Testability of a communicating system through an environment

K. DRIRA[1], P. AZEMA[1], B. SOULAS[2] and A.M. CHEMALI[2]

[1] LAAS du CNRS, 7 avenue du Colonel Roche, F-31077 Toulouse Cedex
[2] EDF-DER, Renardières, BP1 F-77250 Moret sur Loing

Abstract. Testing of a component embedded in a whole system is addressed. The component is not as easy to check as when taken in isolation. The notion of conformance, as introduced by E. Brinksma and G. Scollo, is extended to formalize testing through an environment that does not allow some non conforming implementations to be discarded. A method enabling embedded systems testability to be characterized is proposed. It is based on the refusal graph whose arcs are labeled by events and nodes by subsets of events. An approach is presented to identify erroneous implementations. In particular, the least erroneous implementations discarded by testing through environment are defined and computed.

1 Introduction

Formal Description Techniques, and particularly, process algebra, like CCS [MI 80], CSP [HO 85], ACP [BK 85] and LOTOS [LOT 88], are a well-known mathematical framework permitting communicating systems to be specified by composition and transformation of elementary behaviours. To verify the equivalence of two specifications of the same system, many algebraic relations have been proposed. The best known relations are presented and compared in [DN 87].

The use of formal specifications as a reference model to validate implementations, prompted the introduction of *testing equivalences* and especially *implementation relations* (validity or conformance) [AB 87, DH 84, LE 91]. This led, within the framework of LOTOS, to a formal definition of conformance and a theory for the derivation of corresponding tests [BSS 87, BR 88]. From a system's specification a tester is generated. Applying this tester to an implementation ends with a verdict ('fail', 'pass') that distinguishes nonconforming implementations from the others.

This paper deals with the analysis of the testability of a system embedded in an environment. Such analyses are important because specific constraints are imposed for testing a module embedded whithin a system: The different components (processes or modules), that cooperate to implement the global behaviour, are specified. The delivered system comprises all the components already integrated. It can be shown, by testing, that the global behaviour conforms to the expected one. But what conclusion may be drawn about the conformance of a given component to its specifcation ?

Suppose we are particularly (or only), interested in a component whose implementation and specification are I and M respectively. The problem, to be solved here, can be summarized as follows: what can be decided about the conformance of this component to its specification, M, assuming the global system ('I within E', E being the environment: i.e. the other components are interconnected) passes the tests[3] of the specification 'M within E'. M testability through its environment, E, is said to be good, if testing the whole system ('I within E') discards as many nonconforming M implementations as direct testing of M (i.e. without environment) would.

Here an approach is proposed for the analysis of a component testability when it can only be tested within the whole system. Intuitively this testing is less powerful than the one that has direct access to the isolated component. In practice, this is expressed by non-detection of some erroneous implementations that would have been discarded by directly testing this system. This will be called *testability degradation.*

Testability analysis is easily understood in the idealistic case :
conformance is a total ordering, $\geq$, that places the specification M and its implementations[4] at the same axis: $\{I < M\} \cup \{M\} \cup \{I > M\}$;
testing an implementation I can lead to either of the conclusions: $I > M$ (I conforming) or $I < M$ (I nonconforming).
In this idealistic case, testability (degradation) analysis consists of searching the new reference model, $M'(\leq M)$, such that only '$I < M'$' can be checked when testing I through the environment. M' is called the *limit* of testability degradation (or shortly the limit of testability). Testability degradation is expressed by the widening of the margin of nonconforming implementations not detected through the environment: $\{I, M' < I < M\}$

This paper encompasses this introduction and three sections.

Section 2 describes conformance and other relations as defined in [BR 88] and characterizes erroneous implementations that conformance testing can discard. Testing through an environment is then presented. Testability analysis and related problems are detailed. Finally, the limits of testability are characterized.

Section 3 introduces a behaviour representation structure, referred to as "*Refusal graph*", providing composition and restriction operators together with testing relations ($conf$, red, $\approx_g$) consistent with those initially defined in [BR 88].

Section 4 provides a method for testability analysis based on the refusal graph. An ordered characterization of erroneous implementations allows identification of the least erroneous implementations that testing through an environment can discard.

---

[3] or part of these tests that aim at activating this component

[4] rather than physical implementations, we consider models of these implementations described in the same formalism as the specification. This allows us to compare implementations as well as implementation to a given specification. This identification is discussed in [LE2 91].

2 Conformance through environment

2.1 Preliminary Definitions

This section recalls conformance related definitions first introduced by Brinksma, Scollo and Steenbergen in [BSS 87]. The equivalent definitions used are also employed by Leduc in [LE 90]. This section also presents new relations dealing with non conformance.

Labeled Transition System

A finite Labeled Transition System (LTS) is a quadruple: $\mathcal{S} = (S, \Sigma, \Delta, s_0)$ where:

- S is a finite set of states, and s_0, $s_0 \in S$, is the initial state of $\mathcal{S}$.
- Σ is a finite set of visible actions, or labels
- $\Delta \subseteq S \times (\Sigma \cup \{\tau\}) \times S$: the transitions set, $\tau \notin \Sigma$ is called internal or invisible action. An element $(x, \mu, y) \in \Delta$ is denoted: $x \xrightarrow{\mu} y$

Another transition relation, $\{\stackrel{\mu}{\Rightarrow}\}_{\mu \in \Sigma \cup \{\epsilon\}}$ is defined in a standard way by:

- $s \stackrel{\epsilon}{\Rightarrow} s' : s = s'$ or $s \xrightarrow{\tau} s_1 \xrightarrow{\tau} \cdots \xrightarrow{\tau} s_n \xrightarrow{\tau} s'$
- $s \stackrel{\mu}{\Rightarrow} s' : s \stackrel{\epsilon}{\Rightarrow} s_1 \xrightarrow{\mu} s_1 \stackrel{\epsilon}{\Rightarrow} s'$

The following notations are used:

