INFORMATION AND COMPUTATION **138**, 194–210 (1997)
ARTICLE NO. IC972652

Quiescence, Fairness, Testing, and the Notion of Implementation*

Roberto Segala

Dipartimento di Scienze dell'Informazione, Università di Bologna, Bologna 40127, Italy

Two formalisms for concurrency, the Input/Output automaton model and the theory of testing, are compared and are shown to have common foundations. The relationship between the fair and quiescent preorders of I/O automata is investigated and the two preorders are shown to coincide subject to some restrictions. 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 I/O 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 I/O automata.

1. INTRODUCTION

The algorithms used in modern computers and networks are becoming more and more complex. Since chances of erroneous design or implementation increase with the complexity of a system, the problem of specifying and verifying concurrent systems is receiving increasing attention. The main idea is to specify a system, build an implementation, and then verify that the implementation meets the specification. Implicitly, it is assumed that there is a formalism for the representation of the specification and the implementation, and that there is a notion of implementation within the formalism.

In this paper we compare the notions of implementation that appear in two of the main formalisms for concurrent and distributed systems. Specifically, we compare general labeled transition systems equipped with testing and failure preorders [DH84, DeN85, Hen88] and I/O automata equipped with fair and quiescent preorders [LT87, Vaa91, Seg92]. The former formalism was developed within process algebras with the aim of defining an observational equivalence on labeled transition systems with less discriminating power than bisimulation [Par81] and with a strong intuition of when two processes should be considered equivalent; the latter formalism was developed for the scope of specifying and verifying distributed

* This is a full and revised version of [Seg93].

0890-5401/97 $25.00

systems. Our main result is that, although the two formalisms originate from different backgrounds and apparently address different classes of problems, in many cases the two formalisms capture the same idea.

The main intuition for the testing preorders is that processes should be compared based on their interaction with an external environment, and thus they should be 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 a success of E. Two preorders can be defined over objects: the MAY *preorder*, consisting of inclusion of experiments that may succeed, and the MUST *preorder*, consisting of inclusion of experiments that must succeed.

In [DeN85] it is shown that the MUST preorder coincides with the reversed *failure preorder* of CSP [Hoa85] on objects that never diverge, i.e., objects that never perform infinite computation not visible from the external environment. The failure preorder is proposed as an implementation relation within CSP [BHR84, BR85] and the intuitive idea for its use is that an implementation has to be more deterministic than its specification. Therefore the result of [DeN85] also suggests a possible use of the MUST preorder as an implementation relation.

Within the theory of I/O automata each object is associated with an interface with the external environment. An interface specifies the set of communication *actions* an I/O automaton can engage in, and partitions the communication actions into input, output, and internal. Input actions are always enabled, i.e., they can occur from everywhere, and output actions cannot be blocked by the external environment, i.e., the external environment of an I/O automaton $\mathscr{A}$ should always be ready to engage in any output action of $\mathscr{A}$. In other words, the output and internal actions of an I/O automaton $\mathscr{A}$, also said to be *locally controlled*, are under the sole control of $\mathscr{A}$. An I/O automaton has also subcomponents that are identified through a partition of its locally controlled actions.

The notion of implementation for I/O automata is expressed through fair trace inclusion, where a fair trace of an I/O automaton $\mathscr{A}$ is a sequence of actions (possibly infinite) that originate from a computation where every subcomponent of $\mathscr{A}$ has infinitely many chances to communicate. 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 action the implementation must do the same.

The main criticism against the I/O automaton model is that it is too restrictive: conditions such as 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 [AS85] that are not captured in general by using testing preorders and that have revealed to be important for the verification of many distributed systems [LT87, WLL88, SLL93].

A first step toward the study of the relationship between the process algebraic approach and the I/O automaton approach is in [Vaa91], where the impact of input enabling on the operators of a generic process algebra is analyzed. The analysis of [Vaa91] includes the definition of several preorder relations that gradually approximate the fair preorder. Among these is the *quiescent preorder*, which is a 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 can occur. In [Seg92] the quiescent preorder is studied within a process algebraic theory of I/O automata and a fixpoint theorem is put forward. In [Seg92] there is also an attempt at using the quiescent preorder as an implementation relation; however, it is shown that the quiescent preorder does not provide an intuitively reasonable notion of implementation in general; some restrictions are necessary.

In this paper we study the relationship between the fair preorder and the theory of testing by using the quiescent preorder as an intermediate relation. We first show that the fair and quiescent preorders coincide whenever some continuity properties hold for fair traces. Then we encode I/O automata into the testing framework by expliciting the ideas of internal and external choices that are embedded within the Input/Output structure, and we show that for strongly convergent and finitely branching systems the quiescent preorder of I/O automata coincides with the reversed MUST preorder of the testing theory up to encoding of the former model into the latter. Finally, we define a theory of testing similar to that of [Hen88] directly on I/O automata. The new testing scenario is simple and natural since an experimenter is just an I/O automaton. Once again, the main result is that for strongly convergent and finitely branching I/O automata the quiescent preorder coincides with the reversed MUST preorder.

A problem that we leave open is the issue of divergences. Our equivalence results show that our understanding of convergent processes is sufficiently strong; however, since the theories studied in this paper are different when divergent processes are considered, we do not understand yet how to deal with divergences. The theory that seems to better deal with divergences is that of I/O automata, since it allows one component of a system to diverge while the rest of it is working properly; however, a deadlocked process is equivalent to a divergent process when fair traces are considered, while a key point of the theory of testing is that a divergent process is distinct from a deadlocked process. One of the reasons the approach to divergences of I/O automata is not used within process algebraic theories is that there is no nice mathematical theory for fairness. In particular, there is no fixpoint theorem.

The rest of the paper is organized as follows. Section 2 gives a standard definition of transition systems; Sections 3 and 4 give an overview of I/O automata and the theory of testing; Section 5 studies the relationship between the quiescent and the fair preorders; Section 6 encodes I/O automata into general labeled transition systems and studies the relationship between the quiescent and the must preorders; Section 7 defines a theory of testing directly on I/O automata and shows the equivalence of the quiescent and the reversed MUST preorders; Section 8 gives some concluding remarks.

2. PRELIMINARIES

In this section we give a definition of transition systems in the style of [LT87]. This section is intended mainly as a reference for the terminology and the notation that we use.

DEFINITION 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. We denote by acts(T) the set ext(T) $\cup$ int(T) of actions.
- a transition relation steps(T) $\subseteq$ states(T) $\times$ acts(T) $\times$ states(T).

