Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 99 (2004) 111–126
www.elsevier.com/locate/entcs

Applying Generalized Non Deducibility on Compositions (GNDC) Approach in Dependability

S. Gnesi, G. Lenzini [1]

Istituto di Scienze e Tecnologie Informatiche - C.N.R.
Via G. Moruzzi 1, I-56100 Pisa, Italy

F. Martinelli [2]

Istituto di Informatica e Telematica - C.N.R.
Via G. Moruzzi 1, I-56100 Pisa, Italy

Abstract

This paper presents a framework where dependable systems can be uniformly modeled and dependable properties analyzed within the Generalized Non Deducibility on Compositions (*GNDC*), a scheme that has been profitably used in definition and analysis of security properties. Precisely, our framework requires a systems to be modelled using a formal calculus, here the CCS process algebra, where both the failing behaviour of the system and the related fault-recovering procedures are also explicitly described. An environment able to inject any fault in the system is then defined as a separated component. The parallel composition between the system and the environment represents our scenario of analysis, where some fault tolerance property (*e.g.,* fail stop, safe and silent) are studied as instances of *GNDC* properties. By using different instances of *GNDC* we are able to argue about the availability of effective methodologies of analysis, and on the possibility of applying compositional techniques.

Keywords: Dependability, Fault Tolerance, Security, Non Interference, Formal Verification.

[1] Email: {stefania.gnesi, gabriele.lenzini}@isti.cnr.it
[2] Email: fabio.martinelli@iit.cnr.it

doi:10.1016/j.entcs.2004.02.005

1 Introduction

This paper exploits relationships between mechanisms for dependability analysis and those for security in a formal framework. More precisely it shows how fault tolerance analysis mechanisms can benefit of recent results in security analysis.

First ideas about a relationship between security analysis strategies and mechanisms of dependability analysis can be found in [14,15,10,21,22,25]. For example in a seminal paper [25], Weber found analogies between fault-tolerance specifications and non-interference [8] definitions. He also argued that the formal verification of fault tolerance faces with similar problems and benefits of same solutions as verification of security does. Analogies between non-interferences approaches in security and safety analysis can be found in [22]. Additional efforts in exploiting fault tolerant approaches for security analysis can be found for example in [7], where it is shown that fault tolerance and integrity analysis can be both based on a non-interference notion. In [12], non-interference analysis has been shown to be similar to verification of open systems, *i.e.,* when one needs to verify a system w.r.t. whatever possible execution environment (also named as robustness). Remarkably, the idea of applying dependability methodologies in security had been discussed by Meadows in [14], where she also argued that security analysis might be improved by incorporating techniques typical of reliability formal verification. On the other hand, Meadows and McLean also claim, in [15], that the use of emerging results in security analysis could enrich the taxonomy of fault prevention and removal strategies.

Anyway, as previous works seem to show, for long time security had appeared a restricted paradigm w.r.t the variety of approaches available in dependability. Most of the discussions in the literature have been based more on finding intuitions for security starting from the dependability background and experiences than the opposite. Along the last decade security has been studied in so deep details that contributions of security theory into dependability can be now more than in the past. For example we think that the non-interference notion, widely used in security analysis (*e.g.,* see [4,3,13,20,24]), could be reviewed in the dependability framework by considering the analysis techniques developed so far, for non-interference, in the security area.

Following this direction this paper intends to inquire about the application of the Generalized Non-Deducibility on Compositions (for short, *GNDC*) for the analysis of dependable systems, with particular attention to fault tolerant ones. *GNDC* was first introduced in [6] as a framework where a family of security properties could be uniformly expressed and verified. It has never been applied to dependability so far. Anyway *GNDC* finds its roots in non-

interference analysis, whose intuitions has been claimed to share common ideas with some fault tolerance analysis strategies (*e.g.,* see [7,23]). Generally speaking a *GNDC* property has the following form:

$$P \text{ satisfies } GNDC_{\triangleleft}^{\alpha(P)} \text{ iff } \forall X \in Env : (P \parallel X) \triangleleft \alpha(P) \tag{1}$$

Informally it means that a system P enjoys $GNDC_{\triangleleft}^{\alpha(P)}$ if and only if P shows, w.r.t. a certain behavioral relation $\triangleleft$, the same behaviour of $\alpha(P)$. This must be true even if P is composed, by the parallel operator $\parallel$, with any *environment* X. Property (1) is parametric in $\triangleleft$, a relation among processes representing the notion of "observation" and in $\alpha(P)$, a specification which describes the expected behaviour of system P.

In this paper we will show how such a *GNDC* formulation can be exploited for expressing safety properties peculiar of the dependability analysis: *fail stop*, *fail safe*, *fail silent*. It constitutes a step towards a formal classification of dependable properties, similarly as it happened in security [5]. Moreover we will show how some of the theoretical results of *GNDC*, (*e.g.,* compositional reasoning) originally stated for security analysis (*e.g.,* see [5]), can be re-stated and re-used in the analysis of dependability.