- $s \stackrel{\mu}{\Rightarrow}$ means $\exists s'\ s \stackrel{\mu}{\Rightarrow} s'$. $s \stackrel{\mu}{\not\Rightarrow}$ means $\neg(s \stackrel{\mu}{\Rightarrow})$.
- $out(s) = \{\mu \in \Sigma \mid s \stackrel{\mu}{\Rightarrow}\}$ denotes the set of visible actions that can be performed by the system at the state s

This relation is extended to sequences (i.e. words or strings over Σ: $\sigma \in \Sigma^*$) by:

- if σ is the sequence $\mu_1 \cdots \mu_n$ write $s \stackrel{\sigma}{\Rightarrow} s'$ when $s \stackrel{\mu_1}{\Rightarrow} s_1 \stackrel{\mu_2}{\Rightarrow} \cdots \stackrel{\mu_{n-1}}{\Rightarrow} s_n \stackrel{\mu_n}{\Rightarrow} s'$

The empty sequence is denoted ϵ. As in the case of a state output, "traces of a state" refer to the set of all sequences of visible actions, $\sigma \in \Sigma^*$, that can be performed from this state: $Tr(s) = \{\sigma \in \Sigma^* \mid s \stackrel{\sigma}{\Rightarrow}\}$. The traces of LTS are those of its initial state.

Conforming implementations

In the sequel, A is a set of actions: $A \subseteq \Sigma$; σ is a sequence of actions: $\sigma \in \Sigma^*$. P (resp. P', I, M ...) denotes a behaviour expression associated with a finite Labeled
Transition System whose initial state is P (resp. P', I, M ...).

- $P\ ref\ A$ when $\forall a \in A\ P \stackrel{a}{\not\Rightarrow}$. P has no derivate by any action a among A. Then it is said that P refuses A. Note that if P refuses A then P refuses all subsets of A (i.e. $B \subseteq A$ implies $P\ ref\ B$)
- $P\ after\ \sigma = \{P' : P \stackrel{\sigma}{\Rightarrow} P'\}$: set of all derivates of P via sequence σ.

If $\sigma \notin Tr(P)$ then P has no derivate via σ and then $P\ after\ \sigma = \emptyset$.

- $(P\ after\ \sigma)\ ref\ A$ when $(\exists P' \in P\ after\ \sigma\ , P'\ ref\ A)$

at least one of the derivates of P refuses A.

When $\sigma \notin Tr(P)$, there exists no element in $(P\ after\ \sigma)$ and then $(P\ after\ \sigma)\ ref\ A$ has 'false' as logical value. This substanciates the equivalence of the following two definitions of the conformance relation.

- **conformance:** implementation I is said to be conforming to specification M when I deadlocks less often than M when placed in an environment whose traces

are limited to those of M. Formally $I \underline{conf}\ M \equiv_{df} \forall \sigma \in Tr(M) \cap Tr(I), \forall A \subseteq \Sigma$: if $(I\ after\ \sigma)\ ref\ A$, then $(M\ after\ \sigma)\ ref\ A$. Or equivalently:
$\forall \sigma \in Tr(M), \forall A \subseteq \Sigma$: if $(I\ after\ \sigma)\ ref\ A$, then $(M\ after\ \sigma)\ ref\ A$
• **reduction:** A reduction is a conforming implementation with less traces than the specification. Formally: $I \underline{red}\ M \equiv_{df} (I \underline{conf}\ M) \wedge Tr(I) \subseteq Tr(M)$
• **extension:**An extension is a conforming implementation that has more traces than the specification. Formally: $I \underline{ext}\ M \equiv_{df} (I \underline{conf}\ M) \wedge Tr(I) \supseteq Tr(M)$
• **improvement:** An improvement is a conforming implementation possessing the same traces as the specification. Formally: $I \geq M \equiv_{df} Tr(I) = Tr(M)$ and $I \underline{conf}\ M$. We also say that I is more deterministic than M. The symbol $\leq$ is used to denote $\geq^{-1}$: $I \leq M \stackrel{def}{\Longleftrightarrow} M \geq I$. $I \leq M$ means: I is less deterministic than M.
Note that $(I \geq M) \stackrel{def}{\Longleftrightarrow} (I \underline{red}\ M) \wedge (I \underline{ext}\ M)$.
• **testing equivalent** It is these implementations which are as deterministic as the specification. Formally: $I \underline{te}\ M$ iff $(I \underline{red}\ M) \wedge (M \underline{red}\ I)$.
Non conforming implementations

Definition1 Distortion. Every nonconforming implementation with the same traces as the specification. I is said to be a distortion of M and is denoted $I\ dis\ M$ when $Tr(I) = Tr(M)$ and $I \underline{\neg conf}\ M$.
Definition2 Degradation. Every implementation (strictly) less deterministic than the specification. I is said to be a degradation of M and is denoted $I < M$ when $I \leq M$ and $\neg(M \underline{te}\ I)$. Equivalently: A degradation I of M is a distortion such that $M \underline{conf}\ I$.

Summary of the different relations
Let $\mathcal{I}$ denote the implementation set of a specification M. A conformance test splits $\mathcal{I}$ into two subsets:
• $\mathcal{I}^{\oplus}$ is the set of conforming implementations: $\{I \in \mathcal{I} : I\ conf\ M\} = Conf(M)$ (upper zone of Fig. 1). These implementations pass the conformance test.
• $\mathcal{I}^{\ominus}$ is the set of nonconforming implementations: $\{I \in \mathcal{I} : I\ \neg conf\ M\} = \neg Conf(M)$. (lower zone of Fig. 1). These implementations fail the conformance test.

On the other hand, $\mathcal{I}$ can be partitioned into the set of implementations whose traces are comparable to those of M (inside of the circles of Fig. 1), and its complementary: $\mathcal{I} = (A \cup B) \cup (\mathcal{I} \setminus (A \cup B))$ where
$A = \{I \in \mathcal{I} : Tr(I) \subseteq Tr(M)\}$ is the set of trace reductions of M. This set is denoted Red_tr(M) and is the left circle of Fig. 1.
$B = \{I \in \mathcal{I} : Tr(I) \supseteq Tr(M)\}$ is the set of trace extensions of M. This set is denoted Ext_tr(M) and is the right circle of Fig. 1.