A transition $(q, a, q') \in \text{steps}(T)$ is also denoted by $q \xrightarrow{a} q'$. We extend the notion of transition to finite sequences of symbols by saying that $q \xrightarrow{a_1 \cdots a_n} q'$ iff there is a collection $q_0, ..., q_n$ of states with $q_0 = q$ and $q_n = q'$ such that $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$. A derived transition relation, abstracting from internal computation, is defined as $q \overset{a_1 \cdots a_n}{\Longrightarrow} q'$ iff there is a collection $s_0, s_1, ..., s_n$ of finite sequences of internal actions such that $q \xrightarrow{s_0 a_1 s_1 \cdots a_n s_n} q'$.

An *execution fragment* of a transition system T is a (finite or infinite) sequence of alternate states and actions $\alpha = q_0 a_1 q_1 a_2 q_2 \cdots$ starting with a state and, if the execution fragment is finite, ending in a state, where each (q_i, a_{i+1}, q_{i+1}) is an element of steps(T). An *execution* is an execution fragment whose first state is a start state.

The *external trace* of an execution fragment α of a transition system T, written $\text{etrace}_T(\alpha)$, or just etrace(α) when T is clear, is the list obtained by restricting α to the set of external actions of T, i.e., etrace(α) = $\alpha \lceil \text{ext}(T)$. We say that s is an *external trace* of a transition system T if there exists an execution α of T with etrace(α) = s. We denote by etraces*(T), and etraces(T) the sets of finite, and all external traces of T, respectively.

A state q of a transition systems T is said to *enable* a transition if there is an action a and a state q' such that $(q, a, q') \in \text{steps}(T)$. A transition system T is *finitely branching* iff each state of T enables finitely many transitions. Finite branching is an important property for transition systems since it guarantees that the finite external traces of a transition system are sufficient to characterize the infinite external traces as well.

PROPOSITION 2.2. [LV91]. *Let T be a finitely branching transition system, and let s be an infinite sequence of external actions of T. Suppose that each prefix of s is an external trace of T. Then s is an external trace of T.* ▌

Transition systems can be composed in parallel. We use the CSP synchronization style [Hoa85], where two transition systems synchronize on their common actions

and evolve independently on the others. Two transition systems T_1, T_2 are *compatible* iff $\text{int}(T_1) \cap \text{acts}(T_2) = \text{acts}(T_1) \cap \text{int}(T_2) = \emptyset$. The *parallel composition* $T_1 \| T_2$ of two compatible transition systems T_1, T_2 is the transition system T such that

1. $\text{states}(T) = \text{states}(T_1) \times \text{states}(T_2)$,
2. $\text{start}(T) = \text{start}(T_1) \times \text{start}(T_2)$,
3. $\text{sig}(T) = (\text{ext}(T_1) \cup \text{ext}(T_2), \text{int}(T_1) \cup \text{int}(T_2))$, and
4. $((q_1, q_2), a, (q_1', q_2')) \in \text{steps}(T)$ iff either
   (a) $(q_1, a, q_1') \in \text{steps}(T_1)$ and $(q_2, a, q_2') \in \text{steps}(T_2)$,
   (b) $(q_1, a, q_1') \in \text{steps}(T_1)$, $a \notin \text{acts}(T_2)$, and $q_2 = q_2'$, or
   (c) $(q_2, a, q_2') \in \text{steps}(T_2)$, $a \notin \text{acts}(T_1)$, and $q_1 = q_1'$.

3. I/O AUTOMATA

In this section we give a definition of I/O automata. The reader interested in more information on I/O automata is referred to [LT87].

Given a transition system T, an action $a \in \text{acts}(T)$ is *enabled* from a state $q \in \text{states}(T)$ iff there exists a state $q' \in \text{states}(T)$ such that $(q, a, q') \in \text{steps}(T)$. The set of enabled actions from a state $q \in \text{states}(T)$ is denoted by *enabled*(q).

DEFINITION 3.1. (I/O Automata). An *I/O automaton* $\mathscr{A}$ is a transition system with the following extra structure:

- A partition of $\text{ext}(\mathscr{A})$, called *external action signature*, $\text{esig}(\mathscr{A}) = (\text{in}(\mathscr{A}), \text{out}(\mathscr{A}))$ consisting of input and output actions, respectively. We denote by $\text{local}(\mathscr{A})$ the set $\text{int}(\mathscr{A}) \cup \text{out}(\mathscr{A})$ of *locally controlled* actions of $\mathscr{A}$.
- A partition $\text{part}(\mathscr{A})$ of $\text{local}(\mathscr{A})$.

Furthermore, every state of $\mathscr{A}$ enables all input actions. We say that $\mathscr{A}$ is *input enabled*.

Informally, the input actions of an I/O automaton model the events that are under the control of the external environment (and thus are always enabled), while output actions model external events that are under the control of the system. The partition of the locally controlled actions models the subcomponents of an I/O automaton. Specifically, an I/O automaton $\mathscr{A}$ can be thought of as a collection of subcomponents, each one controlling a part of the locally controlled actions of $\mathscr{A}$.

Parallel composition can be extended easily to I/O automata: the compatibility condition needs to be strengthened by requiring two I/O automata not to have any common output actions. The output actions of the composition are the output actions of the components; the partition of the locally controlled actions of the composition is the union of the partitions of the components. This last condition ensures that the subcomponents of the new I/O automaton are those of the two components.

DEFINITION 3.2. (Quiescent Preorder [Vaa91]). The set of *quiescent traces* of an I/O automaton $\mathscr{A}$ is the set of finite external traces of $\mathscr{A}$ that lead to a state from which only input actions are enabled, i.e.,

$$\text{qtraces}(\mathscr{A}) = \{s \in \text{ext}(\mathscr{A})^* \mid \exists_{q_0 \in \text{start}(\mathscr{A})} \exists_{q \in \text{states}(\mathscr{A})}\, q_0 \stackrel{s}{\Longrightarrow} q \text{ and } \text{enabled}(q) = \text{in}(\mathscr{A})\}.$$

A state q such that $\text{enabled}(q) = \text{in}(\mathscr{A})$ is also called a *quiescent state.*

Given two I/O automata $\mathscr{A}_1$ and $\mathscr{A}_2$ with the same external action signature, the *quiescent preorder* is defined as

$$\mathscr{A}_1 \sqsubseteq_Q \mathscr{A}_2 \text{ iff } \text{etraces}^*(\mathscr{A}_1) \subseteq \text{etraces}^*(\mathscr{A}_2) \text{ and } \text{qtraces}(\mathscr{A}_1) \subseteq \text{qtraces}(\mathscr{A}_2).$$

DEFINITION 3.3. (Fair Preorder [LT87]). An execution α of an I/O automaton $\mathscr{A}$ is *fair* iff either α is quiescent, or α is infinite and for each class $p \in \text{part}(\mathscr{A})$, either actions from p appear infinitely often in α or states from which no action from p is enabled appear infinitely often in α. A *fair trace* of $\mathscr{A}$ is the external trace of a fair execution of $\mathscr{A}$. The set of fair traces of an I/O automaton $\mathscr{A}$ is denoted by $\text{ftraces}(\mathscr{A})$.

