Talking quiescence: a rigorous theory that supports parallel composition, action hiding and determinisation

Gerjan Stokkink, Mark Timmer, and Mariëlle Stoelinga
Formal Methods and Tools, Faculty of EEMCS
University of Twente, The Netherlands
{w.g.j.stokkink, timmer, marielle}@cs.utwente.nl

The notion of quiescence — the absence of outputs — is vital in both behavioural modelling and testing theory. Although the need for quiescence was already recognised in the 90s, it has only been treated as a second-class citizen thus far. This paper moves quiescence into the foreground and introduces the notion of quiescent transition systems (QTSs): an extension of regular input-output transition systems (IOTSs) in which quiescence is represented explicitly, via quiescent transitions. Four carefully crafted rules on the use of quiescent transitions ensure that our QTSs naturally capture quiescent behaviour.

We present the building blocks for a comprehensive theory on QTSs supporting parallel composition, action hiding and determinisation. In particular, we prove that these operations preserve all the aforementioned rules. Additionally, we provide a way to transform existing IOTSs into QTSs, allowing even IOTSs as input that already contain some quiescent transitions. As an important application, we show how our QTS framework simplifies the fundamental model-based testing theory formalised around ioco.

1 Introduction

Quiescence is a fundamental concept in modelling system behaviour. It explicitly represents the fact that, in certain system states, no output is provided. The absence of outputs is often essential: an ATM, for instance, should deliver the requested amount of money only once, not twice (see Figure 1). This means that the ATM's state just after paying out money (s_0 in Figure 1) should be quiescent: it should not produce any output until further input is given. On the other hand, the state before paying out (s_3 in Figure 1) should clearly not be quiescent. Hence, quiescence can also sometimes be considered as erroneous behaviour.

Thus, the notion of quiescence is essential in testing: if a system under test (SUT) does not provide any output, then the test evaluation algorithm must decide whether to produce a pass verdict (allowing quiescence at this point) or a fail verdict (forbidding quiescence at this point).

Origins. The notion of quiescence was first introduced by Vaandrager in [14] to obtain a natural extension of the notion of a terminal or blocking state: if a system is input-enabled (i.e., always ready to receive inputs), then no states are blocking, since each state has outgoing input transitions. However, quiescence can still be used to denote the fact that a state would be blocking when considering only the output actions. Quiescence is explored further in [6, 7].

Tretmans introduced the notion of *repetitive quiescence* [11, 12], which emerged from the need to continue testing, even in a quiescent state: in the ATM example above, we need to test further behaviour that arises from the (quiescent) state after providing money. To accommodate these needs, Tretmans

To appear in EPTCS.

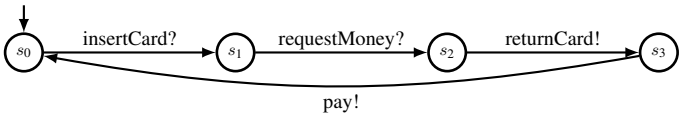


Figure 1: A very basic ATM.

introduced the *suspension automaton* as an auxiliary concept. More recent uses of quiescence include [1], applying it in the context of machine learning.

Example 1.1. Consider the automaton given in Figure 1. The states s_0 and s_1 are quiescent, since they do not have any outgoing output transitions. To obtain the suspension automaton corresponding to such a system, Tretmans adds self-loops, labelled with the quiescence label δ, to each quiescent state. □

Limitations of current treatments. While the papers above all convincingly argued the need for quiescence, none of them presents a comprehensive theory of quiescence. Firstly, quiescence is not treated as a first-class citizen: although the suspension automaton is used during testing, it is not defined as an entity in itself. Therefore, quiescence cannot be used to specify systems, and neither is it clear what properties a suspension automaton satisfies or should satisfy. Since conformance relations such as `ioco` are defined based on 'suspension traces', which are the traces of a suspension automaton, it seems much more appealing to directly start from these suspension automata and base the whole theory on them.

Secondly, basic operators like parallel composition and hiding were only defined for input-output transition systems, but have not been studied for suspension automata at all. Therefore, it was still an open question to what extent these operators could be lifted to the setting of quiescence.

Our approach. The current paper remediates the shortcomings of previous work and presents a comprehensive theory for quiescence, by introducing *quiescent transition systems* (QTSs). These are input-output transition systems in which quiescence can be represented explicitly by δ-transitions, and form a fully-formalised alternative to Tretmans' suspension automata. Whereas suspension automata are always constructed by adding δ-transitions to existing LTSs and subsequently determinising [13], QTSs are defined in a precise manner as a stand-alone entity, can be built from scratch and need not necessarily be deterministic.

As a first step, we handle QTSs that are input-enabled (never reject an input) and most importantly convergent (free of infinite sequences of internal transitions), since the interplay between quiescence and infinite sequences of internal transitions is delicate. Hence, we first focus on the basics. Relaxing these restrictions is an important direction for future work.

Starting point in our theory is the observation that, when treating quiescence as a first-class citizen, restrictions need to be put in place. For instance, it should never be the case that a δ-transition is followed by an output, as this would contradict the meaning of quiescence. As another example, as argued elaborately in Section 3, we do not allow a δ-transition to enable additional behaviour; after all, it would not make much sense if our observation of the absence of outputs impacts the system. In this paper we present and discuss four such rules, that restrict the domain of all possible QTSs to a sensible subclass.

We define three well-known automata-theoretical operations on QTSs: parallel composition, hiding and determinisation. These operations are very important, as they allow a modular approach to system specification. Additionally, we explain how to obtain a QTS from an IOTS by a process called deltafication. We define this process in a liberal way, supporting also the construction of a QTS from an IOTS

that already has some δ-transitions in place. We show that our four requirements on QTSs, which are a key contribution of this paper, are preserved by all of these operations.

This novel theory of QTSs simplifies the theory of model-based testing. Hence, we conclude this paper by showing how QTSs can be used to define the conformance relation `ioco`, and aid in test case generation and evaluation.

Overview of the paper. First, we present some preliminaries on input-output transition systems in Section 2. Then, Section 3 introduces the QTS model and its operations, as well as a variety of important (closure) properties. Section 4 explains how to construct QTSs based on IOTSs, and Section 5 discusses the application of QTSs to test theory. Finally, conclusions and future work are presented in Section 6.

Due to space limitations, we refer to [8] for detailed proofs of all our lemmas, propositions and theorems.

2 Background

2.1 Preliminaries

Given a set L, we denote by L^* the set of all sequences over L. Given a sequence $\sigma = a_1 a_2 \dots a_n$, we define the length of σ, denoted $|\sigma|$, as n. The empty sequence is denoted by ϵ.

Given two sequences $\rho = a_1 a_2 \dots a_n \in L^*$ and $\upsilon = b_1 b_2 \dots b_k \in L^*$, we define the concatenation of ρ and υ, denoted $\rho + \upsilon$ or $\rho\upsilon$, as $a_1 a_2 \dots a_n b_1 b_2 \dots b_k$. The sequence ρ is a *prefix* of υ, denoted $\rho \sqsubseteq \upsilon$, if there is a $\rho' \in L^*$ such that $\rho\rho' = \upsilon$; if $\rho' \neq \epsilon$, then ρ is a *proper prefix* of υ, denoted $\rho \sqsubset \upsilon$.

Given a set $S \subseteq L^*$, a sequence $\sigma \in S$ is called *maximal with respect to* $\sqsubseteq$ if there does not exist a sequence $\rho \in S$ such that $\sigma \sqsubset \rho$. Clearly, such a maximal sequence always exists.

We use $\wp(L)$ to denote the *power set* of L, i.e., $\wp(L)$ is the set of all subsets of L, including the empty set and L itself.

2.2 Input-Output Transition Systems

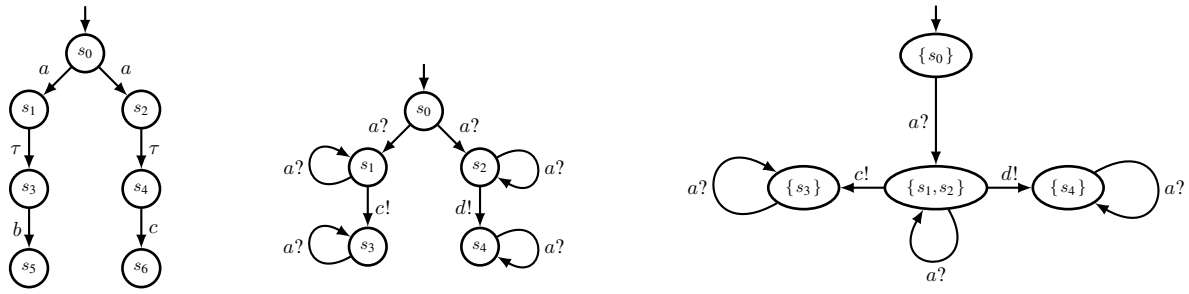
Before we introduce Input-Output Transition Systems, we first describe the modelling formalism they are based on: Labelled Transition Systems.

