Quiescence, Fairness, Testing, and the Notion of Implementation (Extended abstract)

Roberto Segala*

MIT Laboratory for Computer Science
Cambridge, MA 02139

Abstract. Two different formalisms for concurrency are compared and are shown to have common foundations. The Input/Output automaton model and the theory of testing are analyzed in the framework of transition systems. The relationship between the fair and quiescent preorders of I/O automata is investigated and the two preorders are shown to coincide on a large class of automata. I/O automata are encoded into the theory of testing and the reversed MUST preorder is shown to be equivalent to the quiescent preorder for strongly convergent, finitely branching automata up to encoding. Conversely, a theory of testing is defined directly on I/O automata, and the new reversed MUST preorder is shown to coincide with the quiescent preorder on strongly convergent, finitely branching automata. Finally, some considerations are given on the issue of divergence, and on other existing theories with an I/O distinction.

1 Introduction

Several theories of concurrency deal with the idea of *implementation*. Among them, one of the most accepted theories is connected with the *failure preorder* of [1, 8] and with the theory of testing of [3, 7], which are closely related each other [4]. Other theories deal with *receptive systems* [6, 9, 10], where there is a clear separation between the events under the control of the *external environment* of a given object and those events under the control of the object itself. Receptive theories are widely used in practise because they express ideas that are very close to our general intuition and they allow us to express very general liveness properties naturally. However, receptive theories are not sufficiently general, since they do not allow the specification of non-receptive objects at a sufficiently high level of abstraction.

In this paper we investigate how and when receptive theories can be viewed as a special case of a general, non-receptive theory. The formalisms we use for our analysis are the theory of testing [7] and the Input/Output automaton model [10], which has been widely used for the verification of complex distributed algorithms [17, ?]. At the end we also consider other two receptive models that are closely related to the models used in our analysis.

* Supported by NSF grant CCR-89-15206, by DARPA contracts N00014-89-J-1988 and N00014-92-J-4033, and by ONR contract N00014-91-J-1046.

The main intuition at the base of the testing preorders is that objects are compared based on their interactions with an external environment, therefore they are compared based on the success or failure of experiments the external environment performs on them. An experiment E MAY succeed on an object O if there is a sequence of possible interactions between E and O for which E is successful; an experiment E MUST succeed on an object O if all possible interactions between E and O lead to the success of E. Two preorders can then be defined over objects: the MAY preorder, which orders objects by the sets of experiments that may succeed, and the MUST preorder, which orders objects by the sets of experiments that must succeed. In [3] it is shown that the MUST preorder coincides with the reversed failure preorder of CSP [8] on objects that can never diverge. The failure preorder is used as an implementation relation within the CSP community [1, 2] and the intuitive idea at the base of its use is that an implementation has to be more deterministic than its specification. Therefore, the result of [3] also suggests a possible use of the MUST preorder as an implementation relation.

In the I/O automaton model, each object is associated with an explicit interface consisting of input, output and internal event names (*actions*). Input actions are always enabled and the occurrence of output actions cannot be blocked by the external environment. In other words, each object has to be *receptive* [6] on its inputs and each action is the output action of at most one object. The notion of implementation of I/O automata is expressed through fair trace inclusion, where a fair trace of an automaton $\mathcal{A}$ is a sequence of actions (possibly infinite) in which each subcomponent of $\mathcal{A}$ that is continuously willing to perform some output or internal actions will eventually do so. Trivial implementations are avoided by input enabling, since each implementation must accept its external stimuli, and by fairness, since whenever a specification must perform some output actions the implementation must do the same.

The main criticism against the I/O automaton model is that it is too restrictive: conditions like input enabling and actions under the control of at most one component do not allow the specification of several devices at a sufficiently high level of abstraction. A classical example is that of a buffer blocking its inputs whenever it is full. Moreover, since fair traces are not closed under limit, fixpoint reasoning is not possible in general within I/O automata, while fixpoint reasoning is one of the key features of the algebraic theory of processes based on testing preorders. On the other hand fairness allows us to capture some liveness properties that are not captured in general by the testing preorders and that are important for the verification of several distributed systems [10, 17, ?].

A first step toward the study of the relationship between the process algebraic and the I/O automata based approaches is in [16], where the impact of input enabling on the operators of a generic process algebra is analyzed. The analysis of [16] includes the definition of several preorder relations that gradually approximate the fair preorder. Among these is the quiescent preorder, which is the reduction of the fair preorder to the finitary behavior of a system. A quiescent trace is a sequence of actions leading a system to a state from which only input actions are enabled. In [14] the quiescent preorder is studied within a process algebraic theory of I/O automata and a fixpoint theorem is put forward. In [14] there is also an attempt at using the quiescent preorder as an implementation relation; however some counterexamples in [14] show that the quiescent preorder does not provide an intuitively reasonable notion of implementation in general. Thus, some restrictions are necessary.

In this paper we first study the relationship between the fair and the quiescent preorders, and present a large class of automata for which they coincide. We then show that the quiescent preorder of I/O automata is closely related to the theory of testing when dealing with strongly convergent transition systems. We first encode I/O automata as transition systems, making explicit the ideas that are embedded within the Input/Output structure; then, we show that for strongly convergent and finitely branching systems the encodings of I/O automata related by the quiescent preorder are similarly related by the reversed MUST preorder. As a corollary it is possible to identify a class of I/O automata for which the fair preorder coincides with the notion of implementation of the CSP community.