Given two I/O automata $\mathscr{A}_1$ and $\mathscr{A}_2$ with the same external action signature, the *fair preorder* is defined as

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

Based on the fair preorder, [LT87] introduces a notion of implementation on I/O automata: $\mathscr{A}_1$ implements $\mathscr{A}_2$ iff $\mathscr{A}_1 \sqsubseteq_F \mathscr{A}_2$. 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 [LT87] 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." The above justification is rather intuitive; in this paper we enforce it by relating the fair and quiescent preorders to the theory of testing.

4. THE THEORY OF TESTING

Another method for comparing transition systems is based on the observation of the interactions between a transition system and an external *experimenter* [DH84, DeN85, Hen88]. 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.* The experimenter E synchronizes with T on all external actions except for w. In other words, E runs in parallel with T. An *experiment* x is an execution of $T \parallel E$ which is infinite or ends in a deadlocked state (complete execution of $T \parallel E$). 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 $T \parallel E$. We say that T MUST E if each experiment of $T \parallel E$ is successful.

DEFINITION 4.1. (Testing Preorders). Given two transition systems T_1, T_2, define the following preorders:

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

The MAY and MUST preorders can be characterized differently without referring to a notion of external experimenter [DeN85]. In particular the MAY preorder coincides with external trace inclusion; for the MUST preorder we need some more structure.

DEFINITION 4.2. (Convergence). Given a transition system T and a state $q \in \text{states}(T)$, we write $q\uparrow$ if q has an infinite internal computation, i.e., $q \xrightarrow{\tau_1} q_1 \xrightarrow{\tau_2} q_2 \xrightarrow{\tau_3} \ldots$, 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 that q is *divergent*. The same notion can be relativized to sequences of actions by defining

- $q\downarrow\varepsilon$ iff $q\downarrow$
- $q\downarrow as$ iff $q\downarrow$ and $q \stackrel{a}{\Longrightarrow} q'$ implies $q'\downarrow s$.

We write $q\Downarrow$ if $q\downarrow s$ for each $s \in \text{ext}(T)^*$ and we write $T\Downarrow$ if $q\Downarrow$ for each $q \in \text{start}(T)$. If $T\Downarrow$ we say that T is *strongly convergent*. In other words, $T\Downarrow$ means that no divergent state is reached for any sequence of actions s.

DEFINITION 4.3. (Auxiliary Functions). Given a transition system T, a state $q \in \text{states}(T)$, a set of states $Q \subseteq \text{states}(T)$, a sequence of external actions $s \in \text{ext}(T)^*$, and a set of external actions A, we give the following definitions:

1. $\text{wenabled}(q) = \{a \in \text{ext}(T) \mid q \stackrel{a}{\Longrightarrow} q' \text{ for some } q' \in \text{states}(T)\}$
2. $q \text{ after } s = \{q' \in \text{states}(T) \mid q \stackrel{s}{\Longrightarrow} q'\}$
3. $Q \text{ after } s = \bigcup_{q \in Q} (q \text{ after } s)$
4. $T \text{ after } s = \text{start}(T) \text{ after } s$
5. Q MUST A iff $\text{wenabled}(q) \cap A \neq \emptyset$ for each $q \in Q$.

PROPOSITION 4.4. (DeN85) *Given two finitely branching and strongly convergent transition systems T_1 and T_2 with the same external actions, $T_1 \sqsubseteq_{\text{MUST}} T_2$ iff, for each $s \in \text{ext}(T_1)^*$ and each $A \subseteq \text{ext}(T_1)$, T_1 after s MUST A implies T_2 after s MUST A.* ∎

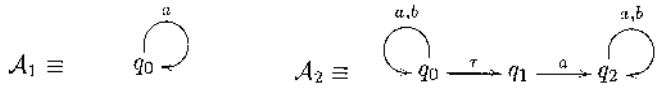
The MUST preorder can be characterized also for non strongly convergent transition systems. We are not interested in such characterization in this paper, and we refer the interested reader to [DeN85, Hen88]. Another characterization of the MUST preorder for strongly convergent transition systems is given by the *failure preorder* [Hoa85]. The failure preorder was first introduced to give a semantics to CSP which is able to detect deadlocks and distinguish between internal and external choice. The observations are based on failures, i.e., finite external traces augmented with sets of actions that a process may refuse to perform afterward. In [BHR84] the failure preorder is claimed to provide some notion of implementation with the informal justification that an implementation is more deterministic than its specification. In [DeN87] the failure preorder is shown to be equivalent to the

reversed MUST preorder on strongly convergent transition systems. Thus, the reversed MUST preorder expresses the same idea of implementation as the failure preorder on strongly convergent transition systems.

5. QUIESCENT AND FAIR PREORDERS

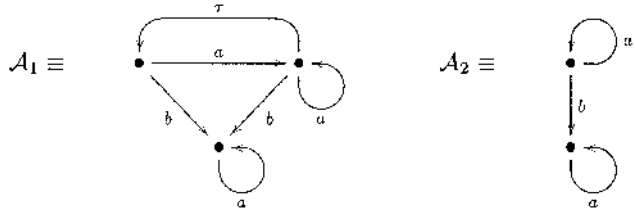
The fair preorder is the only preorder relation among those we consider that observes the infinite behavior of an I/O automaton. It is the basic notion of implementation for I/O automata. The intuitive idea behind its use is that, due to input enabling, an implementation must accept all external stimuli; moreover, due to fairness, an implementation must provide output whenever the specification must do so. However, the infinite fair traces of an I/O automaton cannot be derived from its finite fair traces. This is the main reason for the difference between the fair and quiescent preorders.