Intersection of these different sets is summarized in the following tables and illustrated in Fig. 1.

2.2 Testing through an environment

Here *the test of a system through an environment* is formalized using Lotos operators as a basis for behaviour composition. The system is a finite Lotos

conformance zone: $\oplus$

designation	Symbol	Notation
Conforming	$\mathcal{I}^{\oplus}$	Conf(M)
Reductions	$A \cap \mathcal{I}^{\oplus}$	Red(M)
Extensions	$B \cap \mathcal{I}^{\oplus}$	Ext(M)
Improvements	$A \cap B \cap \mathcal{I}^{\oplus}$	Imv(M)

nonconformance zone : $\ominus$

designation	Symbol	Notation
Nonconforming	$\mathcal{I}^{\ominus}$	$\neg$Conf(M)
Distortions	$A \cap B \cap \mathcal{I}^{\ominus}$	Dis(M)
Degradations	$\subseteq (A \cap B \cap \mathcal{I}^{\ominus})$	Deg(M)

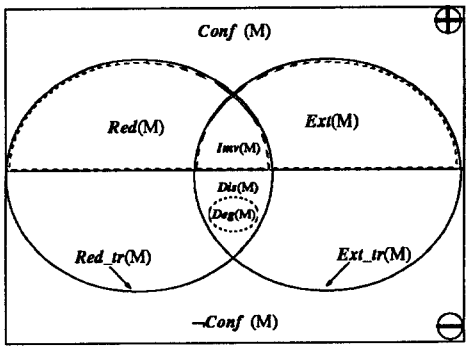


Fig. 1. Zones of the implementations domain

process (which can be represented by a finite LTS). The environment is the particular context *hide* Γ *in* $(\bullet|[\Gamma]|E)$, where E is finite Lotos process.

We suppose that the global implementation results from the composition of an implementation of M (i.e. another process I) with an environment identical[5] to E: Implementation under test is *hide* Γ *in* $(I|[\Gamma]|E)$

The conforming/nonconforming (pass/fail) verdict is considered as a verdict directly concerning M. In other words, a failure only involves the implementation of component M (which is then called component under test).

Definition 3 Conformance through an environment. An implementation I conforms to a specification M through E if *hide* Γ *in* $(I|[\Gamma]|E)$ conforms to *hide* Γ *in* $(M|[\Gamma]|E)$. This will be noted: $I\ conf_E\ M$

2.3 Testability Analysis

Analysis of M testability through environment E is tantamount to introducing, in the implementation domain zones, a new partition of $\mathcal{I}$ given by the conformance verdict when testing implementations through the environment. This leads us to compare M testability after embedding in environment E, and testability of isolated M.

Adopting the intuitive idea that testing through an environment can only degrade testability [6] leads to the following paradox: testing through an environment may evaluate as **nonconforming** some conforming implementations.

Indeed extensions (right upper half of the circle in Fig. 1) may become nonconforming when testing through an environment. This can be explained by considering the objective of conformance testing: conformance testing aims at

[5] This hypothesis can be relaxed without affecting the results : 'identical' can be replaced by 'observationally equivalent' or also another equivalence stronger than testing equivalence and which is a congruence w.r.t. hiding and composition operators (*hide* and ||).

[6] i.e. nonconforming implementations are erroneously evaluated as conforming

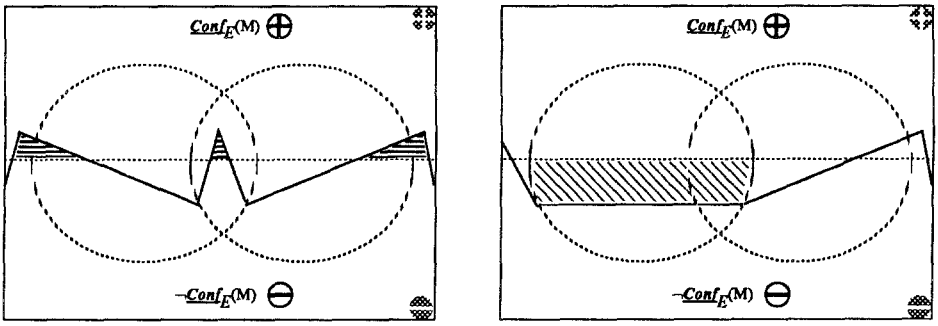
verifying the correct functioning of the implementation with respect to the specified behaviour: A conformance tester accepts implementations that extend the specification traces as soon as they conform to the specified behaviour. Without making any assumptions on the environment behaviour, the latter may have a superset of traces (relativeto synchronization actions) of the specification. The environment therefore participates in *robustness testing*[7] [BR 88]. And testing through this environment rejects conforming implementations that are not reductions of the specification.

This paradox vanishes when the synchronization traces of the environment are restricted to those of the specification.

On the other hand, without assuming that hiding synchronization actions creates no divergence, reductions as well as improvement might be evaluated as nonconforming when checked through an environment.

In Figure 2, the E-conformance boundaries (continuous line) cross the conformance boundaries (horizontal dashed line), which result from a test of the isolated component.

The general case, depicted in part (a) of Fig. 2, illustrates the incomparability of these two conformance tests, because some conforming implementations will be regarded as nonconforming when tested through the environment. These are nonconforming implementations depicted by the horizontally dashed regions in part (a) of the figure.



(a) general case (b) when hiding does not create divergence

Fig. 2. Conformance through environment

Tracing regularity, in the trace inclusion zone, of part (b) of Fig. 2 shows that, in the absence of divergence, conforming implementations with at the most the same traces as the specification (reductions) remain conforming when tested through the environment. This expresses the so-called conformance preservation. It can be formalized as follows

[7] i.e. testing implementation against an environment that behaves incorrectly. This kind of test guarantees that the implementation does not possess unspecified traces.