Definition 2.1 (Labelled Transition Systems)**.** A *Labelled Transition System* (LTS) is a quadruple $\mathcal{A} = \langle S, S^0, L, \rightarrow \rangle$, such that:

- S is a (possibly uncountable) set of states;
- $S^0 \subseteq S$ is a non-empty set of initial states;
- L is a set of labels, each representing a different action. We take $\tau \notin L$ to stand for an internal (unobservable) action and define $L^\tau = L \cup \{\tau\}$;
- $\rightarrow \subseteq S \times L^\tau \times S$ is the transition relation. We use $s \xrightarrow{a} s'$ to denote $(s, a, s') \in \rightarrow$, write $s \xrightarrow{a}$ if there is an $s' \in S$ such that $s \xrightarrow{a} s'$, and $s \not\xrightarrow{a}$ if this is not the case. If $s \xrightarrow{a}$, we say that the action a is *enabled* in state s.

We use $S_\mathcal{A}$, $S^0_\mathcal{A}$, $L_\mathcal{A}$ and $\rightarrow_\mathcal{A}$ to denote the components of an LTS $\mathcal{A}$. These subscripts are left out when it is clear from the context which LTS is referred to.

Example 2.2. Figure 2(a) shows an LTS $\mathcal{A}$. □



(a) $\mathcal{A}$ (b) $\mathcal{B}$ (c) *det*($\mathcal{B}$)

Figure 2: Visual representation of the LTS $\mathcal{A}$ and the IOTSs $\mathcal{B}$ and *det*($\mathcal{B}$). We represent states by circles, and transitions by arrows; each arrow in turn is labelled with the associated action for that particular transition. The initial state is marked by an arrow without a source state. Frow now on, we typically will not label individual states.

Often, in particular in the context of testing, it is desirable to be able to distinguish between actions that are initiated by the environment (inputs), and actions that are initiated by the system itself (outputs). To this end, we introduce Input-Output Transition Systems, which are an extension of regular LTSs.

Definition 2.3 (Input-Output Transition Systems)**.** An *Input-Output Transition System* (IOTS) is a quintuple $\mathcal{A} = \langle S, S^0, L^{\mathrm{I}}, L^{\mathrm{O}}, \rightarrow \rangle$, where L^{I} is a set of input labels and L^{O} a set of output labels such that $L^{\mathrm{I}} \cap L^{\mathrm{O}} = \emptyset$. We define $L = L^{\mathrm{I}} \cup L^{\mathrm{O}}$ and $L^{\tau} = L \cup \{\tau\}$, where $\tau \notin L$. S, S^0 and $\rightarrow$ are as defined for LTSs. Additionally, IOTSs must be *input-enabled*, i.e., $s \xrightarrow{a}$ for all $s \in S, a \in L^{\mathrm{I}}$.

Remark 2.4. Throughout this article we sometimes suffix a question mark (?) to the input labels and an exclamation mark (!) to the output labels, to help differentiating the two types. These are, however, not part of the label.

Note that IOTSs are similar to I/O automata [5, 4], except that the latter allow multiple internal actions, rather than τ only. All our results can easily be phrased in the I/O automata framework.

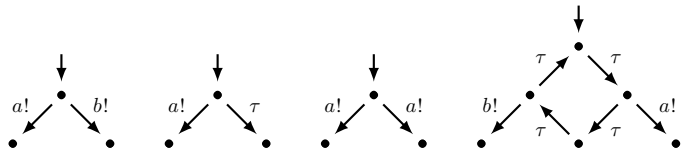
By requiring IOTSs to be input-enabled, any input initiated by the environment is never refused by the system. For deterministic systems (see Definition 2.7), this restriction can easily be lifted by adding a sink state which has self-loops for all possible actions, and adding transitions for the missing inputs to that sink state (so-called *demonic completion* [4, 15]). For nondeterministic systems, a solution is provided in [3].

Example 2.5. Figure 2(b) shows an IOTS $\mathcal{B}$. Note that since $L^{\mathrm{I}} = \{a\}$ and $s \xrightarrow{a}$ for every $s \in S$, $\mathcal{B}$ is input-enabled. □

We introduce the standard language-theoretic concepts for IOTSs.

Definition 2.6 (Notations)**.** Let $\mathcal{A} = \langle S, S^0, L^{\mathrm{I}}, L^{\mathrm{O}}, \rightarrow \rangle$ be an IOTS, then:

- A *path* in $\mathcal{A}$ is a (possibly infinite) sequence $\pi = s_0 a_1 s_1 \ldots s_n$ such that for all $1 \leq i \leq n$ we have $s_{i-1} \xrightarrow{a_i} s_i$ with $a_i \in L^{\tau}$. The set of all paths in $\mathcal{A}$ is denoted *paths*($\mathcal{A}$).
- The path operators *first* and *last* yield the first and last state of a finite path, respectively, e.g., for $\pi = s_0 a_1 s_1 a_2 s_2$ we have *first*(π) $= s_0$ and *last*(π) $= s_2$. A path π is called *initial* if *first*(π) $\in S^0$.
- The path operator *trace* yields the sequence of actions that is obtained by erasing all states and τ-actions from a given path, e.g., for $\pi = s_0 a_1 s_1 \tau s_2 a_2 s_3$ we have *trace*(π) $= a_1 a_2$; we call such a



(a) $\mathcal{A}$ (b) $\mathcal{B}$ (c) $\mathcal{C}$ (d) $\mathcal{D}$

Figure 3: One deterministic ($\mathcal{A}$) and three nondeterministic ($\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$) IOTSs. The IOTS $\mathcal{D}$ is divergent.

sequence of actions a *trace* of $\mathcal{A}$. The *length* of a trace $\sigma = a_1 a_2 \dots a_n$, denoted $|\sigma|$, is the length of the corresponding sequence, i.e., $|\sigma| = |a_1 a_2 \dots a_n| = n$.

- Given an action a and a set of actions P, we denote by $a \restriction P$ the *projection* of a on P, i.e., $a \restriction P = a$ if $a \in P$, and $a \restriction P = \epsilon$ otherwise. The projection of a trace $\sigma = a\sigma'$ on a set of actions P follows naturally from this: $\sigma \restriction P = a\sigma' \restriction P = a \restriction P + \sigma' \restriction P$. Finally, the projection of a set of traces T on a set of actions P is defined as $T \restriction P = \{ \sigma \restriction P \mid \sigma \in T \}$.
- If there is a finite path π in $\mathcal{A}$ such that $\mathit{first}(\pi) = s$, $\mathit{last}(\pi) = s'$ and $\mathit{trace}(\pi) = \sigma$, we write $s \stackrel{\sigma}{\Longrightarrow} s'$; if there exists an $s' \in S$ such that $s \stackrel{\sigma}{\Longrightarrow} s'$, we write $s \stackrel{\sigma}{\Longrightarrow}$, and $s \stackrel{\sigma}{\not\Longrightarrow}$ if this is not the case.
- For a finite trace σ and state $s \in S$, we denote by $\mathit{reach}(s, \sigma)$ the set of states in $\mathcal{A}$ (possibly empty) that can be reached from s via σ, i.e., $\mathit{reach}(s, \sigma) = \{ s' \in S \mid s \stackrel{\sigma}{\Longrightarrow} s' \}$. Similarly, for a given finite trace σ and a set of states $S' \subseteq S$, we denote by $\mathit{reach}(S', \sigma)$ the set of states in $\mathcal{A}$ that can be reached from any of the states in S' via σ, i.e., $\mathit{reach}(S', \sigma) = \{ s \in S \mid \exists s' \in S' \,.\, s' \stackrel{\sigma}{\Longrightarrow} s \}$.
- For a finite trace σ and state $s \in S$, $\mathit{out}(s, \sigma)$ is the set of output actions that are enabled in any of the states reachable from s by σ, i.e., $\mathit{out}(s, \sigma) = \{ a \in L^{\mathrm{O}} \mid \exists s' \in \mathit{reach}(s, \sigma) \,.\, s' \stackrel{a}{\Longrightarrow} \}$. We use the shorthand $\mathit{out}(s)$ for the case $\mathit{out}(s, \epsilon)$, i.e., the set of output actions that are enabled in s itself.
- For every $s \in S$ we denote by $\mathit{traces}(s)$ the set of all traces of $\mathcal{A}$ that correspond to paths that start in s, i.e., $\mathit{traces}(s) = \{ \mathit{trace}(\pi) \mid \pi \in \mathit{paths}(\mathcal{A}) \wedge \mathit{first}(\pi) = s \}$. We denote by $\mathit{traces}(\mathcal{A}) = \bigcup_{s \in S^0} \mathit{traces}(s)$ the set of all traces that correspond to initial paths in $\mathcal{A}$. Two IOTSs $\mathcal{B}$ and $\mathcal{C}$ are *trace equivalent* if $\mathit{traces}(\mathcal{B}) = \mathit{traces}(\mathcal{C})$.

