Inputs and outputs in CSP: a model and a testing theory

ANA CAVALCANTI, University of York, UK
ROBERT M. HIERONS, University of Sheffield, UK
SIDNEY NOGUEIRA, Universidade Federal Rural de Pernambuco, Brazil

This paper addresses refinement and testing based on CSP models, when we distinguish input and output events. In a testing experiment, the tester (or the environment) controls the inputs, and the system under test controls the outputs. The standard models and refinement relations of CSP, however, do not differentiate inputs and outputs and are not, therefore, entirely suitable for testing. Here, we consider an alphabet of events partitioned into inputs and outputs, and present a novel refusal-testing model for CSP with a notion of input-output refusal-traces refinement. We compare that with the ioco relation often used in testing, and find that it is more widely applicable and stronger. This means that mistakes found using traditional ioco testing do indicate mistakes in the development. Finally, we provide a CSP testing theory that takes into account inputs and outputs. With our theory, it becomes feasible to develop techniques and tools for automatic generation of realistic and sound tests from CSP models. Our work reconciles the normally disparate areas of refinement and (formal) testing by identifying how ioco testing can be used to inform refinement-based results, and vice-versa.

CCS Concepts: • **Software and its engineering** → **Formal methods**; **Software testing and debugging**; • **Computing methodologies** → *Concurrent computing methodologies*;

Additional Key Words and Phrases: Exhaustive test set, process algebra, refinement, refusal-testing model

ACM Reference Format:
ANA CAVALCANTI, ROBERT M. HIERONS, and SIDNEY NOGUEIRA. 2020. Inputs and outputs in CSP: a model and a testing theory. *ACM Trans. Comput. Logic* 1, 1 (March 2020), 54 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As a process algebra, CSP [34] is a well established notation, with robust semantics and tools. It has been in use for more than twenty years and the availability of a powerful model checker has ensured that CSP is of interest both in academia and industry [17]. In the public domain, we have reports of applications in hardware and e-commerce [1, 20]. In addition, CSP has been combined with a number of data modelling languages to define notations that cope with state-rich reactive systems [14, 16, 27, 32, 36]. Model-based testing, on the other hand, is not a traditional area of application for CSP. Yet, when a CSP model is available, the possibility of using it for testing is attractive. In fact, CSP has a testing theory [7], and, more recently, its use as part of testing techniques has been explored [9, 24, 30, 35].

A difficulty is that CSP models do not distinguish between input and outputs: they are all synchronisations. In testing, however, there is an asymmetry: the system under test (SUT) controls outputs, while the tester controls inputs.

Authors' addresses: ANA CAVALCANTI University of York, UK, ana.cavalcanti@york.ac.uk; ROBERT M. HIERONS University of Sheffield, UK, r.hierons@sheffield.ac.uk; SIDNEY NOGUEIRAUniversidade Federal Rural de Pernambuco, Brazil, sidney.ufrpe@gmail.com.

In the software testing community, there has been much interest in input-output transition systems (IOTSs) [38] and the ioco implementation relation [3, 25, 42, 43]. In this context, observations are traces that include inputs, outputs, and quiescence, a state in which all enabled events are inputs and it is not possible to take an internal transition. Quiescence is the only type of refusal observed. Implementations must be input enabled, that is, accept any input at any time.

In [12], we follow a suggestion in [33] to define a stable-failures model for CSP, where refusals are observed at the end of traces, parametrised by sets of input and output events. Using the resulting input-output stable-failures model, we define input-output failures refinement and show that it is, in general, incomparable to ioco, and when specifications and implementations are input enabled, ioco is stronger. In addition, we adapt the CSP testing theory to input-output failures refinement, and show that some tests in the exhaustive test set for failures refinement become unnecessary.

Although these results give clear guidance on the impact of a differentiated treatment of inputs and outputs on testing based on CSP, they are unsatisfactory in at least one important respect. Input-output failures refinement is not comparable to ioco (even when refinement is restricted to input-enabled implementations). The wide acceptance of ioco means that a conformance relation that is at least as strong as ioco is more useful for testing.

In this paper, we consider the stable-refusal testing model ($\mathcal{RT}$) [31], instead of the stable-failures model, to study inputs and outputs in CSP. In $\mathcal{RT}$, observations of an execution are recorded in traces including events and refusal sets, that is, sets of events in which the process refuses to engage. A trace of $\mathcal{RT}$ alternates events and refusals, with a refusal at the start and the end. $\mathcal{RT}$ is richer than the stable-failures model, which only records refusals at the end of sequences of events, and than the IOTS model, which only records quiescence, which is a particular form of refusal.

In the presence of inputs and outputs, we define for CSP that a state is stable if it is stable according to the refusal-testing model, and no output is enabled. These are quiescent states, but we can observe the inputs that are enabled: models and implementations need not be input enabled. This notion of stability is the basis of the input-output failures model of [12]. Here, we adopt it to define an input-output refusal-testing model for CSP. We formalise the notion of (refusal) traces in this model and its healthiness conditions, and calculate definitions for the CSP operators.

Others have investigated refusals for inputs [3, 6, 21]. In these lines of work, however, refusals can be observed in states from which an output is possible. The traditional explanation regarding the observation of a refusal set X is that, if the tester offers only events of X, we observe a refusal if the SUT deadlocks. Usually a tester does not block outputs from the SUT, and so the SUT does not deadlock if an output is available. So, we do not consider a state to be stable if an output is possible (since the SUT can change state), and do not allow the observation of refusals in such states.

The testing theory of CSP identifies (typically infinite) test sets that are sufficient and necessary to establish (traces or failures) refinement with respect to a given CSP specification. To take advantage of the knowledge about inputs and outputs, here we adapt that theory for input-output refusal-traces refinement. We identify a test set for a given specification and prove that it is exhaustive for input-output refusal-traces refinement. A novel notion of test uses a version of Roscoe's prioritise operator [34] to ensure that a testing experiment can observe a refusal and continue.

We make the following contributions. First, we give a full account of the $\mathcal{RT}$ model, based on the account of healthiness conditions in [34] and definition of operators in [29], but revisited to handle divergence and termination. Our treatment is compatible with the recent proposal in [34], but has a uniform treatment of refusal traces and includes the definition of CSP operators. We prove that all operator definitions satisfy the healthiness conditions of the model.

Secondly, based on that, we define an input-output refusal-traces model for CSP and an associated notion of refinement; we give healthiness conditions and calculate operator definitions. Crucially, we prove that input-output refusal-traces refinement is stronger than ioco, equivalent for input-enabled processes, and more general, since it does not require implementations to be input enabled. So, a non-conformance identified when testing against a specification

using ioco indicates that the implementation does not refine the specification. Lastly, because input-output refusal-traces refinement is more general, we have also defined a testing theory for this conformance relation.

Refinement is a conformance relation suitable for development, and ioco is suitable for the restricted observability of testing. Here, we have, for the first time, identified how ioco-based testing can be used to shed light on refinement proof, and how design by refinement may influence testing. It makes no sense for developers and testers to use totally unrelated notions of conformance. Our work reconciles these verification approaches.

In addition, the generality of our new conformance relation, input-output refusal-traces refinement, which does not require input enabledness, simplifies modelling. Many systems are not input-enabled. For example, web interfaces often have options that are deliberately disabled; for instance, the options in an online banking system might depend on the account type. Many robotic systems are also not input-enabled and this is, again, deliberate. For instance, it is very common for such systems to have different modes of operation, and switch between them to respond to different environment scenarios, when different sets of sensors can be used. We can certainly model a system that is not input-enabled using input-enabled IOTS by adding transitions that ignore irrelevant inputs. This is not, however, what developers do. They do not implement inputs that are not needed, and doing so may raise issues of efficiency, usability, or even safety. Our results again reconcile the developer and tester views.

With this, we enable practical use of CSP in testing. As already illustrated for the standard CSP theory [8, 15], based on our theory for testing for input-output refusal-traces refinement, it is possible to automate the generation of sound tests. In this case, however, the tests consider inputs and outputs as required in practical approaches.

Next, we present CSP and IOLTS. In Section 3, we revisit the $\mathcal{RT}$ model, and in Section 4, present our new CSP model with inputs and outputs, and its refinement notion. Section 5 discusses the relationship between input-output refusal-traces refinement and ioco. Testing is addressed in Section 6. We conclude in Section 7, where we also discuss related and future work. Appendix A presents definitions of some non-standard operators used here. Appendix B presents proofs for key results that we discuss. A complete set of proofs for all lemmas and theorems is in [10].

2 CSP AND IOLTS

Here, we describe the two notations used in our work: CSP in Section 2.1, and IOLTS and IOTS in Section 2.2.

2.1 CSP

Systems and their components are modelled in CSP using processes. They interact with their environment and each other synchronously via atomic and instantaneous events. In any given model, Σ is the set of all declared events.

A process that is ready to synchronise with the environment on an event a can be written using the prefixing operator $\rightarrow$ as follows $a \rightarrow P$. This process, after engaging on a, when the environment is ready, behaves like the process P.

Basic processes include div, $STOP$, and $SKIP$, which diverge, deadlock, and terminate immediately. A special event $\checkmark$ marks termination, and cannot be explicitly used in process definitions. $\Sigma^{\checkmark}$ includes $\checkmark$ as well as all declared events.

Another core operator is external choice ($\Box$), which offers to the environment the possibility of choosing between processes, via an interaction on events that are initially available. We provide an example next.

Example 2.1. We present below a process that offers the environment a choice.

$$a \rightarrow STOP \;\Box\; b \rightarrow (a \rightarrow c \rightarrow STOP \;\Box\; d \rightarrow c \rightarrow STOP)$$

The environment exercises its choice by interacting on either a or b. If it offers a, then this process deadlocks ($STOP$)

and the behaviour of $b \rightarrow (a \rightarrow c \rightarrow STOP \,\Box\, d \rightarrow c \rightarrow STOP)$ is no longer possible. Equally, if it synchronises on b, the behaviour is then that of $a \rightarrow c \rightarrow STOP \,\Box\, d \rightarrow c \rightarrow STOP$, so a choice based on a and d is offered. □

Another form of choice is internal ($\sqcap$), where the environment has no control.

Example 2.2. We present below a process that makes an internal choice.

$$a \rightarrow STOP \sqcap b \rightarrow (a \rightarrow c \rightarrow STOP \,\Box\, d \rightarrow c \rightarrow STOP)$$

In this case, irrespective of what the environment offers, the process makes its own choice and proceeds. So, even if the environment is ready to synchonise only on a, the process may choose to offer b, and then we have a deadlock. □

Processes can be combined in sequence $P; Q$, where, as usual, the behaviour is that of P, until it terminates, when Q takes over. There are several forms of parallelism. In a generalised parallelism $P \,[\![Z]\!]\, Q$, the processes P and Q must agree on the events in the set Z, and are free to engage on events not in Z independently. As shown in [33], the generalised parallel operator can be used to express other forms of parallelism by using appropriate values for the synchronisation set of events Z. For example, interleaving is the behaviour exhibited when Z is the empty set.
For abstraction, a process can have its events hidden ($\setminus$) or renamed ($\_[[R]]$) as illustrated next.

Example 2.3. Below we apply the hiding operator to the process of Example 2.1 to hide occurrences of c.

$$(a \rightarrow STOP \,\Box\, b \rightarrow (a \rightarrow c \rightarrow STOP \,\Box\, d \rightarrow c \rightarrow STOP)) \setminus \{c\}$$

After the environment interacts with this process on b and then either a or d, there is a deadlock. The inevitable event c becomes internal, and so happens without the need for agreement of the environment, which cannot observe it. □

In a renaming, we provide a total relation R that associates old and new names of events, and ✓ to itself, only.

Example 2.4. We use the relation R that associates a with a itself and c; b and d with themselves; and c with d.

$$(a \rightarrow b \rightarrow STOP)[[R]] = a \rightarrow b \rightarrow STOP \,\Box\, c \rightarrow b \rightarrow STOP$$

Since a is mapped to both itself and c, the prefixing on a becomes an external choice that offers both a and c.

$$(c \rightarrow b \rightarrow STOP \,\Box\, d \rightarrow d \rightarrow STOP)[[R]] = d \rightarrow (b \rightarrow STOP \sqcap d \rightarrow STOP)$$

Now, since both events c and d offered in the choice are associated with d, the choice becomes a prefixing on d. After that event, however, we now have an internal choice to define the behaviour afterwards. Effectively, the renaming takes away the choice from the environment by identifying the events offered in choice. □

The simplest semantic model of CSP is the traces model: $traces[[P]]$ is the set of the sequences of events in which P can engage. This simple model is intuitive, but restricted. It cannot, for example, distinguish external and internal choice. For instance, the processes in Examples 2.1 and 2.2 have the same traces: $\{\langle\rangle, \langle a\rangle, \langle b\rangle, \langle b, a\rangle, \langle b, d\rangle, \langle b, a, c\rangle, \langle b, d, c\rangle\}$.

The richer stable-failures model records the failures $failures[[P]]$ of a process P: pairs formed by a trace that does not lead to a divergent state and a set of events, called a refusal, that P may refuse. A failure (t, Y) is a record of a failed experiment: having engaged in the trace t, and been offered by the environment the events in Y, then P can deadlock.

The definition of a failure depends on the set Σ of events in scope. If, for the process in Example 2.1, Σ is $\{a, b, c, d\}$, we have the failure $(\langle\rangle, \{c, d\})$. This means that, before any events take place, in an environment that offers just c and d, the process deadlocks: $\{c, d\}$ is a refusal, after the empty trace. Of course, if the environment offers just some of these events, the process also deadlocks. So, $\{c\}$, $\{d\}$, and $\emptyset$ are also refusals after the empty trace.

For the internal choice in Example 2.2, we in addition have failures $(\langle\rangle, \{a\})$ and $(\langle\rangle, \{b\})$. In this case, if the environment offers, for example, just a, the internal choice may be resolved in favour of the process that offers just b.

As explained, the standard models of CSP do not distinguish between input and output events: every interaction is just a synchronisation. As a syntactic abbreviation, we can write, for example, a prefixing $in?x \rightarrow P$ to describe a process that takes an input x through a channel in. This is, however, just a shorthand for an external choice over events $in.v$, for every value v of the type of in. An event $in.v$ is composed, but, like any other event, requires synchronisation.

We can also write $out!v$ for a composed event $out.v$. It is used to indicate that $out.v$ is meant to be an output of a value v through a channel out. Again, in the standard semantic models, $out.v$ is just like any other event.

The input output-failures model of CSP is defined in terms of the stable-failures model. As mentioned, in general terms, it takes the view that, if an output is possible, the state is not stable and a refusal cannot be observed. In Example 2.1, if we take that a, for instance, is an output, we have no input-output failures for the empty trace $\langle\rangle$.

The canonic semantics of CSP is the failures-divergences model, which also records the divergences of a process: the set of its traces that can lead to divergent behaviour. This embeds the view that, when there is a possibility of divergence, the behaviour of the process is arbitrary. So, its traces and failures are enriched to record that, after a possible divergence, any events may take place or be refused. For a divergence-free process, this semantics boils down to its stable failures.

Distinctively, CSP is suitable for refinement. Each model defines a refinement relation; we have, for example, $P \sqsubseteq_T Q$, $P \sqsubseteq_F Q$, and $P \sqsubseteq_{FD} TQ$, for traces, failures, and failures-divergence refinement. In each case, refinement holds when the behaviour of the (implementation) process Q is a subset of that of (the specification) P. For instance, traces refinement $P \sqsubseteq_T Q$ requires $traces[[Q]] \subseteq traces[[P]]$. This approach is used to define input-output failures refinement $\sqsubseteq_{IOF}$ [12].

As mentioned, one of our goals is to study the relationship between refinement and ioco, which we discuss next.

2.2 IOLTS and IOTS

An input-output labelled transition system (IOLTS) M is a labelled transition system in which the set of events is partitioned into inputs and outputs [40]. An IOLTS is thus a tuple (I, O, Q, q_0, h) in which I is the set of input events, O is the set of output events, Q is the set of states, $q_0 \in Q$ is the initial state, and h is the transition relation of type $Q \times (I \cup O \cup \{\tau\}) \times Q$, in which the extra event τ represents a silent (internal) event. I and O are required to be disjoint.

A tuple $(q_1, e, q_2) \in h$ is called a *transition*. If $e \in I \cup O$, then the transition (q_1, e, q_2) denotes that M can move from the state q_1 to the state q_2 via the event e. If M has a transition (q_1, τ, q_2), then M can move from q_1 to q_2 without any event being observed. Typically, the name of an input starts with '?' and that of an output starts with '!'.

From a testing perspective, the IOLTS approach has the advantage (over labelled transition systems) of distinguishing between inputs and outputs. As said, this is important since inputs and outputs play different roles in testing. There has thus been significant interest in testing from an IOLTS [5, 37–42]. It is typically assumed that the tester provides inputs, the implementation produces outputs, the tester cannot block outputs, and the implementation cannot block inputs.

During testing, a tester observes inputs and outputs and is normally also allowed to observe quiescence: in a state where the SUT cannot produce an output or change state without first receiving an input. Quiescence is usually represented by δ, where $\delta \notin I \cup O \cup \{\tau\}$. A state q_1 is quiescent if, for all $e \in I \cup O \cup \{\tau\}$ and $q_2 \in Q$, we have that $(q_1, e, q_2) \in h$ implies that $e \in I$. Quiescence is the only type of refusal that the tester is allowed to observe and in practice quiescence is observed through the use of a timeout. There is a test hypothesis stating that if a given time t_δ passes without output being observed then the implementation must be quiescent. Observations made in testing are sequences of events in $I \cup O \cup \{\delta\}$ and these are called suspension traces.

To define the set of suspension traces of an IOLTS $M = (I, O, Q, q_0, h)$, we can first extend M to a model M_δ by adding self-loops denoting quiescence, and then consider the paths through M_δ. To devise M_δ, it is sufficient to add a transition (q, δ, q) for each quiescent state q of M. We then say that $\sigma \in (I \cup O \cup \{\delta\})^*$ is a suspension trace of M if there is some sequence $\langle (q_0, e_1, q_1), (q_1, e_2, q_2), \ldots, (q_{n-1}, e_n, q_n)\rangle$ of consecutive transitions of M such that σ is the sequence formed from $\langle e_1, \ldots, e_n\rangle$ by removing all instances of τ. We write $L(M)$ for the set of suspension traces of IOLTS M.

The observation of quiescence makes it possible to distinguish between some IOLTSs that have the same set of traces.

Example 2.5. We consider IOLTSs M_1 and M_2 defined as follows. After an input $?i$, M_1 moves to a state from which it produces an output $!o$ and deadlocks, while, after $?i$, M_2 either moves to a state q_1 in which it deadlocks or moves to a state q_2 from which it produces $!o$ and then deadlocks. M_1 and M_2 have the same set of traces; this is $\{\langle\rangle, \langle ?i\rangle, \langle ?i, !o\rangle\}$. However, $\langle ?i, \delta\rangle$ is a suspension trace of M_2, but not of M_1. □

An IOLTS $M = (I, O, Q, q_0, h)$ is said to be input enabled if, for all $q_1 \in Q$ and $?i \in I$, there exists some q_1' and q_2 in Q such that $(q_1', ?i, q_2) \in h$, and q_1' can be reached from q_1 via transitions with event τ. In particular, q_1 can be reached from q_1 itself. Much of the work on testing from an IOLTS assumes that the implementation is input-enabled. If an IOLTS is input-enabled then it is an input-output transition system (IOTS) [40].

The testing process is thus seen as testing an implementation that behaves like an unknown IOTS N that has the same input and output sets as the specification IOLTS M. It is usual to assume that these processes are divergence free since, in testing, we cannot distinguish between divergence and deadlock.

Most approaches to testing from an IOLTS use the ioco conformance relation [38–40] or a variant of it. The ioco relation requires that, if a suspension trace σ of the specification M occurs in testing the implementation, and is followed by an event e that is either an output or quiescence, then the specification must be able to perform σ followed by e.

The definition of ioco uses two pieces of notation. First, given an IOLTS M and suspension trace σ we define M *after* σ as the IOLTS whose suspension traces are those that can be produced by M after σ has been observed. Thus, σ_1 is a suspension trace of M *after* σ if, and only if, $\sigma \frown \sigma_1$ is a suspension trace of M. The other piece of notation is $Out(M)$, which denotes the set of observations from $O \cup \{\delta\}$ that M can initially perform. Thus, $e \in O \cup \{\delta\}$ is in $Out(M)$ if, and only if, $\langle e\rangle$ is a suspension trace of M. We then obtain the following definition of ioco [40].

Definition 2.6. Given an IOLTS M and an IOTS N with the same sets of inputs and outputs, we have that N conforms to M under ioco if, and only if, for all $\sigma \in L(M)$ we have that $Out(N \text{ after } \sigma) \subseteq Out(M \text{ after } \sigma)$.

Thus, an implementation N fails according to ioco if, after a trace σ of the specification M, N produces an observation o (that is, an output or quiescence) that is not allowed by M. In contrast, if the tester applies an input $?i$ after the observation of a suspension trace σ and $\sigma \frown \langle ?i\rangle$ is not a suspension trace of M, then N is allowed to do anything.

Example 2.7. We consider again M_1 and M_2 of Example 2.5. It is straightforward to see that every suspension trace of M_1 is also a suspension trace of M_2, and so M_1 conforms to M_2 under ioco. In contrast, if we consider the suspension trace $\sigma = \langle ?i\rangle$, we find that M_2 can be quiescent after σ, but M_1 cannot. Thus, M_2 does not conform to M_1 under ioco. □

Suspension-trace inclusion is a sufficient, but not necessary condition for ioco to hold.

In comparing refinement and ioco, we consider the suspension traces of a process, which we derive from its refusal traces. These are considered and defined in the next few sections.

3 THE $\mathcal{RT}$ MODEL: HEALTHINESS CONDITIONS, CSP OPERATORS, AND REFINEMENT

As already indicated, the basis of our work is the $\mathcal{RT}$ model, which we adapt in the next section to differentiate input and output events. Here, we revisit that model to formalise its traces and healthiness conditions, the refusal-traces semantics of CSP processes, and refusal-traces refinement.

In $\mathcal{RT}$, processes are modelled by sets of refusal traces, which record iterative experiments that proceed as follows in each step. The environment offers a set X of events, and X is recorded in the refusal trace if a deadlock is observed; in that case, the environment offers an event $e \notin X$, and e is recorded in the trace if it occurs. In a refusal trace, $\circ$, read null, records a situation where a refusal set has not been observed: $\circ$ is the least information that an experiment provides.

(We note that, in the literature, $\bullet$ is normally used, instead of $\circ$, for the null refusal. Here, in definitions we use the mathematical notation of Z [45]. So, we avoid use of $\bullet$ for the null refusal, because it is used as a separator in Z. In proofs, here and in [10], we capitalise on the use of a precise notation by resorting to the laws of Z.)

If a deadlock is observed after the environment offers a set X of events, a deadlock is also observed if any subset of X is offered instead. In particular, if the environment offers the empty set $\emptyset$ of events, we can expect a deadlock. We have to note, however, that if the process is unstable, due to divergence or available internal choices, for example, then we cannot observe a refusal set. So, the only record that we can make is $\circ$. Not even $\emptyset$ can be recorded in the trace.

In his account of $\mathcal{RT}$, Roscoe [34] defines the following two forms of refusal traces: $\langle X_0, e_1, X_1, \ldots, e_n, X_n \rangle$ and $\langle X_0, e_1, X_1, \ldots, e_n, \circ, \checkmark \rangle$, where the e_i are events and each X_i is either a standard refusal set or $\circ$. In these traces, none of the e_i can be the special event $\checkmark$, which is also not included in any refusal set X_i.

Roscoe points out alternatives regarding the treatment of termination [34, pg. 260]. We can include a refusal set after the final $\checkmark$; here any set of events is refused. We can also allow any set of events (that does not include $\checkmark$) to be refused before termination, and thus for refusal sets to be observed immediately before $\checkmark$.

Roscoe says that these choices are relatively arbitrary and a matter of taste. Recording a refusal set after the $\checkmark$ event has the advantage that all refusal traces are of the same form: they alternate between events and refusal sets (or $\circ$) and begin and end with refusal sets. This approach to termination had been partially adopted already in [29], where a complete treatment of a refusal testing model, including the definition of the CSP operators, is presented in detail. The model in [29], however, does not cater for divergence (div), and instead treats the most nondeterministic process (that is, *CHAOS*, the process that can terminate, deadlock, accept, or refuse any event at all points) as divergent. Some fundamental properties of processes, like P; $SKIP = P$, for instance, do not hold for the model in [29].

As usual for IOLTS, we assume that both specifications and systems are divergence free. (In specifications, divergence is regarded as a mistake, and, as mentioned, when testing an SUT, divergence cannot be distinguished from deadlock, and so they are identified.) Even so, it is important to have a semantic model compatible with the standard models of CSP; this facilitates the understanding of our results in the context of those models as well as the preservation of laws and properties. In the stable-failures model of CSP, the divergent process div is modelled by the empty set of failures.

Here, we define a refusal-testing model that is similar to that of [29], but follows the treatment of termination and divergence in [34]. We define refusal traces in Section 3.1, consider healthiness conditions in Section 3.2, revisit the definitions of the CSP operators in [29] in Section 3.3, and study refusal-traces refinement in Section 3.4.

3.1 Refusal traces

We adopt the definition below for the set *RTrace* of refusal traces, which corresponds directly to a similar notion in [29]. The symbol == is used in Z abbreviations, to define sets, for instance, as shown below. In CSP, as already illustrated, we

use = to define processes. Finally, later on, we use $\widehat{=}$ when we introduce a new concept, like refinement, for example. In those cases, we use the Z mathematical notation in the definitions, but we do not use Z to introduce (relational) operators. (This would require a deep embedding of CSP in Z, which we avoid for readability.)

We note that, in Z, quantifications are written $\forall\, x : T \mid P(x) \bullet Q(x)$ and $\exists\, x : T \mid P(x) \bullet Q(x)$ to mean 'for all x of type T, $P(x)$ implies $Q(x)$', and 'there exists x of type T such that $P(x)$ and $Q(x)$'. As usual, we can quantify over more than one variable, and the constraint $P(x)$ can be omitted if it is just *true*. Operators available in the Z mathematical toolkit and new operators used in Definition 3.1 are explained afterwards.

In contrast to [29], where the notion of traces is discussed informally, in Definition 3.1 we formalise traces as sequences of elements of a free type *Obs* of observations with constructors *event*, ∘, and *refusal*. We consider ∘ as a special refusal set, and define *Refusal* as a subset of *Obs* containing just ∘ and elements constructed using *refusal*.

Definition 3.1 (Set RTrace of refusal traces).

$$
\begin{array}{lll}
RTrace == & \{\phi : \mathrm{seq}\ Obs \mid & odd\ \#\phi \wedge & (1)\\
 & & \mathrm{ran}(\{i : 1 \mathinner{..} \#\phi \mid odd\ i\} \upharpoonright \phi) \subseteq Refusal \wedge & (2)\\
 & & \mathrm{ran}(\{i : 1 \mathinner{..} \#\phi \mid even\ i\} \upharpoonright \phi) \subseteq \mathrm{ran}\ event \wedge & (3)\\
 & & \forall\, i : 1 \mathinner{..} \#\phi - 2 \mid odd\ i \bullet \phi\ (i+1) \notin_{RT} \phi\ i & (4)\\
 & \} & &
\end{array}
$$

where $Obs ::= event\langle\!\langle \Sigma^{\checkmark} \rangle\!\rangle \mid \circ \mid refusal\langle\!\langle \mathbb{P}\, \Sigma^{\checkmark} \rangle\!\rangle$ and $Refusal == \{\circ\} \cup \mathrm{ran}\ refusal$.

RTrace includes the sequences of observations whose size is an odd number (restriction (1) in Definition 3.1), because a refusal trace includes events and refusals in alternation (restrictions (2) and (3)), but starts and finishes with a refusal. Because of (1), the empty sequence is not a refusal trace; this is in accordance with the intuition that at least ∘ can always be recorded. Finally, (4) establishes that we cannot observe an event e immediately after a refusal that contains e.

The definitions of *odd* and *even* are as expected. These operators and others used here are defined in Appendix A. The sequence operator $I \upharpoonright s$ defines a sequence formed by the elements of s in the positions contained in the set I of indices, in the order in which they appear in s. The function range operator is written ran in Z; it is here applied to a sequence, which is modelled in Z as a function from indices to elements. So, we use ran to specify the set of elements of the sequence. The subscripted operator $\notin_{RT}$ is a version of $\notin$ for elements of *Obs*. It is defined in terms of the $\in_{RT}$ operator (in Appendix A). Obviously, $e \notin_{RT} X$ if, and only if, $\neg\ (e \in_{RT} X)$, and $e \notin_{RT} \circ$ for every e.

Example 3.2. We sketch below the set of refusal traces of the following process *EC*.

$$EC = inA?x \rightarrow STOP \ \Box\ inB?x \rightarrow (inA?x \rightarrow outA!1 \rightarrow STOP \ \Box\ outB!1 \rightarrow outA!1 \rightarrow STOP)$$

For conciseness, in examples, we omit the application of the constructors of *Obs*, and write $\langle\ \circ, a, \emptyset\rangle$ instead of $\langle\ \circ, event\ a, refusal\ \emptyset\rangle$, for instance. Additionally, in characterising below the refusal traces, if the value x communicated in an event $c.x$ does not matter, we write $c?x$. For example, $\langle \{\!|outA, outB, \checkmark|\!\}, inA?x, \{\!|inA, inB, outA, outB, \checkmark|\!\}\rangle$ represents a collection of refusal traces: one for each of the possible values of x in $inA.x$. Since the sets of traces are typically infinite, we cannot list all elements and normally, but not always, avoid inclusion of prefixes of traces already listed.

