\
 & \quad (\exists ref'_{in} \bullet (S[false/wait'] \vee S[A_{out}/ref'_{out}]))) \\
 & \{\text{propositional calculus and definition of } Quiet\} \\
 & \Rightarrow [((\exists ref, ref', wait, wait', ok, ok' \bullet I) \implies \\
 & \quad (\exists ref, ref', wait, wait', ok, ok' \bullet S)) \wedge (Quiet(I) \implies Quiet(S))] \\
 & \{\text{propositional calculus and definition of } Trace\} \\
 & \Rightarrow [\forall t \in \mathcal{A}_\theta^* \bullet (Trace(I)[t/tr'] \implies Trace(S)[t/tr']) \wedge \\
 & \quad \forall t \in \mathcal{A}_\theta^* \bullet (Quiet(I)[t/tr'] \implies Quiet(S)[t/tr'])] \\
 & \{\text{propositional calculus}\} \\
 & \Rightarrow [\forall t \in \mathcal{A}_\theta^*, \forall o \in \mathcal{A}_{out} \bullet (Trace(I)[\widehat{t} o/tr'] \implies Trace(S)[\widehat{t} o/tr']) \wedge \\
 & \quad \forall t \in \mathcal{A}_\theta^* \bullet (Quiet(I)[t/tr'] \implies Quiet(S)[t/tr'])] \\
 & \{\text{propositional calculus and definition of } Trace\} \\
 & \Rightarrow [\forall t \in \mathcal{A}_\theta^*, \forall o \in \mathcal{A}_{out} \bullet \\
 & \quad ((Trace(S)[t/tr'] \wedge Trace(I)[\widehat{t} o/tr']) \implies Trace(S)[\widehat{t} o/tr']) \wedge \\
 & \quad \forall t \in \mathcal{A}_\theta^* \bullet ((Trace(S)[t/tr'] \wedge Quiet(I)[t/tr']) \implies Quiet(S)[t/tr'])] \\
 & \{\text{distributivity of } \forall\} \\
 & = [\forall t \in \mathcal{A}_\theta^*, \forall o \in \mathcal{A}_{out} \bullet \\
 & \quad (((Trace(S)[t/tr'] \wedge Trace(I)[\widehat{t} o/tr']) \implies Trace(S)[\widehat{t} o/tr']) \wedge \\
 & \quad ((Trace(S)[t/tr'] \wedge Quiet(I)[t/tr']) \implies Quiet(S)[t/tr'])] \\
 & \{\text{definition of } \sqsubseteq_{ioco}\} \\
 & = S \sqsubseteq_{ioco} I
 \end{aligned}$$

Although, the definition of \sqsubseteq_{ioco} corresponds to the definition of **ioco** we can reformulate our definition to a more generic version \sqsubseteq_{ioco}^P which corresponds to **ioco_F**. Like \mathcal{F} in **ioco_F**, P is used to select the proper set of traces

Definition 33 (\sqsubseteq_{ioco}^P). *Given an implementation process I and a specification process S , then*

$$\begin{aligned}
 S \sqsubseteq_{ioco}^P I =_{df} [& \forall t \in \mathcal{A}_\theta^*, \forall o \in \mathcal{A}_{out} \bullet \\
 & ((P(S, I, t) \wedge Trace(I)[\widehat{t} o/tr']) \implies Trace(S)[\widehat{t} o/tr']) \wedge \\
 & ((P(S, I, t) \wedge Quiet(I)[t/tr']) \implies Quiet(S)[t/tr'])]
 \end{aligned}$$

This generic version \sqsubseteq_{ioco}^P allows us to define a predicative version of the conformance relations listed in Section 2.