A fundamental concept in automata theory is determinism.

Definition 2.7 (Determinism)**.** An IOTS $\mathcal{A} = \langle S, S^0, L^{\mathrm{I}}, L^{\mathrm{O}}, \rightarrow \rangle$ is *deterministic* if for all $s, s', s'' \in S, a \in L$ we have that $s \stackrel{a}{\rightarrow} s'$ and $s \stackrel{a}{\rightarrow} s''$ imply $a \neq \tau$ and $s' = s''$. Otherwise, $\mathcal{A}$ is *nondeterministic*.

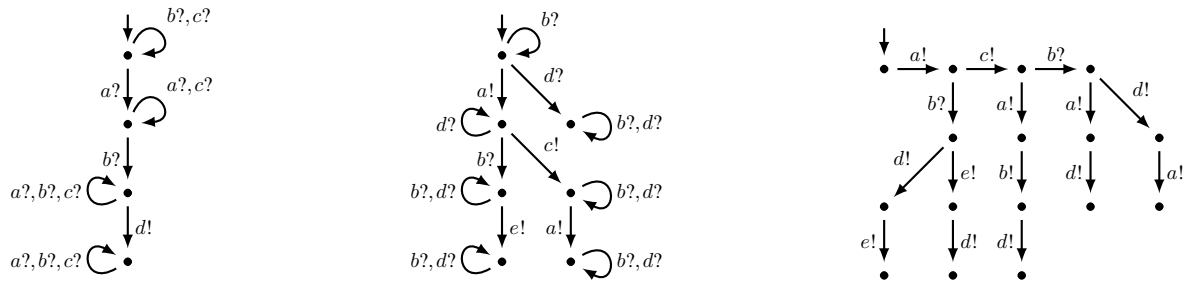
Example 2.8. Figure 3 shows some deterministic and nondeterministic IOTSs. □

Lastly, we introduce the notions of convergence and divergence.

Definition 2.9 (Divergence)**.** Given an IOTS $\mathcal{A} = \langle S, S^0, L^{\mathrm{I}}, L^{\mathrm{O}}, \rightarrow \rangle$, a state $s \in S$ of $\mathcal{A}$ is *divergent* if there is an infinite path $s_0 a_1 s_1 a_2 s_2 \dots$ with $s_0 = s$ and $s_i \in S$, that contains only τ transitions, i.e., $a_i = \tau$ for all i. An IOTS is called divergent if it contains at least one such state, otherwise it is *convergent*.

For the purposes of this paper, we require all IOTSs to be convergent.

Example 2.10. Figure 3(d) shows the divergent IOTS $\mathcal{D}$. Clearly, it is possible for $\mathcal{D}$ to perform an infinite sequence of τ-transitions by continuously looping through the innermost four states. □



(a) $\mathcal{A}$ (b) $\mathcal{B}$ (c) $\mathcal{A} \parallel \mathcal{B}$

Figure 4: The IOTSs $\mathcal{A}$ and $\mathcal{B}$, and their parallel composition $\mathcal{A} \parallel \mathcal{B}$. Note that we have left out some of the $b?$-labelled self-loops from the visualisation of $\mathcal{A} \parallel \mathcal{B}$ to reduce clutter.

2.3 Operations on IOTSs

In this section, we introduce several standard operations on IOTSs. First, every nondeterministic IOTS can be transformed into a deterministic IOTS [9]; the latter is called the determinisation of the original IOTS and is trace equivalent to it [2]. Using this operator, modelling effort is saved since no attention needs to be paid to making the specification deterministic.

Definition 2.11 (Determinisation)**.** The *determinisation* of an IOTS $\mathcal{A} = \langle S, S^0, L^{\mathrm{I}}, L^{\mathrm{O}}, \rightarrow_{\mathcal{A}} \rangle$ is the IOTS $det(\mathcal{A}) = \langle T, \{ S^0 \}, L^{\mathrm{I}}, L^{\mathrm{O}}, \rightarrow_{\mathrm{d}} \rangle$ such that $T = \wp(S) \setminus \emptyset$ and $\rightarrow_{\mathrm{d}} = \{ (U, a, V) \in T \times L \times T \mid V = reach_{\mathcal{A}}(U, a) \wedge V \neq \emptyset \}$.

Example 2.12. Consider the nondeterministic IOTS $\mathcal{B}$ shown in Figure 2(b). Its corresponding determinisation $det(\mathcal{B})$ is shown in Figure 2(c). □

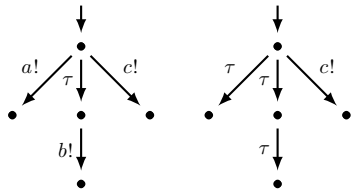
Second, we define the parallel composition operator. This operator is fundamental in modelling frameworks for component-based design. It allows one to build complex system models from smaller ones, thus breaking up the specification of a system into manageable pieces. Parallel composed IOTSs synchronise on shared inputs and complementary input-output pairs [4].

Definition 2.13 (Parallel composition of IOTSs)**.** Given are two IOTSs $\mathcal{A} = \langle S_{\mathcal{A}}, S^0_{\mathcal{A}}, L^{\mathrm{I}}_{\mathcal{A}}, L^{\mathrm{O}}_{\mathcal{A}}, \rightarrow_{\mathcal{A}} \rangle$ and $\mathcal{B} = \langle S_{\mathcal{B}}, S^0_{\mathcal{B}}, L^{\mathrm{I}}_{\mathcal{B}}, L^{\mathrm{O}}_{\mathcal{B}}, \rightarrow_{\mathcal{B}} \rangle$ such that $L^{\mathrm{O}}_{\mathcal{A}} \cap L^{\mathrm{O}}_{\mathcal{B}} = \emptyset$. The *parallel composition* of $\mathcal{A}$ and $\mathcal{B}$ is the IOTS $\mathcal{A} \parallel \mathcal{B} = \langle S_{\mathcal{A}\parallel\mathcal{B}}, S^0_{\mathcal{A}\parallel\mathcal{B}}, L^{\mathrm{I}}_{\mathcal{A}\parallel\mathcal{B}}, L^{\mathrm{O}}_{\mathcal{A}\parallel\mathcal{B}}, \rightarrow_{\mathcal{A}\parallel\mathcal{B}} \rangle$, where $S_{\mathcal{A}\parallel\mathcal{B}} = S_{\mathcal{A}} \times S_{\mathcal{B}}$, $S^0_{\mathcal{A}\parallel\mathcal{B}} = S^0_{\mathcal{A}} \times S^0_{\mathcal{B}}$, $L^{\mathrm{I}}_{\mathcal{A}\parallel\mathcal{B}} = (L^{\mathrm{I}}_{\mathcal{A}} \cup L^{\mathrm{I}}_{\mathcal{B}}) \setminus (L^{\mathrm{O}}_{\mathcal{A}} \cup L^{\mathrm{O}}_{\mathcal{B}})$, and $L^{\mathrm{O}}_{\mathcal{A}\parallel\mathcal{B}} = L^{\mathrm{O}}_{\mathcal{A}} \cup L^{\mathrm{O}}_{\mathcal{B}}$. The transition relation $\rightarrow_{\mathcal{A}\parallel\mathcal{B}}$ is defined as follows:

$$
\begin{aligned}
\rightarrow_{\mathcal{A}\parallel\mathcal{B}} \;=\; & \{ ((s,t), a?, (s',t')) \mid s \xrightarrow{a?}_{\mathcal{A}} s' \wedge t \xrightarrow{a?}_{\mathcal{B}} t' \} \\
\cup\; & \{ ((s,t), a!, (s',t')) \mid s \xrightarrow{a?}_{\mathcal{A}} s' \wedge t \xrightarrow{a!}_{\mathcal{B}} t' \} \\
\cup\; & \{ ((s,t), a!, (s',t')) \mid s \xrightarrow{a!}_{\mathcal{A}} s' \wedge t \xrightarrow{a?}_{\mathcal{B}} t' \} \\
\cup\; & \{ ((s,t), a, (s',t)) \mid s \xrightarrow{a}_{\mathcal{A}} s' \wedge t \in S_{\mathcal{B}} \wedge a \in L^{\tau}_{\mathcal{A}} \setminus L_{\mathcal{B}} \} \\
\cup\; & \{ ((s,t), a, (s,t')) \mid t \xrightarrow{a}_{\mathcal{B}} t' \wedge s \in S_{\mathcal{A}} \wedge a \in L^{\tau}_{\mathcal{B}} \setminus L_{\mathcal{A}} \}
\end{aligned}
$$

Thus, $L_{\mathcal{A}\parallel\mathcal{B}} = L^{\mathrm{I}}_{\mathcal{A}\parallel\mathcal{B}} \cup L^{\mathrm{O}}_{\mathcal{A}\parallel\mathcal{B}} = L_{\mathcal{A}} \cup L_{\mathcal{B}}$.