The notion of prefix for refusal traces is made precise in the sequel.

$$
\begin{array}{l}
\{\ \langle \{\!|\mathit{outA}, \mathit{outB}, \checkmark|\!\}, \mathit{inA}?x, \{\!|\mathit{inA}, \mathit{inB}, \mathit{outA}, \mathit{outB}, \checkmark|\!\}\rangle, \\
\quad \langle \{\!|\mathit{outA}, \mathit{outB}, \checkmark|\!\}, \mathit{inB}?x, \{\!|\mathit{inB}, \mathit{outA}, \mathit{outB}?y : y \neq 1, \checkmark|\!\}\rangle, \ldots, \\
\quad \langle \{\!|\mathit{outA}, \mathit{outB}, \checkmark|\!\}, \mathit{inB}?x, \{\!|\mathit{inB}, \mathit{outA}, \mathit{outB}?y : y \neq 1, \checkmark|\!\}, \mathit{inA}?x, \{\!|\mathit{inA}, \mathit{inB}, \mathit{outA}?y : y \neq 1, \mathit{outB}, \checkmark|\!\}\rangle, \\
\quad \langle\ \{\!|\mathit{outA}, \mathit{outB}, \checkmark|\!\}, \mathit{inB}?x, \{\!|\mathit{inB}, \mathit{outA}, \mathit{outB}?y : y \neq 1, \checkmark|\!\}, \\
\quad\quad \mathit{inA}?x, \{\!|\mathit{inA}, \mathit{inB}, \mathit{outA}?y : y \neq 1, \mathit{outB}, \checkmark|\!\}, \mathit{outA}.1, \{\!|\mathit{inA}, \mathit{inB}, \mathit{outA}, \mathit{outB}, \checkmark|\!\}\rangle, \ldots, \\
\quad \langle\ \{\!|\mathit{outA}, \mathit{outB}, \checkmark|\!\}, \mathit{inB}?x, \{\!|\mathit{inB}, \mathit{outA}, \mathit{outB}?y : y \neq 1, \checkmark|\!\}, \\
\quad\quad \mathit{outB}.1, \{\!|\mathit{inA}, \mathit{inB}, \mathit{outA}?y : y \neq 1, \mathit{outB}, \checkmark|\!\}, \mathit{outA}.1, \{\!|\mathit{inA}, \mathit{inB}, \mathit{outA}, \mathit{outB}, \checkmark|\!\}\rangle, \ldots\}
\end{array}
$$

In spite of our convention, inputs and outputs are not distinguished in the traces. □

Above, we omit refusal traces of *EC* that we can infer to be included in the set from those already listed. For instance, because of $\langle \{\!|\mathit{outA}, \mathit{outB}, \checkmark|\!\}, \mathit{inA}?x, \{\!|\mathit{inA}, \mathit{inB}, \mathit{outA}, \mathit{outB}, \checkmark|\!\}\rangle$, we know that $\langle \{\!|\mathit{outA}, \mathit{outB}, \checkmark|\!\}, \mathit{inA}?x, \emptyset\rangle$ and $\langle\ \circ, \mathit{inA}?x, \circ\ \rangle$ are in the set of traces of *EC*. The healthiness condition that justifies this inference, and a few others, are studied next.

3.2 Healthiness conditions

As said, in $\mathcal{RT}$, a process is represented by a set *RT* of refusal traces. Not all such sets, however, characterise a process: [29] and [34] propose healthiness conditions that these sets must satisfy. We consider both proposals, and adopt the simpler conditions in [34], plus extra conditions to cater for termination and to rule out the empty set.

According to [29], a set *RT* of refusal traces represents a CSP process if, and only if, it satisfies the healthiness conditions in Table 1. There, and in the sequel, we use a variable ϕ to represent sequences of observations, ρ to represent sequences of observations in *RTrace*, X to represent elements of *Refusal*, Y to represent sets of observations in ran *event*, and e to represent an observation in ran *event*. In examples, we use a, b, c, and so on, as particular events of a process or arbitrary elements of Σ. We use subscripts when we need more variables: ϕ_1, ϕ_2, and so on, for example.

MRT0 indicates that the lack of information represented by $\circ$ can be recorded for every process. This ensures that the empty set of refusal traces is not healthy. MRT1 is a related condition that states that if the empty set of refusals *refusal* $\emptyset$ is observed, then $\circ$ can as well. This ensures that, if a process is stable, and so some observation of a refusal can be made, at least the empty set, then the *null* refusal, which is always possible, is also recorded.

MRT2 is related to the treatment of divergence and not considered here. The model in [29] is divergence strict: when divergence is observed, every behaviour becomes possible. Our treatment of divergence is different.

MRT3 is a refusal-trace version of the prefix-closure condition of the traces model. In $\mathcal{RT}$ we cannot just use sequence prefix because a refusal trace that finishes with an event is not well formed. MRT3 therefore considers refusal traces that finish with a pair $\langle e, X\rangle$, and requires the sequence ρ leading up to it to be included in the set of traces. We note that ρ cannot possibly be empty, because every refusal trace starts with a refusal.

MRT4 is a refusal-trace version of the subset-closure condition of the failures model. The $\subseteq_{RT}$ operator is subset inclusion for refusals different from $\circ$. We note that inclusion of $\circ$ refusals is ensured by MRT0 and MRT1. Here, we ensure that if a process is recorded to deadlock when the environment offers events in a set X_1, then it is also recorded to deadlock when potentially fewer events in a subset X_2 are offered.

Finally, MRT5 is the refusal-trace version of the condition that states that an event that is not accepted is refused; $\cup_{RT}$ is union for refusal sets. For convenience, $\cup_{RT}$ applies to an element of *Refusal* and a set of elements in ran *event*. So, MRT5 states that if a set X of events is refused, and no trace records that, after X, an event e can take place, then the refusal obtained by adding e to X is recorded as well. The observation of a refusal after e is unconstrained: just $\circ$.

Table 1. $\mathcal{RT}$ model (Mukaran): healthiness conditions

MRT0	$\langle \circ \rangle \in RT$
MRT1	$\phi_1 \frown \langle \mathit{refusal}\, \emptyset \rangle \frown \phi_2 \in RT \Rightarrow \phi_1 \frown \langle \circ \rangle \frown \phi_2 \in RT$
MRT2	$\phi \frown \langle \circ \rangle \in RT \Rightarrow \phi \frown \langle \mathit{refusal}\, \emptyset \rangle \in RT$
MRT3	$\rho \frown \langle e, X \rangle \in RT \Rightarrow \rho \in RT$
MRT4	$\phi_1 \frown \langle X_1 \rangle \frown \phi_2 \in RT \land X_2 \subseteq_{RT} X_1 \Rightarrow \phi_1 \frown \langle X_2 \rangle \frown \phi_2 \in RT$
MRT5	$X \in \mathrm{ran}\, \mathit{refusal} \land \phi_1 \frown \langle X \rangle \frown \phi_2 \in RT \land \phi_1 \frown \langle X, e, \circ \rangle \notin RT \Rightarrow \phi_1 \frown \langle X \cup_{RT} \{e\} \rangle \frown \phi_2 \in RT$

Table 2. $\mathcal{RT}$ model (Roscoe): healthiness conditions

RT1	$\rho_2 \in RT \land \rho_1 \leq_{RT} \rho_2 \Rightarrow \rho_1 \in RT$
RT2	$\phi_1 \frown \langle X \rangle \frown \phi_2 \in RT \land X \neq \circ \land (\forall e : Y \bullet \phi_1 \frown \langle X, e, \circ \rangle \notin RT) \Rightarrow \phi_1 \frown \langle X \cup_{RT} Y \rangle \frown \phi_2 \in RT$

The set of healthiness conditions in [34] is presented in Table 2. RT1 establishes a sort of prefix-closure. The operator $\rho_1 \leq_{RT} \rho_2$, formalised below, defines prefixing for refusal traces ρ_1 and ρ_2.

$$
\begin{array}{l}
\_ \leq_{RT} \_ : \mathit{seq}\ \mathit{Observation} \leftrightarrow \mathit{seq}\ \mathit{Observation} \\
\hline
\forall\, \rho_1, \rho_2 : \mathit{seq}\ \mathit{Observation} \bullet \rho_1 \leq_{RT} \rho_2 \Leftrightarrow \\
\quad \rho_1 \in \mathit{RTrace} \land \rho_2 \in \mathit{RTrace} \land \#\rho_1 \leq \#\rho_2 \land \\
\quad \forall\, i : 1 \mathinner{..} \#\rho_1 \bullet (\mathit{odd}\ i \Rightarrow \rho_1\ i = \circ \lor \rho_1\ i \subseteq_{RT} \rho_2\ i) \land (\mathit{even}\ i \Rightarrow \rho_1\ i = \rho_2\ i)
\end{array}
$$

Informally, ρ_1 corresponds to an initial subsequence of ρ_2, but its refusal sets are subsets of those in ρ_2 or $\circ$. For instance, $\langle \circ \rangle$, $\langle \circ, a, \emptyset \rangle$ and $\langle \{b\}, a, \{a\} \rangle$ are prefixes of $\langle \{b\}, a, \{a, b\} \rangle$. Above, we require that the size of ρ_1 is at most that of ρ_2. Moreover, for all elements of ρ_1, if it is a refusal (that is, it is in an odd position), then it is either $\circ$ or a subset of the refusal in ρ_2 in the same position. If it is an event (in an even position), then it must be the same event in ρ_2. RT2 states that sets of events that are not accepted must be refused.

In comparing Tables 1 and 2, we have the following results.

Theorem 3.3. *If RT is* RT1*-healthy, then it satisfies all of* MRT1*,* MRT3 *and* MRT4*.*

Proof.

MRT1.

$$
\begin{array}{ll}
\phi_1 \frown \langle \mathit{refusal}\, \emptyset \rangle \frown \phi_2 \in RT & \\
= \phi_1 \frown \langle \mathit{refusal}\, \emptyset \rangle \frown \phi_2 \in RT \land \phi_1 \frown \langle \circ \rangle \frown \phi_2 \leq_{RT} \phi_1 \frown \langle \mathit{refusal}\, \emptyset \rangle \frown \phi_2 & \text{[definition of } \leq_{RT}\text{]} \\
\Rightarrow \phi_1 \frown \langle \circ \rangle \frown \phi_2 \in RT & \text{[RT1]}
\end{array}
$$

MRT3.

$$
\begin{array}{ll}
\rho \frown \langle e, X \rangle \in RT & \\
= \rho \frown \langle e, X \rangle \in RT \land \rho \leq_{RT} \rho \frown \langle e, X \rangle & \text{[definition of } \leq_{RT}\text{]} \\
\Rightarrow \rho \in RT & \text{[RT1]}
\end{array}
$$

MRT4.

$\phi_1 \frown \langle X_1 \rangle \frown \phi_2 \in RT \land X_2 \subseteq_{RT} X_1$

$\Rightarrow \phi_1 \frown \langle X_1 \rangle \frown \phi_2 \in RT \land \phi_1 \frown \langle X_2 \rangle \frown \phi_2 \leq_{RT} \phi_1 \frown \langle X_1 \rangle \frown \phi_2$ [definition of $\leq_{RT}$]

$\Rightarrow \phi_1 \frown \langle X_2 \rangle \frown \phi_2 \in RT$ [RT1]

□

Theorem 3.4. *If RT satisfies all of* MRT0, MRT1, MRT3 *and* MRT4, *then it is* RT1*-healthy.*

Proof. By induction on ρ_1.

Case $\langle \circ \rangle$. $\langle \circ \rangle \in RT$ by MRT0

Case $\langle X_1 \rangle$, *with* $X_1 \neq \circ$.

$\rho_2 \in RT \land \langle X_1 \rangle \leq_{RT} \rho_2$

$\Rightarrow \exists X_2 : Refusal;\ \phi_2 : seq\ Obs \bullet \rho_2 = \langle X_2 \rangle \frown \phi_2 \land \langle X_2 \rangle \frown \phi_2 \in RT \land X_1 \subseteq_{RT} X_2$ [property of $\leq_{RT}$]

$\Rightarrow \langle X_1 \rangle \frown \phi_2 \in RT$ [MRT4]

$\Rightarrow \langle X_1 \rangle \in RT$ [[10, Lemma C.1]]

Case $\rho_1 \frown \langle e_1, \circ \rangle$.

$\rho_2 \in RT \land \rho_1 \frown \langle e_1, \circ \rangle \leq_{RT} \rho_2$

$\Rightarrow \exists \rho_3 : RTrace;\ X_3 : Refusal;\ \phi_3 : seq\ Obs \bullet$ [property of $\leq_{RT}$]
$\rho_2 = \rho_3 \frown \langle e_1, X_3 \rangle \frown \phi_3 \land \rho_3 \frown \langle e_1, X_3 \rangle \frown \phi_3 \in RT \land \rho_1 \leq_{RT} \rho_3 \land \#\rho_1 = \#\rho_3$

$\Rightarrow \exists X_3 : Refusal \bullet \rho_1 \frown \langle e_1, X_3 \rangle \frown \phi_3 \in RT$ [[10, Lemma C.2]]

$\Rightarrow \rho_1 \frown \langle e_1, refusal\ \emptyset \rangle \frown \phi_3 \in RT$ [MRT4]

$\Rightarrow \rho_1 \frown \langle e_1, \circ \rangle \frown \phi_3 \in RT$ [MRT1]

$\Rightarrow \rho_1 \frown \langle e_1, \circ \rangle \in RT$ [[10, Lemma C.1]]

Case $\rho_1 \frown \langle e_1, X_1 \rangle$, *with* $X_1 \neq \circ$.

$\rho_2 \in RT \land \rho_1 \frown \langle e_1, X_1 \rangle \leq_{RT} \rho_2$

$\Rightarrow \exists \rho_3 : RTrace;\ X_3 : Refusal;\ \phi_3 : seq\ Obs \bullet$ [property of $\leq_{RT}$]
$\rho_2 = \rho_3 \frown \langle e_1, X_3 \rangle \frown \phi_3 \land \rho_3 \frown \langle e_1, X_3 \rangle \frown \phi_3 \in RT \land \rho_1 \leq_{RT} \rho_3 \land \#\rho_1 = \#\rho_3 \land X_1 \subseteq_{RT} X_3$

$\Rightarrow \exists X_3 : Refusal \bullet \rho_1 \frown \langle e_1, X_3 \rangle \frown \phi_3 \in RT \land X_1 \subseteq_{RT} X_3$ [[10, Lemma C.2]]

$\Rightarrow \rho_1 \frown \langle e_1, X_1 \rangle \frown \phi_3 \in RT$ [MRT4]

$\Rightarrow \rho_1 \frown \langle e_1, X_1 \rangle \in RT$ [[10, Lemma C.1]]

□

Theorem 3.5. *If RT is* RT1*-healthy, then it satisfies* RT2 *if, and only if, it satisfies* MRT5.

Proof. First, we prove that MRT5 holds if RT2 holds.

$$X \in \mathit{ran\ refusal} \land \phi_1 \frown \langle X \rangle \frown \phi_2 \in RT \land \phi_1 \frown \langle X, e, \circ \rangle \notin RT$$

$$\Rightarrow \phi_1 \frown \langle X \rangle \frown \phi_2 \in RT \land X \neq \circ \land \phi_1 \frown \langle X, e, \circ \rangle \notin RT \qquad \text{[definition of } \mathit{Refusal}\text{]}$$

$$\Rightarrow \phi_1 \frown \langle X \cup_{RT} \{e\} \rangle \frown \phi_2 \in RT \qquad \text{[RT2 with } \{e\} \text{ for } Y\text{]}$$

We now prove that RT2 holds if MRT5 holds, by induction on the cardinality of Y.

Base case: $\#Y = 0$.

$$\phi_1 \frown \langle X \rangle \frown \phi_2 \in RT \land X \neq \circ \land \forall e : \emptyset \bullet \phi_1 \frown \langle X, e, \circ \rangle \notin RT$$

$$\Rightarrow \phi_1 \frown \langle X \rangle \frown \phi_2 \in RT \qquad \text{[predicate calculus]}$$

$$= \phi_1 \frown \langle X \cup_{RT} \emptyset \rangle \frown \phi_2 \in RT \qquad \text{[property of } \cup_{RT}\text{]}$$

Inductive case: $\#Y > 0$. The induction hypothesis is

$$\phi_1 \frown \langle X \rangle \frown \phi_2 \in RT \land X \neq \circ \land \forall e_1 : Y \bullet \phi_1 \frown \langle X, e_1, \circ \rangle \notin RT \Rightarrow \phi_1 \frown \langle X \cup_{RT} Y \rangle \frown \phi_2 \in RT$$

We now prove that the result for $Y \cup \{e_2\}$.

$$\phi_1 \frown \langle X \rangle \frown \phi_2 \in RT \land X \neq \circ \land \forall e_1 : Y \cup \{e_2\} \bullet \phi_1 \frown \langle X, e_1, \circ \rangle \notin RT$$

$$= \phi_1 \frown \langle X \rangle \frown \phi_2 \in RT \land X \in \mathit{ran\ refusal} \land \forall e_1 : Y \cup \{e_2\} \bullet \phi_1 \frown \langle X, e_1, \circ \rangle \notin RT \qquad [X \neq \circ]$$

$$= \phi_1 \frown \langle X \rangle \frown \phi_2 \in RT \land X \in \mathit{ran\ refusal} \land \forall e_1 : Y \bullet \phi_1 \frown \langle X, e_1, \circ \rangle \notin RT \land$$
$$\phi_1 \frown \langle X \rangle \frown \phi_2 \in RT \land X \in \mathit{ran\ refusal} \land \phi_1 \frown \langle X, e_2, \circ \rangle \notin RT \qquad \text{[property of sets and predicate calculus]}$$

$$\Rightarrow \phi_1 \frown \langle X \cup_{RT} Y \rangle \frown \phi_2 \in RT \land \phi_1 \frown \langle X \rangle \frown \phi_2 \in RT \land X \in \mathit{ran\ refusal} \land \phi_1 \frown \langle X, e_2, \circ \rangle \notin RT$$
[induction hypothesis]

$$\Rightarrow \quad \phi_1 \frown \langle X \cup_{RT} Y \rangle \frown \phi_2 \in RT \land \phi_1 \frown \langle X \rangle \frown \phi_2 \in RT \land X \in \mathit{ran\ refusal} \land \phi_1 \frown \langle X \cup_{RT} Y, e_2, \circ \rangle \notin RT$$
[by RT1 and $\phi_1 \frown \langle X, e_2, \circ \rangle \leq_{RT} \phi_1 \frown \langle X \cup_{RT} Y, e_2, \circ \rangle$]

$$\Rightarrow \phi_1 \frown \langle (X \cup_{RT} Y) \cup_{RT} \{e_2\} \rangle \frown \phi_2 \in RT \qquad \text{[MRT5]}$$

$$\Rightarrow \phi_1 \frown \langle X \cup_{RT} (Y \cup_{RT} \{e_2\}) \rangle \frown \phi_2 \in RT \qquad \text{[property of } \cup_{RT}\text{]}$$

□

With these results, we conclude that RT1 and RT2 can be adopted, instead of, MRT1, MRT3, MRT4 and MRT5. We need MRT0 to ensure that the empty set of observations is not a proper model for a process. As said, we disregard MRT2 because it is used by [29] to deal with divergence, and we do not adopt the notion of divergence proposed there, but that in the standard stable-failures model of CSP. In particular, we define div as the process that has the single trace $\langle \circ \rangle$.

Table 3. $\mathcal{RT}$ model: healthiness conditions - termination

RT3 $\quad \rho \frown \langle e, X\rangle \in RT \Rightarrow event(\checkmark) \notin \operatorname{ran}\rho$
RT4 $\quad \phi \frown \langle X_1, event(\checkmark), X_2\rangle \in RT \Rightarrow X_1 =_\circ \wedge\ \phi \frown \langle\, \circ, event(\checkmark), refusal(\Sigma^\checkmark)\rangle \in RT$

Table 4. *rtraces* model of CSP processes

Process P	$rtraces[[P]]$
div	$\{\langle\, \circ\, \rangle\}$
STOP	$\{\, X : Refusal \bullet \langle X\rangle\,\}$
SKIP	$\{\langle\, \circ\, \rangle\} \cup \{X : Refusal \bullet \langle\, \circ, event(\checkmark), X\rangle\}$
$a \rightarrow P$	$\{X : Refusal \mid event(a) \notin_{RT} X \bullet \langle X\rangle\} \cup$ $\{X : Refusal;\ \rho : rtraces[[P]] \mid event(a) \notin_{RT} X \bullet \langle X, event(a)\rangle \frown \rho\,\}$
$P \sqcap Q$	$rtraces[[P]] \cup rtraces[[Q]]$
$P \,\Box\, Q$	$\{\rho : rtraces[[P]] \cup rtraces[[Q]] \mid \langle \rho\, 1\rangle \in rtraces[[P]] \cap rtraces[[Q]]\}$
$P;\ Q$	$\{\rho : rtraces[[P]] \mid event(\checkmark) \notin ran\,\rho\} \cup$ $\{\phi : \text{seq}\ Obs;\ \rho : RTrace \mid \phi \frown \langle\, \circ, event(\checkmark), \circ\,\rangle \in rtraces[[P]] \wedge \rho \in rtraces[[Q]] \bullet \phi \frown \rho\}$
$P \,[\![Z]\!]\, Q$	$\bigcup\{\rho_1 : rtraces[[P]];\ \rho_2 : rtraces[[Q]] \bullet (\rho_1 \,[\![Z]\!]_{RT}\, \rho_2)\}$
$P \setminus Z$	$\bigcup((\!|\ rtraces[[P]]\ |\!) \setminus_{RT} Z)$
$P[[R]]$	$\{\rho_1 : RTrace \mid \exists\, \rho_2 : rtraces[[P]] \bullet \rho_1\ R_{RT}\ \rho_2\}$

In addition, since *RTrace* imposes no restrictions on the use of the ✓ event, we also adopt the healthiness conditions in Table 3. With RT3, we establish that the termination event ✓ always appears as the last event in the refusal traces of *RT*, if at all. With RT4, we establish that if the event ✓ does appear, then no set of events is observed beforehand as a refusal, since termination introduces instability. Moreover, after the ✓, the whole set of events, and, therefore, because of RT1, any of its subsets, can be refused. This means that, after termination, we have a stable state.

In summary, all our refusal traces have the same shape (a sequence of observations that alternate between events and refusals and that start and end with a refusal). The healthiness conditions RT3 and RT4, however, establish that ✓ can only occur as a last event, that it occurs in an unstable state, and that after termination, every refusal is possible. The healthiness conditions that we adopt are MRT0 and RT1 to RT4.

3.3 Operators

For a process P, we write $rtraces[[P]]$ for its set of refusal traces. This is always a subset of *RTrace* that satisfies the healthiness conditions MRT0 and RT1 to RT4 presented in the previous section. This is irrespective of the particular process P under consideration. In what follows, we define $rtraces[[P]]$ for all constructs of CSP presented in Section 2.1. In [10, Appendix D], we prove that, indeed, those definitions characterise healthy sets of traces.

Table 4 defines $rtraces[[P]]$ for every process P. Most of them are just a recast of those in [29], but we provide new definitions for *SKIP*, parallel composition, hiding, and renaming. Below, we explain the definitions for every construct. The only new Z operator used in sequence indexing: $\rho\, 1$ is the first element of the refusal trace ρ.

Core operators. The model for div is $\{\langle\, \circ\, \rangle\}$ as discussed previously. This is the top of the refusal-traces refinement relation $\sqsubseteq_{RT}$ defined in the next section (as well as of the stable-failures refinement relation). The bottom of $\sqsubseteq_{RT}$ is

CHAOS, the process that can nodeterministically choose to terminate, deadlock, accept, or reject any of the events in Σ.

$$CHAOS = SKIP \sqcap STOP \sqcap (\sqcap a : \Sigma \rightarrow CHAOS)$$

Recursion is handled as usual: as the least fixed point with respect to subset inclusion. We can define a special operator $\mu\ X \bullet F(X)$ to define a recursive process in terms of a function F from processes to processes, where references to the local process variable X stands for a recursive call. A straightforward generalisation for functions from vectors of processes to vectors of processes can also be used to deal with mutual recursion. As usual in CSP, however, recursion is defined by direct reference to process names in equations, like, for example, in $P = a \rightarrow P$.

For *STOP*, we have the set of singleton refusal traces $\langle X \rangle$ with arbitrary values for X: any event can be refused, as expected. The traces for a prefixing $a \rightarrow P$ record initially arbitrary refusals that do not include the event a. Longer traces, of size greater than 1, also record a, and then observations from P. These definitions are very similar to those in the standard models of CSP for the same operators. Likewise, nondeterministic choice is modelled by set union.

The model for external choice is the set of refusal traces that belong to either process in the choice, and whose first refusal set can be observed in both processes. This is a remarkably simple definition for a very subtle form of choice.

Termination. As mentioned, our treatment of termination follows [34]. Despite, we adopt a definition for a sequential composition $P;\ Q$ in [29]. There are two; we adopt that used in proofs. The traces of $P;\ Q$ include, first of all, the traces ρ of P that are not terminating, that is, those for which $event(\checkmark) \notin ran\,\rho$. In addition, we have traces obtained from terminating traces $\phi \frown \langle \circ, event(\checkmark), \circ \rangle$ of P by concatenating to ϕ a trace of Q. (The definition in [29] explicitly states that $\checkmark$ is not allowed in the middle of a trace. We do not need this restriction, since $rtraces[[P]]$ is RT4 healthy.) Below, we explain why we reject the approach in [29], present that in [34], and then explain our semantics for *SKIP*.

The refusal traces of *SKIP* are defined as follows in [29].

$$rtraces[[SKIP]] = \{X : Refusal \mid event(\checkmark) \notin_{RT} X \bullet \langle X \rangle\} \cup \{X_1, X_2 : Refusal \mid event(\checkmark) \notin_{RT} X_1 \bullet \langle X_1, event(\checkmark), X_2 \rangle\}$$

Before communicating the event $\checkmark$, the behaviour of *SKIP* is to refuse any event different from $\checkmark$. After termination, it refuses any event. We can, for example, calculate the model of $P1 = a \rightarrow STOP \,\Box\, SKIP$ for $\Sigma^{\checkmark} = \{a, \checkmark\}$ as shown below. We calculate, first, the model for $a \rightarrow STOP$. We recall that we often omit traces included due to prefix closure.

$$rtraces[[a \rightarrow STOP]]$$

$$= \{X : Refusal \mid event(a) \notin_{RT} X \bullet \langle X \rangle\} \cup \{X : Refusal;\ \rho : rtraces[[STOP]] \mid event(a) \notin_{RT} X \bullet \langle X, event(a) \rangle \frown \rho\}$$

[definition of prefixing]

$$= \{\langle \circ \rangle, \langle \emptyset \rangle, \langle \{\checkmark\} \rangle\} \cup \{X : Refusal;\ \rho : \{\langle \{a, \checkmark\} \rangle, \langle \{a\} \rangle, \langle \{\checkmark\} \rangle, \langle \emptyset \rangle, \langle \circ \rangle\} \mid event(a) \notin_{RT} X \bullet \langle X, event(a) \rangle \frown \rho\}$$

[$\Sigma = \{a\}$ and definition of $rtraces[[STOP]]$]

$$= \{\langle \circ \rangle, \langle \emptyset \rangle, \langle \{\checkmark\} \rangle\} \cup \{\langle \{\checkmark\}, a, \{a, \checkmark\} \rangle, \langle \emptyset, a, \{a, \checkmark\} \rangle, \langle \circ, a, \{a, \checkmark\} \rangle, \langle \{\checkmark\}, a, \{a\} \rangle, \langle \{\checkmark\}, a, \{\checkmark\} \rangle, \ldots\}$$

$$= \{\langle \circ \rangle, \langle \emptyset \rangle, \langle \{\checkmark\} \rangle, \langle \{\checkmark\}, a, \{a, \checkmark\} \rangle, \langle \emptyset, a, \{a, \checkmark\} \rangle, \langle \circ, a, \{a, \checkmark\} \rangle, \langle \{\checkmark\}, a, \{a\} \rangle, \langle \{\checkmark\}, a, \{\checkmark\} \rangle, \ldots\}$$

Using the above result, we can calculate the semantics of $P1$ as follows.

$$rtraces[[P1]] = rtraces[[a \rightarrow STOP \,\Box\, SKIP]] \quad \text{[definition of } P1\text{]}$$

$$= \{\rho : rtraces[[a \rightarrow STOP]] \cup rtraces[[SKIP]] \mid \langle \rho\, 1 \rangle \in rtraces[[a \rightarrow STOP]] \cap rtraces[[SKIP]]\} \quad \text{[definition of } \Box\text{]}$$

$$= \{\rho : \{\langle\circ\rangle, \langle\emptyset\rangle, \langle\{\checkmark\}\rangle, \langle\{\checkmark\}, a, \{a,\checkmark\}\rangle, \ldots, \langle\{a\}\rangle, \langle\{a\},\checkmark,\{a,\checkmark\}\rangle, \ldots\} \mid \langle\rho\,1\rangle \in \{\langle\circ\rangle, \langle\emptyset\rangle, \langle\{\checkmark\}\rangle, \langle\{\checkmark\}, a, \{a,\checkmark\}\rangle, \ldots\} \cap \{\langle\circ\rangle, \langle\emptyset\rangle, \langle\{a\}\rangle, \langle\{a\},\checkmark,\{a,\checkmark\}\rangle, \ldots\}\}$$

[above result, and definition of *rtraces*[[*SKIP*]] in [29]]

$$= \{\rho : \{\langle\circ\rangle, \langle\emptyset\rangle, \langle\{\checkmark\}\rangle, \langle\{\checkmark\}, a, \{a,\checkmark\}\rangle, \langle\{a\}\rangle, \langle\{a\},\checkmark,\{a,\checkmark\}\rangle, \ldots\} \mid \langle\rho\,1\rangle \in \{\langle\circ\rangle, \langle\emptyset\rangle\}\}$$

$$= \{\langle\emptyset\rangle, \langle\emptyset,\checkmark,\{a,\checkmark\}\rangle, \langle\emptyset, a, \{a,\checkmark\}\rangle, \ldots\}$$

The trace $\langle\emptyset, a, \{a,\checkmark\}\rangle$ indicates that the environment of $P1$ can observe stability before the communication of a, and, hence, can make a choice between the event a and termination. It is expected, however, that a terminating process and, in particular, *SKIP* is unstable, since that process may internally decide to terminate, and nothing can prevent a process from terminating. So, the fact that *SKIP* is composed in an external choice should not mean that the environment is able to prevent that process from terminating by choosing an event offered in the choice. The external choice with *SKIP* should itself be unstable, since it can be resolved by the termination of *SKIP*.