Proposition 4. *conformance preservation*
In the absence of divergence:
• *If I red M then I* red_E *M (and then I* $conf_E$ *M) [LE 87]*
where $I\ red_E\ M \equiv_{df}$ (*hide* Γ *in* $I|[\Gamma]|E$) *red* (*hide* Γ *in* $M|[\Gamma]|E$),
Reductions are conforming implementations that remain conforming when tested through the environment.

2.4 Limits of testing through an environment

No proposition can be established about the existence of nonconformance detection limits by testing through an environment. This strongly depends on the specification of the component under test and on its environment. Nevertheless, they may have a meaning when they exist.

A limit is an erroneous implementation which may be detected when testing through the environment and such that: only 'more erroneous' implementations will be detected.

The 'more erroneous' relation will be expressed by the pre-order $\leq$; and I_1 is said to be more erroneous than I_2 when $I_1 \leq I_2$.

It can now be stated that the limits are the least erroneous implementations that testing through environment can discard:

An erroneous implementation, I, is a limit of nonconformance detection if
(i) nonconformance of I is detected (through the environment).
(ii) nonconformance of (strictly) less erroneous implementations (i.e. $I' > I$) is not detected (through the environment).

Assuming that hiding creates no divergence, the following proposition shows that implementations that are more erroneous than a detected erroneous implementation (and particularly a limit) will also be detected.

Proposition 5. *non conformance detection*
Given two implementations, I and I' , of specification M
if $(I\ \neg conf_E\ M)$ *then* $(I' \leq I) \Rightarrow (I'\ \neg conf_E\ M)$
if I is detected by testing through E, then all implementations, which are more erroneous than I, will also be detected.

3 Refusal Graphs for computing limits of testability

Labeled transition systems are the initial semantics of LOTOS. Another semantic model (Rooted Failure Tree with divergence) was defined in [LE 90] and proposed to interpret LOTOS specifications. This model was useful for enriching the basic model (Failure tree) of the theory for tests derivation of [BR 88] with *composition* and *restriction* (or *hiding*) operators. The model referred to as *Refusal Graph* presented here, makes the approach for the testability analysis (presented in the next section) operational.

In this section, the refusal graph structure is defined along with the composition and restriction operators. Finally, conformance (*conf*), reduction (*red*)

and equivalence ($\approx_g$) relations are defined directly on the refusal graph structure. These relations are (bi)simulation-like defined [PA 81, MI 80] and therefore easier to check than the initial definitions on transition systems.

3.1 Refusal Graph

The Refusal Graph is a structure specifying the failures of a communicating system [HO 85] (a failure is a couple made up of a sequence of actions in which the system may engage, and a set of actions it can refuse after this sequence).

Definition 6 Refusal Graph. A refusal graph, denoted RG, is a bilabeled graph represented by a 5-tuple $(S, \Sigma, \Delta, Ref, s_0)$ where:
- S is a finite set of states, $s_0 \in S$ is an element of S called initial state.
- $\Sigma \subseteq L$ is a finite set of actions (edge labels), also called the alphabet of RG,
- $\Delta \subseteq (S \times \Sigma \times S)$ is a set of transitions. An element $(s, a, s') \in \Sigma$ is denoted: $s \stackrel{a}{\Rightarrow} s'$. Transitions described in Δ must verify the following determinism property:

$$\forall s \in S,\ \forall a \in \Sigma;\ \exists \text{ at the most one } s' \in S \text{ such that } s \stackrel{a}{\Rightarrow} s'.$$

- $Ref : S \longrightarrow \mathcal{P}(\mathcal{P}(L))$ is an application which defines for each state, the sets of actions that may be refused after the sequence leading to this state.

To avoid redundancy, refusal sets must be minimal w.r.t. set inclusion: $\forall s \in S, \forall X, Y \in Ref(s) : (Y \subseteq X) \Rightarrow (X = Y)$. Or equivalently $\nexists X, Y \in Ref(s) : (X \neq Y) \wedge (Y \subseteq X)$.
i.e. no subset of an element of $Ref(s)$ is in $Ref(s)$. In other words, all elements of a refusal set are pairwise incomparable (w.r.t set inclusion $\subseteq$).

And to avoid describing imaginary systems, one of the following hypothesis is imposed on the refusal graph structure:
h1. $\forall X \in Ref(s)$, $X \subseteq out(s)$. Only refused parts of the output[8] set are considered. Or
h2. $\forall X \in Ref(s)$, $X \cup (L \setminus out(s)) \in Ref(s)$. Refused parts are saturated with respect to output complement. This second hypothesis is used in [BR 88, LE 90]

The changeover from a refusal set, R, built according to h1, to its representation according to h2 is possible by the completion transformation $complete(Ref(s)) = \{X \cup (L \setminus out(s)), X \in Ref(s)\}$. (The reverse changeover corresponding to the reverse transformation $uncomplete(R) = \{X \cap out(s), X \in Ref(s)\}$.)

Let $G_1 = (S_1, \Sigma_1, \Delta_1, Ref_1, s_0^1)$ and $G_2 = (S_2, \Sigma_2, \Delta_2, Ref_2, s_0^2)$ be two refusal graphs such that Ref_1 and Ref_2 are defined with respect to the completion hypothesis (h2.) with alphabet $L = \Sigma_1 \cup \Sigma_2$ as superset of Σ_1 and Σ_2. Let $\Gamma (\subseteq \Sigma_1 \cup \Sigma_2)$ be a set of actions such that $\Sigma_1 \cap \Sigma_2 \subseteq \Gamma$. [9]

Definition 7 refusal graph composition. The composition of G_1 and G_2 is the refusal graph $G = (S, \Sigma, \Delta, Ref, s_0)$ defined by:
- Set of states $S \subseteq S_1 \times S_2$ is such that $(s_0^1, s_0^2) \in S$, and every couple of elements

[8] $out(s) = \{a \in \Sigma, \exists s' \in S : s \stackrel{a}{\Rightarrow} s'\}$ is called output of state s.