Example 2.14. Figure 4 shows two IOTSs $\mathcal{A}$ and $\mathcal{B}$, and their parallel composition $\mathcal{A} \parallel \mathcal{B}$. We have $L^{\mathrm{I}}_{\mathcal{A}} = \{ a, b, c \}$, $L^{\mathrm{O}}_{\mathcal{A}} = \{ d \}$, $L^{\mathrm{I}}_{\mathcal{B}} = \{ b, d \}$, and $L^{\mathrm{O}}_{\mathcal{B}} = \{ a, c, e \}$. Note that indeed $L^{\mathrm{O}}_{\mathcal{A}} \cap L^{\mathrm{O}}_{\mathcal{B}} = \emptyset$, as required; therefore, by Definition 2.13, $L^{\mathrm{I}}_{\mathcal{A}\parallel\mathcal{B}} = \{ b \}$ and $L^{\mathrm{O}}_{\mathcal{A}\parallel\mathcal{B}} = \{ a, c, d, e \}$. □



(a) $\mathcal{A}$ (b) $\mathit{hide}(\mathcal{A}, \{a, b\})$

Figure 5: The IOTSs $\mathcal{A}$ and $\mathit{hide}(\mathcal{A}, \{a, b\})$.

Finally, it is often useful to hide certain actions of a given IOTS, thereby essentially renaming the corresponding labels to τ. For example, when parallel composing two IOTSs, some actions are only used for synchronisation; after composition, they are not needed anymore.

Definition 2.15 (Action hiding in IOTSs)**.** Let $\mathcal{A} = \langle S, S^0, L^{\mathrm{I}}, L^{\mathrm{O}}, \rightarrow_{\mathcal{A}} \rangle$ be an IOTS and $H \subseteq L^{\mathrm{O}}$ a set of output labels, then one can *hide* H in $\mathcal{A}$ to get the IOTS $\mathit{hide}(\mathcal{A}, H) = \langle S, S^0, L^{\mathrm{I}}, L^{\mathrm{O}} \setminus H, \rightarrow_{\mathrm{h}} \rangle$, where $\rightarrow_{\mathrm{h}} = \{ (s, a, s') \in \rightarrow_{\mathcal{A}} | \ a \notin H \} \cup \{ (s, \tau, s') \in S \times \{\tau\} \times S \mid \exists a \in H \,.\, (s, a, s') \in \rightarrow_{\mathcal{A}} \}$.

Thus, we only allow output actions to be hidden. Furthermore, we do not allow action hiding to lead to divergent IOTSs, i.e., the hiding of outputs may not lead to the creation of τ-loops.

Example 2.16. Figure 5 shows the IOTSs $\mathcal{A}$ with $L^{\mathrm{O}}_{\mathcal{A}} = \{a, b, c\}$ and $\mathcal{B} = \mathit{hide}(\mathcal{A}, \{a, b\})$. □

From now on, we typically won't show all input-labelled self-loops in visualisations of IOTSs, to reduce clutter. Thus, we assume that every IOTS is input-enabled (unless mentioned otherwise).

2.4 Properties of IOTSs

IOTSs possess several interesting properties, that will also be of use when working with QTSs later on. We provide three results, showing that (1) hiding of actions corresponds to projection of traces, (2) parallel composition does not introduce new traces when projecting on the alphabet of either one of the components, and (3) parallel composition of components that synchronise on all actions yields the intersection of the traces of the components.

Proposition 2.17. *Given an IOTS $\mathcal{A}$ and a set of labels $H \subseteq L^{\mathrm{O}}_{\mathcal{A}}$, we have $\mathit{traces}(\mathit{hide}(\mathcal{A}, H)) = \mathit{traces}(\mathcal{A}) \restriction (L_{\mathcal{A}} \setminus H)$.*

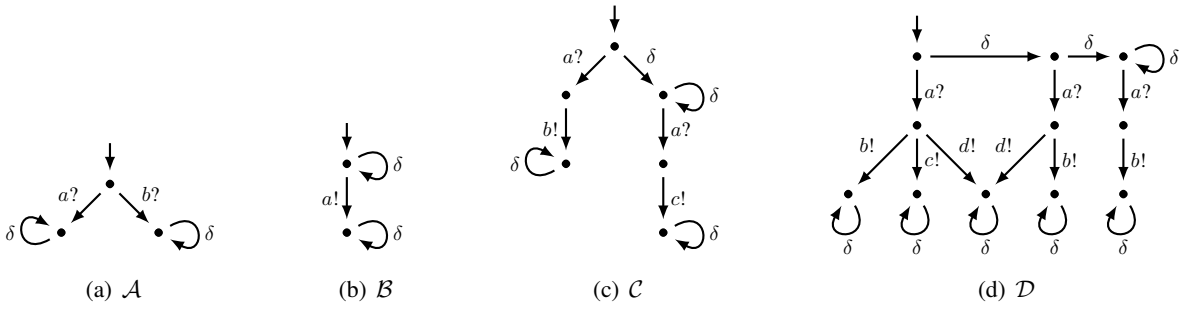
Proposition 2.18. *Given two IOTSs $\mathcal{A}$ and $\mathcal{B}$, we have $\mathit{traces}(\mathcal{A} \parallel \mathcal{B}) \restriction L_{\mathcal{A}} \subseteq \mathit{traces}(\mathcal{A})$ and $\mathit{traces}(\mathcal{A} \parallel \mathcal{B}) \restriction L_{\mathcal{B}} \subseteq \mathit{traces}(\mathcal{B})$.*

Proposition 2.19. *Given two IOTSs $\mathcal{A}, \mathcal{B}$ with $L_{\mathcal{A}} = L_{\mathcal{B}}$, we have $\mathit{traces}(\mathcal{A} \parallel \mathcal{B}) = \mathit{traces}(\mathcal{A}) \cap \mathit{traces}(\mathcal{B})$.*

3 Quiescent Transition Systems

3.1 Basic notions and requirements

IOTSs can be used to model the inputs and outputs of a system, but cannot explicitly express the observation of the absence of outputs, also called the observation of quiescence [14, 11, 7]. To fill this void, we introduce Quiescent Transition Systems. These automata can be used to model all possible observations for a particular system, including quiescence, and can thus be thought of as 'observation automata'.



(a) $\mathcal{A}$ (b) $\mathcal{B}$ (c) $\mathcal{C}$ (d) $\mathcal{D}$

Figure 6: The QTSs $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ that do not satisfy rule R1, rule R2, rule R3 and rule R4, respectively.

They are based on Tretmans' suspension automata [11], in the sense that a δ-transition represents the observation of quiescence. A basic variant of QTSs was already used in [10] in a testing framework. However, restrictions for QTSs to prohibit counterintuitive behaviour, as well as characteristics and closure properties of such models, have never been studied before.

Definition 3.1 (Quiescence)**.** Let $\mathcal{A} = \langle S, S^0, L^{\mathrm{I}}, L^{\mathrm{O}}, \rightarrow \rangle$ be an IOTS. A state $s \in S$ is called *quiescent* if $\nexists a \in L^{\mathrm{O}} \cup \{\tau\} \,.\, s \xrightarrow{a}$, i.e., no outputs or internal transitions can be executed in state s.

A system in a quiescent state will be idle until a new input is supplied. Note that a state s that can still perform a τ-step is not considered quiescent, even if there is no output $a! \in L^{\mathrm{O}}$ such that $s \stackrel{a!}{\Longrightarrow}$. After all, since quiescence signifies that a system is idle indefinitely, it would not make sense if there are still internal steps possible. Moreover, from a more technical point of view, this ensures that QTSs are closed under hiding and that hiding and deltafication (see Section 4) are commutative.

Definition 3.2 (Quiescent Transition Systems)**.** A *Quiescent Transition System* (QTS) is an IOTS $\mathcal{A} = \langle S, S^0, L^{\mathrm{I}}, L^{\mathrm{O}} \cup \{\delta\}, \rightarrow \rangle$, where $\delta \notin L^{\mathrm{I}} \cup L^{\mathrm{O}}$ is a special output label that is used to denote the observation of quiescence. We define $L = L^{\mathrm{I}} \cup L^{\mathrm{O}}$, $L^{\delta}_{\tau} = L^{\mathrm{I}} \cup L^{\mathrm{O}} \cup \{\delta, \tau\}$ and let $\rightarrow \subseteq S \times L^{\delta}_{\tau} \times S$ be the transition relation. Like regular IOTSs, QTSs must be input-enabled, i.e., $s \xrightarrow{a}$ for all $s \in S, a \in L^{\mathrm{I}}$. Furthermore, we also require the following rules to hold for all states $s, s', s'' \in S$:

Rule R1 (Quiescence should be observable)**:** if s is quiescent, then $s \xrightarrow{\delta}$**.**

This rule requires that each quiescent state has an outgoing δ-transition. Consider the QTS $\mathcal{A}$ in Figure 6(a). This QTS does not satisfy this rule, as the topmost state cannot produce any outputs, but neither can execute an outgoing δ-transition.