The paper is organized as follows. Section 2 describes how a fault tolerant system may be modeled in CCS following a uniform modeling style. Section 3 introduces and explains the *GNDC* characterization of fault tolerance. Section 4 provides intuition about how properties, such as fail stop, silent, and safe can be expressed in the *GNDC* scheme. Section 5 discusses the problem of analyzing dependability. It shows how a *GNDC* characterization of fault tolerance properties using trace-based behavioral equivalences, will enjoy some appealing properties in analysis such as effectiveness of analysis procedures and compositionality. Section 6 concludes the paper.

2 Modeling Fault Tolerant Systems

To model a system we will use a process algebra. Here we refer to the CCS [16], but our framework is completely general and it can be easily restated for other process algebras as well (*e.g.,* , CSP [9] or π-calculus [19]).

2.1 CCS background

The CCS language assumes a set of *communication actions* to represent emitted signals or received ones. The special symbol τ is used to model any, unobservable to the environment, *internal action*. Assuming a set $\mathcal{L}$ of *communication labels*, $Act = \mathcal{L} \cup \overline{\mathcal{L}}$ is the set of visible actions (*i.e.,* , actions and

co-actions), and $Act_\tau = Act \cup \{\tau\}$ denotes the full set of possible communication actions. We let a, b to range over Act_τ. In summary the syntax of the language defining all CCS processes is the following grammar:

$$P, Q ::= \mathbf{0} \quad | \quad a.P \quad | \quad P + Q \quad | \quad P \parallel Q \quad | \quad P \setminus \mathcal{L} \quad | \quad P[f] \quad | \quad A$$

Informally $\mathbf{0}$ is the process that does not perform any action; $a.P$ is the action prefix operator; $P + Q$ can, non deterministically, choose between the behaviour the process P or that of the process Q; $\parallel$ is the parallel operator; $P \setminus \mathcal{L}$ is the action restriction operator, meaning that the labels $a \in \mathcal{L}$ can only be performed within a communication. $P[f]$ is the re-labelling operator, renaming each label a into $f(a)$. Finally we assume that every process identifier A has a defining equation of the form $A \stackrel{\text{def}}{=} P$. The semantics of CCS is given operationally over labeled transition systems (see [16]), whose transition relation $\rightarrow$ defines the usual concept of derivation in one step: $P \stackrel{a}{\rightarrow} P'$ means that process P, by executing action $a \in Act_\tau$, evolves in one step in the process P', while we write $P \stackrel{a}{\rightarrow}$ to underline that P can perform action a to evolve in some process. We will write $\rightarrow_*$ the transitive and reflexive closure of $\bigcup_{a \in Act_\tau} \stackrel{a}{\rightarrow}$. In the following we will let $Der(P) = \{P' | P \rightarrow_* P'\}$, $Sort(P)$ is the set of labels used by P, and $\mathcal{E}_\mathcal{F} = \{X : Sort(X) \subseteq \mathcal{F} \cup \{\tau\}\}$, where $\mathcal{F}$ is a set of actions.

2.2 Specifying Fault Tolerant Systems

Using process algebras it is possible to provide a uniform framework for specifying fault tolerant systems. For example in [1,2], a variant of the CCS was used to model a system, its failing behavior, its recovery strategies and its fault assumptions (*i.e.,* assumption on the occurrences of faults); fault tolerant properties expressed as logic formulas are verified on a finite state model of the system through model checking.

Here we will follow a similar modeling approach, but differently from [1,2] we will not require any specific fault assumption to be explicitly described in the formal model of the system. In fact we are mainly interested in evaluating the system behaviour in a general and unspecified *faulty environment*. In other words we look at the fault tolerant analysis as the analysis of an *open system* [13] acting in an environment which is potentially able to induce any fault in the system.

In the following, when talking about formal models of fault tolerant systems, we will refer to the definitions below:

A system model is a CCS process P describing the behaviour of the system through the execution of actions. It is generally a parallel composition of sub-processes, each modeling sub-components of the system, communicating

each other. □

A failing system model is a CCS process $P_{\mathcal{F}}$, obtained by extending the process P with the possibility of executing particular external actions from a set $\mathcal{F}$ of possible *fault actions*. After each fault action, the relative *failure mode* is also specified in $P_{\mathcal{F}}$ for example, by describing the behaviour induced by the occurrence of the fault. □

A fault tolerant (candidate) system model is a CCS process $P_{\mathcal{F}}^{\#}$ obtained by adding to $P_{\mathcal{F}}$ some processes realizing error-recover strategies in accordance to some fault tolerant design strategy (*e.g.,* copies of component, triple modular redundancy, voting etc.). In general the application of fault tolerance techniques in modeling leads to a process $P_{\mathcal{F}}^{\#}$ which may be described in the CCS notation as:

$$P_{\mathcal{F}}^{\#} = (P_{\mathcal{F}}^{(1)} \parallel \cdots \parallel P_{\mathcal{F}}^{(n)} \parallel Q) \setminus \mathcal{A}$$

where (a) $P_{\mathcal{F}}^{(1)} \ldots P_{\mathcal{F}}^{(n)}$ are n copies of $P_{\mathcal{F}}$ composed together (b) Q is a process representing additional components or an *error detection* module, (c) $\mathcal{A} = \{a_1, \cdots, a_n\}$, $a_i \notin \mathcal{F}$, are the labels of synchronization actions between $P_{\mathcal{F}}^{(i)}$'s and Q. □