Most importantly, the law $P;\ SKIP = P$ in [34], does not hold in this setting, as we show below using our example.

$$rtraces[[P1;\ SKIP]]$$

$$= \{\rho : rtraces[[P1]] \mid event(\checkmark) \notin ran\,\rho\} \cup \{\phi : seq\ Obs;\ \rho : RTrace \mid \phi \frown \langle \circ, event(\checkmark), \circ \rangle \in rtraces[[P1]] \wedge \rho \in rtraces[[SKIP]] \bullet \phi \frown \rho\}$$

[definition of *rtraces*[[$P;\ Q$]]]

$$= \{\langle\emptyset\rangle, \langle\emptyset, a, \{a,\checkmark\}\rangle, \ldots\} \cup \{\langle\{a\}\rangle, \langle\{a\},\checkmark,\{a,\checkmark\}\rangle, \ldots\}$$

[definitions of *rtraces*[[$P1$]] and *rtraces*[[*SKIP*]]]

$$= \{\langle\emptyset\rangle, \langle\emptyset, a, \{a,\checkmark\}\rangle, \langle\{a\}\rangle, \langle\{a\},\checkmark,\{a,\checkmark\}\rangle, \ldots\}$$

We observe that $\langle\{a\}\rangle$ belongs to *rtraces*[[$P1;\ SKIP$]], but not to *rtraces*[[$P1$]], so *rtraces*[[$P1;\ SKIP$]] $\neq$ *rtraces*[[$P1$]]. This problem arises whatever the definition for sequential composition from [29] is adopted. Since this is an essential property of CSP processes, we need a different model that supports it.

The definition for *SKIP* proposed in [29], and reproduced and used above, is similar to that in the stable failures model: before $\checkmark$, any event is refused. This is adequate in the context of stable failures, since they cannot record instability. On the other hand, the model in [34] can record only the observation $\circ$ before $\checkmark$; after tick, any event is refused. This captures the expected instability before termination. We show that, by adopting this alternative definition for *SKIP*, which is depicted in Table 4, we have *SKIP* as a unit of sequential composition.

Theorem 3.6. $P;\ SKIP = P$

A proof is in Appendix B. Finally, we note that our definition of *SKIP* is healthy (see [10, Appendix D]).

Parallelism. Instead of recasting the parallel composition operators in [29], namely synchronous and alphabetised parallelism, and interleaving, we define the generalized parallel operator. As said, $P \,[\![\, Z \,]\!]\, Q$ is a composition of P and Q in parallel, synchronising on the events in the set Z. Termination occurs when both P and Q terminate. This property of distributed termination is not valid for the operators in [29], but is valid for those in [34]. For instance, $SKIP \,|||\, SKIP$ is expected to behave like *SKIP*, but this does not hold for the interleave operator defined in [29] (see [10, Lemma C.11]).

$$
\begin{array}{l}
\_ \,[\![\, Z \,]\!]_{RT}\, \_ : RTrace \times RTrace \fun \mathbb{P}(RTrace) \\
\hline
\forall\, X_1, X_2 : Refusal;\ Y : \mathbb{P}(\ran\, event);\ e_1, e_2 : \ran\ event;\ \phi : \seq\, Obs;\ \rho_1, \rho_2 : RTrace \mid \\
\quad Y = event(\!|\, Z \,|\!) \land event(\checkmark) \notin \ran\, \phi \cup \ran\, \rho_1 \cup \ran\, \rho_2 \cup \{e_1, e_2\} \bullet \\
\qquad \rho_1 \,[\![\, Z \,]\!]_{RT}\, \rho_2 = \rho_2 \,[\![\, Z \,]\!]_{RT}\, \rho_1 \\
\\
\qquad (X_1 = \circ \lor X_2 = \circ) \Rightarrow \langle X_1 \rangle \,[\![\, Z \,]\!]_{RT}\, \langle X_2 \rangle = \{\langle\, \circ \,\rangle\} \\
\\
\qquad (X_1 \neq \circ \land X_2 \neq \circ) \Rightarrow \langle X_1 \rangle \,[\![\, Z \,]\!]_{RT}\, \langle X_2 \rangle = \\
\qquad\quad \{\langle\, \circ \,\rangle\} \cup \ \{Z_1, Z_2, Z_3 : \mathbb{P}\, \Sigma^{\checkmark} \mid Z_1 = refusal^{\sim}(X_1) \land Z_2 = refusal^{\sim}(X_2) \land \\
\qquad\qquad\qquad Z_1 \setminus (Z \cup \{\checkmark\}) = Z_2 \setminus (Z \cup \{\checkmark\}) \land Z_3 \subseteq Z_1 \cup Z_2 \\
\qquad\qquad\qquad \bullet \langle refusal(Z_3) \rangle \\
\qquad\qquad\quad \} \\
\\
\qquad e_1 \in Y \Rightarrow \langle X_1, e_1 \rangle \frown \rho_1 \,[\![\, Z \,]\!]_{RT}\, \langle X_2 \rangle = \emptyset \\
\\
\qquad e_1 \notin Y \Rightarrow (\langle X_1, e_1 \rangle \frown \rho_1) \,[\![\, Z \,]\!]_{RT}\, \langle X_2 \rangle = \\
\qquad\quad \{X_3 : Refusal;\ \rho_2 : RTrace \mid \langle X_3 \rangle \in (\langle X_1 \rangle \,[\![\, Z \,]\!]_{RT}\, \langle X_2 \rangle) \land \rho_2 \in (\rho_1 \,[\![\, Z \,]\!]_{RT}\, \langle X_2 \rangle) \bullet \langle X_3, e_1 \rangle \frown \rho_2\} \\
\\
\qquad e_1 \in Y \Rightarrow \langle X_1, e_1 \rangle \frown \rho_1 \,[\![\, Z \,]\!]_{RT}\, \langle X_2, e_1 \rangle \frown \rho_2 = \\
\qquad\quad \{X_3 : Refusal;\ \rho_3 : RTrace \mid \langle X_3 \rangle \in (\langle X_1 \rangle \,[\![\, Z \,]\!]_{RT}\, \langle X_2 \rangle) \land \rho_3 \in (\rho_1 \,[\![\, Z \,]\!]_{RT}\, \rho_2) \bullet \langle X_3, e_1 \rangle \frown \rho_3\} \\
\\
\qquad \{e_1, e_2\} \subseteq Y \land e_1 \neq e_2 \Rightarrow \langle X_1, e_1 \rangle \frown \rho_1 \,[\![\, Z \,]\!]_{RT}\, \langle X_2, e_2 \rangle \frown \rho_2 = \emptyset \\
\\
\qquad e_1 \in Y \land e_2 \notin Y \Rightarrow \langle X_1, e_1 \rangle \frown \rho_1 \,[\![\, Z \,]\!]_{RT}\, \langle X_2, e_2 \rangle \frown \rho_2 = \\
\qquad\quad \{\rho_3, \rho_4 : RTrace \mid \rho_3 \in (\langle X_1 \rangle \,[\![\, Z \,]\!]_{RT}\, \langle X_2 \rangle) \land \rho_4 \in (\langle X_1, e_1 \rangle \frown \rho_1 \,[\![\, Z \,]\!]_{RT}\, \rho_2) \bullet \rho_3 \frown \langle e_2 \rangle \frown \rho_4\} \\
\\
\qquad e_1 \notin Y \land e_2 \notin Y \Rightarrow \langle X_1, e_1 \rangle \frown \rho_1 \,[\![\, Z \,]\!]_{RT}\, \langle X_2, e_2 \rangle \frown \rho_2 = \\
\qquad\quad \{X_3 : Refusal;\ \rho_3 : RTrace \mid \langle X_3 \rangle \in (\langle X_1 \rangle \,[\![\, Z \,]\!]_{RT}\, \langle X_2 \rangle) \land \rho_3 \in (\rho_1 \,[\![\, Z \,]\!]_{RT}\, \langle X_2, e_2 \rangle \frown \rho_2) \\
\qquad\qquad \bullet \langle X_3, e_1 \rangle \frown \rho_3 \\
\qquad\quad \} \cup \\
\qquad\quad \{X_3 : Refusal;\ \rho_3 : RTrace \mid \langle X_3 \rangle \in (\langle X_1 \rangle \,[\![\, Z \,]\!]_{RT}\, \langle X_2 \rangle) \land \rho_3 \in (\langle X_1, e_1 \rangle \frown \rho_1 \,[\![\, Z \,]\!]_{RT}\, \rho_2) \\
\qquad\qquad \bullet \langle X_3, e_2 \rangle \frown \rho_3 \\
\qquad\quad \} \\
\\
\qquad \phi \frown \langle X_1, event(\checkmark), X_2 \rangle \,[\![\, Z \,]\!]_{RT}\, \rho_2 = \bigcup \{X : Refusal \bullet (\phi \frown \langle X \rangle \,[\![\, Z \,]\!]_{RT}\, \rho_2)\} \\
\\
\qquad \rho_1 \frown \langle event(\checkmark), X_1 \rangle \,[\![\, Z \,]\!]_{RT}\, \rho_2 \frown \langle event(\checkmark), X_2 \rangle = \\
\qquad\quad \{\rho_3 : RTrace;\ X_3 : Refusal \mid \rho_3 \in \rho_1 \,[\![\, Z \,]\!]_{RT}\, \rho_2 \bullet \rho_3 \frown \langle event(\checkmark), X_3 \rangle\}
\end{array}
$$

Fig. 1. Parallel composition of traces

The model for $P \,[\![\, Z \,]\!]\, Q$ is obtained by composing the traces from P and Q according to the operator $[\![Z]\!]_{RT}$, defined in Figure 1. Just like in the standard CSP semantics, this operator is used to define the set of refusal traces obtained by the parallel composition of a pair of refusal traces ρ_1 and ρ_2 of the composed processes.

In defining $\rho_1 \,[\![\, Z \,]\!]_{RT}\, \rho_2$, first, we state that this operator is commutative. Proceeding, we consider the various forms that the traces ρ_1 and ρ_2 can take. First, we consider traces $\langle X_1 \rangle$ and $\langle X_2 \rangle$. We define that if stability is not observed in $\langle X_1 \rangle$ and $\langle X_2 \rangle$, then it is not observed in the traces resulting from their composition. So, the only composed trace is

$\langle \circ \rangle$. On the other hand, if both traces $\langle X_1 \rangle$ and $\langle X_2 \rangle$ record sets of events Z_1 and Z_2, then the refusal in the composed trace is defined as in the standard failures model [34, page 239]. If Z_1 and Z_2 contain only events that either are not in the synchronisation set and are refused by both processes or are in the synchronisation set and are refused by one of the processes, then the refusal of the composed trace belongs to the powerset of $Z_1 \cup Z_2$. For these purposes, ✓ is regarded as part of the synchronisation set, to ensure distributed termination.

Corresponding to the synchronisation set Z of events from Σ, we consider a synchronisation set Y of observations in the range of *event*, that is, $Y = event(\| Z \|)$. If a trace $\langle X_1, e_1 \rangle \frown \rho_1$ of P is composed with a trace $\langle X_2 \rangle$ of Q, where no more events are available, but e_1 is in Y, then the parallel composition deadlocks. In this case, there are no composed traces. If, on the other hand, e_1 is not in Y, then the composed traces include e_1, preceded by the refusals X_3 resulting from the composition of $\langle X_1 \rangle$ with $\langle X_2 \rangle$, followed by the traces that arise from the composition of ρ_1 with $\langle X_2 \rangle$. This caters for the fact that if P can proceed independently on e_1, it does, even if Q stops. The composed trace, however, caters for the fact that Q is refusing X_2 at all points: before and after e_1.

Finally, we consider the composition of $\langle X_1, e_1 \rangle \frown \rho_1$ and $\langle X_2, e_2 \rangle \frown \rho_2$. If the events are the same and in the synchronisation set, then it is carried over to the composed traces. If they are in the synchronisation set, but are different, we have a deadlock, and so no composed traces. If one of them is not in the synchronisation set, then the corresponding process can proceed. If neither of them is in the synchronisation set, then both processes can proceed. In this case, the set of composed traces is the union of those that record that P proceeds with those that record that Q proceeds.

To handle termination, we consider a trace $\phi \frown \langle X_1, event(✓), X_2 \rangle$ composed with another trace ρ_2 that does not record a ✓. In this case, there is no agreement to terminate, and so the composed traces belong to the composition of the prefix ϕ followed by an arbitrary refusal with ρ_2. When two terminating traces are composed, then ✓ is present in the composition, followed by an arbitrary refusal set. With our definition, we have distributed termination.

Theorem 3.7. $SKIP \ |[Z]| \ SKIP = SKIP$

A proof is in Appendix B. Finally, we note that our definition of parallelism is healthy (see [10, Appendix D]).

Hiding. Roughly speaking, the refusal traces of the process $P \setminus Z$ are obtained from those of P by internalising (removing) the events in Z. We note, however, that the hiding operator can introduce unstable and divergent behaviour. In [29], the set of observations for $P \setminus Z$ includes those of non-divergent behaviour (after the internalization of the events in Z) and divergence observations due to hiding. If P has a trace that leads to arbitrarily long sequences of hidden events (from Z), then $P \setminus Z$ introduces a divergence after such a trace. In this case, arbitrary extensions of the trace are allowed, to reflect the possibility of arbitrary behaviour upon divergence.

Here, as mentioned, we treat divergence in a different way, akin to that of the stable-failures model. So, we do not enrich divergent traces with arbitrary extensions. Just like stable-failures, the model for $P \setminus Z$ that we adopt, presented in Table 4, does not record divergence arising from hiding. On the other hand, instability is recorded using $\circ$.

The model for $P \setminus Z$ is obtained from that of P by considering the effect of hiding on each of its traces. For a trace ρ, this is captured by the set of traces $\rho \setminus_{RT} Z$ defined below. The model for $P \setminus Z$ is the distributed union of these sets. For generality, useful in proofs, we define $\setminus_{RT}$ for sequences of observations, and not just refusal traces. However, hiding an event potentially has an effect on the refusal that precedes it, so we do not consider events in isolation.

The empty sequence $\langle\rangle$ and $\langle \circ \rangle$ are unaffected by hiding. If instability, recorded by $\circ$, is observed, hiding does not mask it. If we observe a refusal set Z_1 that includes all the events of a set Z_2 being hidden, then the hiding introduces no instability. In this case, the refusal traces obtained from $\langle refusal(Z_1) \rangle$ are those that record as refusal any subset of

Z_1. Consequently, internal events from Z_2, in particular, can be observed in refusals. This is the same view taken in the stable-failures model of CSP, but different from that in [29], where hidden events are excluded from all observations, including refusals. By recording internal events as refused, we ensure that the model of all processes is given in the context of a single alphabet $\Sigma^\checkmark$: hiding introduces refusals, but does not change the alphabet of interest of a process. This simplifies composition of process models necessary to define operators like parallelism, choice, and so on.

$$
\begin{array}{l}
\_ \setminus_{RT} \_ : \mathrm{seq}\, Obs \times \mathbb{P}\, \Sigma^\checkmark \to \mathbb{P}\, \mathrm{seq}\, Obs \\
\hline
\forall\, Z, Z_1, Z_2 : \mathbb{P}\, \Sigma^\checkmark;\ a : \Sigma^\checkmark;\ X : Refusal;\ \phi_1, \phi_2 : \mathrm{seq}\, Obs;\ \rho : RTrace \bullet \\
\quad \langle\,\rangle \setminus_{RT} Z = \langle\,\rangle \wedge \langle\, \circ\,\rangle \setminus_{RT} Z = \{\langle\, \circ\,\rangle\} \\
\quad Z_2 \subseteq Z_1 \Rightarrow \langle refusal(Z_1)\rangle \setminus_{RT} Z_2 = \{Z : \mathbb{P}\, \Sigma^\checkmark \mid Z_1 = Z \cup Z_2 \bullet \langle refusal(Z)\rangle\} \\
\quad \neg\, (Z_2 \subseteq Z_1) \Rightarrow \langle refusal(Z_1)\rangle \setminus_{RT} Z_2 = \{\langle\, \circ\,\rangle\} \\
\quad a \in Z \Rightarrow \langle X, event(a)\rangle \setminus_{RT} Z = \{\, \langle\,\rangle\,\} \wedge (\langle X, event(a)\rangle \frown \rho) \setminus_{RT} Z = \rho \setminus_{RT} Z \\
\quad a \notin Z \Rightarrow \langle X, event(a)\rangle \setminus_{RT} Z = \{\rho : \langle X\rangle \setminus_{RT} Z \bullet \rho \frown \langle event(a)\rangle\,\} \\
\quad a \notin Z \Rightarrow (\langle X, event(a)\rangle \frown \rho) \setminus_{RT} Z = \{\rho_1 : \langle X\rangle \setminus_{RT} Z;\ \rho_2 : \rho \setminus_{RT} Z \bullet \rho_1 \frown \langle event(a)\rangle \frown \rho_2\,\} \\
\quad (\phi_1 \frown \langle X\rangle \frown \phi_2) \setminus_{RT} Z = \{\phi_3 : \phi_1 \setminus_{RT} Z;\ \rho : \langle X\rangle \setminus_{RT} Z;\ \phi_4 : \phi_2 \setminus_{RT} Z \bullet \phi_3 \frown \rho \frown \phi_4\}
\end{array}
$$

If there are hidden events that are not refused, we have an unstable state. In this case, the only refusal trace that can be obtained from $\langle refusal(Z_1)\rangle$ is $\langle\, \circ\,\rangle$. Finally, if an event a is hidden, both that event and its preceding refusal are removed. Otherwise, its observation is unaffected by the hiding.

Example 3.8. The model of $P \setminus \{a\}$, where P is defined as $P = a \to P$, is $\{\langle\, \circ\,\rangle\}$. This is different from its set of stable failures, which is empty. For $P = a \to b \to P$, we have traces of arbitrary length that repeatedly record the refusal $\{a\}$ followed by the event b (and all their prefixes). For example, $\langle \{b\}, a, \{a\}, b, \circ\,\rangle \setminus_{RT} \{a\}$ is the set of traces containing $\langle \{a\}, b, \circ\,\rangle$ and $\langle \emptyset, b, \circ\,\rangle$. From $\langle \{b\}, a, \circ, b, \circ\,\rangle$, we get $\langle\, \circ, b, \circ\,\rangle$. □

Renaming. The form of renaming considered in [29] is $f(P)$, where $f : \Sigma \to \Sigma$ is a function that maps events to events, for a finite Σ. This is a special case of the general form $P[[R]]$, where R is a function; moreover, as we illustrate below, the definition adopted is inaccurate in some cases.

The semantics for $f(P)$ defined in [29] is as follows. It is obtained by renaming each component of the traces in $rtraces[[P]]$ according to f. Only the traces that after renaming are in $RTrace$ are considered.

$$rtraces[[f(P)]] = \{\rho : rtraces[[P]] \mid (\forall\, i : 1\,..\,\#\rho - 1 \mid odd\, i \bullet f_{obs}(\rho\,(i+1)) \notin_{RT} f_{obs}(\rho\, i)) \bullet f_{RT}(\rho)\}$$

The function f_{obs} applies f to an observation, while f_{RT} applies f to a refusal trace.

$$
\begin{array}{l}
f_{obs}\, \_ : Obs \to Obs \\
\hline
\forall\, X : ran\, refusal;\ a : \Sigma \bullet \\
\quad f_{obs}(event(\checkmark)) = event(\checkmark) \wedge f_{obs}(event(a)) = event(f(a)) \\
\quad f_{obs}(\circ) = \circ \wedge f_{obs}(X) = refusal(\{x : \Sigma \mid x \in refusal^{\sim} X \wedge x \neq \checkmark \bullet f(x)\} \cup \{\checkmark : refusal^{\sim} X\})
\end{array}
$$

As probably expected, f_{RT} is just the mapping of f_{obs} over a sequence.

Below, we calculate $rtraces[[f(a \rightarrow STOP)]]$ to illustrate the model of the renaming operator, for $f = \{(a, b), (b, b)\}$ and $\Sigma = \{a, b\}$. First, we consider $rtraces[[a \rightarrow STOP]]$.

$$rtraces[[a \rightarrow STOP]]$$

$$= \{X : Refusal \mid event(a) \notin_{RT} X \bullet \langle X \rangle\} \cup \{X : Refusal;\ \rho : rtraces[[STOP]] \mid event(a) \notin_{RT} X \bullet \langle X, event(a) \rangle \frown \rho\}$$

[definition of prefixing]

$$= \{\langle \circ \rangle, \langle \{\checkmark, b\} \rangle, \ldots\} \cup \{X : Refusal;\ \rho : \{\langle \{a, b, \checkmark\} \rangle, \ldots\} \mid event(a) \notin_{RT} X \bullet \langle X, event(a) \rangle \frown \rho\}$$

[$\Sigma = \{a, b\}$ and $rtraces[[STOP]]$]

$$= \{\langle \circ \rangle, \langle \{\checkmark, b\} \rangle, \ldots\} \cup \{\langle \{\checkmark, b\}, a, \{\checkmark, a, b\} \rangle, \ldots\}$$

$$= \{\langle \circ \rangle, \langle \{\checkmark, b\} \rangle, \ldots, \langle \{\checkmark, b\}, a, \{\checkmark, a, b\} \rangle, \ldots\}$$

Using this result, we obtain the following for the renaming.

$$rtraces[[f(a \rightarrow STOP)]]$$

$$= \{\rho : rtraces[[a \rightarrow STOP]] \mid (\forall\, i : 1 \,..\, \#\rho - 1 \mid odd\ i \bullet f_{obs}(\rho\,(i+1)) \notin_{RT} f_{obs}(\rho\ i)) \bullet f_{RT}(\rho)\}$$

$$= \{\rho : \{\langle \circ \rangle, \langle \{\checkmark, b\} \rangle, \langle \{\checkmark, b\}, a, \{\checkmark, a, b\} \rangle, \ldots\} \mid (\forall\, i : 1 \,..\, \#\rho - 1 \mid odd\ i \bullet f_{obs}(\rho\,(i+1)) \notin_{RT} f_{obs}(\rho\ i)) \bullet f_{RT}(\rho)\}$$

[definition of $rtraces[[a \rightarrow STOP]]$]

$$= \{\langle \circ \rangle, \langle \{\checkmark, b\} \rangle, \ldots, \langle \{\checkmark\}, b, \{\checkmark, b\} \rangle, \ldots\}$$

The behaviour described by $f(a \rightarrow STOP)$ records that b is both refused (in the trace $\langle \{\checkmark, b\} \rangle$) and communicated (in $\langle \{\checkmark\}, b, \{\checkmark, b\} \rangle$). Moreover, the refusals observed after b do not include the refusal of b itself.

The model for renaming we propose (shown in Table 4) addresses these issues and considers $P[[R]]$, for $R : \Sigma \leftrightarrow \Sigma$. Our definition is based on the stable refusals model in [33]. As usual, the traces in $rtraces[[P[[R]]]]$ are derived from mappings in R: events belong to the direct mapping of events, and, refusals (or absence of stability) are those whose inverse mapping are observations in P. Like in [33], we assume that R is total and associates $\checkmark$ only to $\checkmark$ itself. Events that are not renamed are associated to themselves. The definition of R_{ref} is as follows.

$$R_{ref} : Refusal \rightarrow Refusal$$

$$R_{ref}(\circ) =\circ\wedge\ \forall\, Z : \mathbb{P}\, \Sigma^{\checkmark} \bullet R_{ref}(refusal(Z)) = refusal(\{a_1 : \Sigma^{\checkmark} \mid \exists\, a_2 : Z \bullet a_1\ R\ a_2\})$$

R_{ref} associates a refusal of events with new names to a refusal of original events. The effect of renaming on a refusal trace is captured by the relation between traces defined below.

$$\_\ R_{RT}\ \_ : RTrace \leftrightarrow RTrace$$

$$\forall\, \rho_1, \rho_2 : RTrace \bullet \rho_1\ R_{RT}\ \rho_2 \Leftrightarrow \#\rho_1 = \#\rho_2 \wedge$$
$$\forall\, i : 1 \,..\, \#\rho_1 \bullet (even\ i \Rightarrow (\rho_2\ i)\ R\ (\rho_1\ i)) \wedge (odd\ i \Rightarrow \rho_2\ i = R_{ref}(\rho_1\ i))$$

We consider now how our definition characterises the semantics of the renaming above.

Example 3.9. We calculate $rtraces[[(a \rightarrow STOP)[[R]]]]$, for $R = \{(a, b), (b, b), (\checkmark, \checkmark)\}$ with $\Sigma = \{a, b\}$.

$$
\begin{array}{ll}
& rtraces[[(a \rightarrow STOP)[[R]]]] \\
= & \{\rho_1 : RTrace \mid \qquad [rtraces[[P[[R]]]]] \\
& \quad \exists \rho_2 : rtraces[[a \rightarrow STOP]] \bullet \#\rho_1 = \#\rho_2 \land \\
& \quad \forall i : 1 \mathinner{..} \#\rho_1 \bullet (even\ i \Rightarrow (\rho_2\ i)\ R\ (\rho_1\ i)) \land (odd\ i \Rightarrow R_{ref}(\rho_1\ i) = \rho_2\ i) \\
& \} \\
= & \{\rho_1 : RTrace \mid \qquad [rtraces[[a \rightarrow STOP]]] \\
& \quad \exists \rho_2 : \{\langle\circ\rangle, \langle\{\checkmark, b\}\rangle, \ldots, \langle\{\checkmark, b\}, a, \{\checkmark, a, b\}\rangle, \ldots\} \bullet \#\rho_1 = \#\rho_2 \land \\
& \quad \forall i : 1 \mathinner{..} \#\rho_1 \bullet (even\ i \Rightarrow (\rho_2\ i)\ R\ (\rho_1\ i)) \land (odd\ i \Rightarrow R_{ref}(\rho_1\ i) = \rho_2\ i) \\
& \} \\
= & \{\rho_1 : RTrace \mid \qquad [\text{definition of } R_{ref}\text{: } R_{ref}(\circ) = \circ,\ R_{ref}(\{a, \checkmark\}) = \{\checkmark\},\ R_{ref}(\{a, b, \checkmark\}) = \{a, b, \checkmark\}] \\
& \quad \#\rho_1 = 1 \land \rho_1\ 1 = \circ \lor \ldots \lor \#\rho_1 = 1 \land \rho_1\ 1 = \{a, \checkmark\} \lor \ldots \lor \\
& \quad \#\rho_1 = 3 \land \rho_1\ 1 = \{a, \checkmark\} \land \rho_1\ 2 = b \land \rho_1\ 3 = \{a, b, \checkmark\} \lor \ldots \\
& \} \\
= & \{\langle\circ\rangle, \langle\{a, \checkmark\}\rangle, \ldots, \langle\{a, \checkmark\}, b, \{a, b, \checkmark\}\rangle, \ldots\} \qquad [\text{predicate calculus}]
\end{array}
$$

Due to renaming, b happens whenever a does in $a \rightarrow STOP$, thus, after renaming, b can not be refused initially. □

Next, we consider inputs and outputs in the context of refusal traces.

3.4 Refinement

As usual for CSP models, refinement is defined by subset inclusion, and so, entails reduction of nondeterminism.

Definition 3.10 (Refusal-traces refinement). $P \sqsubseteq_{RT} Q \mathrel{\widehat{=}} rtraces[[Q]] \subseteq rtraces[[P]]$

Unlike in the definition of failures refinement, it is not required that the traces of Q are all traces of P, that is, $traces[[Q]] \subseteq traces[[P]]$. Failures refinement includes this requirement because not every trace of a process is involved in a failure: if the trace leads to an unstable state, then no refusal can be observed.

Trace inclusion is not needed for refusal-traces refinement because, with the use of $\circ$, every trace of a process has a corresponding refusal trace. In unstable states, a $\circ$ refusal is recorded. To establish this result we use a function $trace(\rho)$ defined in [29] to map a refusal trace to a standard trace, and the function $RTtoTraces(RT)$, which characterises the set of standard traces corresponding to a set of refusal traces RT. New Z operators used are defined below.

Definition 3.11 (Trace projection). $RTtoTraces(RT) = trace(\!|\ RT\ |\!)$ where

$$
\begin{array}{|l}
trace : RTrace \rightarrow \mathrm{seq}\ \Sigma^{\checkmark} \\
\hline
\forall \rho : RTrace \bullet trace(\rho) = (\{i : 1 \mathinner{..} \#\rho \mid even\ i\} \upharpoonright \rho) \fatsemi event^{\sim}
\end{array}
$$

The sequence $\{i : 1 \mathinner{..} \#\rho \mid even\ i\} \upharpoonright \rho$ is that obtained by extracting from ρ the events. This is a function from indices to elements constructed using *event*. By composing this function with $event^{\sim}$, we have a function from indices to elements of $\Sigma^{\checkmark}$, that is, a sequence of elements from $\Sigma^{\checkmark}$. The operator $R_1 \fatsemi R_2$ is relational composition here applied to two functions. We use $f^{\sim}$ for the inverse of the function f. For the injective constructors f of a free type, $f^{\sim}$ is also a function. Finally, $R(\!|\ S\ |\!)$ is the relational image of a set S through the relation R.

The result that we need can be formalised as follows. As said in Section 2.1, $traces[[P]]$ contains the sequences of events in which P can engage. A definition of $traces[[P]]$ for the process operators in Section 2.1 is in [34].

Theorem 3.12. $traces[[P]] = RTtoTraces(rtraces[[P]])$

Proof. Similar to that presented in [29, Theorem 2.19] for the failures model. □

With this, the following result can be established in a fairly direct way. We recall from Section 2.1 that traces refinement $P \sqsubseteq_T Q$ is defined by subset inclusion just like refusal-traces refinement, as usual in CSP models.

Theorem 3.13. $P \sqsubseteq_{RT} Q \Rightarrow P \sqsubseteq_T Q$.

A proof for this theorem and others omitted in the sequel can be found in [10].

In a similar way, we can obtain the failures of a process from its refusal tests using a function $RTtoFailures(RT)$ defined below. This projection function is different from a corresponding function defined in [29], since we treat divergence differently. For example, as already said, we define div as the process whose only trace is $\langle \circ \rangle$.

The function *failures* used below defines the set of failures characterised by a refusal trace. No failure is characterised by a refusal trace ρ that leads to an unstable state, that is, whose last observation is $\circ$. Moreover, for a termination trace, that is, a trace $\rho_2 \frown \langle event(\checkmark), X\rangle$ whose last event is $\checkmark$, we include failures for the trace of ρ_2 to cover all refusal sets Z that do not include $\checkmark$ and the failure whose trace includes the $\checkmark$ with the refusal defined by X. For a refusal trace that leads to a stable state, we have a single failure characterised by its trace of events and its last refusal set, as expected.