Rule R2 (No outputs after quiescence)**:** if $s \xrightarrow{\delta} s'$, then s' is quiescent**.**

This rule ensures that the system is idle after a δ-transition, i.e., it cannot provide an output (except for δ itself) or execute an internal transition, before another input is provided. In Figure 6(b) the QTS $\mathcal{B}$ is shown which does not satisfy this rule. From the top-most state it is possible to first observe quiescence (the δ-transition) and after that the $a!$ output, without an intermediate input. Since there is no particular observation duration associated with quiescence, but quiescence rather means that the system idles indefinitely, this is clearly counterintuitive and therefore disallowed.

Rule R3 (Quiescence does not enable new behaviour)**:** if $s \xrightarrow{\delta} s'$, then $\mathit{traces}(s') \subseteq \mathit{traces}(s)$**.**

Given a state s' of a QTS that is reached from another state s by a δ-transition (i.e., observation of quiescence), this rule demands that any trace that can be executed starting from state s' can

also be executed in state s, i.e., the observation of quiescence may not introduce any new possible observations. This rule was added to prevent situations like the one depicted in Figure 6(c). For QTS $\mathcal{C}$ it is possible to observe the output $c!$ (after the input $a?$) after first observing quiescence, but if quiescence is not observed (because, for instance, the input $a?$ was directly given) the output $b!$ will be observed after the input $a?$ instead. Thus, the prior observation of quiescence allows new behaviour to be observed later on, which is counterintuitive. This rule therefore ensures that all behaviour that can be observed after observing quiescence can also be observed before.

Rule R4 (Continued quiescence preserves behaviour)**:** if $s \xrightarrow{\delta} s'$ and $s' \xrightarrow{\delta} s''$, then $\mathit{traces}(s') = \mathit{traces}(s'')$**.**

A QTS $\mathcal{D}$ that violates this rule is shown in Figure 6(d). From the initial state an observation of quiescence can be made, which then leads to a new state where the trace ac can no longer be observed. From the latter state another observation of quiescence can be made, which leads to another state where the trace ad can no longer be observed. Rule R3 allows this, but as there is no particular time interval associated with the observation of quiescence, this does not make sense. We therefore have the additional requirement that any observations possible after two (or more) consecutive observations of quiescence should also be possible after a single observation of quiescence, and vice versa.

Just as for IOTSs, we require QTSs to be convergent. The reason for this is that divergent systems have states that can execute internal transitions infinitely often and never output anything. Considering such a state quiescent would be nonintuitive, as it is not idle (and might even be able to provide an output action, even though it does not show it). Not considering it quiescent would also be nonintuitive, because of the possibility that no visible behaviour is observed.

Note that the converse of rule R1 is not required, e.g., we do not forbid that a state has both a δ-transition and an output action enabled. This situation can arise during the determinisation of a QTS, as we will see in Section 4. However, the δ-transition should still end up in a quiescent state, as required by rule R2. Also note that a trace of a QTS can contain a sequence of δ-actions. Although this might seem odd, it corresponds to the practical testing scenario of observing a time-out rather than an output more than once in a row.

Since computing trace inclusion is expensive [1], an easier way to ensure that a QTS complies to rule R3 is to make sure the following alternative rule R3′ holds for all states $s, s', s'' \in S$.

Rule R3′: if $s \xrightarrow{\delta} s'$ and $\exists a? \in L_{\mathrm{I}}$ such that $s' \xrightarrow{a?} s''$ then also $s \xrightarrow{a?} s''$**.**

Clearly, any QTS that satisfies rule R3′ also satisfies rule R3.

Similarly, conformance to rule R4 for a QTS can be achieved by making sure that the following alternative rule R4′ holds for all states $s, s' \in S$ of the QTS.

Rule R4′: if $s \xrightarrow{\delta} s'$ then $s' \xrightarrow{\delta} s'$, and if also $s' \xrightarrow{\delta} s''$ then $s'' = s'$**.**

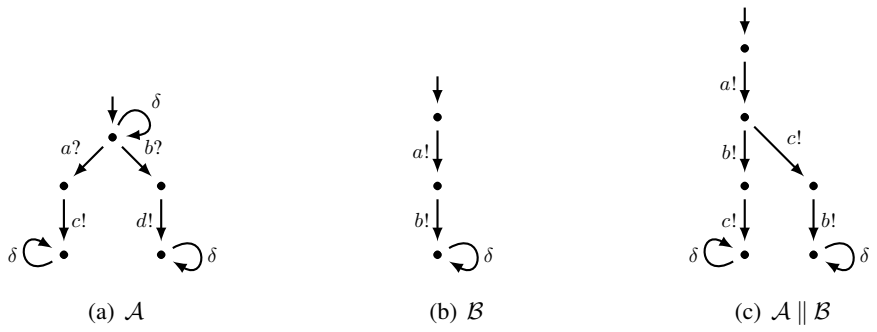
Clearly, any QTS that satisfies rule R4′ also satisfies rule R4.

When comparing the structure of two QTSs $\mathcal{A}$ and $\mathcal{B}$, the notion of isomorphisms can be useful.

Definition 3.3 (Isomorphic QTSs)**.** Two QTSs $\mathcal{A} = \langle S_{\mathcal{A}}, S^0_{\mathcal{A}}, L^{\mathrm{I}}_{\mathcal{A}}, L^{\mathrm{O}}_{\mathcal{A}} \cup \{\delta\}, \rightarrow_{\mathcal{A}} \rangle$ and $\mathcal{B} = \langle S_{\mathcal{B}}, S^0_{\mathcal{B}}, L^{\mathrm{I}}_{\mathcal{B}}, L^{\mathrm{O}}_{\mathcal{B}} \cup \{\delta\}, \rightarrow_{\mathcal{B}} \rangle$ are called *isomorphic*, denoted $\mathcal{A} \cong \mathcal{B}$, if there exists a bijection $h\colon S_{\mathcal{A}} \to S_{\mathcal{B}}$ (called an isomorphism) such that the following holds:

1. for all $s_0 \in S^0_{\mathcal{A}}$ there exists a $t_0 \in S^0_{\mathcal{B}}$ such that $h(s_0) = t_0$, and vice versa;
2. $s \xrightarrow{a}_{\mathcal{A}} s'$ if and only if $h(s) \xrightarrow{a}_{\mathcal{B}} h(s')$, for all $s, s' \in S_{\mathcal{A}}$ and $a \in L_{\mathcal{A}} \cup \{\delta, \tau\}$.

Thus, two isomorphic QTSs are structurally equivalent.



(a) $\mathcal{A}$ (b) $\mathcal{B}$ (c) $\mathcal{A} \parallel \mathcal{B}$

Figure 7: The QTSs $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{A} \parallel \mathcal{B}$.

3.2 Operations on QTSs

Since QTSs are a specialisation of IOTSs, all operations that are applicable to IOTSs (such as determinisation, parallel composition and hiding of actions) are also applicable to QTSs. Determinisation for QTSs is exactly the same as for IOTSs, but there are some minor differences for parallel composition and action hiding.