[9] This hypothesis allows the definition of the composition operator to be simplified compared to the RFT model of [LE 90]. It is equivalent to assuming that only synchronization gates may have the same name in the system and environment specifications which is not restrictive in the framework of testing through an environment.

of S_1 and S_2 which may follow an element of S by one of the transition rules (i), (ii), (iii), given below, is an element of S,

- $\Sigma = \Sigma_1 \cup \Sigma_2$ is the set of actions,
- Δ is defined by: $\forall \alpha \in \Gamma, \forall a \in (\Sigma_1 \setminus \Gamma), \forall b \in (\Sigma_2 \setminus \Gamma)$:

$$\text{(i)}\ \frac{s_1 \stackrel{\alpha}{\Rightarrow} s_1',\ s_2 \stackrel{\alpha}{\Rightarrow} s_2'}{(s_1,s_2) \stackrel{\alpha}{\Rightarrow} (s_1',s_2')}\ \text{(ii)}\ \frac{s_1 \stackrel{a}{\Rightarrow} s_1'}{(s_1,s_2) \stackrel{a}{\Rightarrow} (s_1',s_2)}\ ,\ \text{(iii)}\ \frac{s_2 \stackrel{b}{\Rightarrow} s_2'}{(s_1,s_2) \stackrel{b}{\Rightarrow} (s_1,s_2')}$$

- $(\forall (r,s) \in S)\ Ref((r,s)) = \{((X_1 \cup X_2) \cap \Gamma) \cup (X_1 \cap X_2), X_1 \in Ref_1(r), X_2 \in Ref_2(s)\}$
- $s_0 = (s_0^1, s_0^2)$ is the initial state.

To define the restriction operator on refusal graphs, it is assumed that the restriction creates no divergence. In the opposite case, only upper and lower bounds may be obtained as in [LE 90].

Given a refusal graph $G = (S, \Sigma, \Delta, Ref, s_0)$ and a set of actions $\Gamma \subseteq \Sigma$,

Definition 8 refusal graph restriction. The restriction of Γ in G, is the refusal graph $G' = (S', \Sigma', \Delta', Ref', s_0')$ denoted $G \setminus \Gamma$ and defined by:

- $s_0' = \{s_i : s_0 \stackrel{\gamma}{\Rightarrow} s_i, \gamma \in \Gamma^*\}$ is the initial state. It is the set of all the states reached from s_0 by a sequence of actions in the restriction set.
- $S' \subseteq \mathcal{P}(S)$: whose elements are defined by the series $(s_n')_{n \geq 0} : s_n' = \delta(s_{n-1}')$ where:

$\delta(s') = \bigcup_{s \in s'} \delta(s)$, $\delta(s) = \bigcup_{a \in (\Sigma \setminus \Gamma)} \delta_a(s)$, $\delta_a(s) = \{t \in S, \exists \gamma_1, \gamma_2 \in \Gamma^* : s \stackrel{\gamma_1 a \gamma_2}{\Longrightarrow} t\}$

states of G' are sets of G states that may be reached from s_0' by a sequence where actions in Γ are considered as internal.

- $\Sigma' = \Sigma \setminus \Gamma$: the alphabet of G' is restricted to $\Sigma \setminus \Gamma$
- Δ' is such that: $\forall a \in (\Sigma \setminus \Gamma)\ s_1' \stackrel{a}{\Rightarrow} s_2'$ iff $\exists s_1 \in s_1', \exists s_2 \in s_2', \exists \gamma_1, \gamma_2 \in \Gamma^*$: $s_1 \stackrel{\gamma_1 a \gamma_2}{\Longrightarrow} s_2$
- $Ref'(s') = \{X \setminus \Gamma, X \in Ref(s)\ et\ \Gamma \subset X, s \in s'\} \setminus \{Y \in Ref'(s') \exists X \in Ref'(s') : Y \subset X\}$

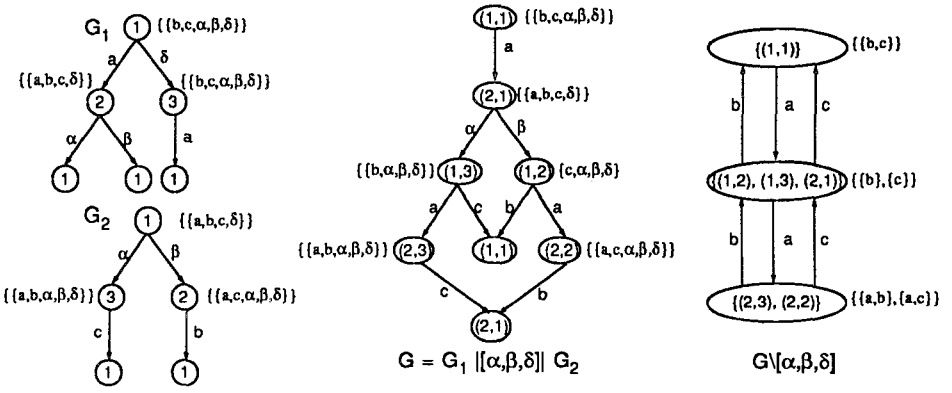


Fig. 3. composition and restriction

Notations: If L is a finite set, then
- $\mathcal{P}(L)$ denotes the power set of L, i.e. the set of subsets of L
- for $A \in \mathcal{P}(\mathcal{P}(L))$ we note $Min(A) = A \setminus \{X,\ \exists Y \in A\ :\ X \subseteq Y, et X \neq Y\}$.
- for $R \subseteq \mathcal{P}(\mathcal{P}(L))$ we note $Min(R) = \{Min(A), A \in R\}$.
- the minimal refusal sets on alphabet $L' \subseteq L$ are elements of $Min(\mathcal{P}(\mathcal{P}(L')))$
- for every minimal refusal set A, $X \in A \equiv_{df} \exists Y \in A\ :\ X \subseteq Y$.
- $A \Subset B \equiv_{df} \forall X,\ (X \in A) \Rightarrow (X \in B)$
- $A \Subset_s B \equiv_{df} (A \Subset B) \wedge (A \neq B)$
- Application Ref_I defines the refusal sets for the states of refusal graph I.
- Application Ref_E^I defines the refusal sets for the states of refusal graph $(I|[\Gamma]|E) \setminus \Gamma$.