The special treatment of termination is required because, in the stable-failures model, termination is, in some sense, regarded as stable. In particular, *SKIP*, has stable failures of the form $(\langle\rangle, Z)$, for any set of events Z not including $\checkmark$. This indicates that the terminating state is stable. In $\mathcal{RT}$, however, as said, termination is unstable as stated in RT4.

Definition 3.14 (Failure projection). $RTtoFailures(RT) = \bigcup failures(\!|\ RT\ |\!)$ where

$$failures : RTrace \rightarrow \mathbb{P}\ Failure$$

$$\begin{array}{l}
\forall\, \rho : RTrace \bullet \\
\quad \mathrm{last}\ \rho = \circ \Longrightarrow failures(\rho) = \emptyset \\
\quad \mathrm{last}\ \rho \neq \circ \Longrightarrow \\
\quad \left(\begin{array}{l}
(\exists\, \rho_2 : RTrace;\ X : Refusal \bullet \rho = \rho_2 \frown \langle event(\checkmark), X\rangle) \Rightarrow \\
\qquad failures(\rho) = \{\, (trace(\rho), refusal^{\sim}(X)) \,\} \cup \{Z : \mathbb{P}\,\Sigma \bullet (trace(\rho_2), Z)\} \\
\neg\, (\exists\, \rho_2 : RTrace;\ X : Refusal \bullet \rho = \rho_2 \frown \langle event(\checkmark), X\rangle) \Rightarrow failures(\rho) = \{\, (trace(\rho), refusal^{\sim}(\mathrm{last}\ \rho)) \,\}
\end{array} \right)
\end{array}$$

and *Failure* is the set of pairs of the standard failures model.

There is, however, a correspondence between $\mathcal{RT}$ and the stable failures model characterised by the function *RTtoFailures* as established below. As mentioned in Section 2.1, $failures[[P]]$ contains the failures of P; a definition of this semantic function for all the process operators in Section 2.1 can be found in [34].

Theorem 3.15. $failures[[P]] = RTtoFailures(rtraces[[P]])$

With this theorem, we know that the failures, as well as the traces, embedded in our refusal-traces semantics is compatible with the stable-failures semantics of CSP. Using these results, we establish a relationship between input-output refusal-traces refinement and traces and failures refinement in Section 4.

4 INPUTS AND OUTPUTS IN THE $\mathcal{RT}$ MODEL

As previously explained, input-output failures involve observing refusal sets at the end of traces and ioco is concerned only with quiescence, but at arbitrary points during a trace execution. As a result $P \sqsubseteq_{IOF} Q$ and Q ioco P are not comparable, in general. Here, first, in Section 4.1, we extend the concepts behind input-output failures to the $\mathcal{RT}$ model defined in the previous section, and in Section 4.2 define input-output refusal-traces refinement as an alternative conformance relation to both $\sqsubseteq_{IOF}$ and ioco. Next, in Section 4.3 we define input-output refusal traces for the CSP operators by calculation from the definitions in Section 3.3. Comparison with ioco is considered in Section 5.

4.1 Input-output refusal traces

To adapt $\mathcal{RT}$ to distinguish between inputs and outputs, we consider that a process is stable when it can engage only in inputs, and do not allow refusal of outputs. This leads to the definition below of the set $IOrtraces^O[[P]]$ of input-output refusal traces for a process P, where O is the subset of Σ of output events. The events in $\Sigma \setminus O$ are the inputs.

Definition 4.1 (Input-output refusal traces). $IOrtraces^O[[P]] \mathrel{\widehat{=}} \{\rho : RTrace \mid iotrace(\rho) \in rtraces[[P]]\}$

In the definition above, we rely on a function *iotrace* that, given a refusal trace ρ, characterises a corresponding refusal trace $iotrace(\rho)$ that can justify ρ as an input-output refusal trace. It is defined below.

Definition 4.2 (Input-output refusal traces). $iotrace(\phi) \mathrel{\widehat{=}} ioobs \circ \phi$

To simplify our notation, we leave an extra parameter O of *iotrace* implicit. Although not needed in Definition 4.1, *iotrace* is defined for arbitrary sequences of observations ϕ, not only refusal traces, to facilitate proofs. To define $iotrace(\phi)$, we compose a function on observations *ioobs*. (We note that a sequence is a function from indices to values.) The composition is the sequence obtained by applying *ioobs* to all elements of ϕ.

The function *ioobs* preserves all observations, except refusals, to which it adds the outputs O. This is an implicit parameter of both *iotrace* and *ioobs*. The formal definition of *ioobs* is presented below.

$$ioobs : Obs \to Obs$$

$$ioobs(\circ) = \circ \land ioobs(event(a)) = event(a) \land ioobs(refusal(Z)) = refusal(Z \cup O)$$

As a result of the above, $iotrace(\rho)$ records exactly the same events as in ρ, and its refusals are either $\circ$, if that is what is recorded in ρ itself, or a superset of the refusal in ρ that includes all outputs O. Accordingly, the set $IOrtraces^O[[P]]$ includes the traces ρ for which $iotrace(\rho)$ is in $rtraces[[P]]$.

Example 4.3. We consider again the process *EC* given in Example 3.2. The events that represent communications over the channels *inA* and *inB* are inputs, and those over *outA* and *outB* are outputs. Accordingly, the value of the parameter O of *IOrtraces* used below is $\{|outA, outB|\}$. We recall that, in *EC*, we have inputs $inA?x$ and $inB?x$ in choice and an input $inA?x$ in choice with an output $outB!1$.

$$\begin{aligned} IOrtraces^O[[EC]] = \{\ & \langle \{|outA, outB, \checkmark|\}, inA?x, \{|inA, inB, outA, outB, \checkmark|\} \rangle, \ldots, \\ & \langle \{|outA, outB, \checkmark|\}, inB?x, \circ, inA?x, \circ, outA.1, \{|inA, inB, outA, outB, \checkmark|\} \rangle, \ldots, \\ & \langle \{|outA, outB, \checkmark|\}, inB?x, \circ, outB.1, \circ, outA.1, \{|inA, inB, outA, outB, \checkmark|\} \rangle, \ldots, \} \end{aligned}$$

We note that the only refusal that we can observe after $inB?x$ or $outB.1$, and after an $inA?x$ event after an $inB?x$, is $\circ$. This is because, after each of them, an output is available. So, the states after these events are not stable. □

For every process P and set of output events O, the set $IOrtraces^O[[P]]$ satisfies the healthiness conditions of the $\mathcal{RT}$ model, namely MRT0 and RT1 to RT4. The proof of this result can be found in [10, Appendix E].

Sets *IORT* of input-output refusal traces that specify a process satisfy also the extra healthiness condition below.

RT5 $\quad \rho \in IORT \Leftrightarrow iotrace(\rho) \in IORT$

This ensures that, if there is an observation in *IORT* of a refusal different from $\circ$, then there is also a corresponding observation where all outputs are refused. So, if a state is recorded as stable, then all outputs are refused.

The sets of input-output refusal traces defined by $IOrtraces^O[[P]]$ satisfy RT5.

Theorem 4.4. *$IOrtraces^O[[P]]$ is RT5-healthy.*

Proof.

$$\begin{aligned}
&\rho \in IOrtraces^O[[P]] \\
&= iotrace(\rho) \in rtraces[[P]] && \text{[definition of } IOrtraces^O[[P]]\text{]} \\
&= iotrace(iotrace(\rho)) \in rtraces[[P]] && \text{[}iotrace\text{ is idempotent]} \\
&= iotrace(\rho) \in IOrtraces^O[[P]] && \text{[definition of } IOrtraces^O[[P]]\text{]}
\end{aligned}$$

□

The subset *IORTrace* of *RTrace* captures that we cannot observe a refusal before an output.

Definition 4.5. $IORTrace == \{\rho : RTrace \mid \forall i : 1 \,..\, \#\rho - 1 \bullet odd\ i \land \rho\ i \neq \circ \Rightarrow event^{\sim}(\rho\ (i+1)) \notin O\}$

Interestingly, if the result of applying *iotrace* to a refusal trace ρ is itself a refusal trace, then ρ is in *IORtrace*.

Lemma 4.6. $iotrace(\rho) \in RTrace \Rightarrow \rho \in IORTrace$

As a direct consequence of this lemma, $IOrtraces^O[[P]] \subseteq IORTrace$. We recall that all proofs omitted here are in [10].

4.2 Input-output refusal-traces refinement

We define a notion of input-output refusal-traces refinement in the expected way.

Definition 4.7 (Input-output refusal-traces refinement). $P \sqsubseteq_{IORT} Q \mathrel{\widehat{=}} IOrtraces^O[[Q]] \subseteq IOrtraces^O[[P]]$

Example 4.8. We consider a process *IC*1 that can initially participate in an output *out*, but not in an input *inp*.

$IC1 = out \rightarrow inp \rightarrow (out \rightarrow STOP \,\Box\, inp \rightarrow STOP) \sqcap out \rightarrow (out \rightarrow STOP \,\Box\, inp \rightarrow inp \rightarrow STOP)$

The maximal traces of *IC*1 are $\langle out, inp, inp\rangle$, $\langle out, inp, out\rangle$, and $\langle out, out\rangle$. There are two ways in which *IC*1 produces $\langle out, inp, inp\rangle$. In one case, we obtain the maximal input-output refusal trace $\langle \circ, out, \{out, \checkmark\}, inp, \circ, inp, \{out, inp, \checkmark\}\rangle$. There is a $\circ$ before the last *inp* since this input is in an external choice with an output and so occurs in an unstable state. In the second case, we obtain the maximal input-output refusal trace $\langle \circ, out, \circ, inp, \{out, \checkmark\}, inp, \{out, inp, \checkmark\}\rangle$.

We now consider the trace $t = \langle out, inp\rangle$. The maximal input-output refusal traces of *IC*1 that have trace t are $\langle \circ, out, \{out, \checkmark\}, inp, \circ\rangle$ and $\langle \circ, out, \circ, inp, \{out, \checkmark\}\rangle$. We consider $\rho = \langle \circ, out, \{out\}, inp, \{out\}\rangle$ and the process *IC*2.

$IC2 = out \rightarrow inp \rightarrow inp \rightarrow STOP \sqcap out \rightarrow (out \rightarrow STOP \,\Box\, inp \rightarrow (out \rightarrow STOP \,\Box\, inp \rightarrow STOP))$

The processes *IC*1 and *IC*2 are both divergence free and have the same sets of traces and failures. However, ρ is an

input-output refusal trace of $IC2$ but not of $IC1$. Thus, $IC2$ is not an input-output failures refinement of $IC1$. □

It might appear that if a process Q is input-enabled then, for it to input-output refusal-trace refine another process P, Q must avoid all states of P in which an input is not enabled. The example below shows that this is not the case.

Example 4.9. We consider a process P with inputs $inp1$ and $inp2$ defined as follows.

$$P = (inp1 \rightarrow inp1 \rightarrow CHAOS \sqcap inp1 \rightarrow inp2 \rightarrow CHAOS)) \Box inp2 \rightarrow CHAOS$$

This is not input-enabled, because after $inp1$ happens, whatever the (nondeterministic) choice, either $inp2$ or $inp1$ is refused. The following is a correct implementation.

$$Q = inp1 \rightarrow (inp1 \rightarrow CHAOS \Box inp1 \rightarrow inp2 \rightarrow CHAOS) \Box inp2 \rightarrow CHAOS$$

This reduces nondeterministic choice. The corresponding IOTS is input-enabled, and does not avoid specification states that are not input enabled. □

Like for refusal-traces refinement, we again do not need to require traces inclusion, as established below by Lemma 4.10. The omitted proofs of all lemmas presented here can be found in [10].

Lemma 4.10. $P \sqsubseteq_{IORT} Q \Rightarrow traces[[Q]] \subseteq traces[[P]]$.

We show below in Theorem 4.14 that the input-output refusal-traces refinement relation is stronger than input-output failures refinement. First we recall the definition of $IOfailures^O(P)$ for a process P defined originally in [12].

Definition 4.11. $IOfailures^O(P) \mathrel{\widehat{=}} \{ (s, X) \mid (s, X \cup O) \in failures[[P]] \}$

We first note that input-output refusal-traces refinement establishes input-output failures inclusion.

Lemma 4.12. $P \sqsubseteq_{IORT} Q \Rightarrow IOfailures^O(Q) \subseteq IOfailures^O(P)$.

We then recall the definition of input-output failure refinement $\sqsubseteq_{IOF}$ also from [12].

Definition 4.13. $P \sqsubseteq_{IOF} Q \mathrel{\widehat{=}} traces[[Q]] \subseteq traces[[P]] \wedge IOfailures^O(Q) \subseteq IOfailures^O(P)$

As usual for failures models, we require subset inclusion of both failures and traces, because traces that lead to divergence have no failures. With these definitions and results, proof of Theorem 4.14 is simple.

Theorem 4.14. $P \sqsubseteq_{IORT} Q \Rightarrow P \sqsubseteq_{IOF} Q$.

Proof. Direct from Lemmas 4.10 and 4.12, and the definition of $\sqsubseteq_{IOF}$. □

The reverse is not true. For instance, for the processes $IC1$ and $IC2$ from Example 4.8, we have $IC1 \sqsubseteq_{IOF} IC2$, but, as observed, $IC1 \sqsubseteq_{IORT} IC2$ does not hold. In summary, $\sqsubseteq_{IORT}$ is stronger than input-output failures refinement, for arbitrary processes. Finally, refusal-traces refinement ensures input-output refusal-traces refinement.

Theorem 4.15. $P \sqsubseteq_{RT} Q \Rightarrow P \sqsubseteq_{IORT} Q$.

Proof.

$$P \sqsubseteq_{RT} Q$$

$$= rtraces[[Q]] \subseteq rtraces[[P]] \qquad \text{[definition of } \sqsubseteq_{RT}\text{]}$$

$\Rightarrow \{\rho : RTrace \mid iotrace(\rho) \in rtraces[[Q]]\} \subseteq \{\rho : RTrace \mid iotrace(\rho) \in rtraces[[P]]\}$ [property of sets]

$= P \sqsubseteq_{IORT} Q$ [definition of $\sqsubseteq_{IORT}$]

□

To conclude, if ρ is in $IOrtraces^{O}[[P]]$, recording a refusal X, then $X \neq \circ$ only if P is quiescent at the point of observation. In this case, $IOrtraces^{O}[[P]]$ also contains an input-output refusal trace in which X is replaced by $X \cup_{RT} event(\!| O |\!)$. This suggests that it is useful to consider the elements of $IOrtraces^{O}[[P]]$ in which all refusal sets are either $\circ$ or include O. The function $IOrtraces^{O}_{M}(RT)$ below characterises the subset of such traces for a given set of refusal traces RT.

Definition 4.16. Given a healthy set RT of refusal traces, $IOrtraces^{O}_{M}(RT) \mathrel{\widehat{=}} \{\rho : RTrace \mid iotrace(\rho) \in RT \bullet iotrace(\rho)\}$.

Input-output refusal-traces refinement can be characterised using $IOrtraces^{O}_{M}$. The model characterised by $IOrtraces^{O}[[P]]$ is more natural than that defined by $IOrtraces^{O}_{M}(P)$, since $IOrtraces^{O}[[P]]$ records all experiments that can be carried out in observing P. This ensures that refusals are subset-closed in the sense of RT1. For reasoning, however, $IOrtraces^{O}_{M}(RT)$ is useful because, when it is applied to a healthy set $IORT$ of refusal traces, it keeps the traces in the range of *iotrace*.

Theorem 4.17. $P \sqsubseteq_{IORT} Q \Leftrightarrow IOrtraces^{O}_{M}(rtraces[[Q]]) \subseteq IOrtraces^{O}_{M}(rtraces[[P]])$

Proof. *Case* ($\Rightarrow$).

$P \sqsubseteq_{IORT} Q$

$= IOrtraces^{O}[[Q]] \subseteq IOrtraces^{O}[[P]]$ [definition of $\sqsubseteq_{IORT}$]

$\Rightarrow \{\rho : RTrace \mid iotrace(\rho) \in rtraces[[Q]]\} \subseteq \{\rho : RTrace \mid iotrace(\rho) \in rtraces[[P]]\}$

[definition of $IOrtraces^{O}[[Q]]$]

$\Rightarrow \{\rho : RTrace \mid iotrace(\rho) \in rtraces[[Q]] \bullet iotrace(\rho)\} \subseteq \{\rho : RTrace \mid iotrace(\rho) \in rtraces[[P]] \bullet iotrace(\rho)\}$

[function application]

$\Rightarrow IOrtraces^{O}_{M}(rtraces[[Q]]) \subseteq IOrtraces^{O}_{M}(rtraces[[P]])$ [definition of $IOrtraces^{O}_{M}(\_)$]

Case ($\Leftarrow$).

$IOrtraces^{O}_{M}(rtraces[[Q]]) \subseteq IOrtraces^{O}_{M}(rtraces[[P]])$

$\Rightarrow \{\rho : RTrace \mid \exists \rho_1 : IOrtraces^{O}_{M}(rtraces[[Q]]) \bullet \rho \leq_{RT} \rho_1\}$ [property of sets]
$\quad \subseteq \{\rho : RTrace \mid \exists \rho_1 : IOrtraces^{O}_{M}(rtraces[[P]]) \bullet \rho \leq_{RT} \rho_1\}$

$\Rightarrow IOrtraces^{O}[[Q]] \subseteq IOrtraces^{O}[[P]]$ [[10, Lemma E.7]]

$= P \sqsubseteq_{IORT} Q$ [definition of $\sqsubseteq_{IORT}$]

□

Below, we give an explicit characterisation of $IOrtraces^{O}_{M}(IORT)$ proved correct in [10].

Lemma 4.18. $IOrtraces^{O}_{M}(IORT) = \{\rho : RTrace \mid iotrace(\rho) \in IORT \wedge iotrace(\rho) = \rho\}$

We use $IOrtraces^{O}_{M}(IORT)$ extensively in calculating, based on Definition 4.1 and Table 4, an input-output refusal-trace semantics for the CSP operators. This is the topic of the next section.

Table 5. *IOrtraces* model of CSP processes

Process P	$IOrtraces^O[[P]]$
div	$\{\langle \circ \rangle\}$
STOP	$\{ X : Refusal \bullet \langle X \rangle \}$
SKIP	$\{\langle \circ \rangle\} \cup \{X : Refusal \bullet \langle \circ, event(\checkmark), X\rangle\}$
$a \rightarrow P$	$\{X : Refusal \mid X =\circ\vee\ event\ a \notin_{RT} X \wedge a \notin O \bullet \langle X \rangle\} \cup$ $\{\rho : IOrtraces^O[[P]];\ X : Refusal \mid (X =\circ\vee\ event\ a \notin_{RT} X \wedge a \notin O) \bullet \langle X, event\ a\rangle \frown \rho\}$
$P \sqcap Q$	$IOrtraces^O[[P]] \cup IOrtraces^O[[Q]]$
$P \,\Box\, Q$	$\{\rho : IOrtraces^O[[P]] \cup IOrtraces^O[[Q]] \mid \langle \rho\, 1\rangle \in IOrtraces^O[[P]] \cap IOrtraces^O[[Q]]\}$
$P;\ Q$	$\{\rho : IOrtraces^O[[P]] \mid event(\checkmark) \notin ran\, \rho\} \cup$ $\{\phi : \text{seq}\ Obs;\ \rho : RTrace \mid \phi \frown \langle \circ, event(\checkmark), \circ \rangle \in IOrtraces^O[[P]] \wedge \rho \in IOrtraces^O[[Q]] \bullet \phi \frown \rho\}$
$P \,[\![Z]\!]\, Q$	$\bigcup\{\rho_1 : IOrtraces^O[[P]];\ \rho_2 : IOrtraces^O[[Q]] \bullet (\rho_1 \,[\![Z]\!]_{RT}\, \rho_2)\}$
$P \setminus Z$	$\bigcup(\{\!|\ IOrtraces^O[[P]]\ |\!\} \setminus_{RT} Z)$
$P[[R]]$	$\{\rho_1 : RTrace \mid \exists\, \rho_2 : IOrtraces^{R_{ref}(O)} P \bullet \rho_1\ R_{RT}\ \rho_2\}$

4.3 Operators

Using Definition 4.1, we can calculate characterisations of input-output refusal traces for CSP processes in terms of their refusal traces as defined in Table 4. A summary of the input-output refusal-traces definitions for each of the CSP operators is provided in Table 5. The calculations are presented in [10, Appendix E].

For *STOP*, the set of input-output refusal traces includes all its refusal traces. Since *STOP* carries out no communications, it has no added instability due to possible outputs. It produces no outputs. The same applies to div and *SKIP*; the instability of divergence or termination is already present in the refusal-traces model for these operators.

For a prefixing $a \rightarrow P$, we have a definition similar to that in the $\mathcal{RT}$ model. We need, however, to consider explicitly $\circ$ as a refusal. Even if a is an output, $\circ$ can be recorded, but, in that case, $\circ$ is the only refusal that can be recorded. We observe that if *out* is an output event, then the semantics of $out \rightarrow P$ is as follows.

$$IOrtraces^O[[out \rightarrow P]] = \{\langle \circ \rangle\} \cup \{\rho : IOrtraces^O[[P]] \bullet \langle \circ, out\rangle \frown \rho\}$$

So, as expected, no stable observation of refusal is possible before the output. The definitions of internal and external choice, sequence, and hiding are in direct correspondence with those in $\mathcal{RT}$. They are unsurprising in this context.

We have to restrict the use of the parallel operator $P \,[\![Z]\!]\, Q$, where we might consider a composition where the inputs and outputs of P and Q are different. If an event e is an input in P and Q, then it is an input in the parallelism; there is no issue. If e is an output in P and Q, then it is an output in the parallelism. We then require that the output is not in Z, so that we have a compositional semantics for $P \,[\![Z]\!]\, Q$. The next example shows the need for this restriction.

Example 4.19. We consider $P1 = out!1 \rightarrow STOP \sqcap out!2 \rightarrow STOP$ and $P2 = out!1 \rightarrow STOP \,\Box\, out!2 \rightarrow STOP$. If the set of outputs is $O = \{\!|out|\!\}$, $P1$ and $P2$ have exactly the same input-output refusal traces. This is because, since their initial states are unstable, all their refusal traces start with $\circ$. We cannot observe the differentiated behaviour of the choices due to lack of stability. In a compositional semantics, we, therefore, expect that $Q1 = out!1 \rightarrow STOP \,[\![\{\!|out|\!\}]\!]\, P1$ and $Q2 = out!1 \rightarrow STOP \,[\![\{\!|out|\!\}]\!]\, P2$ also have the same input-output refusal traces. In $Q1$, however, we have a possible stability, if $P1$ resolves the choice to $out!2 \rightarrow STOP$ and we have a deadlock. In this case, we have singleton input-output refusal traces with all possible refusals. The same stability, however, is not possible for $Q2$. □

So, we define that in a well formed parallelism, Z does not include events e that are outputs in both processes as in the example above. We also have difficulties if e is an input in P and an output in Q. In this case, e is an output in $P \,[\![\, Z \,]\!]\, Q$, if it is in Z. Compositional calculation of input-output refusal traces, however, is not possible, as now illustrated.

Example 4.20. We consider processes $P1 = in!1 \rightarrow STOP \sqcap out!2 \rightarrow STOP$ and $P2 = in!1 \rightarrow STOP \,\Box\, out!2 \rightarrow STOP$. If the set of outputs is $O = \{\!|in, out|\!\}$, $P1$ and $P2$ have exactly the same input-output refusal traces. We, therefore, expect that $Q1 = in?x \rightarrow STOP \,[\![\, \{\!|in, out|\!\} \,]\!]\, P1$ and $Q2 = in?x \rightarrow STOP \,[\![\, \{\!|in, out|\!\} \,]\!]\, P2$ have the same traces. Here, in is an input channel in $in?x \rightarrow STOP$, but an output in $P1$ and $P2$. In $Q1$, we have a possible stability, if $P1$ resolves the choice to $out!2 \rightarrow STOP$ and we have a deadlock. The same stability, however, is not possible for $Q2$. □

The above two examples show that there are processes that have the same input-output refusal traces, but we can distinguish if we allow parallel compositions in which there is synchronisation on events that are outputs in either of them. This is because such a composition allows another process to block an output and, in doing so, make a state that is unstable due to an output become stable. As a result, we cannot expect to have a compositional semantics for parallelism in such situations. This is not a surprise since our semantics reflects the standard assumption in testing that the environment does not block outputs (this is why a process is unstable before an output). Allowing outputs to be blocked, through parallel composition, is incompatible with the aim to have a semantics that supports testing.

Finally, if e is an input in P and an output in Q, or vice-versa, and e is not in Z, then $P \,[\![\, Z \,]\!]\, Q$ is again not well formed, since P and Q have conflicting control over e and $P \,[\![\, Z \,]\!]\, Q$ does not require them to synchronise.

In summary, well formedness of $P \,[\![\, Z \,]\!]\, Q$ requires that P and Q have the same inputs and outputs, and $Z \cap O = \emptyset$. Similar issues arise in the input-output failures semantics [11]. The calculation of $IOrtraces^O[[P[\![Z]\!]Q]]$ is in Appendix B.

For renaming, like in [11], we also make an important assumption: outputs are renamed to outputs, and inputs to inputs. This is an inevitable consequence of the distinction between inputs and outputs. With this, for a set of outputs O, the set of outputs before renaming is $R_{ref}(O)$. The calculation of $IOrtraces^O[[P[[R]]]]$ is below. Since *iotrace* is applied both to original traces and to a renamed traces, we make explicit the set of outputs considered in each application. This is an implicit parameter of *iotrace* that can be inferred from the context in most cases, but here we avoid confusion.

Theorem 4.21.

$$IOrtraces^O[[P[[R]]]] = \{\rho_1 : RTrace \mid \exists\, \rho_2 : IOrtraces^{R_{ref}(O)} P \bullet \rho_1 \; R_{RT} \; \rho_2\}$$

Proof. We prove that $IOrtraces^O_M(IOrtraces^O[[P[[R]]]])$ is equal to

$$IOrtraces^O_M(\{\rho_1 : RTrace \mid \exists\, \rho_2 : IOrtraces^{R_{ref}(O)} P \bullet \rho_1 \; R_{RT} \; \rho_2\})$$

so that the result follows from [10, Theorem E.17].

$$IOrtraces^O_M(IOrtraces^O[[P[[R]]]])$$

$$= \{\rho : RTrace \mid iotrace^O(\rho) \in IOrtraces^O[[P[[R]]]] \wedge iotrace^O(\rho) = \rho\} \qquad \text{[Lemma 4.18]}$$

$$= \{\rho : RTrace \mid iotrace^O(iotrace^O(\rho)) \in rtraces[[P[[R]]]] \wedge iotrace^O(\rho) = \rho\} \qquad \text{[definition of } IOrtraces^O[[P[[R]]]]\text{]}$$

$$= \{\rho : RTrace \mid iotrace^O(\rho) \in rtraces[[P[[R]]]] \wedge iotrace^O(\rho) = \rho\} \qquad \text{[idempotence of } iotrace\text{]}$$

$$= \{\rho : RTrace \mid \exists\, \rho_2 : rtraces[[P]] \bullet iotrace^O(\rho) \; R_{RT} \; \rho_2 \wedge iotrace^O(\rho) = \rho\} \qquad \text{[definition of } rtraces[[P[[R]]]]\text{]}$$

$$= \{\rho : RTrace \mid \exists\, \rho_2 : rtraces[[P]] \bullet iotrace^O(\rho)\ R_{RT}\ \rho_2 \land \rho_2 = iotrace^{R_{ref}(O)}(\rho_2) \land iotrace^O(\rho) = \rho\}$$

[[10, Lemma E.22]]

$$= \{\rho : RTrace \mid \exists\, \rho_2 : RTrace \bullet \quad \text{[predicate calculus]}$$
$$iotrace^{R_{ref}(O)}(\rho_2) \in rtraces[[P]] \land iotrace^O(\rho)\ R_{RT}\ \rho_2 \land \rho_2 = iotrace^{R_{ref}(O)}(\rho_2) \land iotrace^O(\rho) = \rho$$
$$\}$$

$$= \{\rho : RTrace \mid \exists\, \rho_2 : RTrace \bullet iotrace^{R_{ref}(O)}(\rho_2) \in rtraces[[P]] \land iotrace^O(\rho)\ R_{RT}\ \rho_2 \land iotrace^O(\rho) = \rho\}$$

[[10, Lemma E.22]]

$$= \{\rho : RTrace \mid \exists\, \rho_2 : IOrtraces^{R_{ref}(O)}[[P]] \bullet iotrace^O(\rho)\ R_{RT}\ \rho_2 \land iotrace^O(\rho) = \rho\}$$

[definition of $IOrtraces^O[[P]]$]

$$= \{\rho : RTrace \mid iotrace^O(\rho) \in \{\rho_1 : RTrace \mid \exists\, \rho_2 : IOrtraces^{R_{ref}(O)}[[P]] \bullet \rho_1\ R_{RT}\ \rho_2\} \land iotrace^O(\rho) = \rho\}$$

[property of set comprehension]

$$= IOrtraces^O_M(\{\rho_1 : RTrace \mid \exists\, \rho_2 : IOrtraces^{R_{ref}(O)}[[P]] \bullet \rho_1\ R_{RT}\ \rho_2\}) \quad \text{[Lemma 4.18]}$$

□

In addition, $IOrtraces^O[[CHAOS]]$ characterises *CHAOS* as the process that can nondeterministically terminate, deadlock, accept or reject any of the inputs, and produce any of the outputs. Its set of input-output refusal traces is the maximal set of healthy input-output refusal traces. As for $\mathcal{RT}$ and $\sqsubseteq_{RT}$, *CHAOS* is the bottom and div is the top of $\sqsubseteq_{IORT}$. Recursion is again defined using the least-fixed point with respect to $\subseteq$.

Having completed our adaptation of $\mathcal{RT}$ to consider inputs and outputs, we next consider how its input-output refusal-traces refinement relation compares to the standard ioco conformance relation.

5 INPUT-OUTPUT REFUSAL-TRACES REFINEMENT AND IOCO

In this section, we define a notion of suspension traces for CSP processes P using $IOrtraces^O[[P]]$. This notion is the basis of our characterisation of *ioco* in the context of CSP. With that, we can establish that $\sqsubseteq_{IORT}$ is a stronger relation than *ioco*, and, therefore, useful for both development and testing.