We also show that it is possible to define a theory of testing similar to that of [7] directly on I/O automata. Once again, the main theorem is that for strongly convergent and finitely branching I/O automata the quiescent preorder coincides with the reversed MUST preorder.

Finally, we consider two other well known formalisms with an Input/Output distinction: the Receptive Process Theory of Mark Josephs [9], which is used for the specification and verification of delay-insensitive circuits, and the Complete Trace Structures of David Dill [6] which are used for the specification and verification of speed-independent circuits. Both of these formalisms are closely related to I/O automata and to the theory of testing.

The results of this work accomplish three goals at the same time: 1) they show how different theories for the notion of implementation which are based on independent intuitions lead to similar conclusions; 2) they show why, as in practise, it is often possible to avoid dealing explicitly with fairness when specifying systems: the fairness part is already captured by the implementation relations; 3) they show a possible method for eliminating the input enabling constraint of I/O automata by embedding them into a more general framework. Unfortunately the price to pay is the absence of a fully general notion of fairness. Further investigation is necessary in this direction.

The rest of the paper is organized as follows. Section 2 introduces transition systems, I/O automata, and the theory of testing. Section 3 studies the relationship between the quiescent and the fair preorders. Section 4 encodes I/O automata into a general, non input enabled, framework and studies the relationship between the quiescent preorder and testing theory. Section 5 defines a theory of testing directly on I/O automata and shows the equivalence of the quiescent and the reversed MUST preorders. Section 6 addresses the problem of divergent transition systems. Section 7 relates Receptive Process Theory and Complete Trace Structures to I/O automata and the theory of testing. Section 8 presents some concluding remarks.

2 Preliminaries

2.1 Transition Systems

A transition system T consists of four components: a set $states(T)$ of states; a nonempty set $start(T) \subseteq states(T)$ of start states; an action signature $sig(T) = (ext(T), int(T))$ where $ext(T)$ and $int(T)$ are disjoint sets of external and internal actions, respectively; a transition relation $steps(T) \subseteq states(T) \times acts(T) \times states(T)$, where $acts(T)$ denotes the set $ext(T) \cup int(T)$ of actions.

A transition $(q, a, q') \in steps(T)$ is also written as $q \xrightarrow{a} q'$. We extend the notion of transition to finite sequences of actions by taking the transitive closure of $steps(T)$. Two derived transition relations, abstracting from internal computations, are $q \stackrel{a}{\Longrightarrow} q'$ iff $\exists_{s_1,s_2 \in int^*(T)} q \xrightarrow{s_1 a s_2} q'$, and $q \stackrel{a}{\Longrightarrow} q'$ iff $\exists_{s_1 \in int^*(T)}\, q \xrightarrow{s_1 a} q'$. Given a state $q \in states(T)$ we denote by $enabled(q)$ the set $\{a : q \xrightarrow{a} q'$ for some $q'\}$ of enabled actions from q, and we denote by $wenabled(q)$ the set $\{a : q \stackrel{a}{\Longrightarrow} q'$ for some $q'\}$ of enabled actions from q up to internal transitions.

An *execution fragment* of a transition system T is a finite or infinite sequence of alternating states and actions $\alpha = q_0 a_0 q_1 a_1 q_2 \cdots$ starting with a state and, if the execution fragment is finite, ending in a state, where each $(q_i, a_i, q_{i+1}) \in steps(T)$. An *execution* is an execution fragment starting with a start state.

The *external trace* of an execution fragment α of a transition system T, written $etrace_T(\alpha)$, or just $etrace(\alpha)$ when T is clear, is the list obtained by projecting α onto the set of external actions of T, i.e., $etrace(\alpha) = \alpha \restriction ext(T)$. We say that β is a *trace* of a transition system T if there exists an execution α of T with $etrace(\alpha) = \beta$. We denote by $etraces^*(T)$ and $etraces(T)$ the sets of finite and all traces of T, respectively.

Two transition systems T_1, T_2 are *compatible* if $int(T_1) \cap acts(T_2) = acts(T_1) \cap int(T_2) = \emptyset$. The parallel composition $T_1 \| T_2$ of two compatible transition systems T_1, T_2 is the transition system T such that $states(T) = states(T_1) \times states(T_2)$, $start(T) = start(T_1) \times start(T_2)$, $sig(T) = (ext(T_1) \cup ext(T_2), int(T_1) \cup int(T_2))$, and $((q_1, q_2), a, (q_1', q_2')) \in steps(T)$ iff either 1) $(q_1, a, q_1') \in steps(T_1)$ and $(q_2, a, q_2') \in steps(T_2)$, or 2) $(q_1, a, q_1') \in steps(T_1)$, $a \notin acts(T_2)$ and $q_2 = q_2'$, or 3) $(q_2, a, q_2') \in steps(T_2)$, $a \notin acts(T_1)$ and $q_1 = q_1'$. In other words, T_1 and T_2 synchronize on their common actions and evolve independently on the others.

2.2 I/O Automata

An *I/O automaton* $\mathcal{A}$ is a transition system with the following extra structure: a partition of $ext(\mathcal{A})$, $(in(\mathcal{A}), out(\mathcal{A}))$, consisting of input and output actions, respectively; a partition $part(\mathcal{A})$ of $local(\mathcal{A}) = out(\mathcal{A}) \cup int(\mathcal{A})$. The transition relation $steps(\mathcal{A})$ has the property that for each state q and each input action a there is a step from q with action a; we then say that $\mathcal{A}$ is input enabled.

