3

Failure-based Congruences, Unfair Divergences and New Testing Theory

Guy Leduc

Research Associate of the National Fund for Scientific Research (Belgium)
Université de Liège, Institut d'Electricité Montefiore, B 28, B-4000 Liège 1, Belgium

The testing equivalence te that is used as a reference in verification and testing theory in LOTOS is not a congruence, and no explicit definition of the least congruence stronger than te has been found. The critical LOTOS context in which congruence is lost is the hiding context that creates divergence. In this paper we first survey this problem and present three known variants of te that are congruences. Each of them, as well as te, is then related to a particular interpretation of divergences in terms of (un)fairness of divergences. The associated preorders that generate these equivalences are also presented. Based on these results, we propose a new testing theory based on unfair divergences. It defines new equivalence and conformance relations, as well as the associated canonical tester. We also prove that the least congruence stronger than this new testing equivalence is one of three presented failure-based congruences, which thus also deserves the label of testing congruence.

Keyword Codes: F.4.3; D.2.5
Keywords: Formal Languages; Testing and Debugging.

1. Introduction

There exist many variants of testing (or failures) equivalences in the literature. We will review some of them in the sequel. In LOTOS [ISO 8807, BoB 87], the testing equivalence that is used as reference in verification and testing theory was proposed in [BSS 87] and denoted te. Contrary to most other testing equivalences, te is a failures equivalence that does not refer explicitly to divergences in its definition. This does not mean that divergences have no clear interpretation. Indeed, divergences are interpreted as harmless (or fair) in the sense that removing (or adding) a divergence keeps the system unchanged modulo te.

The problem with te is that it is not a congruence in abstraction contexts, i.e. the LOTOS hiding contexts. This means that if one hides some actions in two Labelled Transition Systems (LTS) that are equivalent modulo te, we can get two LTS that are no more equivalent modulo te. As hiding and parallel composition contexts are the most important contexts in complex system design, we consider this non-congruence as a major drawback. For example, let us consider a LOTOS specification in which several processes are interconnected and some interfaces are hidden from the external environment. Replacing one process by another te-equivalent process may have as consequence that the *global* specification is no more te-equivalent to the initial one. This precludes any form of modular design based on te. The situation is also critical for the red and ext preorders of te. Note that this lack of congruence only appears in hiding contexts that create divergence. This would not be too critical if it was possible to give an explicit definition to the weakest congruence stronger than te and to its associated preorder. However, this does not seem to be feasible.

S. T. Vuong et al. (eds.), *Protocol Specification, Testing and Verification XIV*

There exist other variants of testing (or failures) equivalences that are congruences. However, their interpretation of divergence is not intuitively adequate. For example, in the denotational semantics of TCSP [Hoa 85, BrR 85], the divergences are considered catastrophic. This means that when a process starts diverging, its behaviour is considered chaotic in the sense that no information about it is recorded in the model (the process may execute all traces and, after each one, will diverge and may accept or refuse any action). This interpretation is not really adequate w.r.t. the operational interpretation of this phenomenon in terms of infinite internal sequences. The same interpretation applies in the "must" version of the acceptance model of [dNH 84]. To a lesser extent, the catastrophic view is also used in the "may+must" version of the testing theory of CCS [dNH 84, Hen 85, Hen 88, ClH89]. Basically, in this model, only a limited information (viz. the traces) is recorded when a process starts diverging. Moreover, in these models, any trace that *passes through* a state where an unbounded sequence of successive internal steps is enabled is called a divergence. We will refer to this notion as a *subdivergence*, rather than as a divergence. In this paper, a divergence is a trace *leading to* a state where an infinite sequence of successive internal steps is enabled. In the readiness model of TCSP [OlH 86] the catastrophic view is also used. Finally, in the failures model with explicit divergences of [BKO 87], more information (e.g. on stable failures, see later) is recorded when a process starts diverging but again subdivergences are used instead of divergences.

If we focus on failures equivalences that either refer explicitly to divergences (i.e. not to subdivergences), or do not refer explicitly to any form of divergence in their definition (like te), then we know of three proposals that are congruences: the failures equivalence with abstraction of unstable divergences of [BKO 87] and the CFFD (Chaos-Free Failures Divergences) and NDFD (NonDivergent Failures Divergences) equivalences of [KaV 92].

In this paper, we will first present in our notations these three failure-based congruences, denoted $=_{FAUD}$, $=_{CFFD}$ and $=_{NDFD}$. It is also shown that $=_{CFFD}$ is the intersection of $=_{FAUD}$ and $=_{NDFD}$ and thus the strongest equivalence among the three. Also, none of these equivalences are comparable to te, but $=_{NDFD}$ (and thus also $=_{CFFD}$) is stronger than the testing equivalences of [dNH 84]. We also examine their preorders and explain that some or all divergences are considered unfair modulo these congruences.

The testing theory of LOTOS is based on te with its fair interpretation of divergences. In the last part of this paper, we propose a new testing theory based on unfair divergences. It includes a new conformance relation, a new canonical tester and a new testing equivalence. We also prove that the least congruence stronger than this new testing equivalence is $=_{NDFD}$ which is thus our new testing congruence.

2. On the non-congruence of the testing equivalence

Labelled Transition Systems (LTS) are used as models for LOTOS processes. We let the alphabet of actions be $A = L \cup \{i\}$ and $L = G \cup \{\delta\}$ where G is a countable set of observable gates (ranged over by $a, b, c, g \ldots$), L denotes the alphabet of observable actions, δ is the special action denoting successful termination. The symbol i is reserved for the unobservable internal action, and does not belong to L. Capital Greek letters such as Γ will be used to denote subsets of L. L^* denotes the set of strings over L (ranged over by $\sigma, \sigma', \ldots$), and ε the empty string. Note that depending on the context, b may denote the action b or the string

composed of the single action *b*. $\wp$ *(L)* is the power set of *L*, i.e. the set of subsets of *L*. Capital letters *P, P', P_1, Q, ...* are used to represent states (that is LOTOS processes or behaviour expressions).

$P \xrightarrow{a} P'$ means that process P may engage in action a and, after doing so, behave like P'.

$P \not\xrightarrow{a}$ means $\forall P'. \neg (P \xrightarrow{a} P')$

$P \xrightarrow{i^k} P'$ means that process P may engage in the sequence of k internal actions and, after doing so, behave like process P'.

$P \xrightarrow{a.b} P'$ means $\exists P''. P \xrightarrow{a} P'' \wedge P'' \xrightarrow{b} P'$.

$P \stackrel{a}{\Rightarrow} P'$ where $a \in L$, means $\exists k_0, k_1 \in N. P \xrightarrow{i^{k_0}.a.i^{k_1}} P'$

$P \stackrel{a}{\Rightarrow}$ where $a \in L$, means that $\exists P'. P \stackrel{a}{\Rightarrow} P'$, i.e. P may accept the action a.

$P \not\stackrel{a}{\Rightarrow}$ where $a \in L$, means $\neg (P \stackrel{a}{\Rightarrow})$, i.e. P cannot accept (or must refuse) the action a.