Occurrences of faults are induced by a *fault-injector* process $F_{\mathcal{F}}$, that causes faults to happen. It interacts with $P_{\mathcal{F}}^{\#}$ exactly through $f \in \mathcal{F}$ actions. □

Previous definitions converge into the following scenario for fault tolerance analysis, where a system model and a fault injector are composed together:

$$scenario = (P_{\mathcal{F}}{}^{\#} \parallel F_{\mathcal{F}}) \backslash \mathcal{F} \tag{2}$$

It is worth to remark that in (2) fault actions are hidden *i.e.,* restricted. This implies that $P_{\mathcal{F}}$ and $F_{\mathcal{F}}$ have to synchronize on fault events. As a consequence faults are considered internal (*i.e.,* not observable) actions of the failing systems. This means that our analysis lies at the abstraction level where what it is really observable is only the (eventually faulty) behaviour of a system, but not the single occurrence of a fault. Thus in our framework, roughly speaking, fault tolerance means that faults cannot interfere with the normal observable behavior of the system.

Note that, differently from other approaches (*e.g.,* [1]), we assume the faulty behavior is described in the system $P_{\mathcal{F}}{}^{\#}$, rather than also in the fault injector $F_{\mathcal{F}}$. This is the technical trick to encode fault tolerance analysis as non-interference analysis and, eventually, as the GNDC one.

Example 2.1 As a classic example, we will give a CCS model of a simple fault tolerant system. The basic component is a module *Battery* which naively

returns one slot of energy when it receives a request. For sake of conciseness we will write our example in a value passing style CCS, which is known to be a shortcut for CCS [16]. The actions `get` and `ret` are used to communicate module inputs and outputs:

$$Battery \stackrel{\text{def}}{=} \texttt{get}.\overline{\texttt{ret}}(1).Battery$$

In modeling the corresponding failing module we assume that the battery, after having received a request, may crash and non-deterministically produce either a valid energy slot (1) or an invalid energy slot (0):

$$\begin{aligned} &Battery_f \stackrel{\text{def}}{=} \texttt{get}.\\ &(\overline{\texttt{ret}}(1).Battery_f\\ &+f.\textstyle\sum_{x\in\{0,1\}}(\overline{\texttt{ret}}(x).Battery_f)) \end{aligned}$$

In accordance to our modeling framework, the fault is assumed to be caused by a special fault action f triggered by the environment. A fault tolerant battery may be, for example, designed by using redundancy with two batteries, with the additional modules of a splitter *Splitter*, and a controller *Controller*. We require the splitter to deliver the energy request to each of the two redundant instances, respectively $Battery^{(1)}$ and $Battery^{(2)}$, while we require the collector to avoid an overloaded production of energy by absorbing the eventually additional slot. Formally, the two instances of the battery, and their faulty version are, for $i \in \{1, 2\}$:

$$Battery^{(i)} \stackrel{\text{def}}{=} Battery[\texttt{get}_i/\texttt{get}, \texttt{ret}_i/\texttt{ret}]$$

$$Battery_f^{(i)} \stackrel{\text{def}}{=} Battery_f[\texttt{get}_i/\texttt{get}, \texttt{ret}_i/\texttt{ret}]$$

The splitter and the controller are, for example, the following processes:

$$Splitter \stackrel{\text{def}}{=} \texttt{get}.\overline{\texttt{get}}_1.\overline{\texttt{get}}_2.\texttt{ack}.Splitter$$

$$\begin{aligned} Controller \stackrel{\text{def}}{=}\ & \texttt{ret}_1(x_1).\texttt{ret}_2(x_2).\textbf{if } x_1 \textbf{ then } \overline{\texttt{ret}}(x_1).\overline{\texttt{ack}}.Controller\\ & \textbf{else } \overline{\texttt{ret}}(x_2).\overline{\texttt{ack}}.Controller \end{aligned}$$

Note the action `ack` required for synchronizing the splitter and the controller. A resulting fault tolerant model of the battery ${Battery_f}^{\#}$, may be obtained, for example, by inserting a single faulty module in the redundant solution (see also Figure 1):

$$\begin{aligned} {Battery_f}^{\#} \stackrel{\text{def}}{=} (&Splitter \parallel Battery^1 \parallel\\ &Battery_f^2 \parallel Controller) \setminus \{\texttt{get}_1, \texttt{get}_2\texttt{ret}_1, \texttt{ret}_2, \texttt{ack}\} \end{aligned}$$

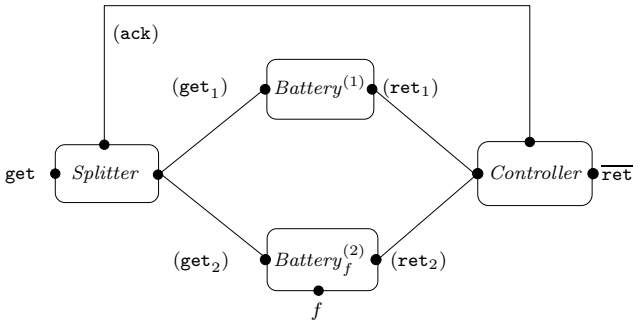

Fig. 1. The flow diagram of $Battery_f{}^{\#}$. Internal action are in brackets.