Precisely, in this section we show that if $P \sqsubseteq_{IORT} Q$, then Q conforms to P under ioco (Theorem 5.13). Moreover, we show that there are processes P and Q related by ioco for which $P \sqsubseteq_{IORT} Q$ does not hold (Theorem 5.14). Together, these results establish that $\sqsubseteq_{IORT}$ is strictly stronger than ioco for input-enabled implementations. We restrict here attention to input-enabled implementations because ioco is only defined for such implementations.

As discussed earlier, ioco is defined in terms of suspension traces. So, to define it in the context of CSP, we need to define a suspension-traces model for CSP. To that end, we first define below the set $Strace^O$ of valid suspension traces for output events in O; we use the set $\Sigma^\delta = \Sigma \cup \{\delta\}$ including the special event δ to represent quiescence.

Definition 5.1.

$$Strace^O == \{\sigma : \mathrm{seq}\, \Sigma^\delta \mid \forall\, i : 1 \,..\, \#\sigma - 1 \bullet \sigma\, i = \delta \Rightarrow \sigma\,(i+1) \notin O\}$$

This set includes the sequences σ of events from Σ and δ, such that, a δ is never followed by an output. This is required because, if we can observe stability, as recorded by δ, then an output cannot be possible.

In standard definitions of suspension traces for an IOLTS Q, occurrences of δ are represented by adding self-loop transitions labelled by δ. Thus, if $\sigma = \sigma_1 \frown \langle\delta\rangle \frown \sigma_2$ is a trace of Q, then all traces of the form $\sigma_1 \frown \delta^m \frown \sigma_2$ are also suspension traces of Q, where δ^m denotes a sequence of m occurrences of δ, for $m \geq 0$. So, we can define an equivalence relation on suspension traces by taking the transitive closure of the reflexive and symmetric relation $\sim_\delta$ below.

Definition 5.2.

$$\sim_\delta : Strace^O \leftrightarrow Strace^O$$

$$\forall \sigma_1, \sigma_2 : ST \bullet \sigma_1 \sim_\delta \sigma_1 \ \wedge \ (\sigma_1 \frown \langle\delta\rangle \frown \sigma_2) \sim_\delta (\sigma_1 \frown \langle\delta,\delta\rangle \frown \sigma_2) \ \wedge \ (\sigma_1 \sim_\delta \sigma_2 \Leftrightarrow \sigma_2 \sim_\delta \sigma_1)$$

If σ is a suspension trace of a process P, then all traces equivalent to σ under the transitive closure $(\sim_\delta)^*$ of $\sim_\delta$ are also traces of P. This is established by the healthiness condition below of the model of sets ST of suspension traces.

ST1 $\quad \sigma_1 \in ST \wedge \sigma_1 \, (\sim_\delta)^* \, \sigma_2 \Rightarrow \sigma_2 \in ST$

We use $[\sigma]$ to denote the equivalence class of a suspension trace σ according to $(\sim_\delta)^*$. As a direct consequence of ST1, a healthy set of suspension traces ST includes either a whole equivalence class $[\sigma]$ of $(\sim_\delta)^*$, or no element of it at all.

Lemma 5.3. $\forall \sigma : STrace \bullet \sigma \in ST \Leftrightarrow [\sigma] \subseteq ST$

The proof of this lemma and others omitted below are in Appendix B. It follows from the above that we have the following property of suspension-trace inclusion. We write σ_1 in σ_2 when the sequence σ_1 occurs in the sequence σ_2.

Lemma 5.4. $ST_2 \subseteq ST_1 \Leftrightarrow \{\sigma : ST_2 \mid \neg \, (\langle\delta,\delta\rangle \text{ in } \sigma)\} \subseteq ST_1$

In words, $ST_2 \subseteq ST_1$ if, and only if, for all $\sigma \in ST_2$ such that σ does not have occurrences of $\langle\delta,\delta\rangle$, we have $\sigma \in ST_1$. As a result, for the purposes of studying ioco, it is sufficient to consider only suspension traces in which there are no consecutive occurrences of δ. So, we adopt the following healthiness condition, instead of ST1.

ST2 $\quad \sigma \in ST \Rightarrow \neg \, (\langle\delta,\delta\rangle) \text{ in } \sigma$

Additional healthiness conditions for suspension traces have been given in [13]. In related work Willemse [44] studies suspension automata, which are used to define the set of suspension traces of an IOLTS specification. That work introduces the notion of a valid suspension automaton, which satisfies four given properties. These properties impose conditions on the sets of suspension traces that can result from suspension automata. One property, for example, states that it is not possible to follow the observation of quiescence with an output. There is thus potential to study the notion of a valid suspension automaton to derive additional healthiness conditions. Here, ST2 is enough to study the relationship between input-output refusal traces refinement and ioco and simplify proofs.

We define below st to characterises a suspension trace corresponding to a refusal trace $\langle X\rangle$ or $\langle X, event\ a\rangle \frown \rho$.

Definition 5.5.

$$st : RTrace \rightarrow Strace^O$$

$$\forall X : Refusal; \ a : \Sigma^\checkmark; \ \rho : RTrace \bullet (st\,\langle X\rangle = \langle\,\rangle \wedge \neg \, (refusal\, O \subseteq_{RT} X) \vee st\,\langle X\rangle = \langle\delta\rangle \wedge refusal\, O \subseteq_{RT} X) \wedge$$
$$st\,(\langle X, event\ a\rangle \frown \rho) = st(\langle X\rangle) \frown \langle a\rangle \frown st(\rho)$$

This function essentially removes all refusals X that do not contain all outputs O (since these do not correspond to the observation of quiescence) and replaces other refusals with δ. Events are untouched. We can use the function st to

define the set of suspension traces of a process. For a process P, we use $straces[[P]]$ to denote the set of suspension traces of P and define it below in terms of $IOrtraces^O[[P]]$.

Definition 5.6. $straces[[P]] \mathrel{\widehat{=}} st(\!| \, IOrtraces^O[[P]] \, |\!)$.

There is no translation from CSP to IOLTS available in the literature. (Such a translation would characterise a semantics for CSP with inputs and outpus.) As a result, it is not possible to directly compare Definition 5.6 with the definition of suspension traces for an IOLTS. We have, however, used Definition 5.6 to calculate a suspension-trace semantics for the CSP operators [13]. Here, we use it just to compare input-output refusal-traces refinement with ioco.

The following establishes that for a process P we have that $straces[[P]]$ is ST2-healthy. This is immediate from the definition of st and the fact that refusal traces in $IOrtraces^O[[P]]$ alternate between events and refusals.

Lemma 5.7. $straces[[P]]$ *is* ST2*-healthy.*

Proof. We provide a proof by contradiction.

$$\sigma_1 \frown \langle \delta, \delta \rangle \frown \sigma_2 \in straces[[P]]$$
$$\Rightarrow \exists \rho : IOrtraces^O[[P]] \bullet st(\rho) = \sigma_1 \frown \langle \delta, \delta \rangle \frown \sigma_2 \quad \text{[definition of } straces[[P]]\text{]}$$
$$\Rightarrow \exists \rho : IOrtraces^O[[P]];\ i : 1..\#\rho \bullet st(\rho)(i) = \delta \wedge st(\rho)(i+1) = \delta \quad \text{[properties of sequences]}$$
$$\Rightarrow \exists \rho : IOrtraces^O[[P]];\ i : 1..\#\rho \bullet \rho(i) \in \operatorname{ran} refusal \wedge \rho(i+1) \in \operatorname{ran} refusal \quad \text{[definition of } st\text{]}$$

This contradicts the definition of *RTrace*, which is the type of ρ, as required. □

We recall that ioco is traditionally defined using functions *after* and *Out* as shown below.

$$Q \operatorname{ioco} P \mathrel{\widehat{=}} \forall \sigma : straces[[P]] \bullet Out(Q \ after \ \sigma) \subseteq Out(P \ after \ \sigma)$$

This can be simply rewritten to the following.

$$Q \operatorname{ioco} P \mathrel{\widehat{=}} \forall \sigma : straces[[P]] \bullet a \in Out(Q \ after \ \sigma) \Rightarrow a \in Out(P \ after \ \sigma)$$

Above, since a is an output, it can be either an output event or δ. By definition, a is in $Out(R \ after \ \sigma)$, for a process R, if, and only if, R can move via σ to a state r (that is, $r \in (R \ after \ \sigma)$) in which the observation of a can be made (that is, $a \in Out(r)$). This is the case if, and only if, $\sigma \frown \langle a \rangle \in straces[[R]]$. As a result, $a \in Out(R \ after \ \sigma)$ if, and only if, $\sigma \frown \langle a \rangle \in straces[[R]]$. Based on this, we obtain the definition below of $Q \operatorname{ioco} P$ in terms of $straces[[P]]$ and $straces[[Q]]$.

Definition 5.8. For an arbitrary process P and an input-enabled process Q,

$$Q \operatorname{ioco} P \mathrel{\widehat{=}} \forall \sigma : straces[[P]];\ a : O \cup \{\delta\} \bullet \sigma \frown \langle a \rangle \in straces[[Q]] \Rightarrow \sigma \frown \langle a \rangle \in straces[[P]]$$

We note that a similar formalisation of ioco in terms of suspension traces has already been given for IOLTS by Tretmans in [38]. Our discussion above justs recasts these results in the context of CSP suspension traces.

Suspension traces that are ST2-healthy (as opposed to those that include occurrences of $\langle \delta, \delta \rangle$) are structurally similar

to refusal traces. Given a suspension trace σ, it is straightforward to define a corresponding refusal trace $rt(\sigma)$.

$$rt : Strace^{O} \to RTrace$$

$$\forall\, a : \Sigma;\ \sigma : Strace^{O} \bullet$$
$$rt(\langle\,\rangle) = \langle\,\circ\,\rangle \land rt(\langle a\rangle \frown \sigma) = \langle\,\circ, event\ a\rangle \frown rt(\sigma) \land$$
$$rt(\langle\delta\rangle) = \langle refusal\, O\rangle \land rt(\langle\delta, a\rangle \frown \sigma) = \langle refusal\, O, event\ a\rangle \frown rt(\sigma)$$

We include in $rt(\sigma)$ the events from σ. Where there is a δ, we include the refusal set O, and, otherwise, use $\circ$.

Example 5.9. For the suspension trace $\sigma_1 = \langle a, b, \delta\rangle$, we have the following.

$$rt(\sigma_1) = \langle\,\circ, a\rangle \frown rt(\langle b, \delta\rangle) = \langle\,\circ, a\rangle \frown \langle\,\circ, b\rangle \frown rt(\langle\delta\rangle) = \langle\,\circ, a\rangle \frown \langle\,\circ, b\rangle \frown \langle O\rangle = \langle\,\circ, a, \circ, b, O\rangle$$

For $\sigma_2 = \langle a, \delta, b\rangle$, we can proceed as follows.

$$rt(s_2) = \langle\,\circ, a\rangle \frown rt(\langle\delta, b\rangle) = \langle\,\circ, a\rangle \frown \langle O, b\rangle \frown rt(\langle\,\rangle) = \langle\,\circ, a\rangle \frown \langle O, b\rangle \frown \langle\,\circ\,\rangle = \langle\,\circ, a, O, b, \circ\,\rangle$$

□

We now establish how *st* and *rt* relate. We show that *rt* and *st* form a Galois connection between refusal traces, ordered by the $\leq_{RT}$ relation, and suspension traces with equality. We recall that the omitted proofs are in [10, Appendix F].

Theorem 5.10. $st(rt(\sigma)) = \sigma \quad and \quad rt(st(\rho)) \leq_{RT} \rho$

There are two reasons why we do not always have $rt(st(\rho)) = \rho$. The first is that *st* removes any record of inputs from refusal sets (although this is not a problem when processes are input-enabled). The second reason is that a refusal set X in ρ that is a proper subset of O is removed by *st*, and then *rt* introduces $\circ$ at the corresponding position in the trace, even if the process considered is quiescent at that point.

We recall that we have used $IOrtraces^{O}_{M}(P)$ to provided an alternative characterisation of input-output refusal traces refinement. We obtain the following stronger result for input-enabled processes and $IOrtraces^{O}_{M}(P)$.

Theorem 5.11. *For an input-enabled process P, if* $\rho \in IOrtraces^{O}_{M}(rtraces[[P]])$, *then* $rt(st(\rho)) = \rho$.

If a process P is input-enabled, then *rt* and *st* are inverses on $IOrtraces^{O}_{M}(P)$.

Interestingly, it also transpires that $rtraces[[P]]$, $IOrtraces^{O}[[P]]$, and $IOrtraces^{O}_{M}(P)$ define the same sets of suspension traces as characterised using the function *st* as established by the theorem below.

Theorem 5.12. $st(\!|\ IOrtraces^{O}[[P]]\ |\!) = st(\!|\ IOrtraces^{O}_{M}(rtraces[[P]])\ |\!) = st(\!|\ rtraces[[P]]\ |\!)$

Finally, we can show that input-output refusal traces refinement implies ioco.

Theorem 5.13. *Given processes P and Q such that Q is input-enabled,*

$$P \sqsubseteq_{IORT} Q \Rightarrow Q \text{ ioco } P.$$

Proof.

$P \sqsubseteq_{IORT} Q$

$\Rightarrow IOrtraces^{O}[[Q]] \subseteq IOrtraces^{O}[[P]]$ [definition of $\sqsubseteq_{IORT}$]

$\Rightarrow st(\!|\ IOrtraces^{O}[[Q]]\ |\!) \subseteq st(\!|\ IOrtraces^{O}[[P]]\ |\!)$ [property of relational image]

$\Rightarrow$ $straces[[Q]] \subseteq straces[[P]]$ [definition of $straces[[P]]$]

$\Rightarrow$ $\forall\, \sigma' : \sigma' \in straces[[Q]] \Rightarrow \sigma' \in straces[[P]]$ [definition of set inclusion]

$\Rightarrow$ $\forall\, \sigma : straces[[P]];\ a : O \cup \{\delta\} \bullet \sigma \frown \langle a \rangle \in straces[[Q]] \Rightarrow \sigma \frown \langle a \rangle \in straces[[P]]$ [substitution of $\sigma \frown \langle a \rangle$ for σ']

$\Rightarrow$ Q ioco P [definition of ioco]

□

We note that, although it could be argued that the proof above of Theorem 5.13 does not use the hypothesis that Q is input-enabled, this hypothesis is needed because ioco is defined only for input-enabled implementations.

It has been shown in [12, Theorems 1 and 2] that there are processes P and Q such that Q ioco P, but where we do not have that $P \sqsubseteq_{IOF} Q$. The following is thus a direct consequence of Theorem 4.14.

Theorem 5.14. *There are P and Q such that Q* ioco *P, but not $P \sqsubseteq_{IORT} Q$.*

As an example, we can take as the specification P the process *SKIP* and let Q be any input-enabled implementation that can produce an output *out* in response to some input *in* but that cannot produce an output before first receiving an input. Under ioco we only need to consider the outputs (and quiescence) of Q after the empty sequence and this is just quiescence. Since the specification is also initially quiescent, we have that Q ioco P as required. However, $P \sqsubseteq_{IORT} Q$ does not hold since the implementation Q has input-output refusal traces that are not input-output refusal traces of P (for example, any that involves the input *inp* followed by the output *out*).

The last two theorems show that input-output refusal traces refinement is strictly stronger than ioco for input-enabled implementations. This is our main result in this section.

We now consider the special case where P and Q are both input-enabled. For input-enabled processes P and Q, we have that Q ioco P implies $P \sqsubseteq_{IOF} Q$ [12, Theorem 3]. We can now strengthen this result for $\sqsubseteq_{IORT}$.

In the proof of Theorem 5.16, we use the lemma below regarding ioco [2].

Lemma 5.15. *Given input-enabled processes P and Q, Q* ioco *$P \Leftrightarrow straces[[Q]] \subseteq straces[[P]]$.*

Theorem 5.16. *Given input-enabled processes P and Q, $P \sqsubseteq_{IORT} Q \Leftrightarrow Q$* ioco *$P$.*

Proof. The left to right implication is given by Theorem 5.13. So, we assume that Q ioco P and prove that $P \sqsubseteq_{IORT} Q$.

Q ioco P

$\Rightarrow$ $straces[[Q]] \subseteq straces[[P]]$ [Lemma 5.15]

$\Rightarrow$ $st(\!|\ IOrtraces^O[[Q]]\ |\!) \subseteq st(\!|\ IOrtraces^O[[P]]\ |\!)$ [definition of *straces*]

$\Rightarrow$ $st(\!|\ IOrtraces^O_M(Q)\ |\!) \subseteq st(\!|\ IOrtraces^O_M(P)\ |\!)$ [Theorem 5.12]

$\Rightarrow$ $rt(\!|\ st(\!|\ IOrtraces^O_M(Q)\ |\!)\ |\!) \subseteq rt(\!|\ st(\!|\ IOrtraces^O_M(P)\ |\!)\ |\!)$ [property of relational image]

$\Rightarrow$ $IOrtraces^O_M(Q) \subseteq IOrtraces^O_M(P)$ [Theorem 5.11]

$\Rightarrow$ $P \sqsubseteq_{IORT} Q$ [Lemma 4.17]

□

To summarise, we have shown that input-output refusal-traces refinement is strictly stronger than ioco, in general,

and equivalent to ioco for input-enabled specifications. This means that failure of an implementation to conform to a specification according to ioco indicates that the implementation is not an input-output refusal-traces refinement of the specification. We next consider how we can test for input-output refusal traces refinement.

6 TESTING AND INPUT-OUTPUT REFUSAL TRACES

This section provides a testing theory for input-output refusal-traces refinement. For that, we consider two processes, an SUT Q and a test case T, and define a process $Execution(Q, T)$ representing Q and T interacting as black boxes. (The compositional nature of the denotational semantics leads to a simple definition of $Execution(Q, T)$). We then define the notion of the SUT failing a test and demonstrate how we can define a test set $Exhaust^{\mathcal{O}}_{RT}(P)$ that is sound (that is, no correct SUT can fail) and exhaustive (that is, all incorrect SUTs will fail).

First of all, we characterise input-output refusal-traces refinement in terms of traces refinement and a new conformance relation $\mathbf{conf}^{\mathcal{O}}_{\mathbf{IO}}$ concerned just with refusals. With this, we can take advantage of results of the CSP testing theory [7] regarding traces refinement, and concentrate our efforts on $\mathbf{conf}^{\mathcal{O}}_{\mathbf{IO}}$, which is simpler than $\sqsubseteq_{IORT}$. As in the stable-failures model, the notion of a trace and, therefore, traces refinement and its associated testing theory are not affected by the distinction between inputs and outputs made here.

Inspired by [4, 7], we define $\mathbf{conf}^{\mathcal{O}}_{\mathbf{IO}}$ to capture the requirements, under $\sqsubseteq_{IORT}$, on the refusals as follows.

$$Q\ \mathbf{conf}^{\mathcal{O}}_{\mathbf{IO}}\ P \mathrel{\widehat{=}} \forall\, t : traces[[P]] \cap traces[[Q]] \bullet Ref^{\mathcal{O}}_{IO}(Q, t) \subseteq Ref^{\mathcal{O}}_{IO}(P, t)$$
$$\text{where } Ref^{\mathcal{O}}_{IO}(P, t) \mathrel{\widehat{=}} \{\rho : RTrace \mid trace(\rho) = t \land \rho \in IOrtraces^{\mathcal{O}}[[P]]\}$$

In this conformance relation, we consider only traces t of both the specification P and the implementation Q. For those, the associated refusal traces of the implementation as defined by $Ref^{\mathcal{O}}_{IO}(Q, t)$ must be traces of the specification. Using $\mathbf{conf}^{\mathcal{O}}_{\mathbf{IO}}$, traces and refusals can be considered separately when establishing refinement.

Theorem 6.1. $P \sqsubseteq_{IORT} Q \Leftrightarrow traces[[Q]] \subseteq traces[[P]] \ \land\ Q\ \mathbf{conf}^{\mathcal{O}}_{\mathbf{IO}}\ P$

Proof.

$$\begin{aligned}
&P \sqsubseteq_{IORT} Q \\
&\Leftrightarrow IOrtraces^{\mathcal{O}}[[Q]] \subseteq IOrtraces^{\mathcal{O}}[[P]] && \text{[definition of } \sqsubseteq_{IORT}\text{]} \\
&\Leftrightarrow traces[[Q]] \subseteq traces[[P]] \ \land\ IOrtraces^{\mathcal{O}}[[Q]] \subseteq IOrtraces^{\mathcal{O}}[[P]] && \text{[Lemma 4.10]} \\
&\Leftrightarrow traces[[Q]] \subseteq traces[[P]] \ \land\ \forall\, t : traces[[Q]] \circ Ref^{\mathcal{O}}_{IO}(Q, t) \subseteq Ref^{\mathcal{O}}_{IO}(P, t) \\
& && \text{[definitions of } Ref^{\mathcal{O}}_{IO}(P, t) \text{ and } Ref^{\mathcal{O}}_{IO}(P, t)\text{]} \\
&\Leftrightarrow \ traces[[Q]] \subseteq traces[[P]] \ \land \\
&\quad \forall\, t : traces[[Q]] \cap traces[[P]] \circ Ref^{\mathcal{O}}_{IO}(Q, t) \subseteq Ref^{\mathcal{O}}_{IO}(P, t) \\
& && [traces[[Q]] \subseteq traces[[P]] \Rightarrow traces[[Q]] = traces[[Q]] \cap traces[[P]]] \\
&\Leftrightarrow traces[[Q]] \subseteq traces[[P]] \ \land\ Q\ \mathbf{conf}^{\mathcal{O}}_{\mathbf{IO}}\ P && \text{[definition of } \mathbf{conf}^{\mathcal{O}}_{\mathbf{IO}}\text{]}
\end{aligned}$$

□

In our approach to generating test cases for $\mathbf{conf}^{\mathcal{O}}_{\mathbf{IO}}$, we identify a set of input-output refusal traces that the SUT should

not have, if it is a correct implementation of the specification. Given such a trace ρ that the SUT should not have, we define a test case $T_F^{IO}(\rho)$ that checks whether ρ is an input-output refusal trace of the SUT. Naturally, the SUT fails this test case if its interactions with $T_F^{IO}(\rho)$ demonstrate that ρ is one of its input-output refusal traces.

For a trace of events t of the specification P, we might consider the whole set $C_{IO}^{\mathcal{O}}(P, t)$ of input-output refusal traces with trace t that are not input-output refusals traces of P. This is the set of input-output refusal traces that have trace t and are not in $Ref_{IO}^{\mathcal{O}}(P, t)$. We could then test the implementation Q to check whether it implements any of the traces in $C_{IO}^{\mathcal{O}}(P, t)$. As illustrated below, however, it is necessary to use the smaller set of traces whose refusals are all either $\circ$ or contain all outputs, and that are minimal under $\leq_{RT}$ (see Section 3.3 for the definition of $\leq_{RT}$).

Example 6.2. We consider as the specification the process EC from Example 3.2, and its trace $t = \langle inB.1 \rangle$. Before $inB.1$, EC can refuse any output, but no inputs. $C_{IO}^{\mathcal{O}}(EC, t)$ contains, for example, the input-output refusal trace $\rho_1 = \langle \{inA.1, outA.1\}, inB.1, \circ \rangle$: this is not an input-output refusal trace of EC, since it records an initial refusal of an event $inA.1$ in which EC can participate. We now suppose that we wish to test to determine whether ρ_1 is an input-output refusal trace of the SUT. If ρ_1 is an input-output refusal trace of the SUT, then the SUT must be able to refuse $\{inA.1, outA.1\}$, in a stable state, before performing $inB.1$. Since no outputs can be enabled in a stable state, by RT2 we know that, if ρ_1 is an input-output refusal trace of the SUT, then $\rho_2 = \langle \{\!| inA.1, outA, outB |\!\}, inB.1, \circ \rangle$, which differs from ρ_1 in that the first refusal contains all outputs, must also be an input-output refusal trace of the SUT. In addition, RT1 tells us that if ρ_2 is an input-output refusal trace of the SUT, then ρ_1 is also an input-output refusal trace of the SUT. Thus, in order to determine whether ρ_1 is an input-output refusal trace of the SUT, it is sufficient to test in order to determine whether ρ_2 is an input-output refusal trace of the SUT. □

We also restrict ourselves to refusal traces that are minimal with respect to $\leq_{RT}$. Since we have a fixed trace of events t, minimality is in relation to the refusals. If a refusal X_1, different from $\circ$, is not an observation of the SUT, then any X_2 containing all events of X_1 is also not a refusal of the SUT (because of RT1). So, it is enough to consider X_1 in tests.

Traces whose refusals are all either $\circ$ or contain all outputs are those in the range of the function *iotrace*. Therefore, given a specification P and a trace t of P we are interested in the following set of input-output refusal traces.

$$A_{IO}^{\mathcal{O}}(P, t) = min_{\leq_{RT}}\{\rho : IORTrace \mid trace(\rho) = t \wedge \rho \notin IOrtraces^{\mathcal{O}}[[P]] \bullet iotrace(\rho)\}$$

The input-output refusal traces in $A_{IO}^{\mathcal{O}}(P, t)$ have the trace of events t, are *not* traces of P, include all outputs $\mathcal{O}$ in its refusals different from $\circ$, and are minimal with respect to $\leq_{RT}$.

Example 6.3. We consider again the process EC from Example 3.2, and the trace $t = \langle inA.1 \rangle$. The set $A_{IO}^{\mathcal{O}}(EC, t)$ contains traces such as $\rho_1 = \langle \{inB.1, outA.1, outB.1\}, inA.1, \circ \rangle$. Since EC deadlocks after $inA.1$, all input-output refusal traces of the form $\langle \circ, inA.1, X \rangle$ are input-output refusal traces of EC. Furthermore, an input-output refusal trace of the form $\langle X, inA.1, X' \rangle$ with $X \neq \circ$ and $X' \neq \circ$ cannot be a minimal input-output refusal trace that is not in $IOrtraces^{\mathcal{O}}[[EC]]$ since $\langle X, inA.1, \circ \rangle \leq_{RT} \langle X, inA.1, X' \rangle$ and $\langle X, inA.1, X' \rangle \notin IOrtraces^{\mathcal{O}}[[EC]] \Leftrightarrow \langle X, inA.1, \circ \rangle \notin IOrtraces^{\mathcal{O}}[[EC]]$. Thus, all traces in $A_{IO}^{\mathcal{O}}(EC, t)$ are of the form $\langle \{\!| e, outA, outB |\!\}, inA.1, \circ \rangle$ where e is an input. □

To check whether an SUT allows a given input-output refusal trace we need to observe a sequence of events and refusals, and so we need to observe a refusal of the SUT and then continue testing for further events and refusals.

Example 6.4. We consider again the processes $IC1$ and $IC2$ from Example 4.8 and the trace $t = \langle out, inp \rangle$. We recall that the maximal input-output refusal traces of $IC1$ with events as in t are $\langle \circ, out, \{out, \checkmark\}, inp, \circ \rangle$ and $\langle \circ, out, \circ, inp, \{out, \checkmark\} \rangle$. So, we obtain a number of input-output refusal traces in $A_{IO}^{\mathcal{O}}(IC1, \langle out, inp \rangle)$ including

$\rho = \langle \circ, out, \{out\}, inp, \{out\}\rangle$. As pointed out in Example 4.8, ρ is a trace of $IC2$, but not of $IC1$. Testing can show that $IC2$ is not an input-output refinement of $IC1$ if it can demonstrate that ρ is a trace of $IC2$. To test $IC2$ against $IC1$ based on $\rho = \langle \circ, out, \{out\}, inp, \{out\}\rangle$, we need to observe a refusal $\{out\}$ after $\langle \circ, out\rangle$ and also after $\langle \circ, out, \{out\}, inp\rangle$. The observation of the refusal after $\langle \circ, out\rangle$ is not sufficient, since $\langle \circ, out, \{out\}\rangle$ is a trace of $IC1$. In addition, the observation of the refusal $\{out\}$ after $\langle \circ, out, \circ, inp\rangle$ would also produce an input-output refusal trace of $IC1$. □

In summary, as perhaps should be expected, it is not enough to observe a refusal after a trace of events (like in the standard CSP testing theory for failures refinement); we need to observe events and refusals in alternation.

Essentially, the observation of a refusal in an input-output refusal trace requires us to observe deadlock before progressing. Potentially, this might be achieved using a timed version of CSP, but instead we use priorities.

The prioritise operator of CSP applies to a process P and a sequence θ of (pairwise disjoint) sets of events whose priority decreases along θ. The process prioritise(P, θ), in which $\theta = \langle Z_1, \ldots, Z_k\rangle$, behaves like P, but prevents an event $a \in Z_i$ if either an event $b \in Z_j$ is possible for $j < i$, or $i > 1$ and P can take an internal event or can terminate. Internal events, termination, and events in Z_1 have the same priority.

We can use priorities to observe a refusal X and, if this is observed, continue with an event a. Essentially we prioritise all elements in X above a, offer all events in X as well as a, and so if a is followed then the set X must have been refused.

To define the notion of test, we follow the approach in the existing testing theory of CSP by including special events in a test case. Namely, we have a set $V = \{inc, pass, fail\}$ of verdict events to indicate an inconclusive, pass, or fail verdict. All events of the specification (and of the SUT) are hidden and, therefore, cannot be observed or blocked by the environment of a testing experiment. The last visible verdict event gives the verdict of the experiment.

Formally, the execution of a test T against an SUT Q can be described as follows.

Definition 6.5. $Execution(Q, T) \mathrel{\widehat=} \text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T, \langle \Sigma, V\rangle) \setminus \Sigma$

As usual, the SUT and the test are run in parallel. The specification events, that is, those in Σ, are given higher priority than the verdict events from V. Due to the form of a test T defined below, this ensures, as required, that the observation of a refusal is possible, before the experiment continues to observe the next event.

The verdict or, more specifically, the failure of a test, is defined as follows. Basically, a test experiment $Execution(Q, T)$ identifies a failure if it has itself a refusal trace ρ that leads to a deadlock (characterised by the refusal $\Sigma^{\checkmark} \cup V\rangle$ of all specification and verdict events) or a termination (characterised that the existence of a refusal trace that ends in $\langle \circ, \checkmark, \circ \rangle$) after the verdict event *fail*. (For simplicity, we do not use the *Observation* constructors like in examples). We note that the semantics taken for the test experiment is the refusal-traces model.