$P \stackrel{\sigma}{\Rightarrow} P'$ means that process P may engage in the sequence of observable actions σ and, after doing so, behave like process P'. More precisely, if $\sigma = a_1 \ldots a_n$ where $a_1, \ldots a_n \in L$:

$\exists k_0, \ldots k_n \in N. P \xrightarrow{i^{k_0}.a_1.i^{k_1}a_2 \ldots a_n.i^{k_n}} P'$

$P \stackrel{\sigma}{\Rightarrow}$ means that $\exists P'. P \stackrel{\sigma}{\Rightarrow} P'$, whereas $P \not\stackrel{\sigma}{\Rightarrow}$ means $\neg (P \stackrel{\sigma}{\Rightarrow})$

Definitions 2.1

$Tr(P) = \{\sigma \mid P \stackrel{\sigma}{\Rightarrow}\}$ i.e. the trace set of P. This set is prefix-closed.

$Ref(P) = \{X \subseteq L \mid \exists P'. P \stackrel{\varepsilon}{\Rightarrow} P' \wedge \forall a \in X. P' \not\stackrel{a}{\Rightarrow}\}$ i.e. Ref (P) is the refusal set of P.

Ref (P) is a set of sets and a subset of $\wp$ (L). A set $X \subseteq L$ belongs to Ref (P) iff P may refuse initially every event of the set X. Ref (P) is subset-closed.

$Fail(P) = \{(\sigma, X) \in L^* \times \wp(L) \mid \exists P'. P \stackrel{\sigma}{\Rightarrow} P' \wedge X \in Ref(P')\}$ i.e. the set of failures of P

Stable (P) iff $P \not\xrightarrow{i}$

The testing equivalence te and the associated relations proposed in [BSS 87] may be expressed in these notations as follows.

Definitions 2.2

P_1 te P_2 iff $Fail(P_1) = Fail(P_2)$

P_1 conf P_2 iff $Fail(P_1) \cap (Tr(P_2) \times \wp(L)) \subseteq Fail(P_2)$

Informally, P_1 conf P_2 iff, placed in any environment whose traces are limited to those of $\mathbf{P_2}$, P_1 cannot deadlock when P_2 cannot deadlock.

P_1 red P_2 iff $Fail(P_1) \subseteq Fail(P_2)$ or equivalently iff $Tr(P_1) \subseteq Tr(P_2) \wedge P_1$ conf P_2

P_1 ext P_2 iff $Tr(P_1) \supseteq Tr(P_2) \wedge P_1$ conf P_2

Definitions 2.3

A context C [.] is a behaviour expression with a formal parameter '.', called a hole.

C [P] is C [.] where all occurrences of '.' have been replaced by P.

For example, C [.] = hide a in (P || .) and C [Q] = hide a in (P || Q).

An equivalence eq is a congruence iff $\forall$ P, Q. P eq Q $\Leftrightarrow$ C[P] eq C[Q] for every context C[.].

A preorder $\leq$ is a congruent preorder (or precongruence) iff

$\forall$ P, Q. $P \leq Q \Leftrightarrow C[P] \leq C[Q]$ for every context C [.]

Unfortunately, te is not a congruence and red is not a precongruence. In [BrS 86], te and red have been strengthened to get closer to such congruence and precongruence. These relations are denoted tc[1] and cred[2] in [BrS 86]. They are (pre)congruent in choice and disabling contexts thanks to the condition on the stabilities, but they still failed to be (pre)congruent in hiding contexts. We illustrate hereafter the problem of the hiding operator [Led 91].

Consider processes *P* and *Q* defined recursively as (depicted on figure 1)

P := a; (b; stop [] i; P)

Q := P [] R where *R := a; R*

It can be shown easily that they have exactly the same failures and are thus te-equivalent. However, *hide a in P* and *hide a in Q* have different refusal sets, thereby showing that hiding does not preserve te (nor tc).

Namely, *Ref (hide a in P) = {∅, {a}}* but *Ref (hide a in Q) = {∅, {a}, {b}, {a, b}}*.

P te Q but ¬*(hide a in P te hide a in Q)*.

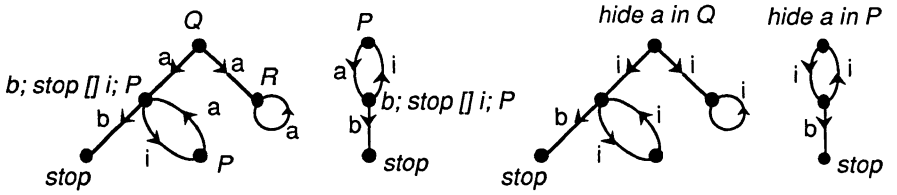

Figure 1 : Non congruence of te in hiding contexts

We are interested in variants of the testing equivalence which are congruences in every LOTOS context, including hiding. Therefore, strengthening the te-equivalence such that, for example, the processes *P* and *Q* above are discriminated is not what we are looking for. In this respect however, note that neither the readiness equivalence [OlH 86], nor the failure trace equivalence[3] [Phi 87, Lan 90], which are stronger than te, are strong enough. By contrast, we are looking for variants of the te-equivalence that accept *hide a in P* and *hide a in Q* in the same equivalence class.

A closer look at this problem shows that it is essentially due to the presence of divergences[4]. For convergent processes, te (or more precisely tc) is adequate, but when hiding creates divergence the congruent character disappears. This problem was first studied in [BKO 87] where it was proved that failure semantics is inconsistent with fair abstraction of divergences.

The reader may think that the creation of divergences in hiding contexts is an academic concern. On the contrary, hiding contexts that create divergences (sometimes also referred to as livelocks) are quite usual as illustrated on the simple example hereafter (shown on fig. 2).

```
Behaviour
hide message, ack, nack in Sender |[message, ack, nack]| Receiver where
Sender := send; S where S := message; (nack; S [] ack; Sender)
Receiver := message; (nack; Receiver [] receive; ack; Receiver)
```

---

[1] P_1 tc P_2 iff Fail (P_1) = Fail (P_2) ∧ (Stable (P_1) ⇔ Stable (P_2)) in our notations

[2] P_1 cred P_2 iff Fail (P_1) ⊆ Fail (P_2) ∧ (Stable (P_1) ⇐ Stable (P_2)) in our notations

[3] This fact has been first communicated to me by Rom Langerak when I sent him this example.

[4] A precise definition of a divergence will be given later. Let us just consider that a divergence is a trace leading to a state where an infinite sequence of internal actions is enabled.

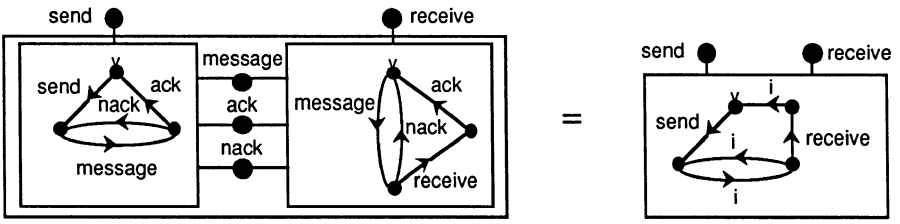

Figure 2: Classical example of a hiding context that creates divergence

3. Failure-based congruences

As mentioned in the introduction, we disregard the failure-based congruences that consider divergences as catastrophic or that refer to subdivergences, because these definitions do not match the operational intuition in terms of infinite sequences of internal actions.