Definition 3.4 (Parallel composition of QTSs)**.** Let $\mathcal{A} = \langle S_{\mathcal{A}}, S^0_{\mathcal{A}}, L^{\mathrm{I}}_{\mathcal{A}}, L^{\mathrm{O}}_{\mathcal{A}} \cup \{\delta\}, \rightarrow_{\mathcal{A}} \rangle$ and $\mathcal{B} = \langle S_{\mathcal{B}}, S^0_{\mathcal{B}}, L^{\mathrm{I}}_{\mathcal{B}}, L^{\mathrm{O}}_{\mathcal{B}} \cup \{\delta\}, \rightarrow_{\mathcal{B}} \rangle$ be two QTSs such that $L^{\mathrm{O}}_{\mathcal{A}} \cap L^{\mathrm{O}}_{\mathcal{B}} = \emptyset$. The *parallel composition* of $\mathcal{A}$ and $\mathcal{B}$ is then the QTS $\mathcal{A} \parallel \mathcal{B} = \langle S_{\mathcal{A}\parallel\mathcal{B}}, S^0_{\mathcal{A}\parallel\mathcal{B}}, L^{\mathrm{I}}_{\mathcal{A}\parallel\mathcal{B}}, L^{\mathrm{O}}_{\mathcal{A}\parallel\mathcal{B}} \cup \{\delta\}, \rightarrow_{\mathcal{A}\parallel\mathcal{B}} \rangle$, where $S_{\mathcal{A}\parallel\mathcal{B}} = S_{\mathcal{A}} \times S_{\mathcal{B}}$, $S^0_{\mathcal{A}\parallel\mathcal{B}} = S^0_{\mathcal{A}} \times S^0_{\mathcal{B}}$, $L^{\mathrm{I}}_{\mathcal{A}\parallel\mathcal{B}} = (L^{\mathrm{I}}_{\mathcal{A}} \cup L^{\mathrm{I}}_{\mathcal{B}}) \setminus (L^{\mathrm{O}}_{\mathcal{A}} \cup L^{\mathrm{O}}_{\mathcal{B}})$, and $L^{\mathrm{O}}_{\mathcal{A}\parallel\mathcal{B}} = L^{\mathrm{O}}_{\mathcal{A}} \cup L^{\mathrm{O}}_{\mathcal{B}}$. $\rightarrow_{\mathcal{A}\parallel\mathcal{B}}$ is defined as follows:

$$
\begin{aligned}
\rightarrow_{\mathcal{A}\parallel\mathcal{B}} \;=\; & \{\, ((s,t), a?, (s',t')) \mid s \xrightarrow{a?}_{\mathcal{A}} s' \wedge t \xrightarrow{a?}_{\mathcal{B}} t' \,\} \\
\cup\; & \{\, ((s,t), a!, (s',t')) \mid s \xrightarrow{a?}_{\mathcal{A}} s' \wedge t \xrightarrow{a!}_{\mathcal{B}} t' \,\} \\
\cup\; & \{\, ((s,t), a!, (s',t')) \mid s \xrightarrow{a!}_{\mathcal{A}} s' \wedge t \xrightarrow{a?}_{\mathcal{B}} t' \,\} \\
\cup\; & \{\, ((s,t), \delta, (s',t')) \mid (s,\delta,s') \in \rightarrow_{\mathcal{A}} \wedge (t,\delta,t') \in \rightarrow_{\mathcal{B}} \,\} \\
\cup\; & \{\, ((s,t), a, (s',t)) \mid s \xrightarrow{a}_{\mathcal{A}} s' \wedge t \in S_{\mathcal{B}} \wedge a \in L^{\tau}_{\mathcal{A}} \setminus L_{\mathcal{B}} \,\} \\
\cup\; & \{\, ((s,t), a, (s,t')) \mid t \xrightarrow{a}_{\mathcal{B}} t' \wedge s \in S_{\mathcal{A}} \wedge a \in L^{\tau}_{\mathcal{B}} \setminus L_{\mathcal{A}} \,\}
\end{aligned}
$$

Thus, when compared to the parallel composition of regular IOTSs, we have the additional requirement that parallel composed QTSs must synchronise on the δ-action, as the observation of quiescence can be made simultaneously for multiple QTSs. Again, we find that $L_{\mathcal{A}\parallel\mathcal{B}} = L^{\mathrm{I}}_{\mathcal{A}\parallel\mathcal{B}} \cup L^{\mathrm{O}}_{\mathcal{A}\parallel\mathcal{B}} = L_{\mathcal{A}} \cup L_{\mathcal{B}}$.

Example 3.5. See Figure 7(a) for the visual representation of a QTS $\mathcal{A}$ which satifies all the requirements for QTSs listed in Definition 3.2. Figure 7(b) shows another QTS $\mathcal{B}$ and Figure 7(c) shows the parallel composition of the QTSs $\mathcal{A}$ and $\mathcal{B}$. □

Definition 3.6 (Action hiding in QTSs)**.** Let $\mathcal{A} = \langle S, S^0, L^{\mathrm{I}}, L^{\mathrm{O}} \cup \{\delta\}, \rightarrow_{\mathcal{A}} \rangle$ be a QTS and $H \subseteq L^{\mathrm{O}}$ a set of labels, then one can *hide* H in $\mathcal{A}$ to obtain the IOTS *hide*$(\mathcal{A}, H) = \langle S, S^0, L^{\mathrm{I}}, (L^{\mathrm{O}} \setminus H) \cup \{\delta\}, \rightarrow_{\mathrm{h}} \rangle$, where $\rightarrow_{\mathrm{h}} = \{\, (s,a,s') \in \rightarrow_{\mathcal{A}} \mid a \notin H \,\} \cup \{\, (s,\tau,s') \in S \times \{\tau\} \times S \mid \exists a \in H \,.\, (s,a,s') \in \rightarrow_{\mathcal{A}} \,\}$.

We do not allow the special output label δ to be hidden, as this label doesn't represent a specific output but rather (the observation of) a lack of outputs. Furthermore, as for IOTSs, we do not allow action hiding to lead to divergent QTSs, i.e., hiding may not lead to the creation of τ-loops.

3.3 Properties of QTSs

In this section, we present several interesting properties of QTSs. First of all, it turns out that our model is closed under all operations defined thus far: determinisation, action hiding and parallel composition. Therefore, these operations are indeed well-defined for QTSs.

Theorem 3.7. *QTSs are closed under determinisation, action hiding and parallel composition. Hence, given two QTSs $\mathcal{A}$, $\mathcal{B}$ and a set of labels $H \subseteq L^{\mathrm{O}}_{\mathcal{A}}$, also $det(\mathcal{A})$, $hide(\mathcal{A}, H)$ and $\mathcal{A} \parallel \mathcal{B}$ are QTSs.*

We also provide two results concerning the traces of parallel compositions of QTSs, generalising the corresponding properties of IOTSs as given in Section 2.4. First, parallel composition does not introduce new traces when projecting on the alphabet of either one of the components. That is, when disregarding the actions of component $\mathcal{B}$ in the traces of $\mathcal{A} \parallel \mathcal{B}$, the resulting set of traces is a subset of the traces of $\mathcal{A}$. It then quite easily follows that, when two parallel QTSs have the same alphabet (and hence synchronise on all actions), we obtain a subset of the intersection of their individual traces.

Proposition 3.8. *Given two QTSs $\mathcal{A}$ and $\mathcal{B}$, we have $traces(\mathcal{A} \parallel \mathcal{B}) \upharpoonright (L_{\mathcal{A}} \cup \{\delta\}) \subseteq traces(\mathcal{A})$ and $traces(\mathcal{A} \parallel \mathcal{B}) \upharpoonright (L_{\mathcal{B}} \cup \{\delta\}) \subseteq traces(\mathcal{B})$.*

Proposition 3.9. *Given two QTSs $\mathcal{A}$, $\mathcal{B}$ with $L_{\mathcal{A}} = L_{\mathcal{B}}$, we have $traces(\mathcal{A} \parallel \mathcal{B}) = traces(\mathcal{A}) \cap traces(\mathcal{B})$.*

4 From IOTS to QTS: deltafication

Usually, the specification and implementation of a system (under development) are given as IOTSs, rather than QTSs. During testing, however, we typically observe the outputs of the system generated in response to inputs from the environment; thus, it is useful to be able to refer to the absence of outputs (i.e., quiescence) explicitly. Hence, we need a way to convert an IOTS to a QTS that captures all possible observations of it, including quiescence; this conversion is called deltafication and is described in [11, 12, 13]. First, however, we need to introduce an additional condition C1 for IOTSs, for every $s, s' \in S$:

Condition C1: if $s \xrightarrow{\delta} s'$, then for all $\sigma \in traces(s')$:

$$\exists t' \in reach(s', \sigma)\ .\ t' \text{ is quiescent } \wedge\ t' \not\xrightarrow{\delta} \ \Rightarrow \forall t \in reach(s, \sigma)\ .\ t \text{ is quiescent } \wedge\ t \not\xrightarrow{\delta}$$

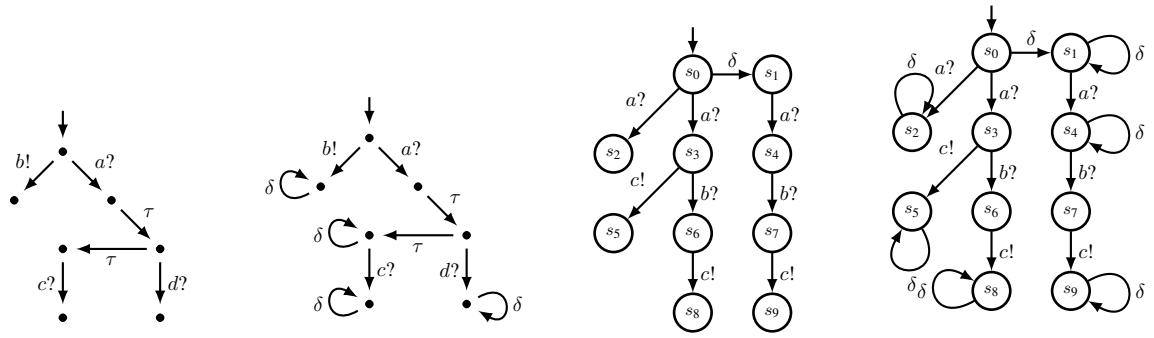
Condition C1 requires that if any trace $\sigma \in traces(s')$, when executed from s', can lead to a state that is quiescent and cannot execute a δ-transition, then it must always lead to a state that is quiescent and cannot execute a δ-transition when executed from s. This condition is weaker than R1, and allows us to determine the deltafication of systems that already contain some δ-transitions without requiring a δ-transition from every quiescent state. Note that any IOTS without δ-transitions vacuously satisfies C1.

