Stepwise Transformations for Fault-Tolerant Design of CCS Processes *

Tomasz Janowski †
Department of Computer Science, University of Warwick,
Coventry CV4 7AL, United Kingdom

This paper provides an approach to the formal design of distributed, fault-tolerant processes, using the language of CCS and the theory of bisimulations. The novel feature of the method is the language by which hypotheses about faults can be specified and also combined. The development of a fault-tolerant process, under a fault hypothesis, makes use of the structure of this hypothesis. This allows to first design a process which does not tolerate any faults and then to stepwise transform this process to tolerate an increasing variety of faults. We illustrate this approach designing a protocol which ensures a reliable transmission for weak assumptions about the faults of the underlying medium.

1. INTRODUCTION

It is widely believed that a retrospective proof of correctness, for programs of realistic size, is infeasible in practice. Much effort is thus spent to ensure that proof systems support the stepwise development of programs, allowing to reason about them compositionally: a program is correct if only its components can be proven correct. The resulting program is then correct by construction. However, the absence of (software) design faults does not guarantee that the program will behave properly in practice. There is a class of faults that is not due to the wrong design decisions but to the malfunction of hardware components upon which the program relies. Such faults are often due to the physical phenomena and only occur under specific environmental conditions e.g. electromagnetic perturbation. Because they do not exist before a system is put into practice, they cannot be removed beforehand but have to be tolerated.

Clearly, it is not possible to tolerate arbitrary faults. Deciding which faults are anticipated (and thus should be tolerated) and which are not (such faults are catastrophical) is the role of the fault hypothesis. The task is then to design a program which behaves properly in the presence of the anticipated hardware faults. A feasible approach to this task is to separate concerns about the correctness of the program in the absence of faults (the functional correctness) and in the presence of the anticipated faults (fault-tolerance). This can be done in two stages:

1. Designing a program which is correct but fault-intolerant.
2. Transforming this program into the one which is correct and fault-tolerant.

*To appear in the Proceedings of the Seventh International Conference on Formal Description Techniques, Berne, Switzerland, October 1994, Chapman & Hall.
†Supported by the University of Warwick, under its Scholarship Scheme for East Europe, and by an Overseas Students Award from the Committee for Vice-Chancellors and Principals.

However, while the standard techniques (e.g. the stepwise development) can be used to support the first development stage, the author is not aware of any technique to handle complexity of the second stage. This complexity stems from the fact that the more faults are anticipated, the more difficult is the task to ensure that all these faults are tolerated. The lack of the appropriate support can be seen as a major obstacle in the formal development of highly-reliable, fault-tolerant systems.

This paper presents a development method for concurrent processes, using the language of CCS [1], which attempts to fill this gap. The idea is to use a language where hypotheses Ψ about faults can be specified and also combined. The development of a fault-tolerant process, under a given fault hypothesis, makes use of the structure of this hypothesis. This allows to first design a process which does not tolerate any faults and then to stepwise transform this process to tolerate an increasing variety of faults.

We use the standard semantic model for concurrent process description languages, the labelled transition system $(\mathcal{P}, \mathcal{A}, \longrightarrow)$ where $\mathcal{P}$ is the set of processes, $\mathcal{A}$ is the set of actions and $\longrightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ is the labelled transition relation. In order to verify correctness in the presence of the specified faults, we need to represent the effect of these faults on the behaviour of processes. We do this first semantically, providing transition relation $\underset{\Psi}{\longmapsto} \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ for Ψ-affected processes Q, and then syntactically, applying process transformation $\mathcal{T}(Q, \Psi)$ [2, 3]. We show that both methods coincide, i.e. that the semantics of Q in the Ψ-affected environment (performing transitions $\underset{\Psi}{\longmapsto}$) is 'the same' as the semantics of the transformed process $\mathcal{T}(Q, \Psi)$ in the fault-free environment (performing transitions $\longrightarrow$). This allows to reason about fault-tolerance using the standard verification techniques e.g. applying bisimulation equivalence [1, 4]:

$$Q \text{ is the } \Psi\text{-tolerant implementation of } P\text{, according } \approx\text{, iff } P \approx \mathcal{T}(Q, \Psi). \tag{1}$$