Definition 34 (Input output testing relation (iot)).

$$S \sqsubseteq_{iot} I =_{df} S \sqsubseteq_{ioco}^{P_{iot}} I \text{ where } P_{iot}(S, I, t) = t \in \mathcal{A}_\theta^*$$

Definition 35 (Input output refusal relation (ior)).

$$S \sqsubseteq_{ior} I =_{df} S \sqsubseteq_{ioco}^{P_{ior}} I \text{ where } P_{ior}(S, I, t) = t \in \mathcal{A}^*$$

Definition 36 (IO-conformance (ioconf)).

$$S \sqsubseteq_{ioconf} I =_{df} S \sqsubseteq_{ioco}^{P_{ioconf}} I \text{ where } P_{ioconf}(S, I, t) = \text{Trace}(S)[t/tr'] \wedge t \in \mathcal{A}^*$$

Definition 37 (Input output conformance (ioco)).

$$S \sqsubseteq_{ioco} I =_{df} S \sqsubseteq_{ioco}^{P_{ioco}} I \text{ where } P_{ioco}(S, I, t) = \text{Trace}(S)[t/tr'] \wedge t \in \mathcal{A}_\theta^*$$

3.5 Test Cases, Test Processes, and Test Suites

Testing for conformance is done by applying a set of test cases in order to assess if an implementation conforms to a specification. This raises the question of what is a test case. Test cases can also be seen as processes satisfying additional properties.

Since the activity of testing should end at some point in time a test process has to have finite behavior⁷.

$$\mathbf{TC1} \quad P(tr, tr') = P \wedge (\exists n \in \mathbb{N} \bullet \text{length}(tr' - tr) \leq n)$$

Furthermore, a test case either accepts all responses from an implementation under test, i.e., inputs from the view of the test case, or it accepts no inputs at all⁷

$$\mathbf{TC2} \quad P = P \wedge (\text{wait}' \implies (\text{ref}'_{in} = \mathcal{A}_{in} \vee \text{ref}'_{in} = \emptyset))$$

If the test case has to provide a particular stimuli to the implementation under test it is always clear which output (from the view of the test case) should be send⁷:

$$\mathbf{TC3} \quad P = P \wedge (\text{wait}' \implies (|\text{ref}'_{out}| \geq |\mathcal{A}_{out}| - 1))$$

Furthermore, testing should be a deterministic activity, i.e. test cases should be deterministic⁷. Determinism includes that a tester can always deterministically decide what to do: send a particular stimuli to the IUT or wait for a possible response⁸. This is known as controllability and is ensured by the following two healthiness conditions.

$$\mathbf{TC5} \quad P = P \wedge (\text{wait}' \implies (|\text{ref}'_{out}| = |\mathcal{A}_{out}| - 1) \iff \text{ref}'_{in} = \mathcal{A}_{in}))$$

$$\mathbf{TC6} \quad P = P \wedge (\text{wait}' \implies ((\text{ref}'_{in} = \emptyset) \iff \text{ref}'_{out} = \mathcal{A}_{out}))$$

After termination a test case should give a verdict about the test execution

$$\mathbf{TC7} \quad P = P \wedge (\neg \text{wait}' \implies (\text{pass}' \iff \neg \text{fail}'))$$

⁷ Definition 5.1 on page 115 of [1]

⁸ 1st column on page 115 of [1]

Definition 38 (Test process). *A test process P is a reactive process, which satisfies the healthiness conditions **TC1**...**TC7** and **IOCO1**. The set of events in which a test case can potentially engage is given by \mathcal{A} , where $\mathcal{A} = \mathcal{A}_{out} \cup \mathcal{A}_{in}$, $\mathcal{A}_{out} \cap \mathcal{A}_{in} = \emptyset$ and $\theta \in \mathcal{A}_{in}$. The observations are extended by:*

- *pass* : *Bool*, which denotes the pass verdict
- *fail* : *Bool*, which denotes that an implementation has failed a test case

Remark 4. Due to the results of Petrenko et al. [11], the properties of test cases have been changed recently [10]. Test cases are now input-enabled, i.e. they are not able to block any input (i.e. outputs of the implementation) anymore. Hence, test cases have now to accept every output of the implementation in every state. Note that this conflicts with healthiness condition **TC5** and **TC6**. Test cases are less controllable in terms of inputs and outputs. During the test execution one has now to decide non-deterministically whether to send an input to the IUT or to wait for an output of the IUT. However, we use the original version of **ioco** in this paper.