Definition 4.1 (Deltafication)**.** Given an IOTS $\mathcal{A} = \langle S, S^0, L^{\mathrm{I}}, L^{\mathrm{O}}, \rightarrow_{\mathcal{A}} \rangle$ that for all $s, s' \in S$ satisfies deltafication condition C1, and rules R2, R3 and R4 (see Definition 3.2), we define the *deltafication* of $\mathcal{A}$ as the QTS $\delta(\mathcal{A}) = \langle S, S^0, L^{\mathrm{I}}, L^{\mathrm{O}} \cup \{\delta\}, \rightarrow_{\delta} \rangle$ where $\rightarrow_{\delta} = \rightarrow_{\mathcal{A}} \cup \{ (s, \delta, s) \in S \times \{\delta\} \times S \mid s \text{ is quiescent } \wedge\ s \not\xrightarrow{\delta}_{\mathcal{A}} \}$.

Example 4.2. An IOTS $\mathcal{A}$ and its deltafication $\delta(\mathcal{A})$ are shown in Figure 8(a) and 8(b), respectively. □

Remark 4.3. To see why condition C1 is necessary, consider the IOTS $\mathcal{B}$ and its deltafication $\delta(\mathcal{B})$ shown in Figure 8(c) and Figure 8(d), respectively; the states have been labelled for convenience. $\mathcal{B}$ does not satisfy condition C1, since $s_0 \xrightarrow{\delta} s_1$, $s_4 \in reach(s_1, a)$ and s_4 is quiescent and $s_4 \not\xrightarrow{\delta}$, but $s_3 \in reach(s_0, a)$ and s_3 is not quiescent. As a consequence, the deltafication $\delta(\mathcal{B})$ is not a valid QTS: for $\delta(\mathcal{B})$ we have $a\delta bc \in traces(s_1)$, but $a\delta bc \notin traces(s_0)$, thereby violating rule R3.

A more liberal version of C1, where the second quantification is changed to an existential one, would not be strong enough to prevent this: it would not forbid this example, as $s_2 \in reach(s_0, a)$ is quiescent and cannot do a δ-transition.



(a) $\mathcal{A}$ (b) $\delta(\mathcal{A})$ (c) $\mathcal{B}$ (d) $\delta(\mathcal{B})$

Figure 8: Deltafications of the IOTSs $\mathcal{A}$ and $\mathcal{B}$.

4.1 Validity of deltafication

Now, we present several interesting properties regarding the deltafication of IOTSs and QTSs. First, we show that deltafication indeed yields a valid QTS, and that it is idempotent.

Lemma 4.4. *Given an IOTS $\mathcal{A}$ that satisfies condition C1 and rules R2, R3 and R4, $\delta(\mathcal{A})$ is a QTS.*

Proposition 4.5. *Deltafication is idempotent, i.e., given an IOTS $\mathcal{A}$ that satisfies condition C1 and rules R2, R3 and R4, we have $\delta(\delta(\mathcal{A})) = \delta(\mathcal{A})$.*

Any IOTS $\mathcal{A}$ with $\delta \notin L_{\mathcal{A}}$ vacuously satisfies condition C1 and rules R2, R3 and R4. Therefore, the following theorem follows directly from Lemma 4.4.

Theorem 4.6. *Given an IOTS $\mathcal{A}$ such that $\delta \notin L_{\mathcal{A}}$, $\delta(\mathcal{A})$ is a QTS.*

By Definition 3.2, QTSs are IOTSs that satisfy rules R1, R2, R3 and R4. Since every state s in a QTS enables at least one output action or δ (due to rule R1), it never occurs that s is quiescent and does not enable a δ-transition, and hence every QTS satisfies condition C1 vacuously.

By Lemma 4.4, this immediately implies the following theorem.

Theorem 4.7. *QTSs are closed under deltafication, i.e., given a QTS $\mathcal{A}$, $\delta(\mathcal{A})$ is also a QTS.*

4.2 Commutativity results

In this section we investigate the commutativity of deltafication with determinisation, action hiding and parallel composition. We will show that parallel composition can safely be swapped with deltafication, but that determinisation has to precede deltafication to get sensible results. Also, we show that action hiding does not commute with deltafication.

Proposition 4.8. *Deltafication and determinisation do not commute, i.e., given an IOTS $\mathcal{A}$ that satisfies condition C1 and rules R2, R3 and R4, it is not necessarily the case that $det(\delta(\mathcal{A})) \cong \delta(det(\mathcal{A}))$.*

Proof. Observe the IOTS $\mathcal{A}$, its determinisation $det(\mathcal{A})$ and deltafication $\delta(\mathcal{A})$ in Figure 9(a,b,c). Clearly, the deltafication of the determinisation of $\mathcal{A}$ (i.e., $\delta(det(\mathcal{A}))$), shown in Figure 9(d), results in an incorrect observation automaton, as it does not model the fact that in the nondeterministic QTS $\delta(\mathcal{A})$ quiescence may be observed after an initial $a?$ input, as required by rule R1.

Contrary to the deltafication of the determinisation of $\mathcal{A}$, the determinisation of the deltafication of $\mathcal{A}$ (i.e., $det(\delta(\mathcal{A}))$), which is shown in Figure 9(e), does preserve the fact that quiescence may be observed

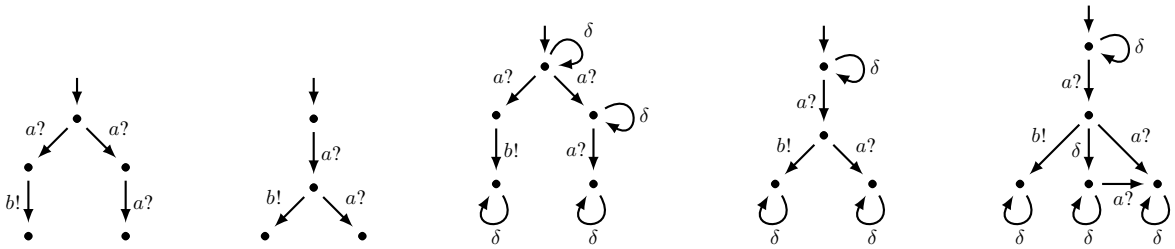


Figure 9: The determinisation and deltafication of IOTS $\mathcal{A}$ do not commute.

after an initial $a?$ input. This shouldn't come as a surprise, since for any IOTS $\mathcal{A}$ the determinisation $det(\mathcal{A})$ is trace equivalent to the original automaton, as was observed earlier. □

Thus, when transforming a nondeterministic IOTS $\mathcal{A}$ to a deterministic QTS, one should take care to first derive $\delta(\mathcal{A})$ and afterwards determinise to obtain $det(\delta(\mathcal{A}))$.

The following results show that deltafication does commute with both action hiding and parallel composition. For action hiding this is trivial. After all, hiding only renames output actions to τ and deltafication only adds δ-loops to states that have no outgoing output transitions, no outgoing τ-transitions and no outgoing δ-transition. Hence, they work on disjoint sets of states; commutativity is therefore immediate.

Theorem 4.9. *Deltafication and action hiding commute, i.e., given an IOTS $\mathcal{A}$ that satisfies condition C1 and rules R2, R3 and R4, and a set of labels $H \subseteq L^{\mathrm{O}}_{\mathcal{A}}$, we have $\delta(hide(\mathcal{A}, H)) \cong hide(\delta(\mathcal{A}), H)$.*

Theorem 4.10. *Deltafication and parallel composition commute, i.e., given two IOTSs $\mathcal{A}$ and $\mathcal{B}$ with $L^{\mathrm{O}}_{\mathcal{A}} \cap L^{\mathrm{O}}_{\mathcal{B}} = \emptyset$ that satisfy condition C1 and rules R2, R3 and R4, we have $\delta(\mathcal{A} \parallel \mathcal{B}) \cong \delta(\mathcal{A}) \parallel \delta(\mathcal{B})$.*

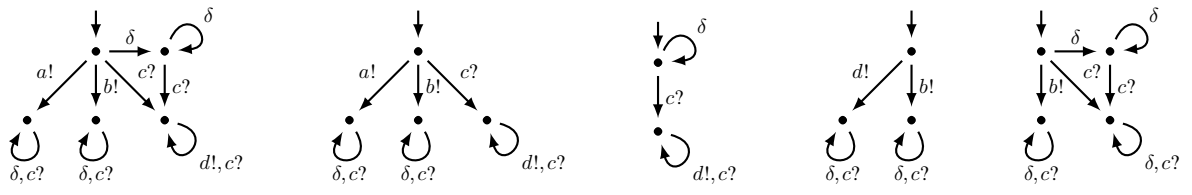
These results are vital, as they allow great modelling flexibility. After all, hiding and parallel composition are often already applied to the IOTSs that describe a specification and its implementation. We now showed that this yields the same QTSs as in case these operations are applied after deltafication.