EXAMPLE 5.1. Consider the I/O automata



where a is an input action, b is an output action, τ is an internal action, and the partitions of the locally controlled actions contain a single class. It is easy to observe that each finite sequence a^n is a quiescent (and thus fair) trace of $\mathcal{A}_2$ since it is enough to loop n times on q_0 to then move to q_1. The sequence a^∞, however, is not a fair trace of $\mathcal{A}_2$ since in any execution action b is enabled in all states but at most one. Therefore, $\mathcal{A}_1 \sqsubseteq_Q \mathcal{A}_2$ while $\mathcal{A}_1 \not\sqsubseteq_F \mathcal{A}_2$.

EXAMPLE 5.2. Consider the I/O automata



where a is an input action, b is an output action, τ is an internal action, and the partitions of the locally controlled actions contain a single class. The I/O automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent according to the quiescent preorder since they have the same external traces and their quiescent traces are all finite external traces containing at least a b action. The external trace a^∞, however, is a fair trace of $\mathcal{A}_1$ but not a fair trace of $\mathcal{A}_2$.

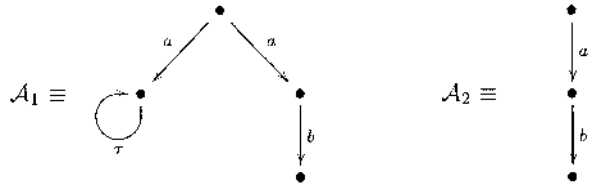
The problem outlined in Example 5.1 is that the limit of a chain of fair traces of an I/O automaton $\mathcal{A}$ is not necessarily a fair trace of $\mathcal{A}$; the problem outlined in Example 5.2 is that an infinite fair trace of an I/O automaton $\mathcal{A}$ is not necessarily the limit of a chain of finite fair traces of $\mathcal{A}$, i.e., it is not necessarily approximable with finite fair traces.

DEFINITION 5.3. (Fair Continuity). An I/O automaton $\mathscr{A}$ is *fair continuous* if the limit of any chain of fair traces of $\mathscr{A}$ is a fair trace of $\mathscr{A}$.

DEFINITION 5.4. (Fair Approximability). An I/O automaton $\mathscr{A}$ is *fair approximable* if each infinite fair trace of $\mathscr{A}$ is the limit of a chain of finite fair traces of $\mathscr{A}$.

Another problem derives from the fact that a finite fair trace is not necessarily a quiescent trace.

EXAMPLE 5.5. Consider the I/O automata



where a and b are output actions and the partitions of the locally controlled actions contain a single class. The I/O automata $\mathscr{A}_1$ and $\mathscr{A}_2$ are equivalent according to the quiescent preorder, but not equivalent according to the fair preorder: the trace a is a fair trace of $\mathscr{A}_1$ but not a fair trace of $\mathscr{A}_2$. ▌

DEFINITION 5.6. (Quiescent Detectability). An I/O automaton $\mathscr{A}$ is *quiescent detectable* if each finite fair trace of $\mathscr{A}$ is also a quiescent trace of $\mathscr{A}$.

Quiescent detectability is implied trivially by strong convergence. In our analysis we study strongly convergent systems only, and thus quiescent detectability is not an important issue.

THEOREM 5.7. (Fair versus Quiescent Preorder). *Consider two strongly convergent* I/O *automata* $\mathscr{A}_1$ *and* $\mathscr{A}_2$. *Then*

1. $\mathscr{A}_1 \sqsubseteq_F \mathscr{A}_2$ *implies* $\mathscr{A}_1 \sqsubseteq_Q \mathscr{A}_2$;
2. *if* $\mathscr{A}_1$ *is fair approximable, and* $\mathscr{A}_2$ *is fair continuous, then* $\mathscr{A}_1 \sqsubseteq_Q \mathscr{A}_2$ *implies* $\mathscr{A}_1 \sqsubseteq_F \mathscr{A}_2$.

Proof. Let $\mathscr{A}_1 \sqsubseteq_F \mathscr{A}_2$ and let s be a quiescent trace of $\mathscr{A}_1$. By definition, s is finite and is a fair trace of $\mathscr{A}_1$, and hence a fair trace of $\mathscr{A}_2$. Since $\mathscr{A}_2$ is strongly convergent, and thus quiescent detectable, s is a quiescent trace of $\mathscr{A}_2$. External trace inclusion follows directly from the facts that each prefix of a fair trace of an I/O automaton $\mathscr{A}$ is an external trace of $\mathscr{A}$ and each external trace of $\mathscr{A}$ can be extended to a fair trace of $\mathscr{A}$ [LT87].

Let $\mathscr{A}_1 \sqsubseteq_Q \mathscr{A}_2$ and let s be a fair trace of $\mathscr{A}_1$. If s is a quiescent trace of $\mathscr{A}_1$, then, from the hypothesis, s is a quiescent trace of $\mathscr{A}_2$, and therefore s is trivially a fair trace of $\mathscr{A}_2$. If s is not quiescent, then, since $\mathscr{A}_1$ is strongly convergent, and thus quiescent detectable, s is an infinite trace. From fair approximability of $\mathscr{A}_1$, s is the limit of a chain $s_1, s_2, \ldots$ of finite fair traces of $\mathscr{A}_1$, and from quiescent detectability of $\mathscr{A}_1$, each s_i is a quiescent trace of $\mathscr{A}_1$. Since $\mathscr{A}_1 \sqsubseteq_Q \mathscr{A}_2$, each s_i is a quiescent, and thus fair, trace of $\mathscr{A}_2$. From fair continuity of $\mathscr{A}_2$, s is a fair trace of $\mathscr{A}_2$. ▌

6. QUIESCENT AND MUST PREORDERS

In this section we compare the quiescent and MUST preorders. We do not consider explicitly the partitions of the locally controlled actions of an I/O automaton, since the partitions do not affect the quiescent traces of an I/O automaton nor the must preorder defined on labeled transition systems.

We start by encoding I/O automata into the testing framework. 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, a nondeterministic choice between input actions is intended to be an external choice, while a nondeterministic choice between output actions is intended to be an internal choice. The following definition formalizes the intuition above by providing an encoding of I/O automata onto general labeled transition systems. The encoded object is still an I/O automaton; however, the I/O structure is exploited. We assume the existence of at least one internal action, denoted by τ.