3.1. Stable failure-based congruence with fair abstraction of unstable divergence [1]

In this section we will present a variant of te which is a congruence. The basic idea is to replace the failures by *stable* failures in te.

Definitions 3.1

$SRef(P) = \{X \subseteq L \mid \exists P'.\ P \stackrel{\varepsilon}{\Rightarrow} P' \wedge Stable(P') \wedge \forall a \in X.\ P' \stackrel{a}{\not\Rightarrow}\}$

$= \{X \subseteq L \mid \exists P'.\ P \stackrel{\varepsilon}{\Rightarrow} P' \wedge \forall a \in X \cup \{i\}.\ P' \stackrel{a}{\not\Rightarrow}\}$;

i.e. the stable refusal set of P. SRef is subset-closed and $SRef \subseteq Ref$.

$SFail(P) = \{(\sigma,X) \in L^* \times \wp(L) \mid \exists P'.\ P \stackrel{\sigma}{\Rightarrow} P' \wedge X \in SRef(P')\}$

i.e. the set of stable failures of P. $SFail \subseteq Fail$

Stable failures have been first used in [BKO 87], but the concept of a stable acceptance set (which is the dual of SRef) was already used in [OlH 86] to define the readiness equivalence. Stable failures have also been used in [ClH89, VaT 91, KaV 92].

$STr(P) = Domain(SFail(P)) = \{\sigma \in L^* \mid (\sigma,\emptyset) \in SFail(P)\}$

i.e. the set of traces leading to stable states, or stable traces for short.

Definitions 3.2

(i) $P_1 =_{FAUD} P_2$ iff $Tr(P_1) = Tr(P_2) \wedge SFail(P_1) = SFail(P_2) \wedge (Stable(P_1) \Leftrightarrow Stable(P_2))$

(ii) $P_1 \leq_{FAUD} P_2$ iff $Tr(P_1) \subseteq Tr(P_2) \wedge SFail(P_1) \subseteq SFail(P_2) \wedge (Stable(P_1) \Leftarrow Stable(P_2))$

Note that traces (and not stable traces) are used in this definition. The conditions on the stabilities are needed to get congruence in choice and disabling contexts, and the replacement of Fail by SFail solves the non congruence in hiding contexts (see below). This $=_{FAUD}$ equivalence is a translation is our setting of the failure semantics with Fair Abstraction of Unstable Divergence (FAUD) presented in [BKO 87]. This terminology will be explained later.

Propositions 3.3

(i) $\leq_{FAUD}$ is a preorder (i.e. a reflexive and transitive relation)

---

[1] The '(un)fair' or '(un)stable' attributes of divergences will be defined and explained later.

(ii) $P_1 =_{FAUD} P_2$ iff $P_1 \leq_{FAUD} P_2 \wedge P_2 \leq_{FAUD} P_1$
(iii) All LOTOS operators are monotonic w.r.t. $\leq_{FAUD}$, i.e. $\leq_{FAUD}$ is a precongruence.
(iv) $=_{FAUD}$ is a congruence
The proofs of (i) and (ii) are obvious. The proof of (iii) can found in [VaT 91], and (iv) is derived directly from (iii).

The next propositions will clearly indicate that the differences between $\leq_{FAUD}$ and $\underline{cred}$, $=_{FAUD}$ and $\underline{tc}$ only appear on divergent processes.

Propositions 3.4
For convergent processes, $\leq_{FAUD} = \underline{cred}$, $=_{FAUD} = \underline{tc}$. Note that for divergent processes these relations are not comparable (see figure 3 for an illustration of $=_{FAUD} \neq \underline{tc}$).

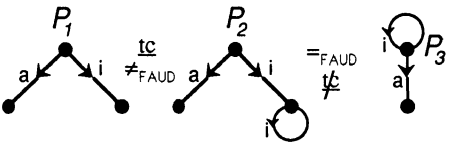

Figure 3: Relation between $\underline{tc}$ and $=_{FAUD}$.

At this stage we have presented the $=_{FAUD}$ equivalence (which is a congruence) and that satisfies our initial requirements. Coming back to our example of figure 1, we have $P =_{FAUD} Q$ and *(hide a in P)* $=_{FAUD}$ *(hide a in Q)*. The reason is that the trace ε cannot lead to a stable state, neither in *hide a in P*, nor in *hide a in Q*, so that $(\varepsilon,\emptyset) \notin$ SFail (hide a in P) and $(\varepsilon,\emptyset) \notin$ SFail (hide a in Q).

3.2. Stable failure-based congruence with unfair interpretation of divergence

In this section, $=_{FAUD}$ will be changed to take account explicitly of divergences. To get this new equivalence, we strengthen $=_{FAUD}$ by requiring that the sets of divergent traces of two processes should also be equal. However, to get congruence on infinite-state systems, the equality of infinite traces is also required.

Definitions 3.5

$Div(P) = \{\sigma \in L^* \mid \exists P'.\ P \stackrel{\sigma}{\Rightarrow} P' \stackrel{i^\infty}{\rightarrow}\}$ i.e. the set of strong divergences of P

$InfTr(P) = \{\sigma \in L^\infty \mid P \stackrel{\sigma}{\Rightarrow}\}$ i.e. the set of infinite traces of P

Note that in finite-state LTSs, the infinite traces can always be derived from the finite traces. However, this is not true in infinite-state LTSs. Consider the following example: P := a; P' where P' := a; P' [] i; stop, and Q := choice n:nat [] Q'(n) where Q'(n) := [n>0] -> a; Q'(n-1). P and Q have the same finite traces (viz. any finite sequence of a's), but P has the infinite trace a^∞ whereas Q has not.

Definitions 3.6

(a) $P_1 =_{CFFD} P_2$ iff $P_1 =_{FAUD} P_2 \wedge Div(P_1) = Div(P_2) \wedge InfTr(P_1) = InfTr(P_2)$
or stated otherwise iff $SFail(P_1) = SFail(P_2) \wedge Div(P_1) = Div(P_2) \wedge$
$(Stable(P_1) \Leftrightarrow Stable(P_2)) \wedge InfTr(P_1) = InfTr(P_2)$
(b) $P_1 \leq_{CFFD} P_2$ iff $P_1 \leq_{FAUD} P_2 \wedge Div(P_1) \subseteq Div(P_2) \wedge InfTr(P_1) \subseteq InfTr(P_2)$

This equivalence is the CFFD equivalence of [KaV 92]. To see why equality of infinite traces is needed when equality of divergence sets is, we simply hide a in both processes P and Q defined above, and get a divergence in hide a in P, but no divergence in hide a in Q.