An execution α of an automaton $\mathcal{A}$ is *quiescent* if it is finite and its final state is quiescent, i.e., its final state does not enable any action from $local(\mathcal{A})$. An execution α is *fair* if either it is quiescent, or it is infinite and for each class

$p \in part(\mathcal{A})$, either actions from p appear infinitely often in α or states from which no action of p is enabled appear infinitely often in α. A *quiescent trace* of $\mathcal{A}$ is the external trace of a quiescent execution of $\mathcal{A}$. A *fair trace* of $\mathcal{A}$ is the external trace of a fair execution of $\mathcal{A}$. The set of quiescent and fair traces of an automaton $\mathcal{A}$ are denoted by $qtraces(\mathcal{A})$ and $ftraces(\mathcal{A})$, respectively.

Based on external, quiescent, and fair traces, three preorder relations can be defined between automata with the same external signature. The *external trace preorder* is defined as

$$\mathcal{A}_1 \sqsubseteq_E^* \mathcal{A}_2 \text{ iff } etraces(\mathcal{A}_1) \subseteq etraces(\mathcal{A}_2)$$

and bases its observations on the sequences of external actions that an automaton may perform; the *quiescent preorder* [16] is defined as

$$\mathcal{A}_1 \sqsubseteq_Q \mathcal{A}_2 \text{ iff } \mathcal{A}_1 \sqsubseteq_E^* \mathcal{A}_2 \text{ and } qtraces(\mathcal{A}_1) \subseteq qtraces(\mathcal{A}_2)$$

and considers also those sequences of actions leading an automaton to a quiescent state; the *fair preorder* [10] is defined as

$$\mathcal{A}_1 \sqsubseteq_F \mathcal{A}_2 \text{ iff } ftraces(\mathcal{A}_1) \subseteq ftraces(\mathcal{A}_2)$$

and considers those sequences of actions generable from executions in which each subcomponent of the automaton that is continuously willing to perform some local action is eventually allowed to proceed. The subcomponents of an automaton $\mathcal{A}$ are identified through the partition $part(\mathcal{A})$ of its locally controlled actions.

The quiescent preorder is slightly complicate since it is based on external and quiescent trace inclusions. It is an approximation of the fair preorder that is based on finite executions only. External trace inclusion is needed to deal correctly with automata that output forever.

The fair preorder is used to express the basic notion of implementation for I/O automata: $\mathcal{A}_1$ implements $\mathcal{A}_2$ iff $\mathcal{A}_1 \sqsubseteq_F \mathcal{A}_2$. Input enabling guarantees that each implementation accepts all external stimuli, while fairness guarantees that each implementation provides some output whenever the specification must provide some output. Mark Tuttle [10] writes: "The requirement that input be constantly enabled ensures that our solutions are able to respond to all patterns of input. The use of fairness ensures that the correctness of a solution will be judged only by those behaviors in which the system is actually given the chance to make progress." Note that the above justifications are rather intuitive, however in this paper we validate them by relating the fair and quiescent preorders to the theory of testing.

2.3 The Theory of Testing

A different method for comparing transition systems is based on the observation of the interactions between a transition system and an external experimenter [3, 5, 7]. An experimenter for a transition system T is a transition system E, compatible with T, whose external actions are those of T plus an action w, called the success action. An experiment x is an execution of $T||E$ which is infinite or ends in a deadlocked state (complete execution). An experiment x is successful if w is enabled in at least one state of x. We say that T MAY E if there is a successful experiment of $T||E$. We say that T MUST E if each experiment of $T||E$ is successful. Two preorder relations can be defined.

1. $T_1 \sqsubseteq_{\text{MAY}} T_2$ iff $\forall_E\, T_1$ MAY E implies T_2 MAY E
2. $T_1 \sqsubseteq_{\text{MUST}} T_2$ iff $\forall_E\, T_1$ MUST E implies T_2 MUST E

The MAY and MUST preorders can be characterized differently without referring to a notion of external experimenter [3]. In particular the MAY preorder coincides with the external trace preorder; for the MUST preorder we need some definitions.

Given a transition system T and given a state $q \in \mathit{states}(T)$, we write $q\uparrow$ if q has an infinite internal computation $q \xrightarrow{\tau_1} q_1 \xrightarrow{\tau_2} q_2 \xrightarrow{\tau_3} \cdots$ where each τ_i is an internal action. We write $q\downarrow$ if q has no infinite internal computation. If $q\downarrow$ we say that q is *convergent* and if $q\uparrow$ we say it is *divergent*. The above notion can be relativized to sequences of actions by defining $q\downarrow\epsilon$ if $q\downarrow$, and $q\downarrow as$ if $q\downarrow$ and $q \stackrel{a}{\Longrightarrow} q'$ implies $q'\downarrow s$. We write $q\Downarrow$ if $q\downarrow s$ for each $s \in \mathit{ext}^*(T)$ and we write $T\Downarrow$ if $q\Downarrow$ for each $q \in \mathit{start}(T)$. If $T\Downarrow$ we say that T is *strongly convergent*. In other words, $T\Downarrow$ means that no divergent state will be reached for any sequence of actions s.

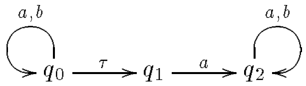
Given a transition system T, a set of states $Q \subseteq \mathit{states}(T)$, a sequence of actions $s \in \mathit{ext}^*(T)$, and a set of external actions A, we define Q *after* $s = \bigcup_{q\in Q}\{q' \in \mathit{states}(T) | q \stackrel{s}{\Longrightarrow} q'\}$ and we say that Q MUST A iff, for each $q \in Q$ and each q' such that $q \Longrightarrow q'$, $\mathit{wenabled}(q') \cap A \neq \emptyset$.