Definition 6.6. Q fails $T \mathrel{\widehat=} \exists \rho : RTrace \bullet \{\rho \frown \langle fail, \Sigma^{\checkmark} \cup V\rangle, \rho \frown \langle fail, \circ, \checkmark, \circ \rangle\} \cap rtraces[[Execution(Q, T)]] \neq \emptyset$

We define below a function T_F^{IO} that takes an input-output refusal trace and returns a corresponding test case. We note that in Definition 6.7(2), we use the iterated external choice $\Box\, x : Z \bullet x \rightarrow pass \rightarrow STOP$ indexed by events x from a set Z. As usual, the external choice is resolved by the environment. If an event x is chosen, this process behaves like $x \rightarrow pass \rightarrow STOP$. If Z is empty, this process behaves just like *STOP*. In Definition 6.7(2), for an empty Z, we get just $fail \rightarrow STOP$, since *STOP* is the unit of an external choice. Similar comments apply to Definition 6.7(5).

Definition 6.7. Below $a \in \Sigma$, that is, it is not $event(\checkmark)$, X is in ran *refusal*, $\rho : RTrace$, and $Z : \Sigma$.

(1) $T_F^{IO}(\langle \circ \rangle) = fail \rightarrow STOP$

(2) $T_F^{IO}(\langle refusal(Z)\rangle) = (\Box\, x : Z \bullet x \rightarrow pass \rightarrow STOP) \,\Box\, inc \rightarrow (SKIP \,\Box\, fail \rightarrow STOP)$

(3) $T_F^{IO}(\langle \circ, event(\checkmark), X\rangle) = fail \rightarrow (SKIP \,\Box\, pass \rightarrow STOP)$
(4) $T_F^{IO}(\langle \circ, event(a)\rangle \frown \rho) = inc \rightarrow a \rightarrow T_F^{IO}(\rho)$
(5) $T_F^{IO}(\langle refusal(Z), event(a)\rangle \frown \rho) = (\Box\, x : Z \bullet x \rightarrow inc \rightarrow STOP) \,\Box\, inc \rightarrow a \rightarrow T_F^{IO}(\rho)$

With the test $T_F^{IO}(\langle \circ \rangle)$ defined in (1), we always have verdict fail. Since all processes have $\langle \circ \rangle$ as an (input-output) refusal trace, we do not really generate a test just for it. $T_F^{IO}(\langle \circ \rangle)$ is used as a base case for the recursive definition.

The test in (2) offers the events in Z. If one of them is accepted by the SUT, then the test execution, which prioritises events of Σ over V, ensures that an event in Z occurs, and so the refusal of Z is not observed. As a result, the event *pass* occurs and the test case deadlocks. Otherwise, the event *inc* occurs and the test experiment terminates with that verdict, or, if termination is not possible, the event *fail* occurs and the test deadlocks with that alternative verdict. This reflects the fact that the forbidden trace $\langle refusal(Z)\rangle$ has been observed.

In the definition of the test case in (3), we rely on the fact that, in a test execution, ✓ has priority over *pass*. Thus, if the SUT can terminate then the test case behaves like *SKIP* after giving verdict *fail*, which is then the last event observed. Otherwise, *pass* occurs and then the test case deadlocks.

The recursive definitions in (4) and (5) consider traces of size three or more. For a trace starting with $\langle \circ, event(a)\rangle$, there is no refusal to test at first. Moreover, we cannot be sure that the SUT can perform a. Due to nondeterminism, it might not need to, and, whether or not a forbidden event can occur is captured by the tests for traces refinement; here, in testing for $\mathbf{conf}_{\mathbf{IO}}^{O}$, we are interested only in refusals. So, the test in (4) gives an inconclusive verdict before a, to indicate that the trace of events in $\langle \circ, event(a)\rangle \frown \rho$ has not yet been observed. If a occurs, we then test for ρ.

Finally, with the test in (5), we try to observe that Z is not a refusal of the SUT. If this is the case, the test deadlocks after the *inc* event, and so is successful: the forbidden trace is not observed. If, however, Z is refused, then a may be accepted, in which case we proceed with the test $T_F^{IO}(\rho)$. If a is not accepted either, again, the last event observed is *inc*.

Example 6.8. We consider $IC1$ and $\rho = \langle \circ, out, \{out\}, inp, \{out\}\rangle$ from Example 4.8. We obtain a test case as follows.

$$
\begin{array}{ll}
T_F^{IO}(\rho) = T_F^{IO}(\langle \circ, out, \{out\}, inp, \{out\}\rangle) & \\
= inc \rightarrow out \rightarrow T_F^{IO}(\langle \{out\}, inp, \{out\}\rangle) & [(4)] \\
= inc \rightarrow out \rightarrow ((\Box\, x : \{out\} \bullet x \rightarrow inc \rightarrow STOP) \,\Box\, inc \rightarrow inp \rightarrow T_F^{IO}(\langle \{out\}\rangle)) & [(5)] \\
= inc \rightarrow out \rightarrow (out \rightarrow inc \rightarrow STOP \,\Box\, inc \rightarrow inp \rightarrow T_F^{IO}(\langle \{out\}\rangle)) & [\text{simplification}] \\
= inc \rightarrow out \rightarrow (\; out \rightarrow inc \rightarrow STOP & [(2)] \\
\qquad\qquad \Box & \\
\qquad\qquad inc \rightarrow inp \rightarrow ((\Box\, x : \{out\} \bullet x \rightarrow pass \rightarrow STOP) \,\Box\, inc \rightarrow (SKIP \,\Box\, fail \rightarrow STOP)) \;) & \\
= inc \rightarrow out \rightarrow (out \rightarrow inc \rightarrow STOP \,\Box\, inc \rightarrow inp \rightarrow (out \rightarrow pass \rightarrow STOP \,\Box\, inc \rightarrow (SKIP \,\Box\, fail \rightarrow STOP))) & \\
\qquad\qquad [\text{simplification}] &
\end{array}
$$

□

There is no *inc* event after the last input, since, all events of the trace have been observed.

Example 6.9. We now suppose that we apply this test case to $IC2$. The following is a calculation of a process that describes one of the possible executions of this experiment, where we resolve the internal choice of $IC2$ in favour of the

first process in the definition of $IC2$. In the calculation, we repeatedly apply step laws of parallelism.

$$
\begin{aligned}
& IC2 \;[\![\, \Sigma \,]\!]\; T_F^{IO}(\rho) \\
& \\
&= inc \rightarrow out \rightarrow (\;\; inp \rightarrow inp \rightarrow STOP \\
&\qquad\qquad [\![\Sigma]\!] \\
&\qquad (out \rightarrow inc \rightarrow STOP \;\Box\; inc \rightarrow inp \rightarrow (out \rightarrow pass \rightarrow STOP \;\Box\; inc \rightarrow (SKIP \;\Box\; fail \rightarrow STOP))) \;) \\
& \\
&= inc \rightarrow out \rightarrow inc \rightarrow (\;\; inp \rightarrow inp \rightarrow STOP \\
&\qquad\qquad\qquad [\![\Sigma]\!] \\
&\qquad\qquad inp \rightarrow (out \rightarrow pass \rightarrow STOP \;\Box\; inc \rightarrow (SKIP \;\Box\; fail \rightarrow STOP)) \;) \\
& \\
&= inc \rightarrow out \rightarrow inc \rightarrow inp \rightarrow (inp \rightarrow STOP \;\; [\![\, \Sigma \,]\!] \;\; (out \rightarrow pass \rightarrow STOP \;\Box\; inc \rightarrow (SKIP \;\Box\; fail \rightarrow STOP))) \\
& \\
&= inc \rightarrow out \rightarrow inc \rightarrow inp \rightarrow inc \rightarrow (inp \rightarrow STOP \;\; [\![\, \Sigma \,]\!] \;\; (SKIP \;\Box\; fail \rightarrow STOP)) \\
& \\
&= inc \rightarrow out \rightarrow inc \rightarrow inp \rightarrow inc \rightarrow (STOP \sqcap fail \rightarrow (inp \rightarrow STOP \;\; [\![\, \Sigma \,]\!] \;\; STOP)) \\
& \\
&= inc \rightarrow out \rightarrow inc \rightarrow inp \rightarrow inc \rightarrow (STOP \sqcap fail \rightarrow STOP)
\end{aligned}
$$

The use of prioritise has no effect. Hiding Σ means that the only maximal traces of events we observe are $\langle inc, inc, inc \rangle$ and $\langle inc, inc, inc, fail \rangle$. This corresponds to a refusal trace that ends in $fail$ followed by a deadlock: a failed experiment. Now, we consider the application of the same test to an implementation of the first process in the choice in $IC1$.

$$
\begin{aligned}
& out \rightarrow inp \rightarrow (out \rightarrow STOP \;\Box\; inp \rightarrow STOP) \;\; [\![\, \Sigma \,]\!] \;\; T_F^{IO}(\rho) \\
& \\
&= inc \rightarrow out \rightarrow (\;\; inp \rightarrow (out \rightarrow STOP \;\Box\; inp \rightarrow STOP) \\
&\qquad\qquad [\![\Sigma]\!] \\
&\qquad (out \rightarrow inc \rightarrow STOP \;\Box\; inc \rightarrow inp \rightarrow (out \rightarrow pass \rightarrow STOP \;\Box\; inc \rightarrow (SKIP \;\Box\; fail \rightarrow STOP))) \;) \\
& \\
&= inc \rightarrow out \rightarrow inc \rightarrow (\;\; inp \rightarrow (out \rightarrow STOP \;\Box\; inp \rightarrow STOP) \\
&\qquad\qquad\qquad [\![\Sigma]\!] \\
&\qquad\qquad inp \rightarrow (out \rightarrow pass \rightarrow STOP \;\Box\; inc \rightarrow (SKIP \;\Box\; fail \rightarrow STOP)) \;) \\
& \\
&= inc \rightarrow out \rightarrow inc \rightarrow inp \rightarrow (\;\; (out \rightarrow STOP \;\Box\; inp \rightarrow STOP) \\
&\qquad\qquad\qquad\qquad [\![\Sigma]\!] \\
&\qquad\qquad\qquad (out \rightarrow pass \rightarrow STOP \;\Box\; inc \rightarrow (SKIP \;\Box\; fail \rightarrow STOP)) \;) \\
& \\
&= inc \rightarrow out \rightarrow inc \rightarrow inp \rightarrow (\;\; out \rightarrow (STOP \;\; [\![\, \Sigma \,]\!] \;\; pass \rightarrow STOP) \\
&\qquad\qquad\qquad\qquad \Box \\
&\qquad\qquad\qquad\qquad inc \rightarrow ((out \rightarrow STOP \;\Box\; inp \rightarrow STOP) \;\; [\![\, \Sigma \,]\!] \;\; (SKIP \;\Box\; fail \rightarrow STOP)) \;)
\end{aligned}
$$

In this case, application of the prioritise operator removes the possibility of the inc event in favour the out event in the external choice. This yields the process $inc \rightarrow out \rightarrow inc \rightarrow inp \rightarrow out \rightarrow pass \rightarrow STOP$ and so a successful verdict. □

As already said, the aim is for $T_F^{IO}(\rho)$ to produce a trace that ends with the event *fail*, when interacting with a process Q, if, and only if, Q can perform the sequence of events and refusals in ρ.

LEMMA 6.10. *Q fails $T_F^{IO}(\rho)$ if, and only if, $\rho \in rtraces[[Q]]$.*

This is proved in Appendix B. A test set is exhaustive if all incorrect implementations fail some test case. The lemma above suggests the following exhaustive test set for $\mathbf{conf}_{\mathbf{IO}}^{\mathcal{O}}$.

Definition 6.11. $Exhaust_{RT}^{\mathcal{O}}(P) \mathrel{\widehat{=}} \{t : traces[[P]];\ \rho : RTrace \mid \rho \in A_{IO}^{\mathcal{O}}(P,t) \bullet T_F^{IO}(\rho)\}$

The following shows that $Exhaust_{RT}^{\mathcal{O}}(P)$ is exhaustive for $\mathbf{conf}_{\mathbf{IO}}^{\mathcal{O}}$.

THEOREM 6.12. *If Q $\mathbf{conf}_{\mathbf{IO}}^{\mathcal{O}}$ P does not hold, there is some T in $Exhaust_{RT}^{\mathcal{O}}(P)$ such that Q fails T.*

PROOF.

$$\neg Q\ \mathbf{conf}_{\mathbf{IO}}^{\mathcal{O}}\ P$$

$$\Rightarrow \exists\, t : traces[[P]] \cap traces[[Q]] \bullet Ref_{IO}^{\mathcal{O}}(Q,t) \not\subseteq Ref_{IO}^{\mathcal{O}}(P,t) \qquad \text{[definition of } Q\ \mathbf{conf}_{\mathbf{IO}}^{\mathcal{O}}\ P\text{]}$$

$$\Rightarrow \exists\, t : traces[[P]] \cap traces[[Q]];\ \rho : RTrace \bullet \rho \in Ref_{IO}^{\mathcal{O}}(Q,t) \land \rho \notin Ref_{IO}^{\mathcal{O}}(P,t) \qquad \text{[property of sets]}$$

$$\Rightarrow \exists\, t : traces[[P]] \cap traces[[Q]];\ \rho : RTrace \bullet \qquad \text{[definition of } Ref_{IO}^{\mathcal{O}}(P,t)\text{]}$$
$$\rho \in \{\rho_1 : IOrtraces^{\mathcal{O}}[[Q]] \mid trace(\rho_1) = t\} \land$$
$$\rho \notin \{\rho_1 : IOrtraces^{\mathcal{O}}[[P]] \mid trace(\rho_1) = t\}$$

$$\Rightarrow \exists\, t : traces[[P]] \cap traces[[Q]];\ \rho : RTrace \bullet \rho \in IOrtraces^{\mathcal{O}}[[Q]] \land \rho \notin IOrtraces^{\mathcal{O}}[[P]] \land trace(\rho) = t$$
[property of sets]

$$\Rightarrow \exists\, t : traces[[P]] \cap traces[[Q]];\ \rho : RTrace \bullet \qquad \text{[RT5]}$$
$$iotrace(\rho) \in IOrtraces^{\mathcal{O}}[[Q]] \land iotrace(\rho) \notin IOrtraces^{\mathcal{O}}[[P]] \land trace(\rho) = t$$

$$\Rightarrow \exists\, t : traces[[P]] \cap traces[[Q]];\ \rho : RTrace \bullet \qquad \text{[definition of } A_{IO}^{\mathcal{O}}(P,t)\text{]}$$
$$iotrace(\rho) \in IOrtraces^{\mathcal{O}}[[Q]] \land iotrace(\rho) \notin IOrtraces^{\mathcal{O}}[[P]] \land trace(\rho) = t \land$$
$$(\exists\, \rho_1 : RTrace \bullet iortrace(\rho_1) \in A_{IO}^{\mathcal{O}}(P,t) \land \rho_1 \leq_{RT} iotrace(\rho))$$

$$\Rightarrow \exists\, t : traces[[P]] \cap traces[[Q]];\ \rho, \rho_1 : RTrace \bullet \qquad \text{[predicate calculus]}$$
$$iotrace(\rho) \in IOrtraces^{\mathcal{O}}[[Q]] \land iotrace(\rho) \notin IOrtraces^{\mathcal{O}}[[P]] \land trace(\rho) = t \land$$
$$iotrace(\rho_1) \in A_{IO}^{\mathcal{O}}(P,t) \land \rho_1 \leq_{RT} iotrace(\rho)$$

$$\Rightarrow \exists\, t : traces[[P]] \cap traces[[Q]];\ \rho, \rho_1 : RTrace \bullet \qquad \text{[definition of } iotrace\text{]}$$
$$iotrace(\rho) \in IOrtraces^{\mathcal{O}}[[Q]] \land iotrace(\rho) \notin IOrtraces^{\mathcal{O}}[[P]] \land trace(\rho) = t \land$$
$$iotrace(\rho_1) \in A_{IO}^{\mathcal{O}}(P,t) \land iotrace(\rho_1) \leq_{RT} iotrace(\rho)$$

$$\Rightarrow \exists\, t : traces[[P]] \cap traces[[Q]];\ \rho, \rho_1, \rho_2 : RTrace \bullet \qquad \text{[predicate calculus]}$$
$$iotrace(\rho) \in IOrtraces^{\mathcal{O}}[[Q]] \land iotrace(\rho) \notin IOrtraces^{\mathcal{O}}[[P]] \land trace(\rho) = t \land$$
$$\rho_2 = iortrace(\rho_1) \land \rho_2 \in A_{IO}^{\mathcal{O}}(P,t) \land \rho_2 \leq_{RT} iortrace(\rho)$$

$$\Rightarrow \exists\, t : traces[[P]] \cap traces[[Q]];\ \rho, \rho_1, \rho_2 : RTrace \bullet \qquad \text{[RT1]}$$
$$iotrace(\rho) \in IOrtraces^{\mathcal{O}}[[Q]] \land iotrace(\rho) \notin IOrtraces^{\mathcal{O}}[[P]] \land trace(\rho) = t \land$$
$$\rho_2 = iortrace(\rho_1) \land \rho_2 \in A_{IO}^{\mathcal{O}}(P,t) \bullet \rho_2 \leq_{RT} iortrace(\rho) \land \rho_2 \in IOrtraces^{\mathcal{O}}[[Q]]$$

$$\Rightarrow \exists\, t : traces[[P]] \cap traces[[Q]];\ \rho_2 : RTrace \bullet \rho_2 \in A^{O}_{IO}(P, t) \land \rho_2 \in IOrtraces^{O}[[Q]] \qquad \text{[predicate calculus]}$$

$$\Rightarrow \exists\, t : traces[[P]] \cap traces[[Q]];\ \rho_2 : RTrace \bullet \qquad \text{[definition of } Exhaust^{O}_{RT}(P)\text{]}$$
$$\rho_2 \in A^{O}_{IO}(P, t) \land \rho_2 \in IOrtraces^{O}[[Q]] \land T^{IO}_{F}(\rho_2) \in Exhaust^{O}_{RT}(P)$$

$$\Rightarrow \exists\, t : traces[[P]] \cap traces[[Q]];\ \rho_2 : RTrace \bullet T^{IO}_{F}(\rho_2) \in Exhaust^{O}_{RT}(P) \land Q \text{ fails } T^{IO}_{F}(\rho_2) \qquad \text{[Lemma 6.10]}$$

$$\Rightarrow \exists\, T : Exhaust^{O}_{RT}(P) \bullet Q \text{ fails } T \qquad \text{[predicate calculus]}$$

□

The result established by the above theorem means that, together with the exhaustive test set for traces refinement [7], the set $Exhaust^{O}_{RT}(P)$ provides exhaustive coverage for input-output refusal traces refinement.

The focus of this section has been the definition of an exhaustive set of test cases. However, there is scope to optimise the proposed approach; we did not consider these since they would complicate the proofs. For example, we can use different exhaustive test sets. For example, for traces t_1 and t_2 of P, with t_1 a proper prefix of t_2, we might have $\rho_1 \in A^{O}_{IO}(P, t_1)$ and $\rho_2 \in A^{O}_{IO}(P, t_2)$, where ρ_1 is a proper prefix of ρ_2. In such situations, there is no need to test with ρ_2. So, instead of using $A^{O}_{IO}(P, t)$ for each trace of events t of P, we might consider the set of minimal input-output refusal traces whose event trace is in $traces[[P]]$ and that are not input-output refusal traces of P to eliminate redundancy.

We might also use adaptive test cases, where the behaviour of the tester depends on previous observations. To illustrate the potential of this approach, we consider that a test case T is used, an input-output refusal trace ρ is observed, the next event in the test case is $a \in O$, and, instead, an output $b \neq a$ occurs, and T terminates with an inconclusive verdict. If there is another test case T' formed from ρ followed by b, we can combine T and T' to produce a single adaptive test case that behaves like T if a is output after ρ, and like T' if b is output instead.

Optimisations such as these are a topic for future work. Here, we have formalised a core notion of a test and associated exhaustive test set to establish a testing theory for input-output refusal traces refinement. Our theory is a solid basis to study the soundness of optimisations, and of test selection and generation techniques in general.

7 CONCLUSIONS

This paper addresses the important problem of model-based testing using CSP models with inputs and outputs. The theory is based on the refusal-testing model of CSP. We have presented a modern complete formalisation of this model, including healthiness conditions and definitions of core CSP operators. Our model gives an appropriate treatment to divergence and termination in line with the standard CSP models.

The work in [26] presents yet another refusal testing model. It admits empty refusal traces, and has a different set of healthiness conditions. The model is expected to be isomorphic to that of [29] and so to ours. Like ours, the healthiness conditions in [26] rule out the empty set of refusal traces. Divergence, deadlock, prefixing, and choice are defined in [26]; due to the different notion of valid sets of refusal traces, these definitions are different. In addition, [26] does not discuss healthiness of operators or termination, but to deal with temporal logic, [26] considers infinite traces.

Based on an earlier approach to inputs and outputs in CSP using the stable-failures model, we have defined an input-output refusal traces model for CSP. This includes the definition of healthiness conditions, the relationship with the refusal traces model, and a calculation of a definition for all the core CSP operators.

Inputs and outputs in CSP are also studied in [23]. The model in that work is for receptive processes, in which inputs and outputs are never blocked. So, unlike ours, that is a model for input-enabled processes, although it admits that

processes may not be ready to accept an input. If such an input is provided, however, instead of a deadlock, like in standard CSP, we have a divergence. The model in [23] includes no refusals, like the suspension-traces model. It is, however, a model for development, rather than testing, and includes a treatment of divergence. The notion of divergence, includes the possibility of an infinite number of outputs. Quiescence is not annotated. A complete set of algebraic laws is presented for this model. Algebraic laws of our (input-output) refusal-traces model is a topic of future work.

We have defined input-output refusal traces refinement in the expected way: subset inclusion. Using the refusal-traces model, we have also characterised the set of suspension traces of a CSP model. With that, we have been able to prove that input-output traces refinement is stronger than ioco. This guarantees that programs developed to conform to a given specification using input-output traces refinement can meaningfully be tested using ioco. Moreover, we note that input-output refusal traces refinement is not a relation restricted to input-enabled implementations.

In the context of the Unifying Theories of Programming (UTP) [22], the work in [43] studies the relationship between ioco and refinement for divergence-free reactive processes. For that, it presents a UTP theory of suspension traces enriched with refusals and divergence. It defines a model for key process operators, input-enabledness, and ioco in this predicative relational setting. It considers new operators for input-enabled processes and proves that ioco is stronger than refinement. This is probably because their model includes general refusals, unlike the pure suspension-trace model.

In the domain of operational models, the works in [18, 19] propose a testing theory based on the iocos relation, which is defined using the notion of simulation over an LTS with inputs and outputs. Like input-output refusal-traces refinement, iocos is a pre-order stronger than ioco. Moreover, it can be used both for refinement and testing. Despite the similarities, there are two essential differences. Quiescence is defined as the absence of outputs, and internal events are not modelled. Moreover, the modelling notation is a simplified subset of CCS [28], less expressive than CSP.

The definitions of the input-output failures and input-output refusal traces of the CSP operators are not congruences. So, they do not provide an appropriate model for the CSP operators in their full generality. There are two main issues, reflected in the restrictions we impose on parallelism and renaming when calculating their input-output semantics.

First, given a very different semantics of inputs and outputs like ours, renaming an input to an output, or vice-versa, is bound to lead to a model with a very different meaning. So, the reuse of models that can be achieved by renaming in CSP has to be restricted in the way we suggest here if we are distinguishing inputs and outputs.

Second, the semantics in this paper does not allow parallel composition in which the synchronisation set contains one or more outputs. This restriction is imposed because such a parallel composition allows outputs to be blocked and so can convert an unstable state into a stable state. We have noted that a semantics for testing should reflect the standard testing assumption, that outputs are not blocked, and the restriction imposed is consistent with this assumption.

As future work, it might be possible to extend the results regarding parallel composition to the case where including an output *out* in a synchronisation set, when composing processes P and Q, cannot lead to *out* being blocked. This is the case, for example, if P (or Q) cannot block *out*, that is, *out* is always possible in P. For example, if *out* is an input in P, then this property is guaranteed if P is input-enabled, a condition under which ioco becomes compositional [42].

Parallelism in CSP can be used to model conjunction of requirements as well as parallel designs. Currently, we can cater for conjunction of requirements on inputs, but not outputs. To deal with synchronisation of outputs in a general way, we are likely to need to resort to the original refusal-traces model. From a pragmatic perspective, we can take the view that testing models use parallelism to reflect concurrent designs.

Finally, we have presented a testing theory for the input-output refusal traces model. We have formalised the notions of test, test execution, verdict, and exhaustive test set. A theory for the refusal-traces model can be very similar. Although

we adopt the use of special verdict events of the standard CSP testing theory for refinement, our tests are very different. Also, our notion of test execution requires the use of priorities.

The use of a denotational semantics has many benefits. For example, it allows the use of a number of tools for reasoning about specifications. In addition, our semantics is compositional - this helps our techniques scale. CSP does have an operational (refusal-testing) semantics (implemented in its model checker FDR); the denotational semantics is congruent to it [34, page 252]. We have also proved how traces and failures can both be derived from our semantics (Theorems 3.12 and 3.15); this helps to justify our derivation of suspension traces from our semantics.

Our results establish a solid foundation for further work on practical and sound test selection and generation based on CSP models. For testing, an appropriate treatment of inputs and outputs is crucial. We now plan to develop a refinement technique, so that we can both develop and test programs using the input-output refusal traces model.

ACKNOWLEDGEMENT

The authors would like to thank James Baxter, Augusto Sampaio, and Jim Woodcock for useful discussions regarding both the technical material and the presentation of this paper. Ana Cavalcanti is funded by the UK EPSRC (Engineering and Physical Sciences Research Council) under Grants No EP/M025756/1 and EP/R025479/1, and by the Royal Academy of Engineering under Grant No CiET1718/45. Robert M. Hierons is funded by the EPSRC, under Grant No EP/R025134/1. Sidney Nogueira is funded by the CNPq (Brazilian National Research Council) under Grant No 249710/2013-7.

REFERENCES

[1] G. Barrett. 1995. Model checking in practice: The T9000 Virtual Channel Processor. *IEEE Transactions on Software Engineering* 21, 2 (1995), 69–78.

[2] M. Bijl, A. Rensink, and J. Tretmans. 2003. *Component Based Testing with ioco.* Technical Report. University of Twente. CTIT Technical Report TR CTIT 03 34.

[3] I. B. Bourdonov, A. Kossatchev, and V. V. Kuliamin. 2006. Formal Conformance Testing of Systems with Refused Inputs and Forbidden Actions. *Electronic Notes in Theoretical Computer Science* 164, 4 (2006), 83–96.

[4] E. Brinksma. 1988. A theory for the derivation of tests. In *Protocol Specification, testing and Verification VIII*. North-Holland, 63–74.

[5] E. Brinksma, L. Heerink, and J. Tretmans. 1998. Factorized Test Generation for Multi-Input/Output Transition Systems. In *11th IFIP Workshop on Testing of Communicating Systems*. Kluwer Academic Publishers, 67–82.

[6] L. B. Briones and E. Brinksma. 2005. Testing Real-Time Multi Input-Output Systems. In *7th International Conference on Formal Engineering Methods (Lecture Notes in Computer Science)*, Vol. 3785. Springer, 264–279.

[7] A. L. C. Cavalcanti and M.-C. Gaudel. 2007. Testing for Refinement in CSP. In *9th International Conference on Formal Engineering Methods (Lecture Notes in Computer Science)*, Vol. 4789. Springer-Verlag, 151–170. https://doi.org/10.1007/978-3-540-76650-6_10

[8] A. L. C. Cavalcanti and M.-C. Gaudel. 2014. Data Flow coverage for *Circus*-based testing. In *Fundamental Approaches to Software Engineering (Lecture Notes in Computer Science)*, Vol. 8441. 415–429. https://doi.org/10.1007/978-3-642-54804-8_29

[9] A. L. C. Cavalcanti, M.-C. Gaudel, and R. M. Hierons. 2011. Conformance Relations for Distributed Testing based on CSP. In *IFIP International Conference on Testing Software and Systems (Lecture Notes in Computer Science)*, B. Wolff and F. Zaidi (Eds.). Springer-Verlag. https://doi.org/10.1007/978-3-642-24580-0_5

[10] A. L. C. Cavalcanti, R. Hierons, and S. Nogueira. 2017. *Input and outputs in CSP: a model and a testing theory.* Technical Report. University of York, Department of Computer Science, York, UK. Available at `www-users.cs.york.ac.uk/~alcc/CHN17.pdf`.

[11] A. L. C. Cavalcanti and R. M. Hierons. 2012. *Testing with inputs and outputs in CSP - Extended version.* Technical Report. Available at www-users.cs.york.ac.uk/ alcc/CH12.pdf.

[12] A. L. C. Cavalcanti and R. M. Hierons. 2013. Testing with Inputs and Outputs in CSP. In *Fundamental Approaches to Software Engineering (Lecture Notes in Computer Science)*, Vol. 7793. 359–374. https://doi.org/10.1007/978-3-642-37057-1_26

[13] A. L. C. Cavalcanti, R. M. Hierons, S. Nogueira, and A. C. A. Sampaio. 2016. A Suspension-Trace Semantics for CSP. In *International Symposium on Theoretical Aspects of Software Engineering*. 3–13. https://doi.org/10.1109/TASE.2016.9 Invited paper.

[14] A. L. C. Cavalcanti, A. C. A. Sampaio, and J. C. P. Woodcock. 2003. A Refinement Strategy for *Circus*. *Formal Aspects of Computing* 15, 2 - 3 (2003), 146–181. https://doi.org/10.1007/s00165-003-0006-5

[15] A. Feliachi, M. C. Gaudel, M. Wenzel, and B. Wolff. 2013. The *Circus* Testing Theory Revisited in Isabelle/HOL. In *15th International Conference on Formal Engineering Methods (Lecture Notes in Computer Science)*, L. Groves and J. Sung (Eds.), Vol. 8144. Springer, 243–260.