Propositions 3.7 [KaV 92]

(i) $P_1 =_{CFFD} P_2$ iff $(P_1 \leq_{CFFD} P_2 \wedge P_2 \leq_{CFFD} P_1)$
(ii) $\leq_{CFFD}$ is a precongruence
(iii) $=_{CFFD}$ is a congruence

A version of $=_{CFFD}$ without equality of infinite traces, denoted $=_{FinCFFD}$, was presented in [VaT 91]. $=_{FinCFFD}$ is a congruence on finite-state LTSs only.

The examples of figure 4 show the discriminative power of $=_{CFFD}$. On the other hand, processes P_2 and P_3 (figure 2) are $=_{CFFD}$-equivalent, which illustrates the limit of this discriminative power.

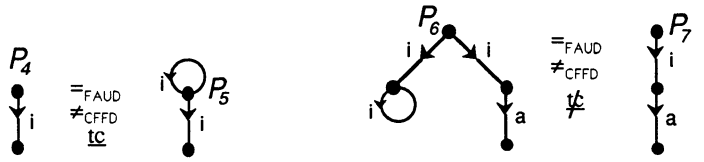

Figure 4: Non $=_{CFFD}$-equivalent processes

3.3. Convergent failure-based congruence with unfair interpretation of divergence

We introduce a third failure-based congruence, denoted $=_{NDFD}$. In this semantics two processes are equivalent if they have the same finite and infinite traces, divergences, stabilities and the same refusals after their *convergent* traces. In this case, stable refusals may also be used because SRef = Ref after convergent traces.

Definitions 3.8

$Conv(P) = Tr(P) - Div(P)$

$CFail(P) = Fail(P) \cap (Conv(P) \times \wp(L)) = SFail(P) \cap (Conv(P) \times \wp(L))$

Definitions 3.9

$P_1 =_{NDFD} P_2$ iff $CFail(P_1) = CFail(P_2) \wedge Div(P_1) = Div(P_2) \wedge (Stable(P_1) \Leftrightarrow Stable(P_2)) \wedge InfTr(P_1) = InfTr(P_2)$

$P_1 \leq_{NDFD} P_2$ iff $CFail(P_1) \subseteq CFail(P_2) \cup (Div(P_2) \times \wp(L)) \wedge Div(P_1) \subseteq Div(P_2) \wedge (Stable(P_1) \Leftarrow Stable(P_2)) \wedge InfTr(P_1) \subseteq InfTr(P_2)$

The first condition in $=_{NDFD}$ means that refusal sets are required to be identical after convergent traces only. No information on the refusals after divergence traces is considered. This equivalence is the NDFD equivalence of [KaV 92].

Propositions 3.10 [KaV 92]

(i) $P_1 =_{NDFD} P_2 \Leftrightarrow (P_1 \leq_{NDFD} P_2 \wedge P_1 \leq_{NDFD} P_2)$
(ii) $\leq_{NDFD}$ is a precongruence
(iii) $=_{NDFD}$ is a congruence

Note that the condition on 'Div' in $=_{NDFD}$ can be replaced by Tr (P_1) = Tr (P_2), but the condition on 'Div' in $\leq_{NDFD}$ cannot be replaced by Tr $(P_1) \subseteq$ Tr (P_2) because this would falsify proposition 3.10 (i). Note also that the first condition in $\leq_{NDFD}$ could be replaced by CFail $(P_1) \subseteq$ CFail (P_2) but this would add an unnecessary constraint.

Figure 5 shows pairs of $=_{NDFD}$-equivalent processes that are not $=_{FAUD}$-equivalent. Processes P_2 and P_3 (figure 3) are $=_{NDFD}$-equivalent and $=_{FAUD}$-equivalent.

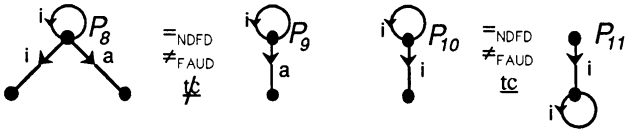

Figure 5: $=_{NDFD}$-equivalent processes

3.4. Comparison of failure-based congruences

Propositions 3.11

(i) $P_1 \leq_{CFFD} P_2$ iff $P_1 \leq_{FAUD} P_2 \wedge P_1 \leq_{NDFD} P_2$

(ii) $P_1 =_{CFFD} P_2$ iff $P_1 =_{FAUD} P_2 \wedge P_1 =_{NDFD} P_2$

Proof. Item (ii) derives directly from (i). For item (i), the ($\Leftarrow$) direction is obvious from the definitions of the preorders. For the ($\Rightarrow$) direction, the non trivial part to prove is $P_1 \leq_{CFFD} P_2 \Rightarrow$ CFail $(P_1) \subseteq$ CFail $(P_2) \cup$ (Div $(P_2) \times \wp$ (L)). So, let $(\sigma, X) \in$ CFail (P_1) and $\sigma \in$ Conv (P_2), otherwise the proof is obvious. Then $(\sigma, X) \in$ CFail (P_1) is equivalent to $(\sigma, X) \in$ SFail (P_1), which implies $(\sigma, X) \in$ SFail (P_2) by $P_1 \leq_{CFFD} P_2$, which is equivalent to $(\sigma, X) \in$ CFail (P_2), which proves the proposition. ❐

Figure 6 shows that none of these three equivalences are comparable with tc. In this figure, each relation is represented as a set (of pairs of behaviours), $=_{CFFD}$ is the intersection of $=_{FAUD}$ and $=_{NDFD}$, and the arrows show some effects of hiding. An arrow originates from a pair of behaviours and leads (when hiding actions) to a new pair of behaviours. When an arrow leaves an equivalence, this means that this equivalence is not a congruence in hiding contexts.

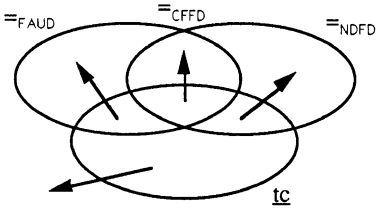

Figure 6: Relative strengths of failure-based equivalences

Other well-known similar equivalences exist but have not been added on figure 6. A survey can be found in [dNi 87]. A first example is the original (chaotic) failures semantics [Hoa 85, BrR 85] and the equivalent "must" testing equivalence of [dNH 84], denoted here $=_{must}$. Another example is the "may+must" testing equivalence of [dNH 84], denoted here $=_t$. We

have that $=_{NDFD}$ is stronger than $=_{\tau}$ which is known to be stronger than $=_{must}$. However, $=_{\tau}$ and $=_{must}$ are not comparable to $\underline{tc}$, nor to $=_{FAUD}$.

In [BKO 87] a failure semantics with explicit divergence, called FS_{Δ}, is proposed, which is similar to the $=_{CFFD}$ equivalence. The differences are: in FS_{Δ} *sub*divergences[1] are used instead of divergences, but the equality of infinite traces is not required.

4. Interpretation of divergences

In this section we define the concepts of (un)stable and (un)fair divergence and give the interpretations of divergences induced by $\underline{te}$ and the three presented congruences.

Proposition 4.1 $\forall$ P. Tr (P) = STr (P) $\cup$ Div (P).

i.e. the traces of P are either stable traces of P or (not exclusive) divergences of P.
The proof is straightforward.