Proposition 1. *Given two finitely branching transition systems T_1, T_2 with the same external alphabet, $T_1 \sqsubseteq_{\text{MUST}} T_2$ iff, for each $s \in \mathit{ext}(T_1)^*$, if $T_1 \downarrow s$ then 1) $T_2 \downarrow s$ and 2) for each finite $A \subseteq \mathit{ext}(T_1)$, T_1 after s* MUST *A implies T_2 after s* MUST *A.* □

3 Quiescent and Fair Preorders

The fair preorder is the only preorder relation among those we consider which respects the infinite behavior of a system. It is the basic notion of implementation for I/O automata. Unfortunately sets of fair traces, ordered by subset, are not closed under limit.

Example 1. Consider the transition system $T \equiv$

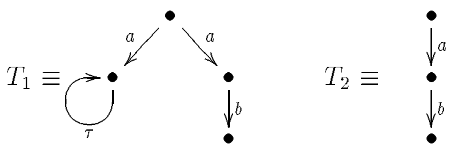


where a is an input action, b is an output action, and q_0 is a start state. It is easy to observe that each finite sequence a^n is a quiescent (and thus fair) trace of T since it is enough to loop on q_0 and then move to q_1. The infinite sequence a^ω, however, is not a fair trace of T since in any execution action b is enabled in all states but (at most) one.

As a consequence of the above observation, standard fixpoint reasoning and the use of finite objects for the description of infinite ones are not possible in general when dealing with fairness.

The quiescent preorder is a simplification of the fair preorder that is based only on the finitary behavior of a transition system. It allows fixpoint reasoning. In [14] it is shown that the quiescent preorder does not coincide with the fair preorder and does not express an intuitively reasonable notion of implementation in general. The rest of this section shows how the two preorders differ and provides some sufficient conditions for them to coincide. We only consider I/O automata without subcomponents (no partitioning of the locally controlled actions). These automata are called automata with a *trivial fairness partition* in [13]. In our examples we also assume to have an internal action τ.

Example 2. The transition systems



are equivalent according to the quiescent preorder if we consider a and b as output actions. However, T_1 and T_2 are not equivalent according to the fair preorder since a is a fair trace of T_1 but not a fair trace of T_2. In other words, T_1 is not an implementation of T_2 according to I/O automata.

The problem outlined in Example 2 is that the quiescent preorder cannot detect the possibility of refusing some output due to a divergence. A sufficient restriction to avoid it is to ensure that all divergences can be detected through a quiescent trace. Formally we give the following

Definition 2. An I/O automaton $\mathcal{A}$ is *quiescent detectable* if each finite fair trace of $\mathcal{A}$ is also a quiescent trace of $\mathcal{A}$.

Example 3. Consider the transition system T of Example 1, and consider a new transition system T' with one state and a unique self-loop transition with action a. If a is an input action and b is an output action, then it is easy to see that $T' \sqsubseteq_Q T$. However, $T' \not\sqsubseteq_F T$ since a^ω is not a fair trace of T.

The problem presented in Example 3 is due to the non closure of quiescent traces under limit. It can be solved by requiring the limit of any chain of quiescent traces to be fair.

Definition 3. An I/O automaton $\mathcal{A}$ is *quiescent continuous* if the limit of any chain of quiescent traces of $\mathcal{A}$ is a fair trace of $\mathcal{A}$.

Quiescent continuity is a strong requirement, however Proposition 7 gives sufficiently general conditions for an automaton to be quiescent continuous.

Example 4. Let a be an input action and b an output action. The transition systems

$$T_1 \equiv \begin{tikzcd} \bullet \arrow[loop above, "a,\tau"] \arrow[r, "b"] & \bullet \arrow[loop above, "a"] \end{tikzcd} \qquad\qquad T_2 \equiv \begin{tikzcd} \bullet \arrow[loop above, "a"] \arrow[r, "b"] & \bullet \arrow[loop above, "a"] \end{tikzcd}$$

are equivalent according to the quiescent preorder since they have the same external traces and their quiescent traces are all traces containing at least a b action. The trace a^ω, however, is a fair trace of T_1 but not a fair trace of T_2.

The problem with Example 4 is that in T_1 it is possible not to execute a continuously enabled output action by letting T_1 advance internally, whereas in T_2 the output action must be executed. The quiescent preorder can identify an infinite fair execution that reaches a quiescent state infinitely many times or contains infinitely many output actions; however, it cannot identify an infinite fair execution that reaches finitely many quiescent states (0 in our example) and contains finitely many output actions (0 in our example).

Definition 4. An I/O automaton $\mathcal{A}$ is *input quiescent detectable* if each infinite fair trace of $\mathcal{A}$ with finitely many output actions has infinitely many prefixes that are quiescent for $\mathcal{A}$.

Input quiescent detectability is a very restrictive requirement and it is complex to verify. An example of a non input quiescent detectable device is an unbounded buffer which performs some internal update after receiving some input. An infinite fair execution leading to an infinite trace with input actions only can be obtained by interleaving each input action with the internal update, however, if the buffer enables some output whenever it is not empty, no finite sequence of input actions is a quiescent trace. Note that in Example 4 fairness is expressing something different than the intuition that each subcomponent of a system is given fair turns to proceed. In other words this is an example in which the basic intuition of fairness for I/O automata is lost.