Test cases are reactive processes, i.e. when writing an action a in the context of a test case we mean $do_{\mathcal{A}}(a)$. For sake of clarity we use abbreviations for indicating pass (\checkmark) and fail (\times) verdicts within our test cases.

$$\checkmark =_{af} (\neg wait' \implies pass') \quad \times =_{af} (\neg wait' \implies fail')$$

Due to the properties of test cases the only choices of a test case are choices between inputs. Since the chosen input to a test case, i.e. output of the IUT, are up to the IUT this choice is given in terms of an external choice:

$$P + Q =_{af} P \wedge Q \triangleleft \delta \triangleright P \vee Q \\ \delta =_{af} \mathbf{R3}(tr' = tr \wedge wait')$$

Example 6. For example, a valid test case T with the alphabet $A_{in} = \{c, t, \theta\}$ and $A_{out} = \{1\}$ that sends a stimuli and subsequently accepts a c -response but neither accepts t nor quiescence is given by

$$T = !1; ((?c \wedge \checkmark) + (?t \wedge \times) + (\theta \wedge \times)) = \{\text{def. of } do_{\mathcal{A}}, \checkmark, \times, \text{ and } +\} \\ = (!1 \notin ref' \wedge tr' = tr \vee (?c \notin ref' \wedge ?t \notin ref' \wedge \theta \notin ref') \wedge tr' = tr \hat{\langle !1 \rangle}) \\ \triangleleft wait' \triangleright (tr' = tr \hat{\langle !1 \rangle} \hat{\langle ?c \rangle} \wedge pass' \vee \\ tr' = tr \hat{\langle !1 \rangle} \hat{\langle ?t \rangle} \wedge fail' \vee tr' = tr \hat{\langle !1 \rangle} \hat{\langle \theta \rangle} \wedge fail')$$

Usually, a test suite is a set of test cases. In our case this means that a test suite is given by the nondeterministic choice of a set of test cases.

Definition 39 (Test suite). *Given a set of N test processes T_1, \dots, T_N , then a test suite TS is defined as:*

$$TS =_{af} \prod_{i=1, \dots, N} T_i$$

3.6 Testing Implementations

Test case execution in the input output conformance testing framework is modeled by executing the test case in parallel to the system under test. We model this parallel execution again as parallel merge (see Section 3.2).

The execution of a test case t on a particular implementation i is denoted by $t \parallel i$. This new process $t \parallel i$ consists of all traces present in both, the test case and the implementation. Furthermore, $t \parallel i$ gives *fail'* and *pass'* verdicts after termination, i.e. in the case of $\neg \text{wait}'$.

Such an execution operator is inspired by CSPs parallel composition [9], i.e. the parallel composition of a test case and an implementation can only engage in a particular action if both processes participate in the communication.

Since the inputs of a test case are the outputs of the implementation under test and the outputs of a test case are inputs to the IUT we need to rename the alphabets. Therefore, we define an alphabet renaming operator for a process P denoted by \overline{P} as follows: $\mathcal{A}\overline{P} =_{df} \mathcal{A}P$; $\mathcal{A}_{out}\overline{P} =_{df} \mathcal{A}_{in}P$; $\mathcal{A}_{in}\overline{P} =_{df} \mathcal{A}_{out}P$

The test case execution operator is given by:

Definition 40 (Test case execution). *Let TC be a test case process and IUT be an implementation process, then*

$$\begin{aligned} \mathcal{A}(TC \parallel IUT) &=_{df} \mathcal{A}\overline{TC} \cup \mathcal{A}IUT \\ TC \parallel IUT &=_{df} (\overline{TC} \triangleleft IUT); M_{ti} \end{aligned}$$

The relation M_{ti} merges the traces of the test case and the implementation. The result comprises the pass and fail verdicts of the test case as well as traces that are allowed in both, the test case and the implementation.

Because of the chosen representation of quiescence, there is no θ that indicates termination of the IUT, i.e., $\neg 1.\text{wait}$. Note that M_{ti} takes care of that when merging the traces.