Definitions 4.2 (Stability and unstability of a divergence)

SDiv (P) $= \text{Div}(P) \cap \{ \sigma \in L^* \mid \forall P'. (P \stackrel{\sigma}{\Rightarrow} P' \Rightarrow \text{SRef}(P') = \varnothing)\}$
$= \text{Div}(P) - \text{STr}(P) = \text{Tr}(P) - \text{STr}(P)$

UnsDiv (P) $= \text{Div}(P) - \text{SDiv}(P) = \text{Div}(P) \cap \text{STr}(P)$

Conv (P) $= \text{Tr}(P) - \text{Div}(P) = \text{STr}(P) - \text{UnsDiv}(P)$

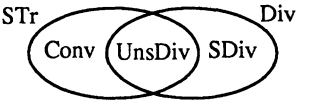

Figure 7: Classification of the traces

SDiv (P) is the set of stable divergences of P. A trace σ is a stable divergence of P if any process (or state) P' reachable from P by executing σ, is divergent (after ε). This means that reaching a divergent state is unavoidable after σ.

UnsDiv (P) is the set of unstable divergences of P. A trace σ is an unstable[2] divergence of P if it is a divergence and if there exists a *stable* process (or state) P' reachable from P by executing σ. This means that reaching a divergent state is possible but not certain after σ.

Conv (P) is the set of convergent traces of P.

Fairness and unfairness of divergences

Propositions 4.3

(i) $P_1 =_{FAUD} P_2 \Rightarrow \text{SDiv}(P_1) = \text{SDiv}(P_2)$

(ii) $P_1 =_{CFFD} P_2 \Rightarrow (\text{SDiv}(P_1)=\text{SDiv}(P_2) \wedge \text{UnsDiv}(P_1)=\text{UnsDiv}(P_2)) \Rightarrow \text{Div}(P_1)=\text{Div}(P_2)$

(iii) $P_1 =_{NDFD} P_2 \Rightarrow \text{Div}(P_1) = \text{Div}(P_2)$

---

[1] σ is a subdivergence if a prefix of σ is a divergence

[2] The expression 'unstable divergence' comes from [BKO 87]. It tends to indicate the possibility of exiting a divergent node silently like in P_5 in fig. 4. However, P_{12} in fig. 8 shows an example of an unstable divergence where there is no explicit silent exit. The same intuition remains however if we consider that P_{12} is equivalent to P_{13} that has a silent exit.

The proofs are straightforward. (i) is derived from def. 4.2 (viz. SDiv (P) = Tr (P) – STr (P)) by noting that the traces and stable traces (derived from the stable failures) are equal if $=_{FAUD}$ holds. (ii) is derived from UnsDiv (P) = STr (P) $\cap$ Div (P), and the divergences and stable traces (derived from the stable failures) are equal if $=_{CFFD}$ holds. (iii) is directly derived from the definition of $=_{NDFD}$. ❐

It is interesting to note that unstable divergences are not determined modulo $=_{FAUD}$. By contrast, modulo $=_{CFFD}$, not only stable divergences are required to be equal, but also unstable divergences. Finally, modulo $=_{NDFD}$, the divergence sets are required to be equal, but their stability does not matter.

Now we discuss how this can be related to the (un)fairness of divergences. Informally, when a divergence may be removed while preserving an equivalence, we will say that this divergence is (considered) fair (or harmless) w.r.t. this equivalence. On the other hand, when a divergence cannot be removed while preserving the equivalence, we will say that this divergence is unfair w.r.t. this equivalence. We will see for example that all divergences are fair w.r.t. te, whereas only unstable divergences are fair w.r.t. $=_{FAUD}$, and all divergences are unfair w.r.t. $=_{CFFD}$ and $=_{NDFD}$.

Definitions 4.4

((Un)stable) divergences are fair or harmless w.r.t. an equivalence eq iff

$\forall$ P. $\exists$ P'. P' eq P $\wedge$ ((Un)S)Div (P') = $\emptyset$.

((Un)stable) divergences are unfair w.r.t. an equivalence eq iff

$\forall$ P. $\nexists$ P'. P' eq P $\wedge$ ((Un)S)Div (P') $\subset$ ((Un)S)Div (P)

Propositions 4.5

(i) Divergences are fair w.r.t. te.

(ii) Unstable divergences are fair w.r.t. $=_{FAUD}$ (Figure 8) [BKO 87]

(iii) Stable divergences are unfair w.r.t. $=_{FAUD}$ (Figure 9)

(iv) Divergences are unfair w.r.t. $=_{CFFD}$ and $=_{NDFD}$ (Figures 9 and 10)

Proof of (i): removing a divergence (i.e. merging all the states which belong to a cycle of internal events, and removing the local internal loops) does not change the failures of a process.

Proof of (ii): removing an unstable divergence does not change the **stable** failures of a process. Removing an unstable divergence means merging all the states which belong to an **unstable** cycle of internal events, and removing the local **unstable** internal loops. An unstable cycle is a cycle from which it is possible to escape by way of a non empty finite sequence of internal steps to reach a stable state.

(iii) is an obvious consequence of proposition 4.3 (i).

(iv) is an obvious consequence of propositions 4.3 (ii) and 4.3 (iii). ❐

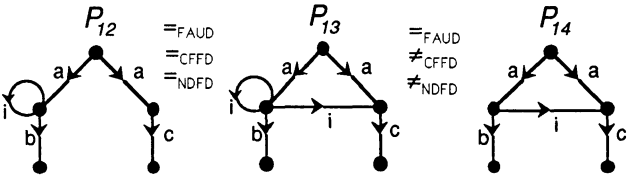

Figure 8: Unstable divergences are fair w.r.t. $=_{FAUD}$

Intuitively, when divergences are fair, the divergent process will eventually exit the internal loop if another action is possible, whereas, when divergences are unfair, the divergent process may also stay in the internal loop forever, which is externally indistinguishable from a potential deadlock (or more precisely a livelock). An illustration of this is given in figures 9 and 10: a process is unaffected modulo $=_{CFFD}$ or $=_{NDFD}$ by the attachment, after a divergent trace, of the process i; Loop where Loop := i; Loop.

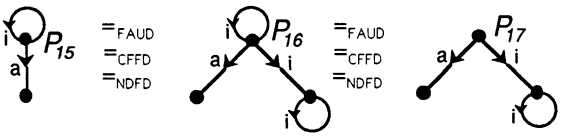

Figure 9: Stable divergences are unfair w.r.t. $=_{FAUD}$, $=_{CFFD}$ and $=_{NDFD}$.

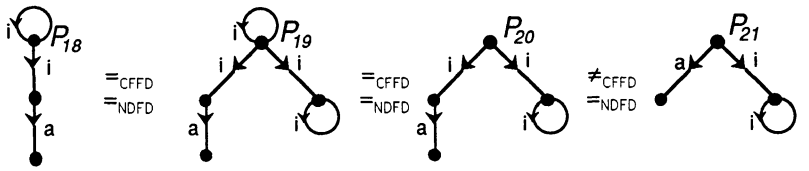

Figure 10: Unstable divergences are unfair w.r.t. $=_{CFFD}$ and $=_{NDFD}$.