A last needed condition concerns the branching structure of a transition system. It provides a sufficient condition for characterizing infinite external traces through finite ones by guaranteeing continuity for external traces.

Definition 5. A transition system T has *finite internal nondeterminism* (FIN) if for each $h \in ext^*(T)$ there are only finitely many states q in T such that $q_0 \stackrel{h}{\Longrightarrow} q$ for some $q_0 \in start(T)$.

Note the use of $\Longrightarrow$ in Definition 5. A transition system with FIN might reach infinitely many states with a trace h; however, the set of states reachable with the last external transition of any given trace h is finite. For this reason our definition of FIN is weaker than that of [11]. Note also that FIN is weaker than *finitely branching* when a transition systems is strongly convergent. In general infinitely branching transition systems are considered to be ill-formed. However, they can be used to model the possibility of receiving infinitely many different inputs. What FIN says is that, after a specific input is chosen, only finitely many possibilities are given.

Theorem 6. *Consider two I/O automata $\mathcal{A}_1, \mathcal{A}_2$ with the trivial fair partition.*

- *If $\mathcal{A}_1$ is quiescent detectable and input quiescent detectable, and $\mathcal{A}_2$ is quiescent continuous and has FIN, then $\mathcal{A}_1 \sqsubseteq_Q \mathcal{A}_2$ implies $\mathcal{A}_1 \sqsubseteq_F \mathcal{A}_2$.*
- *If $\mathcal{A}_2$ is quiescent detectable then $\mathcal{A}_1 \sqsubseteq_F \mathcal{A}_2$ implies $\mathcal{A}_1 \sqsubseteq_Q \mathcal{A}_2$.* □

The proof of Theorem 6 is a simple cases analysis. In order to better understand quiescent continuity, the next proposition gives a sufficient condition.

Proposition 7. *Given a transition system T, if T has FIN and, for each state q, each input action a, and each $q_1, q_2 \in \{q' | q \stackrel{a}{\Longrightarrow} q'\}$, $qtraces(q_1) = qtraces(q_2)$, then T is quiescent continuous.* □

The above condition seems very unnatural, however it captures some interesting cases that generally hold for specifications. One specific case is when for each state q and each input action a the set $\{q' | q \stackrel{a}{\Longrightarrow} q'\}$ contains exactly one element. In this case we say that the automaton is *weakly input deterministic*. A subcase of weak input determinism is *input determinism*, when for each state q and each input action a the set $\{q' | q \stackrel{a}{\Longrightarrow} q'\}$ contains exactly one element.

We conclude this section with a result about general I/O automata.

Proposition 8. *Let $\mathcal{A}$ be an I/O automaton. If for each input transition $q \stackrel{a}{\longrightarrow} q'$ of $steps(\mathcal{A})$ and each class x of $part(\mathcal{A})$, an action of x is enabled from q' whenever an action of x is enabled from q (i.e. input actions do not disable any class of $part(\mathcal{A})$), then $ftraces(\mathcal{A}) \subseteq ftraces(\mathcal{A}')$ where $\mathcal{A}'$ differ from $\mathcal{A}$ only in that $part(\mathcal{A}') = \{local(\mathcal{A})\}$.* □

If an automaton $\mathcal{A}$ is implementing a specification $\mathcal{S}$ with a trivial fairness partition, and if the involved automata satisfy the conditions of Theorem 6, then the above proposition is giving a sufficient condition for deriving the full fair preorder from the quiescent preorder. In fact, from $\mathcal{A}' \sqsubseteq_Q \mathcal{S}$ we derive $\mathcal{A}' \sqsubseteq_F \mathcal{S}$, and, from Proposition 8, we derive $\mathcal{A} \sqsubseteq_F \mathcal{S}$. Examples of systems satisfying the condition of Proposition 8 are the monotone automata for dataflow networks of [15] and the semi-modular, speed independent circuits of [12].

4 From I/O Automata to Testing

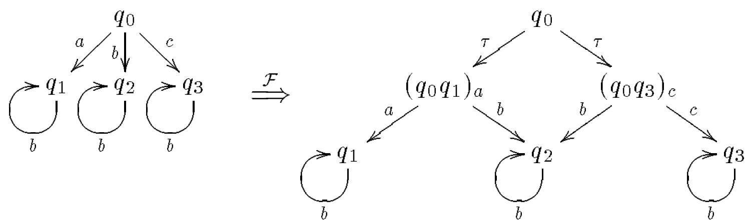
The main intuition behind I/O automata is that input actions are under the control of the external environment while output actions are under the control of the system. In other words, nondeterministic choice between input actions is intended to be an external choice, while nondeterministic choice between output actions is intended to be an internal choice. The following definition formalizes the above idea by providing an encoding of input enabled transition systems onto transition systems that are meant to represent the same object in the more general, non input enabled, framework. We assume the existence of at least one internal action τ.