The resulting scenario for the analysis of fault tolerance is the following:

$$scenario = (Battery_f{}^{\#} \parallel F_f) \setminus \{f\}$$

3 Fault Tolerance Characterization in *GNDC*

This section reports *GNDC* basic ideas and explains how fault tolerance properties can be expressed within the *GNDC* scheme. From Section 1 we already know that a *GNDC* property has the following general form:

$$P \text{ satisfies } GNDC_{\triangleleft}^{\alpha(P)} \text{ iff } \forall X \in Env : (P \parallel X) \triangleleft \alpha(P)$$

This scheme is general enough to implement a wide class of security property definitions. In fact for example, more specific security properties such as the Bisimulation-based Non Deducibility on Compositions (*BNDC*), and the Trace Equivalence-based Non Deducibility on Compositions [3], can be subsumed as *GNDC* properties.

A first step required for instantiating *GNDC* in fault tolerance is to specify what P and Env are in this context. The former is a process $P_{\mathcal{F}}^{\#}$ obtained following the uniform modeling framework introduced in Section 2. The latter is the environment of all fault injectors (*e.g.,* our fault assumption models), that is the set of *CCS* processes whose alphabet of actions is in $\mathcal{F}$. Formally $Env = \mathcal{E}_{\mathcal{F}} = \{X \mid Sort(X) \subseteq \mathcal{F} \cup \{\tau\}\}$. Then the general *GNDC* scheme we propose for fault tolerance, is then the following:

Definition 3.1 [Fault Tolerance Characterisation in *GNDC*]

$$P \text{ satisfies } GNDC_{\triangleleft}^{\alpha(P)} \text{ iff } \forall F_{\mathcal{F}} \in \mathcal{E}_{\mathcal{F}} : (P_{\mathcal{F}}^{\#} \parallel F_{\mathcal{F}}) \setminus \mathcal{F} \triangleleft \alpha(P_{\mathcal{F}}^{\#}) \qquad (3)$$

Let us observe that the clear separation between the system model and the environment made in (2) of Section 2 will allow us to leave $F_{\mathcal{F}}$ unspecified

and to range it over $\mathcal{E}_{\mathcal{F}}$.

Moreover the instantiation of *GNDC* in fault tolerance requires $\alpha(P_{\mathcal{F}})$ to be suitable in expressing some basic property of fault tolerance (*e.g.,* fail safe, fail silent, fail stop). Finally we are interested in understanding what families of equivalences $\triangleleft$ can be suitable for the analysis of such properties. This will be argument of the following sections, where we will write P instead of $P^{\#}_{\mathcal{F}}$ when no explicit reference to the specification framework of Section 2 is required.

4 Fault Tolerance Properties as Instances of *GNDC*

In this section we provide intuitions about how to express some interesting fault tolerance property within the *GNDC* scheme. Generally speaking properties are modeled via a modified version $\alpha(P)$, of P, representing the expected behavior P. For example if we assume a fail stop property, $\alpha(P)$ has to express a modified behavior of P in which any occurrence of a fault necessarily brings to a system stop. We assume that if P is a finite state processes, $\alpha(P)$ must be finite state as well.

Fail Stop A model of a system $P^{\#}_{\mathcal{F}}$ is expected to be fail stop when in case of fault it switches in a stop state, eventually after having performed some internal actions. Intuitively $\alpha(P^{\#}_{\mathcal{F}})$ is the process whose *expanded expression* (*i.e.,* the tree obtained by unfolding the process term by applying the expansion law [16]) is obtained by the one of $P_{\mathcal{F}}$ by substituting every fault action (and relative sub-tree), with a silent action followed by the process 0. For example, referring to our example, we have that:

$$\alpha_{stop}(Battery_f) \stackrel{\text{def}}{=} \mathtt{get}.(\overline{\mathtt{ret}}(1).Battery_f + \tau.0)$$

Fail Safe A model of a system $P^{\#}_{\mathcal{F}}$ is expected to be fail safe if in case of fault it switches in a safe state. In this case $\alpha(P^{\#}_{\mathcal{F}})$ is the process whose expanded expression is obtained from the one of $P_{\mathcal{F}}$ by substituting every fault action (and relative sub-tree) with a silent action followed by some "safe behaviour" (*e.g.,* a shutdown). For example we have that:

$$\alpha_{safe}(Battery_f) \stackrel{\text{def}}{=} \mathtt{get}.(\overline{\mathtt{ret}}(1).Battery_f + \tau.P_{safe})$$

where P_{safe} models a particular "safe" behavior.

Fail Silent A model of a system $P^{\#}_{\mathcal{F}}$ is expected to be fail silent if a fault is ignored. In this case the expanded expression of $\alpha(P^{\#}_{\mathcal{F}})$ is the one obtained from the one of $P_{\mathcal{F}}$ after having substituted every fault action (and relative sub-tree) with a silent action followed by the subtree modeling the non-failing behavior. In other words every fault occurrences and any successive

failing behaviors are replace by a silent action. For example we have:

$$\alpha_{silent}(P) \stackrel{\text{def}}{=} \texttt{get}.(\overline{\texttt{ret}}(1).Battery_f + \tau.Battery_f)$$

Fault tolerance A model of a system $P^{\#}_{\mathcal{F}}$ is expected to be fault tolerant if its behaviour is observationally equal to the behaviour of the module that does not fail at all. In this case then $\alpha_{ft}(P^{\#}_{\mathcal{F}}) = P^{\#}_{\mathcal{F}} \setminus \mathcal{F}$, that is the fault tolerant systems that can never execute any fault action.