5. New testing theory taking divergence into account

In the LOTOS testing theory, which is based on te, the conformance relation, denoted conf, that should hold (and thus be tested) between implementations and specifications is given in def. 2.2. and can be defined equivalently as:

I conf S iff $\forall \sigma \in Tr(S).\ \forall X \subseteq L.\ (\sigma, X) \in Fail(I) \Rightarrow (\sigma, X) \in Fail(S)$

This definition of conformance does not refer explicitly to divergence and leads to the interpretation that divergences (in specifications as well as in implementations) are harmless. In other words, implementations that diverge are always supposed to eventually exit any internal loop if another action is possible. By contrast, in this paper we consider that divergences in implementations are not necessarily harmless. For example, we admit that a divergent implementation may not be able to respond in a finite amount of time. This is an unfair interpretation of divergences which the existing testing theory is not adequate to cope with. In the light of the preceding sections, we propose here a new testing theory that circumvent this limitation.

5.1. New conformance relations

We first define a family of conformance relations as follows.

I conf S iff $\forall \sigma \in$ **Conv** (S). $\forall X \subseteq L$.

(i) $\sigma \in$ **Conv (I)** $\Rightarrow ((\sigma, X) \in Fail(I) \Rightarrow (\sigma, X) \in Fail(S))$, **and**

(ii) $\sigma \in$ **Div (I)** $\Rightarrow$ **a certain condition**

The first idea is to restrict conformance to *convergent* traces of S. This is motivated by the fact that specifying divergences in the *specification* S allows the worst (i.e. divergence and/or deadlock) to be expected in implementations. This hypothesis is aligned with the testing theory of [dNH 84]. Then, we still consider that the inclusion of failures is the relevant criterion

for conformance when there is no divergence, but we consider that, in presence of divergence, this is less obvious. For this reason, we propose and discuss hereafter three conformance relations based on three instantiations of the "certain condition" mentioned in the definition above.

First definition of conformance: "a certain condition" = true

This definition, which is the coarsest of the family and not very interesting, considers, like the usual conf, that an implementation I may diverge anywhere without compromising conformance. So we have:

Definition 5.1
$I \underline{conf_1} S$ iff $\forall \sigma \in Conv(I) \cap Conv(S). \forall X \subseteq L. (\sigma, X) \in Fail(I) \Rightarrow (\sigma, X) \in Fail(S)$

Proposition 5.2 $I \underline{conf} S \Rightarrow I \underline{conf_1} S$

$\underline{Conf_1}$ is not a preorder.

Second definition of conformance: "a certain condition" = $(\sigma, L) \in$ Fail (S)

This definition is an interesting one. It considers that if I diverges, it may not react after a finite amount of time, and therefore be indistinguishable from a deadlock of I. In other words, the condition expresses that *I is only allowed to diverge where it is allowed to deadlock.* This definition does not discriminate livelocks from deadlocks in implementations.

Definition 5.3 $I \underline{conf_2} S$ iff $\forall \sigma \in Conv(S). \forall X \subseteq L.$

(i) $\sigma \in Conv(I) \wedge (\sigma, X) \in Fail(I) \Rightarrow (\sigma, X) \in Fail(S)$, and

(ii) $\sigma \in Div(I) \Rightarrow (\sigma, L) \in Fail(S)$

or equivalently, iff $(CFail(I) \cup (Div(I) \times \wp(L))) \cap (Tr(S) \times \wp(L))$
$\subseteq CFail(S) \cup (Div(S) \times \wp(L))$

Propositions 5.4 $I \underline{conf_2} S \Rightarrow I \underline{conf_1} S$

However, $\underline{conf_2}$ and conf are not comparable. $\underline{Conf_2}$ is not a preorder.

Third definition of conformance: "a certain condition" = false

This definition does not allow an implementation I to diverge when S does not diverge. It is in spirit the idea of the must-preorder of [dNH 84] but with divergences instead of subdivergences. The following definition is indeed similar to the definition in [dNH 84] (with acceptance sets replaced by failures and with a different definition of 'Conv'). However, unlike the must-preorder in [dNH 84], the $\underline{conf_3}$ relation is not a preorder.

Definition 5.5 $I \underline{conf_3} S$ iff $\forall \sigma \in Tr(I) \cap Conv(S). \forall X \subseteq L.$
$\sigma \in Conv(I) \wedge ((\sigma, X) \in Fail(I) \Rightarrow (\sigma, X) \in Fail(S))$

Proposition 5.6 $I \underline{conf_3} S \Rightarrow I \underline{conf_2} S$

However, $\underline{conf_3}$ and conf are not comparable. $\underline{Conf_3}$ is not a preorder.

5.2. New testing preorders and equivalences

From now on, we only consider the $\underline{conf_2}$ and $\underline{conf_3}$ relations, because $\underline{conf_1}$ does not lead to interesting results. If we also require, like in [dNH 84], that an implementation I may pass all

the tests that S may pass, i.e. Tr (I) $\supseteq$ Tr (S), then we get two testing preorders, denoted $\underline{ext_2}$ and $\underline{ext_3}$.

Definitions 5.8

I $\underline{ext_2}$ S iff Tr (I) $\supseteq$ Tr (S) $\wedge$ I $\underline{conf_2}$ S

I $\underline{ext_3}$ S iff Tr (I) $\supseteq$ Tr (S) $\wedge$ I $\underline{conf_3}$ S

Proposition 5.9 $\underline{ext_2}$ and $\underline{ext_3}$ are preorders
The proofs are omitted by lack of place.

The testing equivalences that are generated by these two preorders are as follows.

Definitions 5.10

(i) P $\underline{te_2}$ Q iff Tr (P) = Tr (Q) $\wedge$ CFail (P) $\cup$ (Div(P) $\times$ $\wp$(L)) = CFail (Q) $\cup$ (Div(Q) $\times$ $\wp$(L))

(ii) P $\underline{te_3}$ Q iff Tr (P) = Tr (Q) $\wedge$ Div (P) = Div (Q) $\wedge$ CFail (P) = CFail (Q)

In [Led 91] $\underline{te_2}$ was denoted $\underline{ute}$: the unfair testing equivalence.

Propositions 5.11

(i) P $\underline{te_2}$ Q iff P $\underline{ext_2}$ Q $\wedge$ Q $\underline{ext_2}$ P

(ii) P $\underline{te_3}$ Q iff P $\underline{ext_3}$ Q $\wedge$ Q $\underline{ext_3}$ P

The corresponding reduction preorders are as follows.