For multiple faults Ψ and Φ we provide transition relation $\underset{\Psi,\Phi}{\longmapsto}$ and show that it coincides with transition relation $\underset{\Psi \oplus \Phi}{\longmapsto}$ for the combined fault $\Psi \oplus \Phi$. The same is the case when we compare transformations $\mathcal{T}(\mathcal{T}(Q, \Phi), \Psi)$ and $\mathcal{T}(Q, \Psi \oplus \Phi)$. As a result, if $P \approx \mathcal{T}(Q, \Psi \oplus \Phi)$ then Q is an implementation of P which tolerates (simultaneous) faults Ψ and Φ. This enables to develop such Q in two steps:

$$P \approx \mathcal{T}(\mathcal{R}_1(P), \Psi) \approx \mathcal{T}(\mathcal{R}_2(\mathcal{R}_1(P)), \Psi \oplus \Phi) \tag{2}$$

where $\mathcal{R}_1$ and $\mathcal{R}_2$ are fault-tolerant transformations which may employ various techniques to detect (e.g. coding of data), confine (e.g. atomic actions) and recover (e.g. backward and forward recovery) from erroneous states of P. Thus P is transformed to tolerate $\Psi \oplus \Phi$ in stages, one for each sub-hypothesis Ψ and Φ.

The rest of this paper is as follows. Both specification languages, of processes and of faults are defined in Sections 2 and 3. The verification theory, based on bisimulations and transformations of processes is introduced in Section 4. In Section 5 we present the development method for fault-tolerance by the stepwise transformations of processes and exemplify this method designing a protocol which ensures a reliable transmission for weak assumptions about the faults of the underlying medium. Finally, in Section 6, we draw some conclusions and comment on the directions for future work.

2. SPECIFICATION: PROCESSES

In this section we present the language for describing concurrent processes. The language is a version of CCS which is slightly modified for our purposes. It is given the structured operational semantics [5] in terms of the labelled transition relation $\longrightarrow$. We proceed describing first finite then recursive and finally value-passing processes.

2.1. Finite Processes

The language $\mathcal{E}$ of finite processes is based on a set $\mathcal{A}$ of actions. We have $\tau \in \mathcal{A}$ where τ is internal and represents the outcome of a joint activity (interaction) between two processes. The set $\mathcal{L} =_{def} \mathcal{A} - \{\tau\}$ is partitioned between actions a and their complements $\overline{a}$ where the function $\overline{\cdot}$ is bijective and such that $\overline{\overline{a}} = a$. We extend $\overline{\cdot}$ into $\mathcal{A}$ by allowing $\overline{\tau} = \tau$. Let $\alpha \in \mathcal{A}$, $L \subseteq \mathcal{L}$ and $f : \mathcal{A} \rightarrow \mathcal{A}$ where $f(\tau) = \tau$, $f(a) \neq \tau$ and $f(\overline{a}) = \overline{f(a)}$. $\mathcal{E}$, ranged over by E, is defined by the grammar:

$$E \quad ::= \quad 0 \;\mid\; \alpha.E \;\mid\; E + E \;\mid\; E|E \;\mid\; E\backslash L \;\mid\; E[f] \tag{3}$$

Informally, 0 denotes a process which is incapable of any actions. $\alpha.E$ is a process which performs action α and then behaves like E. $E+F$ behaves either like E or like F where the choice may be nondeterministic. $E|F$ is the parallel composition of E and F where E and F can proceed independently but also synchronise on complementary actions (performing action τ). $E\backslash L$ performs all actions of E except actions in L and their complements. Finally, $E[f]$ behaves like E with all actions a renamed into $f(a)$.

The formal semantics of $E \in \mathcal{E}$ is defined by induction on the structure of E, in terms of the labelled transition relation $\longrightarrow \;\subseteq \mathcal{E} \times \mathcal{A} \times \mathcal{E}$. If $(E, \alpha, E') \in \;\longrightarrow$ then we write $E \xrightarrow{\alpha} E'$ and say that E performs α and evolves into E' (we also use $E \xrightarrow{s} E'$ for $s \in \mathcal{A}^*$). Following [1], $\longrightarrow$ is defined as the least set which satisfies all inference rules in Figure 1.

$$\frac{}{\alpha.E \xrightarrow{\alpha} E} \qquad \frac{E \xrightarrow{\alpha} E'}{E+F \xrightarrow{\alpha} E'} \qquad \frac{F \xrightarrow{\alpha} F'}{E+F \xrightarrow{\alpha} F'}$$

$$\frac{E \xrightarrow{\alpha} E'}{E|F \xrightarrow{\alpha} E'|F} \qquad \frac{F \xrightarrow{\alpha} F'}{E|F \xrightarrow{\alpha} E|F'}$$