DEFINITION 6.1. (I/O Explicit Automata). Let $\mathscr{A}$ be an I/O automaton. The *I/O explicit automaton* associated with $\mathscr{A}$ is defined as $\mathscr{F}(\mathscr{A}) = (Q, \text{start}(\mathscr{A}), ((\text{in}(\mathscr{A}), \text{out}(\mathscr{A})), \text{int}(\mathscr{A})), \text{steps})$ where

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

2.
$$\begin{aligned}\text{steps} = {} & \{q \xrightarrow{\tau} qq'_a \mid q \in \text{states}(\mathscr{A}), qq'_a \in Q\} \cup \\ & \{qq'_a \xrightarrow{a} q' \mid qq'_a \in Q\} \cup \\ & \{qq'_a \xrightarrow{b} q'' \mid q \xrightarrow{b} q'' \in \text{steps}(\mathscr{A}), qq'_a \in Q, b \in \text{in}(\mathscr{A})\} \cup \\ & \{q \xrightarrow{b} q' \in \text{steps}(\mathscr{A}) \mid b \in \text{in}(\mathscr{A})\}.\end{aligned}$$

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

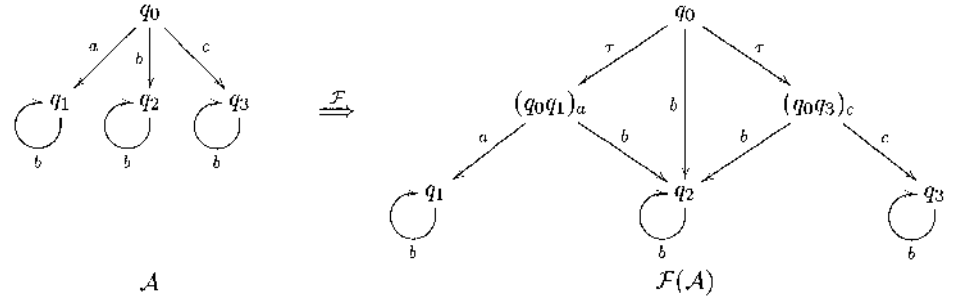


FIG. 1. An example of encoding. $\text{in}(\mathscr{A}) = \{b\}$, $\text{out}(\mathscr{A}) = \{a, c\}$.

EXAMPLE 6.2. Figure 1 shows an example of encoding. The left I/O automaton $\mathscr{A}$ is converted into the I/O automaton $\mathscr{F}(\mathscr{A})$. Actions a and c are output actions, while b is an input action. From state q_0 of $\mathscr{A}$ there are two outgoing transitions labeled with an output action ($q_0 \xrightarrow{a} q_1$ and $q_0 \xrightarrow{a} q_3$), thus two new states ($(q_0q_1)_a$ and $(q_0q_3)_c$) are introduced, and from state q_0 of $\mathscr{F}(\mathscr{A})$ there are two internal transitions to the new states plus all the original transitions labeled with input actions. Thus, the transition system $\mathscr{F}(\mathscr{A})$ decides internally 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.

PROPOSITION 6.3. *Let $\mathscr{A}$ be an I/O automaton. Then*

1. $\text{traces}(\mathscr{A}) = \text{etraces}(\mathscr{F}(\mathscr{A}))$
2. $\text{qtraces}(\mathscr{A}) = \text{qtraces}(\mathscr{F}(\mathscr{A}))$.

Proof. The left to right inclusions are trivial since, if $q \xrightarrow{a} q'$ in $\mathscr{A}$, then $q \stackrel{a}{\Longrightarrow} q'$ in $\mathscr{F}(\mathscr{A})$. For the converse inclusions it is enough to observe that none of the states of the form qq'_a is quiescent and that, if $q \xrightarrow{\tau} qq'_a \xrightarrow{b} q'$ in $\mathscr{F}(\mathscr{A})$, then $q \xrightarrow{b} q'$ in $\mathscr{A}$. ∎

It is already known from [Hen88] that the MAY preorder coincides with external trace inclusion. Here we concentrate on our main theorem, which states that, for strongly convergent and finitely branching I/O automata, the quiescent preorder of I/O automata is equivalent to the reversed MUST preorder up to encoding.

THEOREM 6.4. (Quiescent versus MUST Preorder). *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be strongly convergent, finitely branching* I/O *automata. Then*

$\mathscr{A}_1 \sqsubseteq_Q \mathscr{A}_2$ iff $\mathscr{F}(\mathscr{A}_2) \sqsubseteq_{\text{MUST}} \mathscr{F}(\mathscr{A}_1)$.

Proof. Let $\mathscr{A}_1, \mathscr{A}_2$ be strongly convergent and finitely branching. Let $\mathscr{A}_1 \sqsubseteq_Q \mathscr{A}_2$ and suppose by contradiction that $\mathscr{F}(\mathscr{A}_2) \not\sqsubseteq_{\text{MUST}} \mathscr{F}(\mathscr{A}_1)$. From Proposition 4.4 there is a sequence of actions s and a set of actions A such that $\mathscr{F}(\mathscr{A}_2)$ after s MUST A and not $\mathscr{F}(\mathscr{A}_1)$ after s MUST A. Since $\mathscr{A}_1$ and $\mathscr{A}_2$ are input enabled, the set A contains no input actions. We distinguish two cases:

1. $A = \varnothing$.

Then s is not a trace of $\mathscr{A}_2$ ($\mathscr{A}_2$ after s MUST $\varnothing$) while s is a trace of $\mathscr{A}_1$ (not $\mathscr{A}_1$ after s MUST $\varnothing$), a contradiction.

2. A contains at least one output action.

Since by construction of $\mathscr{F}$ each stable state of $\mathscr{F}(\mathscr{A}_2)$ (a state is stable if it does not enable any internal action) enables at most one output action, A has to be a superset of $A' = \{a \in \text{out}(\mathscr{A}_2) \mid \text{sa} \in \text{etraces}(\mathscr{A}_2)\}$. Observe that s is not a quiescent trace of $\mathscr{A}_2$ since in each state reachable with s it is still possible to perform at least one output action ($\mathscr{A}_2$ after s MUST A and $A \subseteq \text{out}(\mathscr{A}_2)$). On the other hand, since each external trace of $\mathscr{A}_1$ is an external trace of $\mathscr{A}_2$, no state of $\mathscr{A}_1$ after s can enable an output action which is not in A', and therefore, since there is a state of $\mathscr{F}(\mathscr{A}_1)$

after s from which no action of A can be performed, and since $\mathscr{A}_1$ is strongly convergent, there is a quiescent state in $\mathscr{F}(\mathscr{A}_1)$ after s. Thus, s is a quiescent trace of $\mathscr{A}_1$, which contradicts $\mathscr{A}_1 \sqsubseteq_Q \mathscr{A}_2$.