5 Fault Tolerance Analysis Strategies in *GNDC*

This section provides deeper details about the analysis of fault tolerance in the *GNDC* scheme.

5.1 Weak Bisimulation in GNDC for Fault Tolerance

We can use *weak bisimulation* to verify a system model $P^{\#}_{\mathcal{F}}$ to be fault tolerant, w.r.t. the fault tolerant properties previously described in term of $\alpha(P^{\#}_{\mathcal{F}})$. In fact, in case an external observer cannot see τ actions, a natural way of considering that two processes are equivalent is abstracting these actions from their behaviour while preserving the branching structure of the processes. Let us consider the following relation between CCS terms: $P \stackrel{\tau}{\Longrightarrow} P'$ if $P \stackrel{\tau}{\longrightarrow}_* P'$, and $P \stackrel{\alpha}{\Longrightarrow} P'$ if $P \stackrel{\tau}{\longrightarrow}_* \stackrel{\alpha}{\longrightarrow} \stackrel{\tau}{\longrightarrow}_* P'$. We recall from [17] the definition of *weak bisimulation* (*i.e.,* observational) between processes requiring that two bisimilar processes are able to simulate each other step by step:

Definition 5.1 A *symmetric* relation $\mathcal{B}$ on $\mathcal{E} \times \mathcal{E}$ is a weak bisimulation if for each $(P, Q) \in \mathcal{B}$ and for each $a \in Act_\tau$:

$$\text{if } P \stackrel{a}{\longrightarrow} P' \text{ then } \exists Q' : Q \stackrel{a}{\Longrightarrow} Q' \text{ and } (P', Q') \in \mathcal{B}$$

Two processes P and Q are bisimilar (written $P \approx_{bisim} Q$) if a bisimulation $\mathcal{B}$ exists such that $(P, Q) \in \mathcal{B}$.

Weak bisimulation resorts to be useful to detect most of the safety properties we have so far defined, namely fail safe, fail silent, fail stop and fault tolerance (expressed by, resp., $\alpha_{stop}, \alpha_{silent}, \alpha_{safe}$ and α_{ft}). Some liveness properties (*e.g.,* deadlock freedom) may be caught too.

Moreover let us observe that the *GNDC*'s instance where $\triangleleft$ is $\approx_{bisim}$, and where $\alpha(P)$ is $\alpha_{ft}(P^{\#}_{\mathcal{F}}) = P^{\#}_{\mathcal{F}} \setminus \mathcal{F}$ (*i.e.,* exactly the "fault-tolerance" property), is the *BNDC* re-stated in our fault tolerance framework. In fact

$$P \text{ satisfies } GNDC^{\alpha_{ft}(P^{\#}_{\mathcal{F}})}_{\approx_{bisim}} \text{ iff } \forall F_{\mathcal{F}} \in \mathcal{E}_{\mathcal{F}} : (P^{\#}_{\mathcal{F}} \parallel F_{\mathcal{F}}) \setminus \mathcal{F} \approx_{bisim} P^{\#}_{\mathcal{F}} \setminus \mathcal{F} \quad (4)$$

and we remind the following

Definition 5.2 [[3]] P is *BNDC* iff $\forall X \in \mathcal{E}_{\mathcal{F}} : (P \parallel_{\mathcal{F}} X) \approx_{bisim} P \setminus \mathcal{F}$

Proposition 5.3 ([6]) *P is BNDC if and only if P is $GNDC^{P\setminus\mathcal{F}}_{\approx_{bisim}}$.*

As a final observation let us discuss that *BNDC* is not compositional w.r.t. parallel composition (see [12]), that is from $P, P' \in BNDC$ it cannot be deduced that $P \parallel P' \in BNDC$. This could be an undesirable properties in analysis. Anyway there are bisimulation based-equivalences that are compositional (*e.g.,* see *SBSNNI* in [3]) that implied *BNDC*, so that they could be used to prove sufficient condition for fault tolerance, and the formulation of fault-tolerance given in (4) results attractive from this point of view.

The use of *SBSNNI* is not the only way to obtain compositionality in analysis. Alternative strategies for it may be obtained using different observational equivalence than weak bisimulation relation in (3) as we are explaining in the Section 5.2.2.

5.2 Other observational relations in GNDC for Fault Tolerance

Weak bisimulation resorts to be useful to detect most of the properties defined so far. Anyway let observe that in a practical situation we expect that many systems will be fault tolerant when weaker condition are satisfied. Generally it may be not a problem that one can deduce the existence of faults, so long the system behaviour response to those faults is "good enough". For example, the definition of fault tolerance given in (4) seems too strong and prevents the observer to deduce that *any* faults have occurred. Out of deadlock detection, for the all safety properties defined in this paper the ability to distinguish the branching structures is not required or necessary, or even dangerous: in fact safety properties, that informally can be read [11] like "nothing bad can happen", require to be satisfied independently the (branching) path conducting to a fault. This makes us to resort to weak form of observational equivalences, such as *trace equivalence* and *simulation* that will also have, within *GNDC* theory, positive effect on compositionality and on avoiding the universal quantification of fault injectors over the faulty environment. In the following we will write $P \parallel_{\mathcal{F}} Q$ as a shortcut for $(P \parallel Q) \setminus \mathcal{F}$, and we will refer to a generic $\alpha(\_)$ function. Obviously the results will also hold for all $\alpha(\_)$'s of our.