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{\overline{a}} F'}{E|F \xrightarrow{\tau} E'|F'}$$

$$\frac{E \xrightarrow{\alpha} E'}{E\backslash L \xrightarrow{\alpha} E'\backslash L}, \quad \alpha, \overline{\alpha} \notin L \qquad \frac{E \xrightarrow{\alpha} E'}{E[f] \xrightarrow{f(\alpha)} E'[f]}$$

Figure 1. Operational semantics of finite processes $\mathcal{E}$

We use one derived operator, $E \frown F$, where E and F proceed in parallel with actions *out* of E and *in* of F 'joined' and restricted (*mid* is not used by E or F):

$$E \frown F =_{def} (E[mid/out]|F[mid/in])\backslash\{mid\} \tag{4}$$

2.2. Recursion

It is most common to model hardware devices by cyclic, possibly nonterminating processes. In order to specify such processes consider a set $\mathcal{X}$ of process identifiers and, with slight abuse of notation, a new grammar for $\mathcal{E}$ which extends (3) by $X \in \mathcal{X}$:

$$E \quad ::= \quad \ldots \mid X \tag{5}$$

We call such E a process expression and use $\mathcal{X}(E) \subseteq \mathcal{X}$ for the set of identifiers occurring in E. We also use $E\{Y/X\}$ for process expression E where all identifiers X are replaced by Y. The semantics of E is well-defined by $\longrightarrow$ (Figure 1). Such a relation however treats X like 0, as incapable of any actions. In order to interpret X, we will use declarations of the form $X \mathrel{\widehat{=}} E$. If $X \in \mathcal{X}(E)$ then such X is defined by recursion. Given $X \mathrel{\widehat{=}} E$ and $Y \mathrel{\widehat{=}} F$ where $X \in \mathcal{X}(F)$ and $Y \in \mathcal{X}(E)$, such X and Y are defined by the mutual recursion. In the sequel we will often need to manipulate declarations for mutually recursive identifiers. Then, it will be helpful to use a simple language $\mathcal{D}$ for specifying collections of such declarations. $\mathcal{D}$, ranged over by Δ and ∇, is defined by the following grammar:

$$\Delta \quad ::= \quad [\,] \ \mid \ \Delta[X \mathrel{\widehat{=}} E] \ \mid \ \Delta \oplus \nabla \tag{6}$$

Informally, $[\,]$ is an empty declaration and X is defined as E in $\Delta[X \mathrel{\widehat{=}} E]$ and as $E + F$ in $(\Delta[X \mathrel{\widehat{=}} E]) \oplus (\nabla[X \mathrel{\widehat{=}} F])$. Formally, Δ is assigned a denotation $[\![\Delta]\!]$ which is a partial function from $\mathcal{X}$ to $\mathcal{E}$. We use $dom(\Delta)$ and $ran(\Delta)$ for the domain and the range of $[\![\Delta]\!]$ respectively, and we define $[\![\Delta]\!]$ in Figure 2, by induction on the structure of Δ.

$$
\begin{array}{lll}
dom([\,]) & =_{def} & \emptyset \\[2ex]
[\![\Delta[Y \mathrel{\widehat{=}} E]]\!](X) & =_{def} & \left\{ \begin{array}{ll} E & \text{if } X = Y \\ [\![\Delta]\!](X) & \text{if } X \neq Y,\ X \in dom(\Delta) \end{array} \right. \\[3ex]
[\![\Delta \oplus \nabla]\!](X) & =_{def} & \left\{ \begin{array}{ll} [\![\Delta]\!](X) & \text{if } X \in dom(\Delta) - dom(\nabla) \\ [\![\Delta]\!](X) + [\![\nabla]\!](X) & \text{if } X \in dom(\Delta) \cap dom(\nabla) \\ [\![\nabla]\!](X) & \text{if } X \in dom(\nabla) - dom(\Delta) \end{array} \right.
\end{array}
$$

Figure 2. Denotational semantics of declarations $\mathcal{D}$.

We say that the declaration Δ is closed if all identifiers of right-hand side expressions of Δ are interpreted by Δ: $\bigcup_{E \in ran(\Delta)} \mathcal{X}(E) \subseteq dom(\Delta)$. A useful abbreviation is to take $[X \mathrel{\widehat{=}} E, Y \mathrel{\widehat{=}} F]$ instead of $[\,][X \mathrel{\widehat{=}} E][Y \mathrel{\widehat{=}} F]$ and $[X \mathrel{\widehat{=}} E \mid p]$ for all definitions $X \mathrel{\widehat{=}} E$ such that the predicate p holds.