5 Application to testing

Our main motivation for introducing and studying the QTS model was to enable a clean theoretical framework for model-based testing. In this section, we illustrate how the model can be incorporated in the `ioco` (input-output conformance) testing theory [13].

5.1 A conformance relation based on QTSs

To interpret the results of testing, we need to know which implementations are considered correct. For this, we use a conformance relation, such as `ioco`, that relates specifications to implementations if and only if the latter is 'correct' with respect to the former. For `ioco`, this is the case if the implementation never provides an unexpected output when it is only fed inputs that are allowed according to the specification. In this setting, an unexpected absence of outputs of the implementation is also considered to be unexpected output. This can be formalised very nicely using QTSs, as they already model the expected absence of outputs by explicit δ-transitions.



(a) $\mathcal{A}_{\text{spec}}$ (b) $\mathcal{A}_{\text{impl}_1}$ (c) $\mathcal{A}_{\text{impl}_2}$ (d) $\mathcal{A}_{\text{impl}_3}$ (e) $\mathcal{A}_{\text{impl}_4}$

Figure 10: A specification with two correct and two erroneous implementations.

Definition 5.1. Let $\mathcal{A}_{\text{impl}}, \mathcal{A}_{\text{spec}}$ be QTSs over the same alphabet $L^{\text{O}} \cup L^{\text{I}} \cup \{\delta\}$. Then

$$\mathcal{A}_{\text{impl}} \sqsubseteq_{\texttt{ioco}} \mathcal{A}_{\text{spec}} \text{ if and only if } \forall \sigma \in \mathit{traces}(\mathcal{A}_{\text{spec}}) \,.\, \mathit{out}_{\mathcal{A}_{\text{impl}}}(\sigma) \subseteq \mathit{out}_{\mathcal{A}_{\text{spec}}}(\sigma),$$

where $\mathit{out}_{\mathcal{A}}(\sigma) = \{a! \in L^{\text{O}} \cup \{\delta\} \mid \sigma a! \in \mathit{traces}(\mathcal{A})\}$.

Since we require all QTSs to be input-enabled, it is easy to see that ioco-conformance precisely corresponds to traditional trace inclusion over QTSs.

Example 5.2. Consider the specification $\mathcal{A}_{\text{spec}}$ given in Figure 10. It allows the initial state to either be quiescent, output an $a!$ or output a $b!$. We present four implementations. The first two implementations are ioco-correct with respect to $\mathcal{A}_{\text{spec}}$: although they omit some of the traces of the specification, they never provide an unexpected output after a trace that is in the specification. The third implementation is erroneous since it can provide a $d!$ output from the initial state, while the specification does not allow this. The fourth implementation is erroneous since it is unexpectedly quiescent after the trace $c?$. □

Note that QTSs allowed us in this example to explicitly model the fact that both quiescence and some output actions are considered correct behaviour of a system. Also, note that the unexpected quiescence of the fourth implementation is clearly marked by a δ-transition in the QTS.

5.2 Testing using QTSs

Using the notion of ioco-correspondence, it is quite easy to derive test cases for QTSs. Basically, at each point in time we choose to either try to provide an input, observe the behaviour of the system or stop testing. As long as the trace we obtain in this way (including the δ-actions) is also a trace of the specification, the implementation is correct. Due to the explicit presence of quiescence in the QTS model of the specification, it is easy to see that this straightforward way of testing precisely corresponds to checking ioco-conformance.

6 Conclusions and Future Work

We introduced the notion of quiescent transition systems (QTSs), explicitly modelling the absence of outputs as a first-class citizen. We provided four restrictions for QTSs, to eliminate counterintuitive behaviours. Also, we defined the common automaton operations — parallel composition, determinisation and action hiding — directly on QTSs, and showed that all of our restrictions are indeed preserved by the operations. We presented a way to obtain a QTS from a traditional input-output transition system (IOTS), even allowing the situation in which the IOTS already partially models quiescence. Finally,

we illustrated how our novel theory of QTSs can be used to greatly simplify the theory of model-based testing, defining the conformance relation ioco in terms of QTSs.

So far, we only allowed input-enabled and convergent QTSs; i.e., systems that cannot perform an endless series of unobservable transitions. Future work will focus on extending our framework to divergent systems that are not necessarily input-enabled. Also, we plan on linking QTSs to timed automata, to explicitly represent δ-transitions as finite timeouts, bridging the gap between formal and practical testing.

Acknowledgements This research has been partially funded by NWO under grants 612.063.817 (SYRUP) and Dn 63-257 (ROCKS).

References

[1] F. Aarts & F. W. Vaandrager (2010): *Learning I/O Automata*. In: *Proc. of the 21th Int. Conf. on Concurrency Theory (CONCUR)*, *LNCS* 6269, Springer, pp. 71–85, doi:10.1007/978-3-642-15375-4_6.

[2] C. Baier & J.-P. Katoen (2008): *Principles of Model Checking*. The MIT Press.

[3] H. C. Bohnenkamp & M. I. A. Stoelinga (2008): *Quantitative testing*. In: *Proc. of the 8th ACM & IEEE Int. Conf. on Embedded software (EMSOFT)*, ACM, pp. 227–236, doi:10.1145/1450058.1450089.

[4] R. De Nicola & R. Segala (1995): *A process algebraic view of input/output automata*. *Theoretical Computer Science* 138, pp. 391–423, doi:10.1016/0304-3975(95)92307-J.

[5] N. A. Lynch & M. R. Tuttle (1987): *Hierarchical Correctness Proofs for Distributed Algorithms*. In: *Proc. of the 6th Annual ACM Symp. on Principles of Distributed Computing (PODC)*, pp. 137–151, doi:10.1145/41840.41852.

[6] R. Segala (1993): *Quiescence, Fairness, Testing, and the Notion of Implementation*. In: *Proc. of 4th Int. Conf. on Concurrency Theory (CONCUR)*, *LNCS* 715, Springer, pp. 324–338, doi:10.1007/3-540-57208-2_23.

[7] R. Segala (1997): *Quiescence, Fairness, Testing, and the Notion of Implementation*. *Information and Computation* 138(2), pp. 194 – 210, doi:10.1006/inco.1997.2652.

[8] W. G. J. Stokkink, M. Timmer & M. I. A. Stoelinga (2012): *Talking quiescence: a rigorous theory that supports parallel composition, action hiding and determinisation (extended version)*. Technical Report TR-CTIT-12-05, CTIT, University of Twente.

[9] T. A. Sudkamp (2006): *Languages and machines*. Pearson Addison Wesley.

[10] M. Timmer, H. Brinksma & M. I. A. Stoelinga (2011): *Model-Based Testing*. In: *Software and Systems Safety: Specification and Verification*, *NATO Science for Peace and Security Series D: Information and Communication Security* 30, IOS Press, Amsterdam, pp. 1–32, doi:10.3233/978-1-60750-711-6-1.

[11] G. J. Tretmans (1996): *Test Generation with Inputs, Outputs, and Quiescence*. In: *Proceedings of the 2nd Int. Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, *LNCS* 1055, Springer, pp. 127–146, doi:10.1007/3-540-61042-1_42.

[12] G. J. Tretmans (1996): *Test Generation with Inputs, Outputs and Repetitive Quiescence*. *Software - Concepts and Tools* 17(3), pp. 103–120.

[13] G. J. Tretmans (2008): *Model Based Testing with Labelled Transition Systems*. In: *Formal Methods and Testing*, *LNCS* 4949, Springer, pp. 1–38, doi:10.1007/978-3-540-78917-8_1.

[14] F. W. Vaandrager (1991): *On the Relationship Between Process Algebra and Input/Output Automata (Extended Abstract)*. In: *Proc. of 6th Annual Symposium on Logic in Computer Science (LICS)*, IEEE, pp. 387–398, doi:10.1109/LICS.1991.151662.

[15] H. M. van der Bijl, A. Rensink & G. J. Tretmans (2004): *Compositional Testing with ioco*. In: *Formal Approaches to Software Testing (FATES)*, *LNCS* 2931, Springer Verlag, Berlin, pp. 86–100, doi:10.1007/978-3-540-24617-6_7.