5.2.1 Most General (Faulty) Environment

The possibility of avoiding universal quantifier in expression (3) is based on the following theory about pre-congruences. We will say that $\lhd$ is a *pre-congruence* w.r.t. $\parallel_{\mathcal{F}}$ if for every $P, P', X, \in \mathcal{E}_{\mathcal{F}}$ if $P \lhd P'$ then $P \parallel_{\mathcal{F}} X \lhd P' \parallel_{\mathcal{F}} X$. The

following results hold for pre-congruences:

Proposition 5.4 ([6]) *Let be $\lhd$ a pre-congruence w.r.t. $\|_{\mathcal{F}}$. If there exists a process $Top \in \mathcal{E}_{\mathcal{F}}$ such that for every process $X \in \mathcal{E}_{\mathcal{F}}$ we have $X \lhd Top$, then:*

$$P \in GNDC_{\lhd}^{\alpha(P)} \text{ iff } (P \|_{\mathcal{F}} Top) \lhd \alpha(P)$$

In particular, if the hypothesis of the proposition above holds then it is sufficient to check that $\alpha(P)$ is satisfied when P is composed with the *most general environment*, *Top*. In our fault tolerance analysis context it would permit to make only one single check, in order to prove that a fault tolerance property holds in every fault scenario. We have also the following corollary for the congruence induced by $\lhd$:

Corollary 5.5 ([6]) *Let $\lhd$ to be a pre-congruence w.r.t. $\|_{\mathcal{F}}$ and let $\bowtie$ be defined as $\lhd \cap \lhd^{-1}$. If there exist two processes $Bot, Top \in \mathcal{E}_{\mathcal{F}}$ such that for every process $X \in \mathcal{E}_{\mathcal{F}}$ we have $Bot \lhd X \lhd Top$ then*

$$P \in GNDC_{\bowtie}^{\alpha} \text{ iff } (P \|_{\mathcal{F}} Bot) \bowtie (P \|_{\mathcal{F}} Top) \bowtie \alpha(P)$$

We show that whenever we are interested in properties based on the notion of trace equivalence, Proposition 5.4 and Corollary 5.5 hold. Let $\boldsymbol{a} = a_1 \dots a_n \in Act^*$ be a sequence of actions. We write $P \stackrel{\boldsymbol{a}}{\Longrightarrow} P'$ if and only if there exists $P_1, \dots, P_n \in \mathcal{E}$ such that $P \stackrel{a_1}{\Longrightarrow} P_1 \stackrel{a_2}{\Longrightarrow} \dots \stackrel{a_n}{\Longrightarrow} P_n$. Let be $T(P) = \{\boldsymbol{a} \in Act^* : \exists P', P \stackrel{\boldsymbol{a}}{\Longrightarrow} P'\}$ the set of traces associated to a process P. We have:

Definition 5.6 Let P and Q be two processes. Moreover Q can execute all the traces of P (written $P \leq_{trace} Q$) if and only if $T(P) \subseteq T(Q)$. P and Q are said *trace equivalent* (written $P \approx_{trace} Q$) if and only if $T(P) = T(Q)$.

It is easy to prove that $\leq_{trace}$ is a pre-congruence with respect to the CCS operators.

Proposition 5.7 *$\leq_{trace}$ is a pre-congruence w.r.t. $\|_{\mathcal{F}}$*

Proof. By reasoning of transition rules structures. □

In addition we can prove the existence of the most general (failing) environment, and provide its description. Let us consider the following process:

$$Top^{\mathcal{F}} = \sum_{f \in \mathcal{F}} \mathtt{f}.Top^{\mathcal{F}} + \overline{\mathtt{f}}.Top^{\mathcal{F}} \tag{5}$$

It is easy to demonstrate that:

Proposition 5.8 *If $X \in \mathcal{E}_{\mathcal{F}}$ then $X \leq_{trace} Top^{\mathcal{F}}$.*

Proof. By induction on the length of the traces generated by X. □

So we have proved that there exists a most general environment with respect to $\leq_{trace}$. A similar conclusion can be obtained when the following *simulation* relation is considered:

Definition 5.9 [[18]] Let $\mathcal{S}$ a binary relation on $\mathcal{E} \times \mathcal{E}$. Then $\mathcal{S}$ is said a *simulation* if for each $(P, Q) \in \mathcal{S}$ and for each $a \in Act_{\tau}$, if $P \xrightarrow{a} P'$ then there exists Q' such that $Q \stackrel{a}{\Longrightarrow} Q'$ and $(P', Q') \in \mathcal{S}$.

We write $Q \leq_{sim} P$ if there exists a simulation $\mathcal{S}$ such that $(P, Q) \in \mathcal{S}$. It is easy to prove that $\leq_{sim}$ is a pre-congruence with respect to the CCS operators, admitting the same most general environment in (5).