A process $P \in \mathcal{P}$ is finally the pair $\langle E, \Delta \rangle$ of the process expression E and the closed declaration Δ for all identifiers of E: $\mathcal{X}(E) \subseteq dom(\Delta)$. The semantics of $\langle E, \Delta \rangle$ is defined, with slight abuse of notation, by transition relation $\longrightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ which is the least set which satisfies all inference rules in Figure 3. It is easy to see that Δ persists through the transitions of $\langle E, \Delta \rangle$ i.e. if $\langle E, \Delta \rangle \xrightarrow{\alpha} P'$ then there exists $E' \in \mathcal{E}$ such that $P' \equiv \langle E', \Delta \rangle$ ($\equiv$ is the syntactic identity).

$$\frac{\langle E_i, \Delta\rangle \xrightarrow{\alpha_i} \langle E_i', \Delta\rangle, \quad i \in I}{\langle E, \Delta\rangle \xrightarrow{\alpha} \langle E', \Delta\rangle} \quad \text{for} \quad \frac{E_i \xrightarrow{\alpha_i} E_i', \quad i \in I}{E \xrightarrow{\alpha} E'} \quad \text{in Figure 1}$$

$$\frac{\langle [\![\Delta]\!](X), \Delta\rangle \xrightarrow{\alpha} \langle E, \Delta\rangle}{\langle X, \Delta\rangle \xrightarrow{\alpha} \langle E, \Delta\rangle} \quad \text{for} \quad X \in dom(\Delta)$$

Figure 3. Operational semantics of processes $\mathcal{P}$.

Example 1 Consider $n \in \mathbb{N} \cup \{\infty\}$ and let R_n be a process which performs actions a and b; the first at any time; the second no more than n times in a row (R_0 can never perform b and R_∞ can perform b at any time). We have:

$$\begin{array}{lll} R_n & =_{def} & \langle X_0, \Delta \oplus \nabla\rangle \\ \text{where } \Delta & =_{def} & [X_i \mathrel{\widehat{=}} a.X_0 \quad | \quad 0 \leq i \leq n] \\ \nabla & =_{def} & [X_i \mathrel{\widehat{=}} b.X_{i+1} \quad | \quad 0 \leq i < n] \end{array}$$

□

Operators (3) will be also used to combine processes, under syntactic restriction that processes involved have disjoint sets of identifiers. If $dom(\Delta) \cap dom(\nabla) = \emptyset$ then:

$$\begin{array}{lll} \alpha.\langle E, \Delta\rangle & =_{def} & \langle \alpha.E, \Delta\rangle \\ \langle E, \Delta\rangle \backslash L & =_{def} & \langle E\backslash L, \Delta\rangle \\ \langle E, \Delta\rangle[f] & =_{def} & \langle E[f], \Delta\rangle \\ \langle E, \Delta\rangle + \langle F, \nabla\rangle & =_{def} & \langle E + F, \Delta \oplus \nabla\rangle \\ \langle E, \Delta\rangle \mid \langle F, \nabla\rangle & =_{def} & \langle E \mid F, \Delta \oplus \nabla\rangle \end{array} \tag{7}$$

2.3. Value-Passing

In the language defined so far, processes interact by synchronising on complementary actions a and $\overline{a}$. There is no directionality or value which passes between them. In practice however, we may find it convenient to use a value-passing language for the set V of values (we assume, for simplicity, that V is finite).

To this end we introduce value constants, value variables x, value and boolean expressions e and p built using constants, variables and any function symbols we need. These include ε as the empty sequence; $\sharp s$ as the length of the sequence s; s_0, its first element; s', all but the first element; $s : x$, the sequence s with x appended. We also introduce parameters into process identifiers: $X(e1, .., en)$ for X of arity n. Then we extend $\mathcal{E}$ by input prefixes $a(x).E$, output prefixes $\overline{a}(e).E$ and by conditionals if p then E else F. The semantics of the resulting language is defined by translation into the basic one [1].