Definition 5.12

(i) I $\underline{red_2}$ S iff Tr (I) $\subseteq$ Tr (S) $\wedge$ I $\underline{conf_2}$ S

(ii) I $\underline{red_3}$ S iff Tr (I) $\subseteq$ Tr (S) $\wedge$ I $\underline{conf_3}$ S

Propositions 5.13

(i) $\underline{red_2}$ and $\underline{red_3}$ are preorders

(ii) P $\underline{te_2}$ Q iff P $\underline{red_2}$ Q $\wedge$ Q $\underline{red_2}$ P

(iii) P $\underline{te_3}$ Q iff P $\underline{red_3}$ Q $\wedge$ Q $\underline{red_3}$ P

(iv) I $\underline{red_2}$ S iff Tr (I) $\subseteq$ Tr (S) $\wedge$ CFail (I) $\cup$ (Div (I) $\times$ $\wp$(L)) $\subseteq$ CFail(S) $\cup$ (Div (S) $\times$ $\wp$(L))

(v) I $\underline{red_3}$ S iff Tr (I) $\subseteq$ Tr (S) $\wedge$ Div (I) $\subseteq$ Div (S) $\wedge$ CFail (I) $\subseteq$ CFail (S) $\cup$ (Div (S) $\times$ $\wp$(L))

(vi) I $\underline{red_3}$ S iff I $\underline{red_2}$ S $\wedge$ Div (I) $\subseteq$ Div (S)

The proofs are straightforward.

None of these preorders are precongruences, but the least precongruences stronger than these preorders are easy to find according to the results presented in section 3.

Propositions 5.14

(i) The least precongruence stronger than $\underline{red_3}$ is $\leq_{NDFD}$.

(ii) The least precongruence stronger than $\underline{ext_3}$ is $\leq_{NDFD} \cap \underline{Tr\text{-}eq}$
where P $\underline{Tr\text{-}eq}$ Q iff Tr (P) = Tr (Q)

(iii) The least congruence stronger than $\underline{te_3}$ is $=_{NDFD}$

Proofs: (i) It is well-known that monotonicity of the choice (and also disabling) operator requires the condition on the stabilities. The inclusion of infinite traces is needed in infinite-state systems as shown in [KaV 92]. (ii) The inclusion of finite traces is required in the parallel + hiding + disable contexts: by contradiction, let $\sigma \in$ Tr (P) $\cap$ Tr (Q) and $\sigma.a \in$ Tr (P) – Tr (Q), and the context C [.] = hide a in ((. [> b; stop) |[a,b]| a^n; (a; b; stop [] b; c; stop))

where b, c are not in σ, a^n is a sequence of n a's where n is the number of occurrences of a in σ. We have immediately that $\neg$ (C[P] $\underline{ext_3}$ C[Q]) because $((\sigma.b)\backslash a, L) \in$ CFail (C[P]) whereas $(\sigma.b)\backslash a \in$ Conv (C [Q]) and $((\sigma.b)\backslash a, L) \notin$ CFail (C [Q]). The reason is that C[Q] cannot refuse action c. Together with $\underline{ext_3}$ this leads to equality of finite traces. (iii) is derived from any of (i) or (ii). ❐

It is very interesting to note that a weaker condition than *equality* of traces in proposition 5.14 (ii) would be possible if the disable operator would not be part of LOTOS. This disable operator is indeed necessary in the proof of 5.14 (ii) to have $(\sigma.b)\backslash a \in$ Conv (C [Q]) when $\sigma \in$ Div (Q).

Proposition 5.15
(i) The least precongruence stronger than $\underline{red_2}$ is $\leq_{NDFD}$.
(ii) The least precongruence stronger than $\underline{ext_2}$ is $\leq_{NDFD} \cap \underline{Tr\text{-}eq}$
(iii) The least congruence stronger than $\underline{te_2}$ is $=_{NDFD}$
Proofs: (i) $\underline{red_2}$ is weaker than $\underline{red_3}$, and thus there may exist a precongruence stronger than $\underline{red_2}$ that is weaker than $\leq_{NDFD}$. To prove that this is not possible, it suffices (due to propositions 5.13 (vi) and 5.14 (i)) to prove that inclusion of divergence sets is necessary. Let P $\underline{red_2}$ Q, and suppose there is a trace $\sigma \in$ Div (P) – Div (Q) such that $(\sigma, L) \in$ CFail (Q). Consider the interleaving context C[.] = (. ||| a; stop) with a not in σ. We have immediately that $\neg$ (C[P] $\underline{red_2}$ C[Q]) because $\sigma \in$ Div (C[P]), $\sigma \in$ Conv (C[Q]) and $(\sigma, L) \notin$ CFail (C [Q]). The reason is that C[Q] cannot refuse action a. The proof of (ii) and (iii) are like in proposition 5.14 ❐

5.3. New canonical testers

The $\underline{conf_2}$ relation leads to a new canonical tester that we propose hereafter. The canonical tester associated with $\underline{conf_3}$ can be derived from the canonical tester associated with $\underline{conf_2}$ but requires additional testing power with respect to the detection of divergences in implementations.

First let us recall the usual definition of the canonical tester associated with $\underline{conf}$ [BSS 87]. It is denoted T (S) and is defined modulo $\underline{te}$ by its failures as follows:
$\forall \sigma \in$ Tr (S). $\forall X \subseteq L$. $(\sigma, X) \in$ Fail (T (S)) iff $((\sigma, L - X) \notin$ Fail (S) $\vee$ $(\sigma, L) \in$ Fail (S))

This tester has the property that, when synchronized with an implementation I, it deadlocks with I iff $\neg$ (I $\underline{conf}$ S). In other words the tester reaches a final state (i.e. a state where $(\sigma, L) \in$ Fail (T(S))) iff the implementation is conformant. Every final state of T (S) has thus an implicit "pass" verdict associated with it.

We give now the definition modulo $\underline{te_3}$ of the canonical tester T_2 (S) associated with $\underline{conf_2}$.

Definition 5.16
(i) Tr (T_2 (S)) = Tr (S) $\wedge$ Div (T_2 (S)) = $\varnothing$
(ii) $\forall \sigma \in$ Tr (S). $\forall X \subseteq L$.

$(\sigma,X) \in$ CFail (T_2(S)) iff $((\sigma, L-X) \notin$ CFail (S) $\vee$ $(\sigma, L) \in$ CFail(S))

Intuitively if there is no divergence in the specification S, then T_2 (S) is identical to T (S). But if σ is a divergence in S, then T_2 has an additional correct final state after σ.

The next proposition ensures that T_2 (S) is the canonical tester of S according to $\underline{conf_2}$. It says first that T_2 (S) can test all the traces of S. It also says that, if synchronized with an implementation I, not conforming to S, then I ‖ T_2 (S) may engage in a sequence σ and then deadlock or diverge after σ whereas the tester is still offering interactions. Conversely, if the canonical tester is synchronized with an implementation conforming to S, then a deadlock or a divergence is only possible if the tester has reached a correct final state.

Propositions 5.17

$\forall$ I. I $\underline{conf_2}$ S iff $\forall \sigma \in L^*$.

$$((\sigma, L) \in CFail\ (I \| T_2\ (S)) \vee \sigma \in Div\ (I \| T_2\ (S))) \Rightarrow (\sigma, L) \in CFail\ (T_2\ (S)).$$

We omit the proof by lack of place.