[16] C. Fischer. 1998. How to Combine Z with a Process Algebra. In *The Z Formal Specification Notation*, J. Bowen, A. Fett, and M. Hinchey (Eds.). Springer-Verlag.
[17] T. Gibson-Robinson, P. Armstrong, A. Boulgakov, and A. W. Roscoe. 2014. FDR3 - A Modern Refinement Checker for CSP. In *Tools and Algorithms for the Construction and Analysis of Systems*. 187–201.
[18] C. Gregorio-Rodríguez, L. Llana, and R. Martínez. 2018. An axiomatic semantics for iocos conformance relation. *Journal of Logical and Algebraic Methods in Programming* (2018).
[19] C. Gregorio-Rodríguez, L. Llana, and R. Martínez-Torres. 2013. Input-Output Conformance Simulation (iocos) for Model Based Testing. In *Formal Techniques for Distributed Systems*, D. Beyer and M. Boreale (Eds.). Springer, 114–129.
[20] A. Hall and R. Chapman. 2002. Correctness by construction: Developing a commercial secure system. *IEEE Software* 19, 1 (2002), 18–25.
[21] L. Heerink and J. Tretmans. 1997. Refusal Testing for Classes of Transition Systems with Inputs and Outputs. In *Formal Description Techniques and Protocol Specification, Testing and Verification (IFIP Conference Proceedings)*, Vol. 107. Chapman & Hall, 23–38.
[22] C. A. R. Hoare and He Jifeng. 1998. *Unifying Theories of Programming*. Prentice-Hall.
[23] M. B. Josephs. 1992. Receptive Process Theory. *Acta Informatica* 29, 1 (1992), 17–31.
[24] T. Kahsai, M. Roggenbach, and B.-H. Schlingloff. 2007. Specification-based testing for refinement. In *5th IEEE International Conference on Software Engineering and Formal Methods*. IEEE Computer Society, 237–246.
[25] M. Krichen. 2010. A Formal Framework for Conformance Testing of Distributed Real-Time Systems. In *Principles of Distributed Systems*, C. Lu, T. Masuzawa, and M. Mosbah (Eds.). Lecture Notes in Computer Science, Vol. 6490. Springer, 139–142.
[26] G. Lowe. 2008. Specification of communicating processes: temporal logic versus refusals-based refinement. *Formal Aspects of Computing* 20, 3 (2008), 277–294.
[27] B. P. Mahony and J. S. Dong. 1998. Blending Object-Z and Timed CSP: An Introduction to TCOZ. In *The 20th International Conference on Software Engineering (ICSE'98)*. IEEE Computer Society Press, 95–104.
[28] R. Milner. 1989. *Communication and Concurrency*. Prentice-Hall.
[29] A. Mukarram. 1993. *A Refusal Testing Model for CSP*. Ph.D. Dissertation. Department of Computer Science, University of Oxford.
[30] S. Nogueira, A. C. A. Sampaio, and A. C. Mota. 2014. Test generation from state based use case models. *Formal Aspects of Computing* 26, 3 (2014), 441–490.
[31] I. Phillips. 1987. Refusal testing. *Theoretical Computer Science* 50, 3 (1987), 241–284.
[32] M. Roggenbach. 2006. CSP-CASL: a new integration of process algebra and algebraic specification. *Theoretical Computer Science* 354, 1 (2006), 42–71.
[33] A. W. Roscoe. 1998. *The Theory and Practice of Concurrency*. Prentice-Hall.
[34] A. W. Roscoe. 2011. *Understanding Concurrent Systems*. Springer.
[35] A. C. A. Sampaio, S. Nogueira, A. Mota, and Y. Isobe. 2014. Sound and mechanised compositional verification of input-output conformance. *Software Testing, Verification and Reliability* 24, 4 (2014), 289–319.
[36] S. Schneider and H. Treharne. 2002. Communicating B Machines. In *ZB'2002: Formal Specification and Development in Z and B (Lecture Notes in Computer Science)*, D. Bert, J. Bowen, M. Henson, and K. Robinson (Eds.), Vol. 2272. 416–435.
[37] J. Tretmans. 1992. *A formal approach to conformance testing*. Ph.D. Dissertation. University of Twente, Enschede, The Netherlands.
[38] J. Tretmans. 1996. Test Generation with Inputs, Outputs, and Quiescence. In *Tools and Algorithms for the Construction and Analysis of Systems (Lecture Notes in Computer Science)*, Vol. 1055. Springer-Verlag, 127–146.
[39] J. Tretmans. 1996. Test Generation with Inputs, Outputs and Repetitive Quiescence. *Software - Concepts and Tools* 17, 3 (1996), 103–120.
[40] J. Tretmans. 2008. Formal Methods and Testing. Springer-Verlag, Chapter Model Based Testing with Labelled Transition Systems, 1–38.
[41] J. Tretmans and E. Brinksma. 2003. TorX: Automated Model Based Testing. In *1st European Conference on Model-Driven Software Engineering*. 13–25.
[42] M. van der Bijl, A. Rensink, and J. Tretmans. 2004. Compositional Testing with ioco. In *Formal Approaches to Software Testing*, A. Petrenko and A. UlrichAndreas (Eds.). Lecture Notes in Computer Science, Vol. 2931. Springer, 86–100.
[43] M. Weiglhofer and B. Aichernig. 2010. Unifying Input Output Conformance. In *Unifying Theories of Programming*, A. Butterfield (Ed.). Lecture Notes in Computer Science, Vol. 5713. Springer, 181–201.
[44] T. A. C. Willemse. 2006. Heuristics for ioco-Based Test-Based Modelling. In *Formal Methods: Applications and Technology (Lecture Notes in Computer Science)*, L. Brim, B. Haverkort, M. Leucker, and J. van de Pol (Eds.), Vol. 4346. Springer, 132–147.
[45] J. C. P. Woodcock and J. Davies. 1996. *Using Z - Specification, Refinement, and Proof*. Prentice-Hall.

A AUXILIARY DEFINITIONS

Here, we define operators used in our work that are not directly available in Z.

$$odd\_, even\_ : \mathbb{P}\,\mathbb{N}$$

$$odd = \{ n : \mathbb{N} \mid n \bmod 2 \neq 0 \} \land even = \{ n : \mathbb{N} \mid n \bmod 2 = 0 \}$$

$$\_ \in_{RT} \_ : Observation \leftrightarrow Observation$$

$$\forall e, X : Observation \bullet e \in_{RT} X \Leftrightarrow e \in \operatorname{ran} event \land X \in \operatorname{ran} refusal \land event^{\sim} e \in refusal^{\sim} X$$

$$\_ \subseteq_{RT} \_ : Refusal \leftrightarrow Refusal$$

$$\forall X_1, X_2 : Refusal \bullet X_1 \subseteq_{RT} X_2 \Leftrightarrow X_1 \neq \circ \land X_2 \neq \circ \land refusal^{\sim} X_1 \subseteq refusal^{\sim} X_2$$

$$\_ \cup_{RT} \_ : \operatorname{ran} refusal \times \mathbb{P}(\operatorname{ran} event) \rightarrow Refusal$$

$$\forall X : \operatorname{ran} refusal;\ Y : \mathbb{P}(\operatorname{ran} event) \bullet X \cup_{RT} Y = refusal\,(refusal^{\sim} X \cup event^{\sim}(\!|\ Y\ |\!))$$

$$lastevent : RTrace \nrightarrow \Sigma^{\checkmark}$$

$$\operatorname{dom} lastevent = \{\rho : RTrace \mid \#\rho \geq 3\} \land \forall \rho : RTrace \mid \#\rho \geq 3 \bullet lastevent\ \rho = event^{\sim}(\rho\ (\#\rho - 1))$$

B ELECTRONIC APPENDIX: PROOF OF KEY RESULTS

We present proof for key lemmas and theorems in the paper. In these proofs we sometimes refer to results found in [10].

B.1 *SKIP*

Theorem 3.6. $P;\ SKIP = P$

Proof.

$rtraces[[P;\ SKIP]]$

$$\begin{array}{ll}
= & \{\rho : rtraces[[P]] \mid event(\checkmark) \notin ran\,\rho\} \cup \qquad \text{[definition of } rtraces[[P;\ Q]]\text{]} \\
& \{\phi : \operatorname{seq} Obs;\ \rho : RTrace \mid \phi \frown \langle \circ, event(\checkmark), \circ \rangle \in rtraces[[P]] \land \rho \in rtraces[[SKIP]] \bullet \phi \frown \rho\}
\end{array}$$

$$\begin{array}{ll}
= & \{\rho : rtraces[[P]] \mid event(\checkmark) \notin ran\,\rho\} \cup \qquad \text{[definition of } rtraces[[SKIP]]\text{]} \\
& \{\phi : \operatorname{seq} Obs;\ \rho : RTrace \mid \\
& \quad \phi \frown \langle \circ, event(\checkmark), \circ \rangle \in rtraces[[P]] \land (\rho = \langle \circ \rangle \lor (\exists X : Refusal \bullet \rho = \langle \circ, event(\checkmark), X \rangle)) \bullet \phi \frown \rho \\
& \}
\end{array}$$

$$\begin{array}{ll}
= & \{\rho : rtraces[[P]] \mid event(\checkmark) \notin ran\,\rho\} \cup \qquad \text{[property of sets]} \\
& \{\rho : RTrace \mid \exists \phi : \operatorname{seq} Obs;\ \rho_1 : RTrace \bullet \rho = \phi \frown \rho_1 \land \\
& \quad \phi \frown \langle \circ, event(\checkmark), \circ \rangle \in rtraces[[P]] \land (\rho_1 = \langle \circ \rangle \lor (\exists X : Refusal \bullet \rho_1 = \langle \circ, event(\checkmark), X \rangle)) \\
& \}
\end{array}$$

$$\begin{array}{ll}
= & \{\rho : rtraces[[P]] \mid event(\checkmark) \notin ran\,\rho\} \cup \qquad \text{[predicate calculus]} \\
& \{\rho : RTrace \mid \\
& \quad (\exists \phi : \operatorname{seq} Obs \bullet \phi \frown \langle \circ, event(\checkmark), \circ \rangle \in rtraces[[P]] \land \rho = \phi \frown \langle \circ \rangle) \lor \\
& \quad (\exists \phi : \operatorname{seq} Obs;\ X : Refusal \bullet \phi \frown \langle \circ, event(\checkmark), \circ \rangle \in rtraces[[P]] \land \rho = \phi \frown \langle \circ, event(\checkmark), X \rangle)\ ) \\
& \}
\end{array}$$

$$
\begin{array}{ll}
= & \{\rho : rtraces[[P]] \mid event(\checkmark) \notin ran\,\rho\} \cup \qquad \text{[RT1 and RT4]} \\
& \{\rho : rtraces[[P]] \mid \\
& \quad (\exists \phi : \text{seq } Obs \bullet \phi \frown \langle \circ, event(\checkmark), \circ \rangle \in rtraces[[P]] \land \rho = \phi \frown \langle \circ \rangle) \lor \\
& \quad (\exists \phi : \text{seq } Obs;\ X : Refusal \bullet \phi \frown \langle \circ, event(\checkmark), \circ \rangle \in rtraces[[P]] \land \rho = \phi \frown \langle \circ, event(\checkmark), X \rangle)\ ) \\
& \}
\end{array}
$$

$$
\begin{array}{ll}
= & \{\rho : rtraces[[P]] \mid event(\checkmark) \notin ran\,\rho\} \cup \qquad \text{[RT1]} \\
& \{\rho : rtraces[[P]] \mid \\
& \quad (\exists \phi : \text{seq } Obs \bullet \phi \frown \langle \circ, event(\checkmark), \circ \rangle \in rtraces[[P]] \land \rho = \phi \frown \langle \circ \rangle) \lor \\
& \quad (\exists \phi : \text{seq } Obs;\ X : Refusal \bullet \rho = \phi \frown \langle \circ, event(\checkmark), X \rangle) \\
& \}
\end{array}
$$

$$
\begin{array}{ll}
= & \{\rho : rtraces[[P]] \mid \qquad \text{[property of sets]} \\
& \quad event(\checkmark) \notin ran\,\rho \lor \\
& \quad (\exists \phi : \text{seq } Obs \bullet \phi \frown \langle \circ, event(\checkmark), \circ \rangle \in rtraces[[P]] \land \rho = \phi \frown \langle \circ \rangle) \lor \\
& \quad (\exists \phi : \text{seq } Obs;\ X : Refusal \bullet \rho = \phi \frown \langle \circ, event(\checkmark), X \rangle) \\
& \}
\end{array}
$$

$$
\begin{array}{ll}
= & \{\rho : rtraces[[P]] \mid \qquad \text{[RT3]} \\
& \quad event(\checkmark) \notin ran\,\rho \lor \\
& \quad (\exists \phi : \text{seq } Obs \bullet \phi \frown \langle \circ, event(\checkmark), \circ \rangle \in rtraces[[P]] \land \rho = \phi \frown \langle \circ \rangle \land event(\checkmark) \notin ran\,\rho) \lor \\
& \quad (\exists \phi : \text{seq } Obs;\ X : Refusal \bullet \rho = \phi \frown \langle \circ, event(\checkmark), X \rangle) \\
& \}
\end{array}
$$

$$= \{\rho : rtraces[[P]] \mid event(\checkmark) \notin ran\,\rho \lor (\exists \phi : \text{seq } Obs;\ X : Refusal \bullet \rho = \phi \frown \langle \circ, event(\checkmark), X \rangle)\}$$

[predicate calculus]

$$= rtraces[[P]] \qquad \text{[RT3]}$$

□

Theorem 3.7. $SKIP \,[\![\, Z \,]\!]\, SKIP = SKIP$

$$rtraces[[SKIP \,[\![\, Z \,]\!]\, SKIP]]$$

$$= \{\rho_1 : RTrace \mid \rho_2 \in rtraces[[SKIP]] \land \rho_3 \in rtraces[[SKIP]] \land \rho_1 \in (\rho_2 \,[\![\, Z \,]\!]_{RT}\, \rho_3)\}$$

[definition of $rtraces[[P \,[\![\, Z \,]\!]\, Q]]$]

$$
\begin{array}{ll}
= \{\rho_1 : RTrace \mid & (\rho_2 = \langle \circ \rangle \lor \exists X : Refusal \bullet \rho_2 = \langle \circ, event(\checkmark), X \rangle) \land \qquad \text{[definition of } rtraces[[SKIP]]\text{]} \\
& (\rho_3 = \langle \circ \rangle \lor \exists X : Refusal \bullet \rho_3 = \langle \circ, event(\checkmark), X \rangle) \land \\
& \rho_1 \in (\rho_2 \,[\![\, Z \,]\!]_{RT}\, \rho_3)\}
\end{array}
$$

$$
\begin{array}{ll}
= & \{\rho_1 : RTrace \mid \\
& \quad (\ \rho_2 = \langle\ \circ\ \rangle \land \rho_3 = \langle\ \circ\ \rangle \lor \\
& \quad\ \ \rho_2 = \langle\ \circ\ \rangle \land (\exists X : Refusal \bullet \rho_3 = \langle\ \circ, event(\checkmark), X\rangle) \lor \\
& \quad\ \ (\exists X : Refusal \bullet \rho_2 = \langle\ \circ, event(\checkmark), X\rangle) \land \rho_3 = \langle\ \circ\ \rangle \lor \\
& \quad\ \ (\exists X : Refusal \bullet \rho_2 = \langle\ \circ, event(\checkmark), X\rangle) \land \\
& \quad\ \ (\exists X : Refusal \bullet \rho_3 = \langle\ \circ, event(\checkmark), X\rangle)\ ) \land \\
& \quad \rho_1 \in (\rho_2\ [\![\ Z\ ]\!]_{RT}\ \rho_3) \\
& \}
\end{array}
$$

$$= \{\rho_1 : RTrace \mid \rho_1 \in \{\langle\ \circ\ \rangle\} \lor \rho_1 \in \{\langle\ \circ\ \rangle\} \lor \rho_1 \in \{\langle\ \circ\ \rangle\} \lor \rho_1 \in \{X : Refusal \mid \langle\ \circ, event(\checkmark), X\rangle\}\}$$

$$= \{\langle\ \circ\ \rangle\} \cup \{X : Refusal \mid \langle\ \circ, event(\checkmark), X\rangle\} \qquad \text{[predicate calculus]}$$

$$= rtraces[[SKIP]] \qquad \text{[definition of } rtraces[[SKIP]]\text{]}$$

B.2 Input-output refusal traces

Lemma 4.6. $iotrace(\rho) \in RTrace \Rightarrow \rho \in IORTrace$

Proof.

$$
\begin{array}{ll}
& iotrace(\rho_1) \in RTrace \\
= & \exists \rho_2 : RTrace \bullet \#\rho_1 = \#\rho_2 \land \forall i : 1\,..\,\#\rho_2 \bullet \qquad \text{[definition of } iotrace\text{]} \\
& \quad (even\ i \Rightarrow \rho_2\ i = \rho_1\ i) \land (odd\ i \Rightarrow \rho_1\ i =\circ \land\ \rho_2\ i =\circ \lor\ \rho_2\ i = refusal(refusal^{\sim}(\rho_1 i) \cup O)) \land \rho_2 \in RTrace \\
\Rightarrow & \exists \rho_2 : RTrace \bullet \#\rho_1 = \#\rho_2 \land \forall i : 1\,..\,\#\rho_2 \bullet \qquad \text{[definition of } RTrace \text{ and } \rho_2 \in RTrace\text{]} \\
& \quad (even\ i \lor \rho_1\ i =\circ \Rightarrow \rho_2\ i = \rho_1\ i) \land \\
& \quad (odd\ i \land \rho_1 \neq \circ \Rightarrow \rho_2\ i = refusal(refusal^{\sim}(\rho_1 i) \cup O) \land i < \#\rho_2 \Rightarrow event^{\sim}(\rho_2\ (i+1)) \notin O) \\
= & \exists \rho_2 : RTrace \bullet \#\rho_1 = \#\rho_2 \land \forall i : 1\,..\,\#\rho_2 \bullet \qquad \text{[}odd\ i \Rightarrow even\ (i+1) \land \rho_2\ (i+1) = \rho_1\ (i+1)\text{]} \\
& \quad (even\ i \lor \rho_1\ i =\circ \Rightarrow \rho_2\ i = \rho_1\ i) \land \\
& \quad (odd\ i \land \rho_1 \neq \circ \Rightarrow \rho_2\ i = refusal(refusal^{\sim}(\rho_1 i) \cup O) \land i < \#\rho_2 \Rightarrow event^{\sim}(\rho_1\ (i+1)) \notin O) \\
\Rightarrow & \forall i : 1\,..\,\#\rho_1 - 1 \bullet odd\ i \land \rho_1 \neq \circ \Rightarrow event^{\sim}(\rho_1\ (i+1)) \notin O \qquad \text{[}\rho_1 = \rho_2 \text{ and predicate calculus]} \\
= & \rho_1 \in IORTrace \qquad \text{[definition of } IORTrace\text{]}
\end{array}
$$

□

Lemma 4.10. $P \sqsubseteq_{IORT} Q \Rightarrow traces[[Q]] \subseteq traces[[P]]$.

Proof.

$$
\begin{array}{ll}
& P \sqsubseteq_{IORT} Q \\
= & IOrtraces^{O}[[Q]] \subseteq IOrtraces^{O}[[P]] \qquad \text{[definition of } \sqsubseteq_{IORT}\text{]}
\end{array}
$$

$= \forall\, \rho_1 : RTrace \bullet iotrace(\rho_1) \in rtraces[[Q]] \Rightarrow iotrace(\rho_1) \in rtraces[[P]]$

[definition of *IOrtraces* and property of sets]

$= \forall\, \rho_1 : RTrace \bullet (\exists\, \rho_2 : rtraces[[Q]] \bullet \rho_2 = iotrace(\rho_1)) \Rightarrow iotrace(\rho_1) \in rtraces[[P]]$ [predicate calculus]

$= \forall\, \rho_1 : RTrace;\ \rho_2 : rtraces[[Q]] \bullet \rho_2 = iotrace(\rho_1) \Rightarrow iotrace(\rho_1) \in rtraces[[P]]$ [predicate calculus]

$= \forall\, \rho_1 : RTrace;\ \rho_2 : rtraces[[Q]] \bullet \rho_2 = iotrace(\rho_1) \Rightarrow \rho_2 \in rtraces[[P]]$ [predicate calculus]

$= \forall\, \rho_2 : rtraces[[Q]] \bullet (\exists\, \rho_1 : RTrace \bullet \rho_2 = iotrace(\rho_1)) \Rightarrow \rho_2 \in rtraces[[P]]$ [predicate calculus]

$= \forall\, \rho_2 : rtraces[[Q]] \bullet \rho_2 \in \operatorname{ran} iotrace \Rightarrow \rho_2 \in rtraces[[P]]$ [property of functions]

$= \forall\, \rho_2 : RTrace \bullet \rho_2 \in rtraces[[Q]] \cap \operatorname{ran} iotrace \Rightarrow \rho_2 \in rtraces[[P]]$ [predicate calculus]

$= rtraces[[Q]] \cap \operatorname{ran} iotrace \subseteq rtraces[[P]]$ [property of sets]

$\Rightarrow trace(\!|\ rtraces[[Q]] \cap \operatorname{ran} iotrace\ |\!) \subseteq trace(\!|\ rtraces[[P]]\ |\!)$ [property of relational image]

$\Rightarrow trace(\!|\ rtraces[[Q]]\ |\!) \subseteq trace(\!|\ rtraces[[P]]\ |\!)$ [[10, Lemma E.3]]

$= traces[[Q]] \subseteq traces[[P]]$ [Theorem 3.12]

□

Lemma 4.12. $P \sqsubseteq_{IORT} Q \Rightarrow IOfailures^{O}(Q) \subseteq IOfailures^{O}(P)$.

Proof.

$P \sqsubseteq_{IORT} Q$

$= rtraces[[Q]] \cap \operatorname{ran} iotrace \subseteq rtraces[[P]]$ [as shown in the proof of Lemma 4.10]

$\Rightarrow \bigcup failure(\!|\ rtraces[[Q]] \cap \operatorname{ran} iotrace\ |\!) \subseteq \bigcup failure(\!|\ rtraces[[P]]\ |\!)$ [property of relational image and sets]

$= \{t : \operatorname{seq} \Sigma^{\checkmark};\ Z : \mathbb{P}\, \Sigma^{\checkmark} \mid (t, Z) \in failures[[Q]] \wedge O \subseteq Z\} \subseteq \bigcup failure(\!|\ rtraces[[P]]\ |\!)$ [[10, Lemma E.4]]

$\Rightarrow \{t : \Sigma^{\checkmark};\ Z : \mathbb{P}\, \Sigma^{\checkmark} \mid (t, Z) \in failures[[Q]] \wedge O \subseteq Z\} \subseteq failures[[P]]$ [Theorem 3.15]

$\Rightarrow \{t : \Sigma^{\checkmark};\ Z : \mathbb{P}\, \Sigma^{\checkmark} \mid (t, Z) \in failures[[Q]] \wedge O \subseteq Z\} \subseteq \{t : \Sigma^{\checkmark};\ Z : \mathbb{P}\, \Sigma^{\checkmark} \mid (t, Z) \in failures[[P]] \wedge O \subseteq Z\}$

[property of sets]

$= IOfailures^{O}(Q) \subseteq \{t : \Sigma^{\checkmark};\ Z : \mathbb{P}\, \Sigma^{\checkmark} \mid (t, Z) \in failures[[P]] \wedge O \subseteq Z\}$ [[10, Lemma E.5]]

$= IOfailures^{O}(Q) \subseteq IOfailures^{O}(P)$ [definition of $IOfailures^{O}(P)$ and properties of sets]

□

Theorem B.1.

$$IOrtraces^{O}[[P \,[\![\, Z \,]\!]\, Q]] = \bigcup\{\rho_1 : IOrtraces^{O}[[P]];\ \rho_2 : IOrtraces^{O}[[Q]] \bullet (\rho_1 \,[\![\, Z \,]\!]_{RT}\, \rho_2)\}$$

Proof. We prove that $IOrtraces^{O}_{M}(IOrtraces^{O}[[P \,[\![\, Z \,]\!]\, Q]])$ is equal to

$$IOrtraces^{O}_{M}(\bigcup\{\rho_1 : IOrtraces^{O}[[P]];\ \rho_2 : IOrtraces^{O}[[Q]] \bullet (\rho_1 \,[\![\, Z \,]\!]_{RT}\, \rho_2)\})$$

so that the result follows from [10, Theorem E.17], which is similar to Lemma 4.17, but applies to arbitrary healthy sets of input-output refusal traces.

$$IOrtraces^{O}_{M}(IOrtraces^{O}[[P \,[\![\, Z \,]\!]\, Q]])$$

$$= \{\rho : RTrace \mid iotrace(\rho) \in IOrtraces^{O}[[P \,[\![\, Z \,]\!]\, Q]] \wedge iotrace(\rho) = \rho\} \qquad \text{[Lemma 4.18]}$$

$$= \{\rho : RTrace \mid iotrace(iotrace(\rho)) \in rtraces[[P \,[\![\, Z \,]\!]\, Q]] \wedge iotrace(\rho) = \rho\}$$
$$\text{[definition of } IOrtraces^{O}[[P \,[\![\, Z \,]\!]_{RT}\, Z]]\text{]}$$

$$= \{\rho : RTrace \mid iotrace(\rho) \in rtraces[[P \,[\![\, Z \,]\!]\, Q]] \wedge iotrace(\rho) = \rho\} \qquad \text{[idempotence of } iotrace\text{]}$$

$$= \{\rho : RTrace \mid iotrace(\rho) \in \bigcup\{\rho_1 : rtraces[[P]];\ \rho_2 : rtraces[[Q]] \bullet (\rho_1 \,[\![\, Z \,]\!]_{RT}\, \rho_2)\} \wedge iotrace(\rho) = \rho\}$$
$$\text{[definition of } rtraces[[P \,[\![\, Z \,]\!]_{RT}\, Z]]\text{]}$$

$$= \{\rho : RTrace \mid \exists\, \rho_1 : rtraces[[P]];\ \rho_2 : rtraces[[Q]] \bullet iotrace(\rho) \in (\rho_1 \,[\![\, Z \,]\!]_{RT}\, \rho_2) \wedge iotrace(\rho) = \rho\}$$
$$\text{[property of set comprehension]}$$

$$\begin{aligned} = \ & \{\rho : RTrace \mid \exists\, \rho_1 : rtraces[[P]];\ \rho_2 : rtraces[[Q]] \bullet \\ & \quad iotrace(\rho) \in (\rho_1 \,[\![\, Z \,]\!]_{RT}\, \rho_2) \wedge iotrace(\rho) = \rho \wedge \\ & \quad \exists\, \rho_3, \rho_4 : RTrace \bullet \rho_3 \leq_{RT} \rho_1 \wedge \rho_4 \leq_{RT} \rho_2 \wedge \\ & \qquad iotrace(\rho) \in (\rho_3 \,[\![\, Z \,]\!]\, \rho_4) \wedge iotrace(\rho_3) = \rho_3 \wedge iotrace(\rho_4) = \rho_4 \\ & \} \end{aligned} \qquad \text{[[10, Lemma E.19]]}$$

$$\begin{aligned} = \ & \{\rho : RTrace \mid \exists\, \rho_1 : rtraces[[P]];\ \rho_2 : rtraces[[Q]] \bullet \\ & \quad iotrace(\rho) \in (\rho_1 \,[\![\, Z \,]\!]_{RT}\, \rho_2) \wedge iotrace(\rho) = \rho \wedge \\ & \quad \exists\, \rho_3 : rtraces[[P]];\ \rho_4 : rtraces[[Q]] \bullet \rho_3 \leq_{RT} \rho_1 \wedge \rho_4 \leq_{RT} \rho_2 \wedge \\ & \qquad iotrace(\rho) \in (\rho_3 \,[\![\, Z \,]\!]\, \rho_4) \wedge iotrace(\rho_3) = \rho_3 \wedge iotrace(\rho_4) = \rho_4 \\ & \} \end{aligned} \qquad \text{[}rtraces[[P]] \text{ and } rtraces[[Q]] \text{ are RT1]}$$

$$\begin{aligned} = \ & \{\rho : RTrace \mid iotrace(\rho) = \rho \wedge \exists\, \rho_3 : rtraces[[P]];\ \rho_4 : rtraces[[Q]] \bullet \\ & \quad (\exists\, \rho_1 : rtraces[[P]];\ \rho_2 : rtraces[[Q]] \bullet iotrace(\rho) \in (\rho_1 \,[\![\, Z \,]\!]_{RT}\, \rho_2) \wedge \rho_3 \leq_{RT} \rho_1 \wedge \rho_4 \leq_{RT} \rho_2) \wedge \\ & \quad iotrace(\rho) \in (\rho_3 \,[\![\, Z \,]\!]\, \rho_4) \wedge iotrace(\rho_3) = \rho_3 \wedge iotrace(\rho_4) = \rho_4 \\ & \} \end{aligned} \qquad \text{[predicate calculus]}$$

$$
\begin{array}{ll}
= & \{\rho : RTrace \mid iotrace(\rho) = \rho \land \exists\, \rho_3 : rtraces[[P]];\ \rho_4 : rtraces[[Q]] \bullet \\
& \quad iotrace(\rho) \in (\rho_3 \,[\![\, Z \,]\!]\, \rho_4) \land iotrace(\rho_3) = \rho_3 \land iotrace(\rho_4) = \rho_4 \\
& \} \\
& \qquad\qquad [\text{predicate calculus: take } \rho_1 \text{ and } \rho_2 \text{ to be } \rho_3 \text{ and } \rho_4] \\
= & \{\rho : RTrace \mid iotrace(\rho) = \rho \land \exists\, \rho_3, \rho_4 : RTrace \bullet \qquad [\text{predicate calculus}] \\
& \quad iotrace(\rho_3) \in rtraces[[P]] \land iotrace(\rho_4) \in rtraces[[Q]] \land \\
& \quad iotrace(\rho) \in (\rho_3 \,[\![\, Z \,]\!]\, \rho_4) \land iotrace(\rho_3) = \rho_3 \land iotrace(\rho_4) = \rho_4 \\
& \} \\
= & \{\rho : RTrace \mid iotrace(\rho) = \rho \land \exists\, \rho_3, \rho_4 : RTrace \bullet \qquad [\text{predicate calculus: take } \rho_5 \text{ and } \rho_6 \text{ to be } \rho_3 \text{ and } \rho_4] \\
& \quad iotrace(\rho_3) \in rtraces[[P]] \land iotrace(\rho_4) \in rtraces[[Q]] \land \\
& \quad iotrace(\rho) \in (\rho_3 \,[\![\, Z \,]\!]\, \rho_4) \land iotrace(\rho_3) = \rho_3 \land iotrace(\rho_4) = \rho_4 \\
& \quad \exists\, \rho_5, \rho_6 : RTrace \bullet \\
& \qquad iotrace(\rho_5) \in rtraces[[P]] \land iotrace(\rho_6) \in rtraces[[Q]] \land \\
& \qquad iotrace(\rho) \in (\rho_5 \,[\![\, Z \,]\!]\, \rho_6) \land \rho_3 \leq_{RT} \rho_5 \land \rho_4 \leq_{RT} \rho_6 \\
& \} \\
= & \{\rho : RTrace \mid iotrace(\rho) = \rho \land \exists\, \rho_5, \rho_6 : RTrace \bullet \qquad [\text{predicate calculus}] \\
& \quad (\exists\, \rho_3, \rho_4 : RTrace \bullet \\
& \qquad iotrace(\rho_3) \in rtraces[[P]] \land iotrace(\rho_4) \in rtraces[[Q]] \land \\
& \qquad iotrace(\rho) \in (\rho_3 \,[\![\, Z \,]\!]\, \rho_4) \land iotrace(\rho_3) = \rho_3 \land iotrace(\rho_4) = \rho_4 \land \rho_3 \leq_{RT} \rho_5 \land \rho_4 \leq_{RT} \rho_6) \land \\
& \quad iotrace(\rho_5) \in rtraces[[P]] \land iotrace(\rho_6) \in rtraces[[Q]] \land iotrace(\rho) \in (\rho_5 \,[\![\, Z \,]\!]\, \rho_6) \\
& \} \\
= & \{\rho : RTrace \mid iotrace(\rho) = \rho \land \exists\, \rho_5, \rho_6 : RTrace \bullet \\
& \quad (\exists\, \rho_3, \rho_4 : RTrace \bullet \\
& \qquad iotrace(\rho) \in (\rho_3 \,[\![\, Z \,]\!]\, \rho_4) \land iotrace(\rho_3) = \rho_3 \land iotrace(\rho_4) = \rho_4 \land \\
& \qquad \rho_3 \leq_{RT} \rho_5 \land \rho_4 \leq_{RT} \rho_6) \land \\
& \quad iotrace(\rho_5) \in rtraces[[P]] \land iotrace(\rho_6) \in rtraces[[Q]] \land iotrace(\rho) \in (\rho_5 \,[\![\, Z \,]\!]\, \rho_6) \\
& \} \\
& \qquad\qquad [\rho_3 \leq_{RT} \rho_5 \leq_{RT} iotrace(\rho_5) \text{ and RT1 imply } \rho_3 = iotrace(\rho_3) \in rtraces[[P]]] \\
& \qquad\qquad [\rho_4 \leq_{RT} \rho_6 \leq_{RT} iotrace(\rho_6) \text{ and RT1 imply } \rho_4 = iotrace(\rho_4) \in rtraces[[Q]]] \\
= & \{\rho : RTrace \mid iotrace(\rho) = \rho \land \exists\, \rho_5, \rho_6 : RTrace \bullet \qquad [[10, \text{Lemma E.19}]] \\
& \quad iotrace(\rho_5) \in rtraces[[P]] \land iotrace(\rho_6) \in rtraces[[Q]] \land iotrace(\rho) \in (\rho_5 \,[\![\, Z \,]\!]\, \rho_6) \\
& \} \\
= & \{\rho : RTrace \mid \exists\, \rho_5 : IOrtraces^{O}[[P]];\ \rho_6 : IOrtraces^{O}[[Q]] \bullet iotrace(\rho) \in (\rho_5 \,[\![\, Z \,]\!]\, \rho_6) \land iotrace(\rho) = \rho\} \\
& \qquad\qquad [\text{definition of } IOrtraces^{O}[[P]]]
\end{array}
$$

$= \{\rho : RTrace \mid iotrace(\rho) \in \bigcup\{\rho_5 : IOrtraces^O[[P]];\ \rho_6 : IOrtraces^O[[Q]] \bullet (\rho_5 \,[\![\, Z \,]\!]_{RT}\, \rho_6)\} \wedge iotrace(\rho) = \rho\}$ [property of sets]

$= IOrtraces^O_M(\bigcup\{\rho_5 : IOrtraces^O[[P]];\ \rho_6 : IOrtraces^O[[Q]] \bullet (\rho_5 \,[\![\, Z \,]\!]_{RT}\, \rho_6)\})$ [Lemma 4.18]

□

B.3 Suspension traces

Lemma 5.3. $\forall \sigma : STrace \bullet \sigma \in ST \Leftrightarrow [\sigma] \subseteq ST$

Proof. The reverse implication follows directly from the property $\sigma \in [\sigma]$ of equivalence classes. For the implication, we have the following.