3.2 Binary relations over refusal graphs

Consider two refusal graphs defined over the same alphabet[10] L: $I = (S_I, L, \Delta_I, Ref_I, I)$ and $M = (S_M, L, \Delta_M, Ref_M, M)$.

Definition 9 .
- conformance $I\ conf\ M \equiv_{df}$
  - (i) $Ref_I(I) \Subset Ref_M(M)$
  - (ii) $\forall a \in out(I) \cap out(M)\ :\ if\ I \stackrel{a}{\Rightarrow} I'\ then\ M \stackrel{a}{\Rightarrow} M'\ and I'\ conf\ M'$.
- reduction $I\ red\ M \equiv_{df}$
  - (i) $Ref_I(I) \Subset Ref_M(M)$
  - (ii) $\forall a \in L\ :\ if\ I \stackrel{a}{\Rightarrow} I'\ then\ M \stackrel{a}{\Rightarrow} M'\ and I'\ red\ M'$.
- testing equivalence $I \approx_g M \equiv_{df} I\ red\ M$ et $M\ red\ I$

The refusal graph corresponding to a transition system is obtained by making deterministic the transition system considered as an automaton whose every state is terminal. This provides a mapping, *newState*, which associates each state of the refusal graph with a set of states of the transition system. The refusal sets of a state g are given by: $Ref(g) = Min(\{out(g) \setminus out(s), s \in newState(g)\})$.

Proposition 10 is related to the compatibility of the composition and restriction operators of the refusal graphs with those defined on transition systems. The latter are similar to basic LOTOS operators: a transition system is viewed as a set of processes, S, executing actions in $\Sigma \cup \{i = \tau\}$ according to the transition rules defined by Δ. The initial behaviour being the initial state of the transition system, i.e. s_0.

For any transition systems, S_1 and S_2, we have:

Proposition 10. *operator compatibility*
- *composition:* $rg(S_1|[\Gamma]|S_2) \approx_g rg(S_1)|[\Gamma]|rg(S_2)$
- *restriction:* $rg(hide\ \Gamma\ in\ S_1) \approx_g rg(S_1) \setminus \Gamma$

Relations $\geq$, $\leq$, and *dis* are also defined over refusal graphs, by replacing the definition of *conf* of transition systems by its dual relation of refusal graphs. For these two types of relations the same symbols are kept.

[10] in practice L is the union of the alphabets of the two refusal graphs

4 Testability through an environment

This section is dedicated to the use of the refusal graphs structure for computing the limits of nonconformance detection when testing through an environment.

4.1 Ordered erroneous implementations

This section introduces the transformations employed to simulate the three types of implementations presented earlier (paragraph 2.1), namely, '*improvements*', '*degradations*', and '*distortions*'[11].

With respect to testability analysis, we want particularly to order the erroneous implementations (degradations and distortions) such that they can then be classified in agreement with the degradation relation ($<$). Thus an approach, based on an ordered analysis of erroneous implementations, can be developed to identify the limits.

As illustrated in Fig. 4, testability analysis consists of tracing limits (painted ovals) of nondetection of degradations (and distortions) over the set of ordered erroneous implementations.

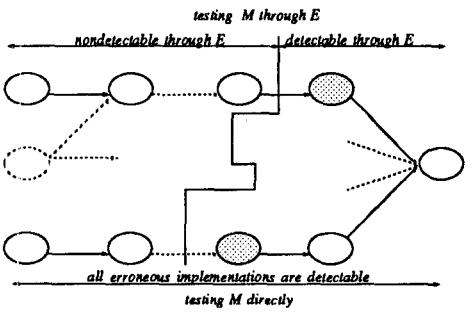


Fig. 4. testability Degradation

For every state, i, of the specification refusal graph, $M = (S, \Sigma, \Delta, Ref, s_0)$, a mapping, Ψ_i is defined associating with each refusal set, $A \in Min(\mathcal{P}(\mathcal{P}(out(i))))$, an erroneous implementation (degradation or distorsion), or a conforming implementation (improvement). Let $\mathcal{G}$ denote the set of implementations having the same refusal graph as M except for the refusal sets (i.e. the same state space, the same edge labels, but not the same state labels).

Let $(\mathcal{T}, \mathbb{C})$ denote the lattice $(Min(\mathcal{P}(\mathcal{P}(out(i)))), \mathbb{C})$. [12] And $\forall A \in \mathcal{T}$, $\forall \mathcal{E} \subseteq \mathcal{T}$

$\underline{A} = \{B \in \mathcal{T}, B \mathbb{C}_s A\}$, $\overline{A} = \{B \in \mathcal{T}, A \mathbb{C}_s B\}$ et $inf(\mathcal{E}) = \{A \in \mathcal{E}, \nexists B \in \mathcal{E}, B \mathbb{C}_s A\}$.

[11] For the latter, we consider only those which may not be captured as degradations.

[12] In practice it is possible to use the same ordered set $Min(\mathcal{P}(\mathcal{P}(\Sigma))), \mathbb{C}$, for all the states.