Proposition 5.10 *The following results hold*

(i) *$\leq_{sim}$ is a pre-congruence w.r.t. $\|_{\mathcal{F}}$*
(ii) *if $X \in \mathcal{E}_{\mathcal{F}}$ then $X \leq_{sim} Top^{\mathcal{F}}$.*

Proof. (1) by reasoning of transition rules structures; (2) by induction on the length of the traces generated by X. □

As a conclusion, when $\leq_{trace}$ and $\leq_{sim}$ are used as process relations, to check that P satisfies *GNDC* properties can be carried out only against the "most general (faulty) environment", for example by running some existing tool for non-interference [12].

5.2.2 Compositional Analysis of Fault Tolerance

This section illustrates that, when $\leq_{trace}$ and $\leq_{sim}$ are used as process equivalences in our analysis scheme, compositional proof rules for establishing that a system enjoys *GNDC* can be applied. Compositionality is a desirable property in verification to infer a global fault tolerance exploiting local fault tolerance results. Let us show it with a simple example, obtained by extending example 2.1 with the following processes

$$Torch \stackrel{\text{def}}{=} \overline{\texttt{get}}.\texttt{ret}(x).[\textbf{if } x = 1 \textbf{ then } \overline{\texttt{flash}} \textbf{ else } \overline{\texttt{no\_flash}}].\mathbf{0}$$

$$S \stackrel{\text{def}}{=} (Torch \parallel {Battery_f}^{\#}) \setminus \{\texttt{get}, \texttt{ret}\}$$

representing the behaviour of a flashing torch *Torch* using the fault tolerant energizer of Example 2.1. The energizer is expected to furnish one units of energy, even in case of fault. The flashing torch *Torch* emits a $\overline{\texttt{flash}}$ action whenever it receives exactly one unit of energy, $\overline{\texttt{no\_flash}}$ otherwise. What

an observer watching at the system S, obtained by composing the torch and the energizer, expects is to see only $\overline{\texttt{flash}}$ actions. (Recall that the system ${Battery_f}^{\#}$ provides only $\overline{\texttt{ret}}(1)$.) This safety properties can be formalized as:

$$S \in GNDC^{\alpha(S)}_{\leq_{sim}} \text{ iff } \forall F_f \in \mathcal{E}_f : S \parallel_f F_f \leq_{sim} \alpha(S) \stackrel{\text{def}}{=} \overline{\texttt{flash}}.\mathbf{0}$$

where the $\leq_{sim}$ relation has been used. In this case the expected behaviour (given through $\alpha(S)$) is that one unit of energy is furnished (and so one $\overline{\texttt{flash}}$ is observed). It is easy to convince us that the given specification of the system enjoys our safety property. Let us now consider a system S^n obtained by composing (in parallel) n instances of the system S and a similar safety property, on the S^n, that reflects the question "only at most n flashes are observed". In our scheme this is equivalent to prove that:

$$S^n \in GNDC^{\alpha(S)^n}_{\leq_{sim}} \text{ iff } \forall F_f \in \mathcal{E}_f : S^n \parallel_f F_f \leq_{sim} \alpha(S)^n \stackrel{\text{def}}{=} \underbrace{\alpha(S) \parallel \ldots \parallel \alpha(S)}_{n}$$

Compositionality would made the previous statement true, for any fixed n, without the needs of any additional check. In the following we prove that it is really the case when $\leq_{trace}$ or $\leq_{sim}$ are used. The following results hold:

Proposition 5.11 *Let P_1 and P_2 be two processes such that $P_i \in GNDC^{\alpha(P_i)}_{\leq_{trace}}$ for $i = 1, 2$. Then*

- $P_1 \parallel P_2 \in GNDC^{\alpha(P_1)\|\alpha(P_2)}_{\leq_{trace}}$
- $P_1 \parallel P_2 \in GNDC^{\alpha(P_1)\|\alpha(P_2)}_{\leq_{sim}}$

Proof. Exploiting the existence of the most general environment *Top*, and the fact that $\leq_{trace}$ (resp.$\leq_{sim}$) is a pre-congruence. □

A final discussion about the applicability of compositionality must be done. We have affirmed that using the results of this section, global fault tolerance can be deduced from local fault tolerance. Here with local fault tolerance we intend the property enjoyed by the formal specifications of sub-systems which are required to be fault tolerant by their own. With global fault tolerance we intend the property enjoyed by the specification of a system which is obtained by the composition of such sub-systems *without* the adjoint of any other global modules, such as a voter. Obviously we do not expect compositionality hold in such cases. This discussion would merit to be better developed, but being far from the scope of this paper we leave it for future works.

6 Conclusions

In this paper we have shown how to formalize fault tolerance within the *GNDC* framework *i.e.,* a general scheme for expressing security properties in a process algebra context.

Once characterized as *GNDC* property, fault tolerance can benefit of various of theoretical results and analysis techniques. Those results and strategies are peculiar of the research background in security analysis. Specifically when either a trace or a simulation relations are used to characterized fault tolerance properties, the *GNDC* theory assures efficient analysis procedures to exist. Specifically it benefits both of a static characterization (which gets rid off the universal quantification over all the possible fault scenarios), and of compositionality proofs. Another advantage of our uniform characterization is the possibility to compare dependability properties as done for the security ones in the *GNDC* framework. Potentially, this could be a preliminary step towards a formal and uniform taxonomy of dependability and security properties.