Definition 9. Given an I/O automaton $\mathcal{A}$, the *interface free automaton* associated with $\mathcal{A}$ is defined as $\mathcal{F}(\mathcal{A}) = (Q, start(\mathcal{A}), (ext(\mathcal{A}), int(\mathcal{A})), steps)$ where

1. $Q = states(\mathcal{A}) \cup \{qq'_a | q \xrightarrow{a} q' \in steps(\mathcal{A}), a \in local(\mathcal{A})\}$ where for each pair $q, q' \in states(A)$ and each $a \in local(A)$ the expression qq'_a denotes a new state not occurring in $states(A)$,
2. $steps = \{q \xrightarrow{\tau} qq'_a | q \in states(\mathcal{A}), qq'_a \in Q\} \cup$
$\{qq'_a \xrightarrow{a} q' | qq'_a \in Q\} \cup$
$\{qq'_a \xrightarrow{b} q'' | q \xrightarrow{b} q'' \in steps(\mathcal{A}), qq'_a \in Q, b \in in(\mathcal{A})\} \cup$
$\{q \xrightarrow{b} q' \in steps(\mathcal{A}) | b \in in(\mathcal{A}), q \text{ is quiescent in } \mathcal{A}\}$

At each state q an automaton decides which local action to perform by internally moving to a new state from which only the selected local action is enabled. The new state also enables all input actions since only the external environment can decide which input to provide.

Example 5. The figure below shows an example of encoding.



The left automaton $\mathcal{A}$ is converted into the transition system $\mathcal{F}(\mathcal{A})$. Actions a and c are output actions, while b is an input action. From state q_0 of $\mathcal{A}$ there are two outgoing transitions labeled with an output action ($q_0 \xrightarrow{a} q_1$ and $q_0 \xrightarrow{c} q_3$), therefore two new states ($(q_0q_1)_a, (q_0q_3)_c$) are introduced, and from state q_0 of $\mathcal{F}(\mathcal{A})$ there are two internal transitions to the new states. In doing so the transition system $\mathcal{F}(\mathcal{A})$ internally decides which output action to perform. From the new states there is the preselected outgoing output transition together with all the outgoing input transitions of q_0.

The transformation $\mathcal{F}$ preserves the external and quiescent traces of an automaton $\mathcal{A}$, where a quiescent traces of $\mathcal{F}(\mathcal{A})$ is the external traces of an executions of $\mathcal{F}(\mathcal{A})$ that ends in a state enabling only the input actions of $\mathcal{A}$. Also, although the encoding of an I/O automaton is not input enabled in general, a weaker notion of input enabling is preserved. The key idea behind the new input enabledness condition is that a system is weakly input enabled whenever it cannot prevent the environment from providing input. The above condition is met even if all input actions are enabled from any state up to internal transition.

Proposition 10. *For each I/O automaton $\mathcal{A}$, $\mathcal{F}(\mathcal{A})$ is weakly $in(\mathcal{A})$-enabled, i.e., for each state $q \in states(\mathcal{F}(\mathcal{A}))$ and for each $a \in in(\mathcal{A})$, $q \stackrel{a}{\Longrightarrow}$.* □

Note that, if we only consider the reachable states of T, the definition of weak enabledness is equivalent to "T is A-enabled if for each $h \in ext^*(T)$ and for each $a \in A$, T after h MUST a".

Our main theorem states the relationship between the quiescent and the MUST preorders. Its proof strongly relies on the fact that the stable states of $\mathcal{F}(\mathcal{A})$ enable at most one output action. Thus, the simple knowledge that $\mathcal{F}(\mathcal{A})$ *after* s MUST A allows us to conclude very strong properties of A by only looking at the external traces of $\mathcal{F}(\mathcal{A})$.

Theorem 11. *Let $\mathcal{A}_1, \mathcal{A}_2$ be strongly convergent, finitely branching I/O automata. Then $\mathcal{A}_1 \sqsubseteq_Q \mathcal{A}_2$ iff $\mathcal{F}(\mathcal{A}_2) \sqsubseteq_{\mathrm{MUST}} \mathcal{F}(\mathcal{A}_1)$.* □

5 A Theory of Testing for I/O Automata

In Section 4 we have related the quiescent preorder of I/O automata to the theory of testing by encoding I/O automata into general labeled transition systems. However, we can reduce the power of an experimenter according to the interaction schemas of I/O automata and define a theory of testing directly on input enabled transition systems. In this section we show that also this approach leads us to the quiescent preorder when dealing with strongly convergent and finitely branching I/O automata.

An experimenter $\mathcal{E}$ for an I/O automaton $\mathcal{A}$ is an I/O automaton, compatible with $\mathcal{A}$, whose input actions are the output actions of $\mathcal{A}$ and whose output actions are the input actions of $\mathcal{A}$ plus an action w, called the success action. The experimenters for I/O automata are less powerful than those of Section 4 since an experimenter $\mathcal{E}$ can only control the input actions of an automaton $\mathcal{A}$. We denote the new testing preorders with $\sqsubseteq'_{\mathrm{MAY}}$ and $\sqsubseteq'_{\mathrm{MUST}}$.

The alternative characterization of the MUST preorder given in Proposition 1 is still valid, however the definition of Q MUST A has to take into account the fact that an automaton is in full control of its output actions and does not have any control on its input actions.

Definition 12. Given an I/O automaton $\mathcal{A}$, a set of states $Q \subseteq states(\mathcal{A})$ and a set of external actions A, we say that Q MUST A iff either 1) $A \cap in(\mathcal{A}) \neq \emptyset$ or 2) for each $q \in Q$, $wenabled(q) \cap out(\mathcal{A}) \subseteq A$ and, for each q' such that $q \Longrightarrow q'$, $wenabled(q') \cap A \neq \emptyset$.

The first condition says that any automaton must perform its input actions; the second condition says that any automaton internally decides wich one of its possible output actions to perform.