$\Psi_i : \mathcal{T} \longrightarrow \mathcal{G}$

$A \longrightarrow \Psi_i(A) = (S, \Sigma, \Delta, Ref', s_0)$ such that:

$$\begin{cases} \forall j \in S, j \neq i, Ref'(j) = Ref(j) \\ Ref'(i) = A \end{cases}$$

Thus a lattice $(\Psi_i(\mathcal{T}), \geq)$ isomorphic to $(\mathcal{T}, \mathbb{C})$ is built. In particular we verify that: $B \mathbb{C} A$ iff $\Psi_i(B) \geq \Psi_i(A)$, meaning: $\Psi_i(B)$ is more deterministic than $\Psi_i(A)$. When $B \mathbb{C}_s A$ then $\Psi_i(B) > \Psi_i(A)$ and $\Psi_i(A)$ is a degradation of $\Psi_i(B)$.

- $I \in \Psi_i(\underline{Ref(i)})$ implies $I > M$. *These are M improvements.*
- $I \in \Psi_i(\overline{Ref(i)})$ implies $I < M$. *These are M degradations.*
- $I \in \Psi_i(\mathcal{T} \setminus (\overline{Ref(i)} \cup \underline{Ref(i)} \cup \{Ref(i)\}))$ implies I *dis* M. *These are M distortions.*

By way of example, consider the state 2 of the refusal graph M represented by G_1 in the Fig. 3. $out(2) = \{\alpha, \beta\}$ et $Ref(2) = \{\{\}\}$

According to the lattice depicted in Fig. 5, M has four degradations [13]: $M_{1,1}$ is the one having $\{\{\alpha\}\}$ as refusal set at the state 2. $M_{2,1}$ has $\{\{\beta\}\}$ as refusal set. $M_{2,2} = M_{1,2}$ have $\{\{\alpha\}, \{\beta\}\}$ as refusal set. And finally $M_{2,3} = M_{1,3} = M_{min}$ have $\{\{\alpha, \beta\}\}$ (i.e. $\{out(2)\}$) as refusal set.

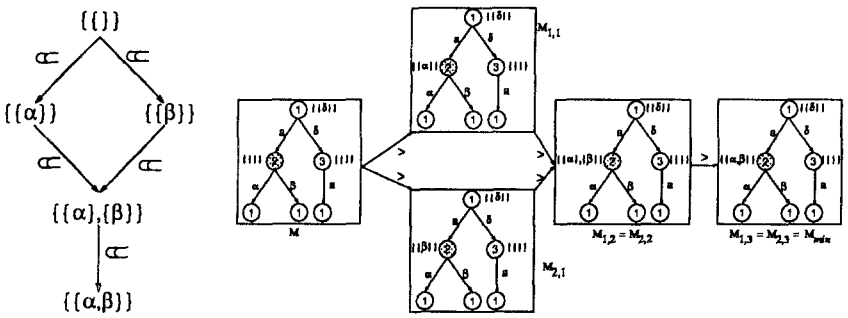


Fig. 5. Ordered refusal sets and induced Ordered degradations

4.2 Use of refusal graphs for limits location

Refusal graphs allow erroneous implementation characterization problem to be reduced to elementary operations on the refusal set structure. Just as in the case of testability degradation limit identification. Indeed, the definition on refusal graphs of the composition and restriction operators allows us to decide if an error is detectable through the environment by comparing (two) refusal sets w.r.t. order relation $\mathbb{C}$.

[13] to simplify we present refusal sets according to the first hypothesis h1 of refusal graphs definition

Let $Deg(i) = \overline{Ref_M(i)}$, $Imv(i) = \underline{Ref_M(i)}$, $Dis(i) = \mathcal{T} \setminus (Deg(i) \cup Imv(i) \cup \{Ref_M(i)\})$, and $Err(i) = Deg(i) \cup \overline{Dis(i)}$

Simulating an error related to state i, consists in replacing the refusal set at this state, $Ref_M(i)$, by one, say A, of the lower bounds of $Err(i)$.

To verify whether this error is a limit (of conformance testing through environment E) the following steps are executed:

- choose a state s of $(M|[\Gamma]|E) \setminus \Gamma$ which could have been "disturbed" by this error, and which includes a couple of the form $(i, *)$, ($*$ matching any E state).

– compute the new refusal set of s by replacing, $Ref_M(i)$ by the (completed) refusal set $complete(A)$.
– test whether the obtained set is included (in the sense of $\mathbb{C}$) in the initial refusal set. If it is not (the corresponding erroneous implementation is a limit), stop the exploration (of the matching states, s) and add this error to the set of limits sought. Remove A and all its successors (i.e. all sets B such that $A \mathbb{C} B$) from set of errors to be explored. The new set to explore is then $Err(i) \setminus (\overline{A} \cup \{A\})$.
In the opposite case (the erroneous implementation corresponding to A cannot be detected in any state s), only A is removed from the set of errors to be explored.

Formalization: For every state, i, of the refusal graph, let (G, V) denote the oriented graph structure that represents the lattice $(Min(\mathcal{P}(\mathcal{P}(out(i)))), \mathbb{C})$.
• G is a set of nodes representing the lattice elements.
These refusal sets are candidate for replacing the refusal set $Ref(i)$ that will be referred to as e_i.
• V is the set of edges representing the (strict) inclusion of two refusal sets.
• (G, V^*) is the transitive closure of (G, V).
For every couple of (distinct) nodes (e, e'),
• $(e, e') \in V$ means: $e \mathbb{C} e'$ and $\nexists e''$ such that $e \mathbb{C}_s e'' \mathbb{C}_s e'$
• $(e, e') \in V^*$ means: $\exists e_1, .., e_k : e \mathbb{C}_s e_1 \mathbb{C}_s \cdots \mathbb{C}_s e_k \mathbb{C}_s e'$

The search for the limits of degradations and distortions that cannot be detected when testing through the environment can be formalized by the calculus of the set **Limits** returned by the following algorithm:

Distortions := $G \setminus \{e_i\} \setminus \{e : (e, e_i) \in V^*$,or, $(e_i, e) \in V^*\}$
Degradations := $\{e : (e_i, e) \in V^*\}$
OrderedErrors := Distortions ∪ Degradations
Limits := ∅
PotLimits:= **inf**(OrderedErrors)
while PotLimits ≠ ∅ **do**
  forall $e \in$ PotLimits **do**
    detected(e) := **VerifyBySubstitution**(i,e)
    if detected(e) **then**
      OrderedErrors:= OrderedErrors\{e}\ $\{e_k : (e, e_k) \in V^*\}$
      Limits := Limits ∪{e}

```
            else
              OrderedErrors:= OrderedErrors\{e}
            endif
        done
        PotLimits:=inf(OrderedErrors)
done
```