Anyway the general contribution of this paper aims to be wider. In fact by this work we intend to sustain the trend that watches at security as a research field whose applications in dependability analysis are still emerging. This is with greatest reason true if one considers the great deal of results around security recently appeared in the literature. We leave as a future work to extend our study over the characterization of other dependability properties.

Acknowledgements. All the authors thank anonymous referees for their useful suggestions that have contributed to improve the overall presentation of the work. Gnesi and Lenzini are partially supported by the MIUR-CNR project SP4 and MIUR project MEFISTO. Martinelli is supported by MIUR project "Tools, techniques and methodologies for the information society" and MIUR project MEFISTO.

References

[1] C. Bernardeschi, A. Fantechi, and S. Gnesi. Model checking fault tolerant systems. *Software Testing, Verification and Reliability*, 12:1–25, November 2002.

[2] C. Bernardeschi, A. Fantechi, and L. Simoncini. Formally verifying fault tolerant system designs. *The Computer Journal*, 3(43):191–205, 2000.

[3] R. Focardi and R. Gorrieri. A classification of security properties. *Journal of Computer Security*, 3(1):5–33, 1995.

[4] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Proc. of the 27th Colloquium in Automata, Languages and Programming*, pages 354–372, 2000.

[5] R. Focardi, R. Gorrieri, and F. Martinelli. Classification of security properties – part ii: Network security. In R. Gorrieri and R. Focardi, editors, *Foundations of Security Analysis and Design II*, volume 2946 of *LNCS*, pages 139–185. Springer-Verlag, 2004. FOSAD 2001/2002 Tutorial Lectures.

[6] R. Focardi and F. Martinelli. A uniform approach for the definition of security properties. In *Proc. of Congress on Formal Methods (FM'99)*, volume 1708 of *Lectures Notes in Computer Science*, pages 794–813. Springer-Verlag, 1999.

[7] S. N. Foley. External consistency and the verification of security protocols. In *Proc. of the 6th International Workshop on Security of Protocols*, volume 1550 of *Lecture Notes in Computer Science*, pages 28–33. Springer-Verlag, 1998.

[8] J. Goguen and J. Meseguer. Security policy and security models. In *Proc. of IEEE Symposium in Security and Privacy*. IEEE Computer Society Press, 1978.

[9] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, NJ, 1985.

[10] E. Johnsson, L. Stromberg, and S. Lindskog. On the functional relation between security and dependability impairment. In *Proc. of the New Security Paradigms Workshop*, Ontario, Canada, 1999.

[11] L. Lamport. What good is temporal logic. In R. E. Manson, editor, *Proc. of the IFIP Congress on Information Processing*, pages 657–668, Amsterdam, 1983. North Holland.

[12] F. Martinelli. Partial model checking and theorem proving for ensuring security properties. In *Proc. of the 11th IEEE Computer Security Foundation Workshop*, pages 44–52. IEEE, Computer Society, 1998.

[13] F. Martinelli. Analysis of security protocols as open systems. *Theoretical Computer Science*, 1(290):1057–1106, 2003.

[14] C. Meadows. Applying the dependability paradigm to computer security. In *Proc. of the New Security Paradigms Workshop*, pages 75–81, 1996.

[15] C. Meadows and J. McLean. Security and dependability: Then and now. In *Proc. of the Workshop on Computer Security, Dependability, and Assurance: From Needs to Solutions*, pages 166–70. Los Alamitos, CA, USA : IEEE Comput. Soc, 1999, 1998.

[16] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[17] R. Milner. Operational and algebraic semantics of concurrent processes. In J. van Leewen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 19, pages 1201–1242. The MIT Press, New York, N.Y., 1990.

[18] R. Milner. *Communication and Mobile systems: the π-calculus*. Cambridge University Press, 1999.

[19] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100(1):1–77, 1992.

[20] A. W. Roscoe, J. P. C. Wookcock, and L. Wulf. Non-interference through determinism. *Journal of Computer Security*, 1(4):27–54, 1996.

[21] J. Rushby. Critical system properties: Survey and taxonomy. *Reliability Engineering and System Safety*, 43(2):189–219, 1994.

[22] J. Rushby. Partitioning for safety and security: Requirements, mechanisms, and assurance. Technical Report CR-1999-209347, NASA Langley Research Center, June 1999.

[23] J. Rushby. Security, safety and partitioning. In *Proc. of the International Symposium on Software Security (ISSS 2003), Tokyo, Japan, November 4-6, 2003*, 2003. to appear in Lecture Notes on Computer Science.

[24] A. Simpson, J. Woodcock, and J. Davis. Safety through security. In *Proc. of the 9th International Workshop on Software Specification and Design, April 16-18, 1998, Ise-Shima (Isobe), Japan*, pages 18–23. IEEE Computer Society, 1998.

[25] D. G. Weber. Formal specification of fault tolerance and its relation to computer security. *ACM SIGSOFT Software Engineering Notes*, 14(3):273–277, 1989.