Definition 41 (Test case/impl. merge).

$$\begin{aligned} M_{ti} &=_{df} \text{pass}' = 0.\text{pass} \wedge \text{fail}' = 0.\text{fail} \wedge \text{wait}' = (0.\text{wait} \wedge 1.\text{wait}) \wedge \\ &\quad \text{ref}' = (0.\text{ref} \cup 1.\text{ref}) \wedge \\ &\quad (\exists u \bullet ((u = (0.\text{tr} - \text{tr}) \wedge u = (1.\text{tr} - \text{tr}) \wedge \text{tr}' = \text{tr} \hat{u}) \vee \\ &\quad (u \hat{\langle \theta \rangle} = (0.\text{tr} - \text{tr}) \wedge u = (1.\text{tr} - \text{tr}) \wedge \text{tr}' = \text{tr} \hat{u} \hat{\langle \theta \rangle}) \wedge \neg 1.\text{wait})) \end{aligned}$$

Due to the lack of symmetry of our merge operator the test case execution operator \parallel is not symmetric. However, it still satisfies one important law. Let T_1, T_2 be test cases and let P be an implementation, then

$$\text{LAW L2} \quad (T_1 \sqcap T_2) \parallel P = (T_1 \parallel P) \sqcap (T_2 \parallel P)$$

This law allows one to run a set of N test cases T_1, \dots, T_N , i.e. a test suite $TS =_{df} \prod_{i=1, \dots, N} T_i$, against an implementation process P :

$$TS \parallel P = \prod_{i=1, \dots, N} T_i \parallel P = (T_1 \sqcap \dots \sqcap T_N) \parallel P$$

Since our test cases do not consist of a single trace but of several traces there may be different verdicts given at the end of different traces. An implementation passes a test case if all possible test runs lead to the verdict pass:

Definition 42 (Global verdict). *Given a test process (or a test suite) T and an implementation process IUT , then*

$$\begin{aligned} IUT \text{ passes } T &=_{df} \forall r \in \mathcal{A}_\theta^* \bullet (((T||IUT)[r/tr'] \wedge \neg wait') \implies pass') \\ IUT \text{ fails } T &=_{df} \exists r \in \mathcal{A}_\theta^* \bullet (((T||IUT)[r/tr'] \wedge \neg wait') \implies fail') \end{aligned}$$

Example 7. Now we can calculate verdicts by executing test cases on implementations. For example, consider the test case of Example 6, i.e. $T = !1; ((?c \wedge \checkmark) + (?t \wedge \mathbf{X}) + (\theta \wedge \mathbf{X}))$, and the implementation $\mathcal{P}_w = \iota_{\mathcal{A}}^\theta(?1); \iota_{\mathcal{A}}^\theta(!c)$ (representing implementation w of Figure 3). Executing t on the implementation \mathcal{P}_w , i.e. $T||\mathcal{P}_w$, is conducted as follows: .

$$\begin{aligned} T||\mathcal{P}_w &= && \{\text{def. of } ||\} \\ &= \overline{(!1; ((?c \wedge \checkmark) + (?t \wedge \mathbf{X}) + (\theta \wedge \mathbf{X}))} \triangleleft \iota_{\mathcal{A}}^\theta(?1); \iota_{\mathcal{A}}^\theta(!c); M_{ti} && \{\text{def. of renaming}\} \\ &= (?1; ((!c \wedge \checkmark) + (!t \wedge \mathbf{X}) + (\theta \wedge \mathbf{X})) \triangleleft \iota_{\mathcal{A}}^\theta(?1); \iota_{\mathcal{A}}^\theta(!c); M_{ti} && \{\text{def. of } \triangleleft \text{ and } M_{ti}\} \\ &= (?1 \notin ref' \wedge tr' = tr \vee !c \notin ref' \wedge tr' = tr \widehat{(?1)}) \\ &\quad \triangleleft wait' \triangleright tr' = tr \widehat{(?1)} \widehat{(?c)} \wedge pass' \end{aligned}$$

Thus, we have \mathcal{P}_w passes T because

$$\begin{aligned} \mathcal{P}_w \text{ passes } T & && \{\text{def. of passes}\} \\ &= \forall r \in \mathcal{A}_\theta^* \bullet (((T||\mathcal{P}_w)[r/tr'] \wedge \neg wait') \implies pass') && \{t||\mathcal{P}_w\} \\ &= \forall r \in \mathcal{A}_\theta^* \bullet ((\neg wait' \wedge r = tr \widehat{(?1)} \widehat{(?c)} \wedge pass') \implies pass') && \{\text{prop. calc.}\} \\ &= \forall r \in \mathcal{A}_\theta^* \bullet TRUE = TRUE \end{aligned}$$

4 Conclusion and Future Work

This paper lifts the input output conformance (**ioco**) theory of Tretmans [1] to UTP's reactive processes [6], including all underlying assumptions of **ioco**.