Conversely, let $\mathscr{F}(\mathscr{A}_2) \sqsubseteq_{\text{MUST}} \mathscr{F}(\mathscr{A}_1)$ and suppose by contradiction that $\mathscr{A}_1 \not\sqsubseteq_Q \mathscr{A}_2$. Since from [Hen88] the MUST preorder implies the reverse external trace inclusion on strongly convergent transition systems, there exists a quiescent trace s of $\mathscr{A}_1$ which is an external trace of $\mathscr{A}_2$ but not a quiescent trace of $\mathscr{A}_2$. Thus, from each state of $\mathscr{F}(\mathscr{A}_2)$ reachable via s it is possible to perform at least one output action, which means that $\mathscr{F}(\mathscr{A}_2)$ after s MUST out$(\mathscr{A}_2)$. On the other hand, since s is a quiescent trace of $\mathscr{A}_1$, there is a state of $\mathscr{F}(\mathscr{A}_1)$ reachable via s from which no output action is possible; therefore, not $\mathscr{F}(\mathscr{A}_1)$ after s MUST out$(\mathscr{A}_2)$. This contradicts $\mathscr{F}(\mathscr{A}_2) \sqsubseteq_{\text{MUST}} \mathscr{F}(\mathscr{A}_1)$. ▮

7. TESTING FOR I/O AUTOMATA

In Section 6 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 also reduce the power of an experimenter according to the interaction schemas of I/O automata and define a theory of testing directly on I/O automata. In this section we show that also this approach leads to the quiescent preorder when dealing with strongly convergent and finitely branching I/O automata. Thus, by reducing the power of an experimenter we can eliminate the encoding.

An experimenter $\mathscr{E}$ for an I/O automaton $\mathscr{A}$ is an I/O automaton compatible with $\mathscr{A}$ whose input actions are the output actions of $\mathscr{A}$ $(\text{in}(\mathscr{E}) = \text{out}(\mathscr{A}))$ and whose output actions are the input actions of $\mathscr{A}$ plus an action w, called the success action $(\text{out}(\mathscr{E}) = \text{in}(\mathscr{A}) \cup \{w\})$. The experimenters for I/O automata are less powerful than those of Section 6: an experimenter $\mathscr{E}$ can only control the input actions of an automaton $\mathscr{A}$. We denote the new testing preorders by $\sqsubseteq'_{\text{MAY}}$, $\sqsubseteq'_{\text{MUST}}$, and $\sqsubseteq'_{\text{TEST}}$.

The alternative characterization of the MUST preorder given in Proposition 4.4 is still valid; however, the definition of Q MUST A has to take into account that an automaton is in full control of its output actions and does not have any control on its input actions.

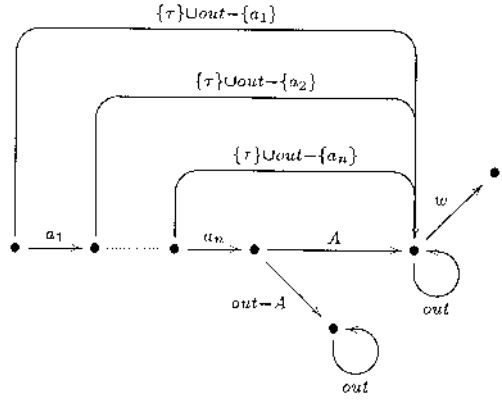
DEFINITION 7.1. Given an I/O automaton $\mathscr{A}$, a set of states $Q \subseteq \text{states}(\mathscr{A})$, and a set of external actions A, we say that Q MUST′ A iff either

1. $A \cap \text{in}(\mathscr{A}) \neq \varnothing$ or
2. for each $q \in Q$,
   (a) $\text{wenabled}(q) \cap \text{out}(\mathscr{A}) \subseteq A$, and
   (b) $\text{wenabled}(q) \cap A \neq \varnothing$.

Item 1 says that any I/O automaton must perform its input actions; Item 2 says that any I/O automaton internally decides which one of its possible output actions to perform.

PROPOSITION 7.2. *Given two finitely branching and strongly convergent I/O automata* $\mathscr{A}_1$ *and* $\mathscr{A}_2$ *with the same external signature,* $\mathscr{A}_1 \sqsubseteq'_{\text{MUST}} \mathscr{A}_2$ *iff* $\mathscr{A}_1 \sqsubseteq'_{\text{M}} \mathscr{A}_2$, *where* $\mathscr{A}_1 \sqsubseteq'_{\text{M}} \mathscr{A}_2$ *iff for each* $s \in \text{ext}(\mathscr{A}_1)^*$ *and each* $A \subseteq \text{ext}(\mathscr{A}_1)$, $\mathscr{A}_1$ *after* s MUST′ A *implies* $\mathscr{A}_2$ *after* s MUST′ A.

Proof. Let *out* denote $\text{out}(\mathscr{A}_1)$. Suppose that $\mathscr{A}_1 \sqsubseteq'_{\text{MUST}} \mathscr{A}_2$. Let $s = a_1 \cdots a_n$ be a sequence of external actions, let A be a set of external actions, and suppose that $\mathscr{A}_1$ after s MUST′ A. If $A \cap \text{in}(\mathscr{A}_1) \neq \emptyset$, then $\mathscr{A}_2$ after s MUST′ A trivially. Thus, suppose that $A \cap \text{in}(\mathscr{A}_1) = \emptyset$. Consider the experimenter $\mathscr{E}$,



CLAIM. *For each finitely branching and strongly convergent I/O automaton* $\mathscr{A}$ *with the same external signature as* $\mathscr{A}_1$, $\mathscr{A}$ after s MUST′ A *iff* $\mathscr{A}$ MUST′ $\mathscr{E}$.

Proof. Suppose that $\mathscr{A}$ after s MUST′ A, and suppose by contradiction that there exists an unsuccessful execution α of $\mathscr{A} \parallel \mathscr{E}$. Since both $\mathscr{A}$ and $\mathscr{E}$ are strongly convergent, α does not diverge. If α is of infinite length, then the trace of α is an extension of s followed by an action from $\text{out} - A$. This means that there is a state of $\mathscr{A}$ after s that enables some action from $\text{out} - A$, contradicting the fact that $\mathscr{A}$ after s MUST′ A. Thus, α is finite and leads to a state not enabling any action. This means that $\mathscr{E}$ ends either in the state reached after a_n, or in the state reached after the occurrence of w, or in the state reached after performing s followed by an action from $\text{out} - A$. In the first case there is a state of $\mathscr{A}$ after s that does not enable any output action, and thus not $\mathscr{A}$ after s MUST′ A; in the second case α would be successful; in the third case it would not be the case that $\mathscr{A}$ after s MUST′ A once again since an output action not in A is enabled after s. In all cases we have a contradiction.