Where:

• **VerifyBySubstitution**(i,e) is a procedure that returns **true** if the replacement of $Ref(i)$ by e in the calculus of $Ref_E^M(s)$ gives a refusal set r such that $r \notin Ref_E^M(s)$.
where s is a state (of $M|[\Gamma]|E \setminus \Gamma$) containing a couple $(i, *)$.
• for $\mathcal{E} = \{e_1, e_2, .., e_n\} \subseteq G$, $\mathbf{inf}(\mathcal{E})$ returns the set $\{e \in \mathcal{E} : \nexists e_k \in \mathcal{E} \ : \ (e, e_k) \in V\}$

$\forall e \in$ Limits, we have :
$\Psi_i(e)$ $\neg conf\ M$, $\Psi_i(e)$ $\neg conf_E\ M$
$\Psi_i(e)$ is an erroneous implementation of M detectable by the test through E.
$\forall e' \in G \ : e' \mathbb{C}_s\ e$ implies $\Psi_i(e')\ conf_E\ M$
No implementation less erroneous than $\Psi_i(e)$ can be detected through E.
$\forall e' \in G \ : e \mathbb{C}_s\ e'$ implies $\Psi_i(e')$ $\neg conf_E\ M$
Any implementation more erroneous than $\Psi_i(e)$ can be detected through E.

5 Conclusion

In this paper, we have presented a formalization of the notion of degradation of the testablity of a system when the latter is embedded in an environment that prevents it from being directly tested. We relied on a formal definition of the conformance of an implementation to a given specification. This conformance definition supports automatic tests generation, for details see [BR 88, LE2 91]. We explained the testability degradation by the existence of nonconforming implementations (i.e. erroneous implementations which are discarded by conformance testing when no constraints are imposed by an environment) that the testing through an environment procedure cannot detect.

Assuming hiding synchronization actions creates no divergence, the notion of testability degradation limits was put forward. Limits was characterized as erroneous implementations before which nonconformance cannot be detected when testing through an environment. An original presentation of the conformance and related notions has been given. To simulate erroneous implementations and determine these limits, we proposed transformations by 'degradation' and 'distorsion' of the specification described by a bilabeled graph structure referred to as refusal graph. Note that characterizing ordered erroneous implementations can be used for more general purposes, e.g. for test selection. A framework for test selection, proposed in [BTV 91], assumes error characterization as we detailed in this paper.

The refusal graph structure yields composition and restriction operators that lead to an operational approach for the testability analysis. Furthermore, this

structure, together with the bisimulation relation, $\approx_g$, defined on, facilitate checking of testing equivalence of two systems.

Finally an approach for simulation of erroneous implementations and computation of the so-called limits of testability has been presented.

Acknowledgment: We thank Guy Leduc of Université de Liège for his relevant comments on the draft version of this paper.

References

[AB 87] S. ABRAMSKY *Observation equivalence as a testing equivalence* Theoretical Computer Science 53 (1987), pp. 225-241.

[BK 85] J.A. BERGSTRA, J.W. KLOP *Algebra of Communicating Processes with Abstraction.* Theoretical Computer Science 37 (1985). 77-121

[BSS 87] E. BRINKSMA, G. SCOLLO AND C. STEENBERGEN *Lotos Specifications, their implementations and their tests.* Protocol Specification Testing and Verification, VI. B. Sarikaya and G.V. Bochmann (editors) Elsevier Science Publishers B.V. (North-Holland) 1987

[BR 88] E. BRINKSMA *A theory for the derivation of tests.* Protocol Specification Testing and Verification, VIII. S. Aggrawal and K.Sabani (editors) Elsevier Science Publishers B.V. (North-Holland) 1988.

[BTV 91] E. BRINKSMA, J. TRETMANS and L. VERHAARD *A Framework for Test Selection* Proceedings of the 11th international IFIP WG6.1 Symposium on Protocol Specification, Testing and Verification. Stockholm, June 17-20 1991.

[DH 84] R. DE NICOLA, M.C.B. HENNESSY *Testing equivalences for processes.* Theorical Computer Science 34 (1984). 83-133.

[DN 87] R. DE NICOLA *Extensional Equivalences for Transition Systems.* Acta Informatica 24, (1987), pp. 211-237.

[HO 85] C.A.R. HOARE *Communicating Sequential Processes.* Printice-Hall International series in computer science, New York 1985

[LE 87] G. LEDUC *The Interwriting of Data Types and Processes in LOTOS.* Protocol Specification Testing and Verification, VII. H. Rudin and C.H. West (editors) Elsevier Science Publishers B.V. (North-Holland) 1987

[LE 90] G. LEDUC *On the role of Implementation Relations in the Design of Distributed Systems using LOTOS.* dissertation d'agrégation, Université de Liège, Juillet 1990

[LE 91] G. LEDUC *A Framework based on implementation relations for implementing LOTOS specifications.* Computer Networks & ISDN Systems, 1991.

[LE2 91] G. LEDUC *Conformance relation, associated equivalence, and new canonical tester in LOTOS.* Proceedings of the 11th international IFIP WG6.1 Symposium on Protocol Specification, Testing and Verification. Stockholm, June 17-20 1991.

[LOT 88] INTERNATIONAL STANDARD ISO8807 *Information processing systems, Open systems interconnection, A formal description technique based on the temporal ordering of observational behaviour*

[MI 80] R. MILNER *A Calculus of Communicating System,* LNCS, Vol. 64, 1980.

[PA 81] D. PARK *Concurrency and Automata on Infinite Sequences,* LNCS, Vol. 104, 1981.