Theorem 13. *Let $\mathcal{A}_1, \mathcal{A}_2$ be strongly convergent, finitely branching I/O automata. Then $\mathcal{A}_1 \sqsubseteq_Q \mathcal{A}_2$ iff $\mathcal{A}_2 \sqsubseteq'_{\mathrm{MUST}} \mathcal{A}_1$.* □

6 Divergent Transition Systems

All the preorders that we have shown to coincide on strongly convergent transition systems are incomparable when dealing with divergent transition systems.

The MUST preorder considers a divergent state as chaotic, therefore nothing can be guaranteed to happen after a divergence has occurred. A purely divergent transition system is the minimal element of the MUST preorder, hence any other transition system is considered to be an implementation of it.

The fair preorder of I/O automata, instead, has some distinguishing power even in the presence of divergences. For example within the I/O automata theory it is possible to verify the correctness of a component which is running in parallel with a purely divergent one. As a consequence of the fair preorder, a purely divergent transition system is equivalent to a deadlocked one.

The quiescent preorder does not represent any intuition when dealing with divergent transition systems. It is just a finite approximation of the fair preorder which does not seem to work properly in the presence of divergences. Its usefulness is due to the fact that it allows fixpoint reasoning and that it represents a well known intuition when the involved transition systems satisfy the conditions of Theorem 6.

It is difficult at this stage to say what is the best approach to divergent transition systems. The approach of testing leads to a very neat theory while the approach of I/O automata (fairness) leads to a notion that is very closed to our intuition. On the other hand Example 4 has shown that the fairness of I/O automata may not be appropriate to every situation. It is the scope of further research to find a reasonable extension of the analyzed preorder relations to divergent transition systems.

7 Other Theories with I/O Distinction

7.1 Receptive Process Theory

Receptive Process Theory (RPT) [9] is an algebraic theory of receptive (input enabled) objects which is used for the description of delay insensitive circuits. Each object has an interface consisting of input and output actions and is receptive on its inputs. As for I/O automata, two RPT objects can be composed in parallel only if they do not have any common output actions.

The semantics of an RPT expression is denotational and is given in terms of "failures", however these failures are not those of [8]. The failures of [9] coincide with the quiescent traces of [16] (see Section 2) if we assume that a system performing infinitely many output actions without being provided with any input has a chaotic behavior (it is divergent).

An RPT object is considered to be divergent if "it can output forever or if it can become quiescent in infinitely many different ways". The reason for this choice is to avoid an hiding operator to introduce new divergences. RPT does not consider infinite sequences of output actions to be desirable.

Another reason for the choice of [9] about divergence of infinite output sequences, and probably the most important one, is that it gives a simple way of handling systems that output forever. The quiescent preorder of [16], in fact, has to consider quiescent and external traces at the same time in order to correctly handle transition systems that can output forever.

7.2 Complete Trace Structures

In his PhD thesis [6] David Dill introduces *Complete Trace Structures* for the description of speed-independent circuits. Formally, a complete trace structure is a quadruple (I, O, S, F) where I is a set of input actions, O is a set of output actions, S and F are two sets of finite and infinite strings (traces) over $I \cup O$. Both S and F have to be *mixed regular sets*, i.e., sets obtained from the union of a *regular* and an *ω-regular* language over $I \cup O$. In other words S and F are languages accepted by finite state machines. The set S is called the *success set* and represents those traces that are considered to be successful for the described system; the set F is called the *failure set* and represents those traces that lead to a failure of the described system. A failure is used to model unexpected behaviors from the environment.

A complete trace structure has to be *receptive*, i.e., it has not to control its inputs. Formally, receptiveness is defined by means of games between the *system* and the *environment*. The environment is allowed to add any finite number (possibly 0) of input actions at each move, while the system is allowed to add at most one output action at each move. A *strategy* for the system is a function giving an output action (or nothing) for each finite sequence of environment moves. A strategy for the system is a winning strategy if it leads to a trace of $P = S \cup F$ for any behavior of the environment. A complete trace structure is said to be receptive if, for each prefix x of a trace of P, there exists a winning strategy for the system starting the game after x has occurred.

The parallel composition of two complete trace structures $\mathcal{T} = (I, O, S, F)$ and $\mathcal{T}' = (I', O', S', F')$, denoted by $\mathcal{T} \| \mathcal{T}'$, is a new complete trace structure $((I \cup I') \setminus (O \cup O'), O \cup O', S'', F'')$ where S'' and F'' are those traces that projected on $I \cup O$ give traces of S and F respectively, and projected on $I' \cup O'$ give traces of S' and F' respectively. If $F'' = \emptyset$, we say that $\mathcal{T} \| \mathcal{T}'$ is failure free, and this means that $\mathcal{T}'$ can safely interact with $\mathcal{T}$. We say that a complete trace structure $\mathcal{T}_1$ *conforms* to a complete trace structure $\mathcal{T}_2$ ($\mathcal{T}_1 \preceq \mathcal{T}_2$) if each complete trace structure $\mathcal{T}'$ that can safely interact with $\mathcal{T}_2$ can also safely interact with $\mathcal{T}_1$. The relation $\preceq$ is called *conformation preorder* and expresses the notion of implementation for complete trace structures.

The idea of a failure, which is not the same as the idea of [8], is the aspect of complete trace structures that has no corresponding notion within the other theories we are considering in this paper. In fact, I/O automata and Receptive Process Theory are receptive in the sense that they always respond to input stimuli. Unexpected inputs are modeled by moving to special trap states from which any behavior is possible. By ignoring failures it is possible to show that each finite state I/O automaton can be described as a failure free trace structure and that each failure free trace structure can be described by an I/O automaton. Moreover, the conformation preorder coincides with the fair preorder.