Conversely, suppose that not $\mathscr{A}$ after s MUST′ A. We build an unsuccessful execution of $\mathscr{A} \parallel \mathscr{E}$. First of all, observe that s is a trace of $\mathscr{A}$. If there is a state q of $\mathscr{A}$ after s that does not enable any action from out, then the unsuccessful computation

spans s and then moves $\mathscr{A}$ to q; otherwise, there is at least one state q of $\mathscr{A}$ after s that enables some action of $\text{out} - A$. In this case the unsuccessful computation is any complete extension of the execution that spans s, moves $\mathscr{A}$ to q, and then performs an action from $\text{out} - A$.

Since $\mathscr{A}_1$ after s MUST′ A, then $\mathscr{A}_1$ MUST′ $\mathscr{E}$. Since $\mathscr{A}_1 \sqsubseteq_{\text{MUST}'} \mathscr{A}_2$, then $\mathscr{A}_2$ MUST′ $\mathscr{E}$. Thus, $\mathscr{A}_2$ after s MUST′ A. ▮

Conversely, suppose that $\mathscr{A}_1 \sqsubseteq'_{\text{M}} \mathscr{A}_2$. We show that for each experimenter $\mathscr{E}$, if not $\mathscr{A}_2$ MUST′ $\mathscr{E}$, then not $\mathscr{A}_1$ MUST′ $\mathscr{E}$ as well. Specifically, we show that if there exists an unsuccessful execution α of $\mathscr{A}_2 \parallel \mathscr{E}$, then there exists an unsuccessful execution α' of $\mathscr{A}_1 \parallel \mathscr{E}$. An unsuccessful execution α can exist for two reasons.

1. $\alpha = (q_0, e_0)\, a_1(q_1, e_1) \cdots a_n(q_n, e_n)$, (q_n, e_n) does not enable any action, and none of the e_i's enable w.

2. $\alpha = (q_0, e_0)\, a_1(q_1, e_1) \cdots a_n(q_n, e_n)\, a_{n+1}(q_{n+1}, e_{n+1}) \cdots$ where none of the e_i's enable w.

In each one of the cases above we build a nonsuccessful execution of $\mathscr{A}_1 \parallel \mathscr{E}$.

1. In this case there exists a finite trace $s = \text{etrace}(\alpha)$ such that $(q_0, e_0) \stackrel{s}{\Longrightarrow} (q_n, e_n)$. Since e_n enables all the output actions of $\mathscr{A}$, then q_n does not enable any action from *out*, which means that not q_0 after s MUST′ *out*. Thus, since $\mathscr{A}_1 \sqsubseteq'_{\text{M}} \mathscr{A}_2$, there is a start state q'_0 of $\mathscr{A}_1$ such that not q'_0 after s MUST′ *out*. This means that there is a finite execution from q'_0 whose trace is s and whose final state does not enable any output action of $\mathscr{A}_1$. Thus, by keeping the same execution of $\mathscr{E}$ from e_0, we obtain an unsuccessful execution from (q'_0, e_0).

2. In this case there is a trace $s = \text{etrace}(\alpha)$ spanned by α. Let s' be any prefix of s. Then, s' is a trace of q_0, which means that not q_0 after s' MUST′ $\varnothing$. Since $\mathscr{A}_1 \sqsubseteq'_{\text{M}} \mathscr{A}_2$, then not $\mathscr{A}_1$ after s' MUST′ $\varnothing$, which means that s' is a trace of $\mathscr{A}_1$. From Proposition 2.2, s is a trace of $\mathscr{A}_1$; i.e., there is a start state q'_0 of $\mathscr{A}_1$ such that s is a trace of q'_0. Thus, we can build an unsuccessful execution of $\mathscr{A}_1 \parallel \mathscr{E}$ from (q'_0, e_0) by keeping the same execution of $\mathscr{E}$ from e_0. ▮

The equivalence theorem is then straightforward.

THEOREM 7.3. (Quiescent versus MUST preorder). *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be finitely branching and strongly convergent I/O automata. Then*

$$\mathscr{A}_1 \sqsubseteq_Q \mathscr{A}_2 \text{ iff } \mathscr{A}_2 \sqsubseteq'_{\text{MUST}} \mathscr{A}_1.$$

Proof. We use the alternative characterization of the MUST preorder. Suppose that $\mathscr{A}_1 \sqsubseteq_Q \mathscr{A}_2$, and suppose by contradiction that there exist s and A such that $\mathscr{A}_2$ after s MUST′ A and not $\mathscr{A}_1$ after s MUST′ A. If $A = \varnothing$, then s is a trace of $\mathscr{A}_1$ and not a trace of $\mathscr{A}_2$. This leads to a contradiction since each finite trace of $\mathscr{A}_1$ is also a trace of $\mathscr{A}_2$. Thus $A \neq \varnothing$. Furthermore, since not $\mathscr{A}_1$ after s MUST′ A, s is a trace of $\mathscr{A}_1$ and there is a state $q \in \mathscr{A}_1$ after s such that not q MUST′ A. We distinguish two cases.

1. q is a quiescent state.

In this case s is a quiescent trace of $\mathscr{A}_1$, which means that s is a quiescent trace of $\mathscr{A}_2$ ($\mathscr{A}_1 \sqsubseteq_Q \mathscr{A}_2$). Thus, there is a quiescent state in $\mathscr{A}_2$ after s, which means that not $\mathscr{A}_2$ after s MUST′ A, a contradiction.

2. q enables an output action $b \notin A$.

In this case sb is a trace of $\mathscr{A}_1$, which means that sb is a trace of $\mathscr{A}_2$ ($\mathscr{A}_1 \sqsubseteq_Q \mathscr{A}_2$). Thus, there is a state q' of $\mathscr{A}_2$ after s that enables action b, which means that not $\mathscr{A}_2$ after s MUST′ A, again a contradiction.