$\sigma_1 \in ST \Rightarrow [\sigma_1] \subseteq ST$

$= \sigma_1 \in ST \Rightarrow \forall \sigma_2 : [\sigma_1] \bullet \sigma_2 \in ST$ [property of sets]

$= \forall \sigma_2 : [\sigma_1] \bullet \sigma_1 \in ST \Rightarrow \sigma_2 \in ST$ [predicate calculus]

$= \forall \sigma_2 : STrace \bullet \sigma_1 \in ST \wedge \sigma_1 \,(\sim_\delta)^*\, \sigma_2 \Rightarrow \sigma_2 \in ST$ [predicate calculus]

$= true$ [ST1]

□

Lemma 5.4. $ST_2 \subseteq ST_1 \Leftrightarrow \{\sigma : ST_2 \mid \neg\ (\langle \delta, \delta \rangle \text{ in } \sigma)\} \subseteq ST_1$

Proof. For every equivalence class $[\sigma]$, we use $\sigma \downarrow \delta$ to denote is canonical representative characterised by $\sigma \downarrow \delta \in [\sigma]$ and $\neg\ (\langle \delta, \delta \rangle \text{ in } \sigma \downarrow \delta)$.

$\{\sigma_1 : ST_2 \mid \neg\ (\langle \delta, \delta \rangle \text{ in } \sigma_1)\} \subseteq ST_1$

$= \{\sigma_1 : ST_2 \mid \exists \sigma_2 : STrace \bullet \sigma_2 \downarrow \delta = \sigma_1\ \} \subseteq ST_1$ [definition of $\sigma_2 \downarrow \delta$]

$= \forall \sigma_1 : ST_2 \bullet (\exists \sigma_2 : STrace \bullet \sigma_2 \downarrow \delta = \sigma_1) \Rightarrow \sigma_1 \in ST_1$ [property of sets]

$= \forall \sigma_1 : ST_2;\ \sigma_2 : STrace \bullet \sigma_2 \downarrow \delta = \sigma_1\ \Rightarrow \sigma_1 \in ST_1$ [predicate calculus]

$= \forall \sigma_2 : STrace \bullet \sigma_2 \downarrow \delta \in ST_2 \Rightarrow \sigma_2 \downarrow \delta \in ST_1$ [predicate calculus]

$= \forall \sigma_2 : STrace \bullet [\sigma_2 \downarrow \delta] \subseteq ST_2 \Rightarrow [\sigma_2 \downarrow \delta] \subseteq ST_1$ [Lemma 5.3]

$= \forall \sigma_2 : STrace \bullet [\sigma_2] \subseteq ST_2 \Rightarrow [\sigma_2] \subseteq ST_1$ [property of equivalence relations]

$= \forall \sigma_2 : STrace \bullet \sigma_2 \in ST_2 \Rightarrow \sigma_2 \in ST_1$ [Lemma 5.3]

$= ST_2 \subseteq ST_1$ [property of sets]

□

B.4 Testing

In the proof that our test cases perform as expected, we use a new notation to define processes: Q after $\langle X, e \rangle$. This is the process reached by Q after observing X and then e. Importantly, we have the following definition. (This is similar to

Roscoe's definition of after for traces and failures [34]).

$$rtraces[[Q \text{ after } \langle X, e\rangle]] = \{\rho \mid \langle X, e\rangle \frown \rho \in rtraces[[Q]]\}$$

If $rtraces[[Q \text{ after } \langle X, e\rangle]]$ is non-empty, that is, if there are traces of Q starting with $\langle X, e\rangle$ then the corresponding set of refusal traces satisfies the healthiness conditions as established by the lemma below.

Lemma B.2. *If $\langle X, e\rangle \frown \rho \in rtraces[[Q]]$, then $rtraces[[Q \text{ after } \langle X, e\rangle]]$ is healthy.*

It is proved in [10, Appendix C]. For our lemma below on test failure, we use a formalisation of the refusal traces of the prioritise(P, θ) operator in terms of a relation ρ_1 prioritise$_\theta$ ρ_2 between refusal traces as follows.

$$\text{prioritise}(P, \theta) = \{\rho_1, \rho_2 : RTrace \mid \rho_2 \in rtraces[[P]] \land \rho_1 \text{ prioritise}_\theta\ \rho_2 \bullet \rho_1\}$$

We require the following properties of prioritise$_\theta$.

(1) $\langle\ \circ\ \rangle$ prioritise$_\theta$ $\langle X\rangle$
(2) $\langle X_1\rangle$ prioritise$_\theta$ $\langle X_2\rangle$ provided $X_1 \subseteq_{RT} X_2$
(3) $\langle\ \circ, v\rangle \frown \rho_1$ prioritise$_\theta$ $\langle \Sigma, v\rangle \frown \rho_2$ if ρ_1 prioritise$_\theta$ ρ_2 for $\theta = \langle \Sigma, V\rangle$ and $v \in V$.
(4) $\langle X, v\rangle \frown \rho_1$ prioritise$_\theta$ $\langle \Sigma, v\rangle \frown \rho_2$ if $X \subseteq_{RT} \Sigma$ and ρ_1 prioritise$_\theta$ ρ_2 for $\theta = \langle \Sigma, V\rangle$ and $v \in V$.
(5) $(\langle X_1, e_1\rangle \frown \rho_1)$ prioritise$_\theta$ $(\langle X_2, e_1\rangle \frown \rho_2)$ if e_1 is maximal, $X_1 \subseteq_{RT} X_2 \cup_{RT} \{e_2 : \Sigma^\checkmark \mid e_2 \leq_\theta e_1\}$ and ρ_1 prioritise$_\theta$ ρ_2.
(6) $\langle\ \circ, \checkmark\rangle \frown \rho_1$ prioritise$_\theta$ $\langle\ \circ, \checkmark\rangle \frown \rho_2$ if ρ_1 prioritise$_\theta$ ρ_2

We write $e_2 \leq_\theta e_1$ when the priority of the event e_2 is strictly lower than that of e_1 according to the sequence θ. We assume that $\checkmark$ always has maximal priority.

Lemma 6.10. *Q fails $T_F^{IO}(\rho)$ if, and only if, $\rho \in rtraces[[Q]]$.*

Proof. By induction on ρ. For clarity, we omit the *Obs* constructors when applied to specific values, like *fail* or $\Sigma^\checkmark$.

Case $\langle\ \circ\ \rangle$ *($\Rightarrow$).* $\langle\ \circ\ \rangle \in rtraces[[Q]]$ by MRT0.

Case $\langle\ \circ\ \rangle$ *($\Leftarrow$).*

$\langle\ \circ\ \rangle \in rtraces[[Q]]$

$\Rightarrow rtraces[[fail \rightarrow STOP]] \subseteq rtraces[[Q\ [\![\ \Sigma\ ]\!]\ T_F^{IO}(\langle\ \circ\ \rangle)]]$ [property of $rtraces[[Q\ [\![\ \Sigma\ ]\!]\ T_F^{IO}(\langle\ \circ\ \rangle)]]$]

$\Rightarrow \langle \Sigma, fail, \Sigma^\checkmark \cup V\rangle \in rtraces[[Q\ [\![\ \Sigma\ ]\!]\ T_F^{IO}(\langle\ \circ\ \rangle)]]$ [definition of $rtraces[[fail \rightarrow STOP]]$]

$\Rightarrow \langle \Sigma, fail, \Sigma^\checkmark \cup V\rangle \in rtraces[[\text{prioritise}(Q\ [\![\ \Sigma\ ]\!]\ T_F^{IO}(\langle\ \circ\ \rangle), \langle \Sigma, V\rangle)]]$

[definition of $rtraces[[\text{prioritise}(P, \theta)]]$ and $\langle \Sigma, fail, \Sigma^\checkmark \cup V\rangle$ prioritise$_\theta$ $\langle \Sigma, fail, \Sigma^\checkmark \cup V\rangle$ by (4) and (2)]

$\Rightarrow \langle \Sigma, fail, \Sigma^\checkmark \cup V\rangle \in rtraces[[\text{prioritise}(Q\ [\![\ \Sigma\ ]\!]\ T_F^{IO}(\langle\ \circ\ \rangle), \langle \Sigma, V\rangle) \setminus \Sigma]]$

$[\langle \Sigma, fail, \Sigma^\checkmark \cup V\rangle \in \langle \Sigma, fail, \Sigma^\checkmark \cup V\rangle \setminus_{RT} \Sigma]$

$\Rightarrow Q$ fails $T_F^{IO}(\langle\ \circ\ \rangle)$ [definition of fails]

Case $\langle X \rangle$ *(*$\Rightarrow$*).*

Q fails $T_F^{IO}(\langle X \rangle)$

$\Rightarrow \exists\, \rho : RTrace \bullet \rho \frown \langle fail, \Sigma^{\checkmark} \cup V \rangle \in rtraces[[\text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle X \rangle), \langle \Sigma, V \rangle) \setminus \Sigma]]$

[definition of fails, $\forall\, \rho : RTrace \bullet \rho \frown \langle fail, \circ, \checkmark, \circ \rangle \notin rtraces[[\, T_F^{IO}(\langle X \rangle)]]$, and definition of parallelism]

$\Rightarrow \langle \circ, inc, \circ, fail, \Sigma^{\checkmark} \cup V \rangle \in rtraces[[\text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle X \rangle), \langle \Sigma, V \rangle) \setminus \Sigma]]$

[RT1, $fail \notin initials(T_F^{IO}(\langle X \rangle))$, $fail \notin initials(T_F^{IO}(\langle X \rangle) \text{ after } \langle \circ, event\, x \rangle)$ for $x \in_{RT} X$,]

[$fail \in initials(T_F^{IO}(\langle X \rangle) \text{ after } \langle \circ, inc \rangle)$ and $fail \notin \bigcup \text{ran}(\!|\, rtraces[[\, T_F^{IO}(\langle X \rangle) \text{ after } \langle \circ, inc \rangle \text{ after } \langle \circ, fail \rangle]] \,|\!)$]

$\Rightarrow \exists\, \rho : rtraces[[\text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle X \rangle), \langle \Sigma, V \rangle)]] \bullet \langle \circ, inc, \circ, fail, \Sigma^{\checkmark} \cup V \rangle \in \rho \setminus_{RT} \Sigma$ [definition of hiding]

$\Rightarrow \exists\, \rho : rtraces[[\text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle X \rangle), \langle \Sigma, V \rangle)]];\ X_1, X_2, X_3 : Refusal \bullet$ [property of $\setminus_{RT}$]
$\rho = \langle X_1, inc, X_2, fail, X_3 \rangle \wedge \langle \circ, inc, \circ, fail, \Sigma^{\checkmark} \cup V \rangle \in \langle X_1, inc, X_2, fail, X_3 \rangle \setminus_{RT} \Sigma$

$\Rightarrow \exists\, \rho : rtraces[[\text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle X \rangle), \langle \Sigma, V \rangle)]];\ X_1, X_2, X_3 : Refusal \bullet$ [definition of $\setminus_{RT}$]
$\rho = \langle X_1, inc, X_2, fail, X_3 \rangle \wedge (X_1 = \circ \vee \neg\, (\Sigma \subseteq X_1)) \wedge (X_2 = \circ \vee \neg\, (\Sigma \subseteq X_2)) \wedge X_3 = \Sigma^{\checkmark} \cup V$

$\Rightarrow \langle \circ, inc, \circ, fail, \Sigma^{\checkmark} \cup V \rangle \in rtraces[[\text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle X \rangle), \langle \Sigma, V \rangle)]]$ [predicate calculus and RT1]

$\Rightarrow \langle \Sigma, inc, \Sigma, fail, \Sigma^{\checkmark} \cup V \rangle \in rtraces[[\, Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle X \rangle)]]$

[$\langle \circ, inc, \circ, fail, \Sigma^{\checkmark} \cup V \rangle$ prioritise$_\theta$ $\langle \Sigma, inc, \Sigma, fail, \Sigma^{\checkmark} \cup V \rangle$ by (3) and (2)]

$\Rightarrow \exists\, \rho : rtraces[[Q]] \bullet X \subseteq_{RT} \rho\, 1$

[definition of $rtraces[[\, Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle X \rangle)]]$ and $\langle refusal(\{e\}) \rangle \notin rtraces[[\, T_F^{IO}(\langle X \rangle)]]$ for any $event\ e \in_{RT} X$]

$\Rightarrow \langle X \rangle \in rtraces[[Q]]$ [RT1]

Case $\langle X \rangle$ *(*$\Leftarrow$*).* Similar to that for $\langle \circ \rangle$.

Case $\langle \circ, \checkmark, X \rangle$ *(*$\Rightarrow$*).*

Q fails $T_F^{IO}(\langle \circ, \checkmark, X \rangle)$

$\Rightarrow \exists\, \rho : RTrace \bullet \rho \frown \langle fail, \circ, \checkmark, \circ \rangle \in rtraces[[\text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle \circ, \checkmark, X \rangle), \langle \Sigma, V \rangle) \setminus \Sigma]]$ [definition of fails,]

[definition of $rtraces[[\, Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle \circ, \checkmark, X \rangle)]]$ (*pass* is not refused by $T_F^{IO}(\langle \circ, \checkmark, X \rangle)$,)]

[$\langle \circ, \checkmark, \circ \rangle$ prioritise$_\theta$ $\langle \circ, \checkmark, \circ \rangle$ (by (6) and (1)), $\checkmark \notin \Sigma$, and definition of $\setminus_{RT}$]

$\Rightarrow \langle \circ, fail, \circ, \checkmark, \circ \rangle \in rtraces[[\text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle \circ, \checkmark, X \rangle), \langle \Sigma, V \rangle) \setminus \Sigma]]$

[RT1, $fail \in initials(T_F^{IO}(\langle \circ, \checkmark, X \rangle))$, and $fail \notin \bigcup \text{ran}(\!|\, traces[[\, T_F^{IO}(\langle \circ, \checkmark, X \rangle) \text{ after } \langle fail \rangle \,]\!|)]]$

$\Rightarrow \exists\, \rho : rtraces[[\text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle\, \circ, \checkmark, X\rangle), \langle \Sigma, V\rangle)]] \bullet \langle\, \circ, fail, \circ, \checkmark, \circ\, \rangle \in \rho \setminus_{RT} \Sigma$ [definition of hiding]

$\Rightarrow \exists\, \rho : rtraces[[\text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle\, \circ, \checkmark, X\rangle), \langle \Sigma, V\rangle)]];\ X_1, X_2, X_3 : Refusal \bullet$ [property of $\setminus_{RT}$]
$\rho = \langle X_1, fail, X_2, \checkmark, X_3\rangle \wedge \langle\, \circ, fail, \circ, \checkmark, \circ\, \rangle \in \langle X_1, fail, X_2, \checkmark, X_3\rangle \setminus_{RT} \Sigma$

$\Rightarrow \exists\, \rho : rtraces[[\text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle\, \circ, \checkmark, X\rangle), \langle \Sigma, V\rangle)]];\ X_1, X_2, X_3 : Refusal \bullet$ [definition of $\setminus_{RT}$]
$\rho = \langle X_1, fail, X_2, \checkmark, X_3\rangle \wedge X_1 = \circ \wedge X_2 = \circ \wedge X_3 = \circ$

$\Rightarrow \langle\, \circ, fail, \circ, \checkmark, \circ\, \rangle \in rtraces[[\text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle\, \circ, \checkmark, X\rangle), \langle \Sigma, V\rangle)]]$ [predicate calculus]

$\Rightarrow \langle \Sigma, fail, \circ, \checkmark, \circ\, \rangle \in rtraces[[Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle\, \circ, \checkmark, X\rangle)]]$

[$\langle\, \circ, fail, \circ, \checkmark, \circ\, \rangle$ prioritise$_\theta$ $\langle \Sigma, fail, \circ, \checkmark, \circ\, \rangle$ by (3), (6), and (1)]

$\Rightarrow \exists\, X_1 : Refusal \bullet \langle\, \circ, \checkmark, X_1\rangle \in rtraces[[Q]]$ [definition of $rtraces[[Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle\, \circ, \checkmark, X\rangle)]]$]

$\Rightarrow \langle\, \circ, \checkmark, X\rangle \in rtraces[[Q]]$ [RT4 and RT1]

Case $\langle\, \circ, \checkmark, X\rangle$ ($\Leftarrow$).

$\langle\, \circ, \checkmark, X\rangle \in rtraces[[Q]]$

$\Rightarrow rtraces[[fail \rightarrow SKIP]] \subseteq rtraces[[Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle\, \circ, \checkmark, X\rangle)]]$ [property of $rtraces[[Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle\, \circ, \checkmark, X\rangle)]]$]

$\Rightarrow \langle \Sigma, fail, \circ, \checkmark, \circ\, \rangle \in rtraces[[Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle\, \circ, \checkmark, X\rangle)]]$ [definition of $rtraces[[fail \rightarrow SKIP]]$]

$\Rightarrow \langle \Sigma, fail, \circ, \checkmark, \circ\, \rangle \in rtraces[[\text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle\, \circ, \checkmark, X\rangle), \langle \Sigma, V\rangle)]]$

[$\langle \Sigma, fail, \circ, \checkmark, \circ\, \rangle$ prioritise$_\theta$ $\langle \Sigma, fail, \circ, \checkmark, \circ\, \rangle$ by (4), (6), and (1)]

$\Rightarrow \langle \Sigma, fail, \circ, \checkmark, \circ\, \rangle \in rtraces[[\text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle\, \circ, \checkmark, X\rangle), \langle \Sigma, V\rangle) \setminus \Sigma]]$

[$\langle \Sigma, fail, \circ, \checkmark, \circ\, \rangle \in \langle \Sigma, fail, \circ, \checkmark, \circ\, \rangle \setminus_{RT} \Sigma$]

$\Rightarrow Q$ fails $T_F^{IO}(\langle\, \circ, \checkmark, X\rangle)$ [definition of fails]

Case $\langle\, \circ, e\rangle \frown \rho$. Similar to that for $\langle X, e\rangle \frown \rho_1$ shown below.

Case $\langle X, e\rangle \frown \rho_1$.

$\langle X, e\rangle \frown \rho_1 \in rtraces[[Q]]$

$= \rho_1 \in rtraces[[Q \text{ after } \langle X, e\rangle]]$ [definition of after]

$= (Q \text{ after } \langle X, e\rangle) \text{ fails } T_F^{IO}(\rho_1)$ [induction hypothesis]

$= \exists\, \rho_2 : RTrace \bullet \{\rho_2 \frown \langle fail, \Sigma^\checkmark \cup V\rangle, \rho_2 \frown \langle fail, \circ, \checkmark, \circ\, \rangle\} \cap rtraces[[Execution(Q \text{ after } \langle X, e\rangle, T_F^{IO}(\rho_1))]] \neq \emptyset$

[definition of fails]

$= \exists\, \rho_2 : RTrace \bullet \quad \{\rho_2 \frown \langle fail, \Sigma^{\checkmark} \cup V\rangle, \rho_2 \frown \langle fail, \circ, \checkmark, \circ\rangle\} \cap \quad$ [definition of $Execution(Q \text{ after } \langle X, e\rangle, T_F^{IO}(\rho_1))$]
$rtraces[[\text{prioritise}(Q \text{ after } \langle X, e\rangle \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\rho_1), \langle \Sigma, V\rangle) \setminus \Sigma]] \neq \emptyset$

$= \exists\, \rho_2 : RTrace \bullet$ [refusal traces of hiding and priority]
$\{\rho_2 \frown \langle fail, \Sigma^{\checkmark} \cup V\rangle, \rho_2 \frown \langle fail, \circ, \checkmark, \circ\rangle\} \cap$
$\bigcup (\!|\, \{\rho_3, \rho_4 : RTrace \mid \rho_4 \in rtraces[[Q \text{ after } \langle X, e\rangle \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\rho_1)]] \wedge \rho_3 \text{ prioritise}_\theta\ \rho_4 \bullet \rho_3\} \,|\!)\, \setminus_{RT} \Sigma \neq \emptyset$

$= \exists\, \rho_2 : RTrace \bullet$ [refusal traces of after, parallelism, and prefixing]
$\{\rho_2 \frown \langle fail, \Sigma^{\checkmark} \cup V\rangle, \rho_2 \frown \langle fail, \circ, \checkmark, \circ\rangle\} \cap$
$\bigcup (\!|\, \{\rho_3, \rho_4 : RTrace \mid$
$\langle X, e\rangle \frown \rho_4 \in rtraces[[Q \,[\![\, \Sigma \,]\!]\, e \longrightarrow T_F^{IO}(\rho_1)]] \wedge \rho_3 \text{ prioritise}_\theta\ \rho_4$
$\bullet\ \rho_3$
$\} \,|\!)\, \setminus_{RT} \Sigma \neq \emptyset$

$= \exists\, \rho_2 : RTrace \bullet$ [refusal traces of parallelism and prefixing, and $inc \notin \alpha Q$]
$\{\rho_2 \frown \langle fail, \Sigma^{\checkmark} \cup V\rangle, \rho_2 \frown \langle fail, \circ, \checkmark, \circ\rangle\} \cap$
$\bigcup (\!|\, \{\rho_3, \rho_4 : RTrace \mid$
$\langle \Sigma, inc, X, e\rangle \frown \rho_4 \in rtraces[[Q \,[\![\, \Sigma \,]\!]\, inc \longrightarrow e \longrightarrow T_F^{IO}(\rho_1)]] \wedge \rho_3 \text{ prioritise}_\theta\ \rho_4$
$\bullet\ \rho_3$
$\} \,|\!)\, \setminus_{RT} \Sigma \neq \emptyset$

$= \exists\, \rho_2 : RTrace \bullet$ [definition of $T_F^{IO}(\langle X, e\rangle \frown \rho_1)$]
$\{\rho_2 \frown \langle fail, \Sigma^{\checkmark} \cup V\rangle, \rho_2 \frown \langle fail, \circ, \checkmark, \circ\rangle\} \cap$
$\bigcup (\!|\, \{\rho_3, \rho_4 : RTrace \mid$
$\langle \Sigma, inc, X, e\rangle \frown \rho_4 \in rtraces[[Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle X, e\rangle \frown \rho_1)]] \wedge \rho_3 \text{ prioritise}_\theta\ \rho_4$
$\bullet\ \rho_3$
$\} \,|\!)\, \setminus_{RT} \Sigma \neq \emptyset$

$= \exists\, \rho_5 : RTrace \bullet$ [property of $\setminus_{RT} \Sigma$ and take $\rho_5 = \Sigma, inc, \circ, e\rangle \frown \rho_2$]
$\{\rho_5 \frown \langle fail, \Sigma^{\checkmark} \cup V\rangle, \rho_5 \frown \langle fail, \circ, \checkmark, \circ\rangle\} \cap$
$\bigcup (\!|\, \{\rho_3, \rho_4 : RTrace \mid$
$\langle \Sigma, inc, X, e\rangle \frown \rho_4 \in rtraces[[Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle X, e\rangle \frown \rho_1)]] \wedge \rho_3 \text{ prioritise}_\theta\ \rho_4$
$\bullet\ \langle \Sigma, inc, X, e\rangle \frown \rho_3$
$\} \,|\!)\, \setminus_{RT} \Sigma \neq \emptyset$

$= \exists\, \rho_5 : RTrace \bullet$ [properties (4) and (5) of prioritise_θ]
$\{\rho_5 \frown \langle fail, \Sigma^{\checkmark} \cup V\rangle, \rho_5 \frown \langle fail, \circ, \checkmark, \circ\rangle\} \cap$
$\bigcup (\!|\, \{\rho_3, \rho_4 : RTrace \mid$
$\langle \Sigma, inc, X, e\rangle \frown \rho_4 \in rtraces[[Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle X, e\rangle \frown \rho_1)]] \wedge$
$\langle \Sigma, inc, X, e\rangle \frown \rho_3 \text{ prioritise}_\theta\ \langle \Sigma, inc, X, e\rangle \frown \rho_4$
$\bullet\ \langle \Sigma, inc, X, e\rangle \frown \rho_3$
$\} \,|\!)\, \setminus_{RT} \Sigma \neq \emptyset$

$= \quad \exists\, \rho_5 : RTrace \bullet$ [property of sets (take $\rho_6 = \langle \Sigma, inc, X, e\rangle \frown \rho_3$ and $\rho_7 = \langle \Sigma, inc, X, e\rangle \frown \rho_4$)]
$\{\rho_5 \frown \langle fail, \Sigma^{\checkmark} \cup V\rangle, \rho_5 \frown \langle fail, \circ, \checkmark, \circ\rangle\} \cap$
$\bigcup (\!| \{\rho_6, \rho_7 : RTrace \mid \rho_7 \in rtraces[[Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle X, e\rangle \frown \rho_1)]] \wedge \rho_6 \text{ prioritise}_\theta\ \rho_7 \bullet \rho_6\} |!) \setminus_{RT} \Sigma \neq \emptyset$

$=$ $\exists\, \rho_5 : RTrace \bullet$ $\{\rho_5 \frown \langle fail, \Sigma^{\checkmark} \cup V\rangle, \rho_5 \frown \langle fail, \circ, \checkmark, \circ\rangle\} \cap$ [refusal traces of hiding and priority]
$rtraces[[\text{prioritise}(Q \,[\![\, \Sigma \,]\!]\, T_F^{IO}(\langle X, e\rangle \frown \rho_1), \langle \Sigma, V\rangle) \setminus \Sigma]] \neq \emptyset$

$=$ $\exists\, \rho_5 : RTrace \bullet$ $\{\rho_5 \frown \langle fail, \Sigma^{\checkmark} \cup V\rangle, \rho_5 \frown \langle fail, \circ, \checkmark, \circ\rangle\} \cap$ [definition of $Execution(Q, T_F^{IO}(\langle X, e\rangle \frown \rho_1))$]
$rtraces[[Execution(Q, T_F^{IO}(\rho_1))]] \neq \emptyset$

$= Q$ fails $T_F^{IO}(\langle X, e\rangle \frown \rho_1)$ [definition of fails]

□