Proposition 14. *Given a finite state I/O automaton $\mathcal{A}$, the tuple $Ctrace(\mathcal{A}) = (in(\mathcal{A}), out(\mathcal{A}), ftraces(\mathcal{A}), \emptyset)$ is a complete trace structure.* □

Proposition 15. *Given a failure free trace structure $\mathcal{T} = (I, O, S, \emptyset)$ there exists an I/O automaton $Autom(\mathcal{T})$ with input actions I, output actions O, and such that $ftraces(Autom(\mathcal{T})) = S$.* □

The proof of Proposition 14 is constructive and essentially transforms a finite state I/O automaton into a finite state machine with normal and Büchi acceptance states. The proof of Proposition 15 is based on the results about *union-game realizable* languages of [13] and builds an infinite state automaton starting from a complete trace structure $\mathcal{T}$. An open problem is to find a construction giving a finite state I/O automaton from a complete trace structure or at least show whether or not such a finite state automaton exists.

Proposition 16. *Given two finite state I/O automata $\mathcal{A}_1, \mathcal{A}_2$, $\mathcal{A}_1 \sqsubseteq_F \mathcal{A}_2$ iff $Ctrace(\mathcal{A}_1) \preceq Ctrace(\mathcal{A}_2)$. Given two failure free complete trace structures $\mathcal{T}_1, \mathcal{T}_2$, $\mathcal{T}_1 \preceq \mathcal{T}_2$ iff $Autom(\mathcal{T}_1) \sqsubseteq_F Autom(\mathcal{T}_2)$.* □

8 Conclusion

We have analyzed the fair and quiescent preorders of I/O automata and the theory of testing in the common framework of transition systems. The two theories, although apparently different, are based on similar intuitions.

As a result of this paper we have given a class of I/O automata for which the quiescent preorder is equivalent to a simple fair preorder not distinguishing between the subcomponents of a system. We have also considered some cases in which the quiescent preorder is equivalent to the standard fair preorder. Secondly we have shown the relationship between the theory of I/O automata and that of testing both by encoding the information contained in the interfaces of I/O automata into general labeled transition systems and by defining testing preorders directly on I/O automata. Our main result is that for strongly convergent and input enabled transition systems the quiescent preorder of I/O automata coincides with the reversed MUST preorder. Finally we have shown how other two widely known theories of receptive systems [6, 9] relate to the above models.

We also have outlined some of the problems of divergences. It is not clear yet what a divergence should really represent. The theory of testing deals with divergences in a way similar to that of standard denotational semantics for sequential processes by considering a divergence as an absence of information; I/O automata deal with divergences by identifying pure divergences with deadlocks. This problem will be the argument of further research.

Acknowledgments: I am grateful to Rocco De Nicola, Roberto Gorrieri, Alber Meyer, Jørgen Søgaard-Andersen, Frits Vaandrager and David Wald for helpful discussions.

References

1. S. Brookes, C. Hoare, and A. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.
2. S. Brookes and A. Roscoe. An improved failures model for communicating processes. In S. Brookes, A. Roscoe, and G. Winskel, editors, *Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 281–305. Springer-Verlag, july 1984. Lecture and Notes in Computer Science, Vol 197.
3. R. De Nicola. *Testing Equivalences and Fully Abstract Models for Communicating Processes.* PhD thesis, Department of Computer Science, University of Edinburgh, 1985.
4. R. De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, 24:211–237, 1987.
5. R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
6. D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* ACM Distinguished Dissertations. MIT Press, 1988.
7. M. Hennessy. *Algebraic Theory of Processes.* MIT Press, Cambridge, Massachusetts, 1988.
8. C. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, Englewood Cliffs, 1985.
9. M. Josephs. Receptive process theory. *Acta Informatica*, 29:17–31, 1992.
10. N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the* 6^{th} *Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, Vancouver, Canada, August 1987. A full version is available as MIT Technical Report MIT/LCS/TR-387.
11. N. Lynch and F. Vaandrager. Forward and backward simulations for timing-based systems. In J. de Bakker, C. Huizing, W. d. Roever, and G. Rozenberg, editors, *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lecture Notes in Computer Science*, pages 397–446. Springer-Verlag, 1991.
12. D. Muller and W. Bartky. A theory of asynchronous circuits. *Annals of the Computation Laboratory of Harvard University. Volume XXIX: Proceedings of an International Symposium on the Theory of Switching, Part I*, pages 204–243, 1959.
13. N. Reingold, D. Wang, and L. Zuck. Games I/O automata play. In W. Cleaveland, editor, *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science*, pages 325–339. Springer-Verlag, 1992.
14. R. Segala. A process algebraic view of I/O automata. Technical Memo MIT/LCS/TR-557, Laboratory for Computer Science, MIT, Cambridge, MA 02139, October 1992.
15. E. Stark. On the relations computable by a class of concurrent automata. In *Proceedings of the 1990 SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, 1990.
16. F. Vaandrager. On the relationship between process algebra and Input/Output automata. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science*, 1991.
17. J. Welch, L. Lamport, and N. Lynch. A lattice-structured proof technique applied to a minimum spanning tree algorithm. Technical Report MIT/LCS/TM-361, Laboratory for Computer Science, MIT, June 1988.