Conversely, suppose that $\mathscr{A}_2 \sqsubseteq'_{\text{MUST}} \mathscr{A}_1$, and suppose by contradiction that $\mathscr{A}_1 \not\sqsubseteq_Q \mathscr{A}_2$. If there exists s which is a trace of $\mathscr{A}_1$ but not a trace of $\mathscr{A}_2$, then $\mathscr{A}_2$ after s MUST′ $\varnothing$ and not $\mathscr{A}_1$ after s MUST′ $\varnothing$, a contradiction. Thus, trace inclusion holds and there exists an s such that s is a quiescent trace of $\mathscr{A}_1$ but not a quiescent trace of $\mathscr{A}_2$. Then $\mathscr{A}_2$ after s MUST′ *out*, while not $\mathscr{A}_1$ after s MUST′ *out*, again a contradiction. ∎

8. CONCLUDING REMARKS

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. We have given a class of I/O automata for which the quiescent preorder is equivalent to the fair preorder. Secondly, we have shown the relationship between the theory of I/O automata and the theory 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.

The problem is still open for divergent systems since the theories we have studied are quite different in this respect. The MUST preorder considers a divergent state as chaotic, and therefore nothing is guaranteed to happen after a divergence has occurred. A purely divergent process 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 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 runs in parallel with a purely divergent component. As a consequence of the fair preorder, a purely divergent process is equivalent to a deadlocked process. In fact, any hypothetical external process which is waiting for some output does not distinguish between a pure divergence and a deadlock. The quiescent preorder does not represent any intuition when dealing with divergent processes. 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 Section 5.

It is difficult at this stage to say what is the best approach to divergent transition systems. The approach of testing leads to a neat theory while the approach of I/O automata leads to a notion that is closer to our intuition. Further research should focus on clarifying the issue of divergence. Is there a unifying theory that deals with divergences in the way of I/O automata and that at the same time enjoys most of the nice algebraic properties of the theory of testing? What is the correct interpretation of a divergence?

In summary, I/O automata and testing express the same notion of implementation whenever fairness is not essential and the I/O structure is either encoded or enforced on transition systems. Whenever a system is verified through testing, liveness has to be considered aside unless results like Theorem 5.7 hold; whenever a system is verified through I/O automata, liveness is implicit, but some I/O structure must be imposed. The choice of the formalism to use depends on the system to analyze: if some I/O structure can be imposed, then I/O automata appear to be more suitable; if no I/O structure can be imposed, then only testing can be used. However, if conditions like those stated in Section 5 hold, then testing appears to be preferable to I/O automata due to the availability of powerful algebraic tools. Two such cases are the delay insensitive circuits verified by Dill [Dil88] and Josephs [Jos92]. For the circuits analyzed by Josephs a comparison between process algebras and I/O automata is carried out in [LS95].

ACKNOWLEDGMENTS

The author thanks the anonymous referees for insightful comments.

Received August 3, 1995; final manuscript received June 3, 1997

REFERENCES

[AS85] Alpern, B., and Schneider, F. B. (1985), Defining liveness, *Inform. Process. Lett.* **21**(4), 181–185.

[BHR84] Brookes, S. D., Hoare C. A. R., and Roscoe, A. W., (1984), A theory of communicating sequential processes, *J. Assoc. Comput. Mach.* **31**(3), 560–599.

[BR84] Brookes, S. D., and Roscoe, A. W. An improved failures model for communicating processes, *in* "Seminar on Concurrency" (S. D. Brookes, A. W. Roscoe, and G. Winskel, Eds.), Lecture Notes in Computer Science, Vol. 197, pp. 281–305, Springer-Verlag, Berlin/New York.

[DeN85] De Nicola, R. (1985),"Testing Equivalence and Fully Abstract Models for Communicating Processes," Ph.D. Thesis, Department of Computer Science, University of Edinburgh.

[DeN87] De Nicola, R. (1987), Extensional equivalence for transition systems, *Acta Informat.* **24**, 211–237.

[DH84] De Nicola, R., and Hennessy, M. (1984), Testing equivalence for processes, *Theoret. Comput. Sci.* **34**, 83–133.

[Dil88] Dill, D. (1988), "Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits," ACM Distinguished Dissertations, MIT Press, Cambridge, MA.

[Hen88] Hennessy, M. (1988), "Algebraic Theory of Processes," MIT Press, Cambridge, MA.

[Hoa85] Hoare, C. A. R. (1985), "Communicating Sequential Processes," Prentice–Hall International, Englewood Cliffs, NJ.

[Jos92] Josephs, M. B. (1992), Receptive process theory, *Acta Informat.* **29**(1), 17–31.

[LS95] Lynch, N. A., and Segala, R. (1995), A comparison of simulation techniques and algebraic techniques for verifying concurrent systems, *J. Formal Aspects Comput. Sci.* **7**(3), 231–265.

[LT87] Lynch, N. A., and Tuttle, M. R. (1987), Hierarchical correctness proofs for distributed algorithms, *in* "Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing, Vancouver, 1987," pp. 137–151. A full version is available as MIT Technical Report MIT/LCS/TR-387.

[LV91] Lynch, N. A., and Vaandrager, F. W. (1991), Forward and backward simulations for timing-based systems, *in* "Proceedings of the REX Workshop Real-Time: Theory in Practice," (J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, Eds.), Lecture Notes in Computer Science, Vol. 600, pp. 397–446, Springer-Verlag, Berlin/New York.

[Par81] Park, D. M. R. (1981), Concurrency and automata on infinite sequences, *in* "5th GI Conference" (P. Deussen, Ed.), Lecture Notes in Computer Science, Vol. 104, pp. 167–183, Springer-Verlag, Berlin/New York.

[Seg92] Segala, R. (1992), "A Process Algebraic View of I/O Automata," Technical Report MIT/LCS/TR-557, MIT Laboratory for Computer Science, Cambridge, MA.

[Seg93] Segala, R. (1993), Quiescence, fairness, testing and the notion of implementation, *in* "Proceedings of CONCUR 93, Hildesheim" (E. Best, Ed.), Lecture Notes in Computer Science, Vol. 175, Springer-Verlag, Berlin/New York.

[SLL93] Søgaard-Andersen, J. F., Lampson, B., and Lynch, N. A. (1993), Correctness of at-most-once message delivery protocols, *in* "FORTE'93–Sixth International Conference on Formal Description Techniques."

[Vaa91] Vaandrager, F. W. (1991), On the relationship between process algebra and Input/Output automata, *in* "Proceedings of the Sixth Annual Symposium on Logic in Computer Science."

[WLL88] Welch, J. L., Lamport, L., and Lynch, N. A. (1988), "A Lattice-Structured Proof Technique Applied to a Minimum Spanning Tree Algorithm," Technical Report MIT/LCS/TM-361, MIT Laboratory for Computer Science.