Now we look at the canonical tester associated with $\underline{conf_3}$. Since the $\underline{conf_3}$ relation is stronger than the $\underline{conf_2}$ relation, we may expect that the canonical tester associated with $\underline{conf_3}$ will require more testing power. This is indeed the case. To test the conformance of implementations in the $\underline{conf_3}$ sense, the implementation under test (IUT) must be equipped with a "divergence light" that is 'on' when the IUT diverges, so that the canonical tester be able to discriminate between livelocks and deadlocks of the IUT under some circumstances. This was not needed to test according to $\underline{conf_2}$. The following definition of T_3 (S) shows how a canonical tester based on $\underline{conf_3}$ can be obtained from the canonical tester T_2 (S).

Definition 5.18

T_3 (S) is defined like T_2 (S), but in every final state of T_3 (S), the implicit verdict is not necessarily "pass". Here the verdict has to be defined explicitly as follows: if $\sigma \in$ Div (S) then "pass", else "pass if the divergence light is off".

This additional discrimination when the tester reaches a final state allows testing according to the finer notion of conformance $\underline{conf_3}$. However, considering that the presence of some "divergence light" on the IUT is unlikely, we do not investigate further in this direction.

7. Conclusion

Divergences are often considered as undesirable. The best and most extreme example of this interpretation is certainly the catastrophic interpretation of divergences of TCSP [BrR 85]. At the opposite side, we find the fair interpretation of divergences of $\underline{te}$ in which all divergences are simply considered harmless. The catastrophic interpretation of divergences leads to nice mathematical properties, but has a poor operational interpretation. The opposite is true for $\underline{te}$. Between these two extreme cases, there are however some other failure-based congruences. In this paper, three of them, denoted $=_{FAUD}$, $=_{CFFD}$ and $=_{NDFD}$ have been presented and compared. We have shown that some or all divergences are considered unfair modulo these congruences. Intuitively, when divergences are fair, the divergent process will eventually exit the internal loop if another action is possible, whereas, when divergences are unfair, the divergent process may also stay in the internal loop forever even if another action is enabled, which is externally indistinguishable from a potential deadlock.

In the second part of this paper, we have reconsidered the existing LOTOS testing theory with a different point of view on divergences. The fair interpretation of divergences leads to the well-known $\underline{te}$-equivalence and $\underline{conf}$ relation [BSS 87]. With an unfair interpretation of divergences, a new testing equivalence $\underline{te_2}$ and a new conformance relation $\underline{conf_2}$ have been

derived. $\underline{Conf_2}$ considers divergences as potential livelocks. The testing equivalence $\underline{te_2}$ clearly shows that deadlocks and divergences (or livelocks) are not discriminated. Finally, a canonical tester T_2 has been associated with $\underline{conf_2}$ and the weakest congruence stronger than $\underline{te_2}$ has an easy and explicit definition: $=_{NDFD}$. For this reason the failure-based $=_{NDFD}$-equivalence can be considered as a testing congruence. Furthermore, from [KaV 92] we know that $=_{NDFD}$ is the *weakest* congruence that has the following property: no nexttime-less linear temporal logic formula can distinguish two $=_{NDFD}$-equivalent LTSs. These two properties position $=_{NDFD}$ as an important equivalence.

References

[BKO 87] J.A. Bergstra, J. W. Klop, E.-R. Olderog, *Failures without chaos : a new process semantics for fair abstraction,* in: M. Wirsing, ed., Formal Description of Programming Concepts, III (North-Holland, Amsterdam, 1987) 77-103.

[BoB 87] T. Bolognesi, E. Brinksma, *Introduction to the ISO Specification Language LOTOS,* Computer Networks and ISDN Systems 14 (1) 25-59 (1987).

[BrR 85] S.D. Brookes, A.W.Roscoe, *An Improved Failures Model for Communicating Sequential Processes,* in: S.D. Brookes, A.W. Roscoe, G. Winskel, eds., Seminar on Concurrency, LNCS 197 (Springer-Verlag, Berlin Heidelberg New York Tokyo, 1985) 281-305.

[BrS 86] E. Brinksma, G. Scollo, *Formal notions of implementation and conformance in LOTOS,* Rept. INF-86-13, Twente Univ., Dept. of Inform., Enschede, The Netherl., Dec. 86.

[BSS 87] E. Brinksma, G. Scollo, C. Steenbergen, *Process specification, their implementations and their tests,* in: G.v. Bochmann, B. Sarikaya, eds., Protocol Specification, Testing and Verification, VI (North-Holland, Amsterdam, 1987) 349-360.

[ClH 89] R. Cleaveland, M. Hennessy,*Testing Equivalence as a Bisimulation Equivalence,* in: J. Sifakis, ed., Automatic Verification Methods for Finite State Systems, LNCS 407 (Springer - Verlag, Berlin Heidelberg New York, 1990) 11-23.

[dNH 84] R. De Nicola, M.C.B. Hennessy, *Testing equivalences for processes,* Theoretical Computer Science 34 (1984) 83-133 (North-Holland, Amsterdam).

[dNi 87] R. De Nicola, *Extensional Equivalences for Transition Systems,* Acta Informatica 24 (1987) 211-237 (Springer - Verlag, Berlin Heidelberg).

[Hen 85] M. Hennessy, *Acceptance Trees,* Journal of the ACM, Vol. 32, No. 4, Oct. 85, 896 - 928.

[Hen 88] M. Hennessy, *Algebraic Theory of Processes,* (MIT Press, Cambridge, London, 1988).

[Hoa 85] C.A.R. Hoare, *Communicating Sequential Processes,* (Prentice-Hall Int., London, 85).

[ISO 8807] ISO/IEC-JTC1/SC21/WG1/FDT/C, *Information Processing Systems - Open Systems Interconnection - LOTOS, a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour,* IS 8807, February 1989.

[KaV 92] R. Kaivola, A. Valmari, *The Weakest Compositional Semantic Equivalence Preserving Nexttime-less Linear Time Temporal Logic,* in: R. Cleaveland, ed., CONCUR '92 (LNCS 630, Springer-Verlag, Berlin Heidelberg New York, 1992) 207-221.

[Lan 90] R. Langerak, *A Testing Theory for LOTOS using Deadlock Detection,* in: E. Brinksma, G. Scollo, C. A. Vissers, eds., Protocol Specification, Testing, and Verification, IX (North-Holland, Amsterdam, 1990, ISBN 0-444-88343-6), 87-98.

[Led 91] G. Leduc, *On the role of implementation relations in the design of distributed systems,* in: Collection des Publications de la Faculté des Sciences Appliquées de l'Université de Liège, n° 130 (ISSN 0075-9333, Liège, 1991), Thèse d'agrégat. de l'enseign. supérieur, 283 p.

[OlH 86] E.-R. Olderog and C.A.R. Hoare, *Specification-Oriented Semantics for Communicating Processes,* Acta Informatica 23 (1986) 9-66 (Springer - Verlag, Berlin).

[Phi 87] I. Phillips, *Refusal Testing,* Theoretical Computer Science 50 (1987) 241 - 284

[VaT 91] A. Valmari, M. Tienari, *An Improved Failures Equivalence for Finite-State Systems with a reduction algorithm,* in: B. Jonsson, J. Parrow, B. Pehrson, eds., Protocol Specification, Testing and Verification, XI (North-Holland, Amsterdam, 1991) 3-18.