The presented operators allow our processes to make the absence of output events observable. Furthermore, we showed how to express input enabled processes and introduced a notion of fairness. Input enabled, fair processes are used to reason about implementations.

By the use of specification processes and implementation processes we defined \sqsubseteq_{ioco} . This conformance relation gives a notion of correctness of an implementation with respect to a specification in terms of the observable behavior.

Although, this paper gives the basic notion of specifications, implementations, test cases and conformance there is plenty of work left. One issue is the formulation of laws regarding the conformance relation. Another open task is to instantiate this framework for a particular process algebra, e.g. CSP or ACP.

Furthermore, there are many extensions to the original input output conformance. For example, Lestiennes and Gaudel [13] presented the *rioco* relation which allows to relax the property of input-enabledness. Another variation of **ioco** is input output conformance under the presence of time. This has been considered in [14] and results into timed-**ioco** (*tioco*). It would be interesting to study these conformance relations in terms of UTP and compare arising healthiness conditions to the healthiness conditions presented in this paper.

Acknowledgments The research herein is partially conducted within the competence network Softnet Austria and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT). Additionally, this research is partially funded by the EU project ICT-216679, Model-based Generation of Tests for Dependable Embedded Systems (MOGENTES).

References

1. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. *Software - Concepts and Tools* **17**(3) (1996) 103–120
2. Tretmans, J., Brinksma, E.: TorX: Automated model based testing. In: 1st European Conference on Model-Driven Software Engineering. (2003) 13–25
3. Jard, C., Jéron, T.: TGV: theory, principles and algorithms. *International Journal on Software Tools for Technology Transfer* **7**(4) (August 2005) 297–315
4. de Vries, R.G., Belinfante, A., Feenstra, J.: Automated testing in practice: The highway tolling system. In: 14th International Conference on Testing Communicating Systems. Volume 210 of IFIP Proceedings. (2002) 219–234
5. Aichernig, B.K., Peischl, B., Weiglhofer, M., Wotawa, F.: Protocol conformance testing a SIP registrar: An industrial application of formal methods. In: 5th International Conference on Software Engineering and Formal Methods, London, UK, IEEE (2007) 215–224
6. Hoare, C., He, J.: *Unifying Theories of Programming*. Prentice Hall (1998)
7. Bernot, G.: Testing against formal specifications: A theoretical view. In: International Joint Conference on Theory and Practice of Software Development. Volume 494 of LNCS., Springer (1991) 99–119
8. Tretmans, G.J.: *A Formal Approach to Conformance Testing*. PhD thesis, University of Twente, Enschede (December 1992)
9. Hoare, C.: *Communicating Sequential Processes*. Prentice Hall (1985)
10. Tretmans, J.: Model based testing with labelled transition systems. In: *Formal Methods and Testing*. Volume 4949 of LNCS., Springer (2008) 1–38
11. Petrenko, A., Yevtushenko, N., Huo, J.L.: Testing transition systems with input and output testers. In: 15th International Conference on Testing of Communication Systems. Volume 2644 of LNCS., Springer (May 2003) 129–145
12. He, J., Sanders, J.W.: Unifying probability. In: 1st International Symposium on Unifying Theories of Programming. Volume 4010 of LNCS. (2006) 173–199
13. Lestiennes, G., Gaudel, M.C.: Test de systèmes réactifs non réceptifs. *Journal Européen des Systèmes Automatisés, Modélisation des Systèmes Réactifs* **39**(1–3) (2005) 255–270 in French, Technical Report in English available.
14. Krichen, M., Tripakis, S.: Black-box conformance testing for real-time systems. In: 11th International SPIN Workshop. Volume 2989 of LNCS. (2004) 109–126