Example 2 Consider a buffer of capacity $m > 0$, Buf_m, which receives (by action in) and subsequently transmits (by action $\overline{out}$) all values x unchanged, in the same order and with at most m of them received but not sent. We have:

$$\begin{array}{lll} Buf_m & =_{def} & \langle X(\varepsilon), \Delta \oplus \nabla\rangle \\ \text{where } \Delta & =_{def} & [X(s) \mathrel{\widehat{=}} in(x).X(s : x) \quad | \quad 0 \leq \sharp s < m] \\ \nabla & =_{def} & [X(s) \mathrel{\widehat{=}} \overline{out}(s_0).X(s') \quad | \quad 0 < \sharp s \leq m] \end{array}$$

□

3. SPECIFICATION: FAULTS

We treat transition relation $\longrightarrow$ as the semantics of processes in the idealised, fault-free environment. The primary effect of faults however is that processes no longer behave according to $\longrightarrow$. We use $\longmapsto$ to model the fault-affected semantics and assume that $\longmapsto \supseteq \longrightarrow$. In the first part of this section we show how to specify such relations. The idea is to use process identifiers as 'states' which can be potentially affected by faults. Each process $\langle E, \Delta\rangle$ provides its own declaration Δ for process identifiers. In contrast to this, 'normal' declaration, we specify faults by an alternative, 'faulty' declaration Ψ.

For certain faults, fault-tolerance cannot be ensured in full. In these circumstances, we may be satisfied with its conditional version, for certain assumptions about the quantity of faults. Even when the 'full' fault-tolerance is (in theory) possible, we may choose its conditional version to first design a process for restricted assumptions about faults and then to stepwise transform this process to ensure fault-tolerance for more relaxed assumptions. For these reasons, it is important that in addition to possible 'faulty' transitions $\dashrightarrow \;=_{def}\; \longmapsto - \longrightarrow$, we can also specify assumptions about the quantity of such transitions. Such assumptions are introduced in the second part of this section.

3.1. Qualitative Assumptions

In order to specify faults we will use declarations $\mathcal{D}$. As defined in Figure 3, transitions of the process $\langle X, \Delta\rangle$ are determined by the transitions of $\langle [\![\Delta]\!](X), \Delta\rangle$ where $[\![\Delta]\!](X)$ is the process expression assigned to X by Δ. Consider $\Psi \in \mathcal{D}$ which assigns yet another expression $[\![\Psi]\!](X)$ to X. Such Ψ specifies the following transitions $\underset{\Psi}{\longmapsto}$ of $\langle X, \Delta\rangle$:

if $\langle [\![\Psi]\!](X), \Delta\rangle \overset{\alpha}{\underset{\Psi}{\longmapsto}} \langle E, \Delta\rangle$ then $\langle X, \Delta\rangle \overset{\alpha}{\underset{\Psi}{\longmapsto}} \langle E, \Delta\rangle$.

Ψ is not assumed to be closed. Instead, we assume that all process identifiers in the right-side expressions of Ψ are declared in Δ, so that $\underset{\Psi}{\longmapsto}$ does not lead from the well-defined process to the ill-defined one: $\bigcup_{F \in ran(\Psi)} \mathcal{X}(F) \subseteq dom(\Delta)$. We use $\mathcal{P}_\Psi$ for the set of all such $\langle E, \Delta\rangle$. Given Ψ which specifies anticipated faults, we can define a new, Ψ-affected semantics of processes, in terms of transition relation $\underset{\Psi}{\longmapsto} \subseteq \mathcal{P}_\Psi \times \mathcal{A} \times \mathcal{P}_\Psi$ which is the least set which satisfies inference rules in Figure 4. If $(P, \alpha, P') \in \underset{\Psi}{\longmapsto}$ then we write $P \overset{\alpha}{\underset{\Psi}{\longmapsto}} P'$ (we also use $P \overset{s}{\underset{\Psi}{\longmapsto}} P'$ for $s \in \mathcal{A}^*$).

$$\frac{\langle E_i, \Delta\rangle \overset{\alpha_i}{\underset{\Psi}{\longmapsto}} \langle E_i', \Delta\rangle, \quad i \in I}{\langle E, \Delta\rangle \overset{\alpha}{\underset{\Psi}{\longmapsto}} \langle E', \Delta\rangle} \quad \text{for} \quad \frac{E_i \overset{\alpha_i}{\longrightarrow} E_i', \quad i \in I}{E \overset{\alpha}{\longrightarrow} E'} \quad \text{in Figure 1}$$

$$\frac{\langle [\![\Delta]\!](X), \Delta\rangle \overset{\alpha}{\underset{\Psi}{\longmapsto}} \langle E, \Delta\rangle}{\langle X, \Delta\rangle \overset{\alpha}{\underset{\Psi}{\longmapsto}} \langle E, \Delta\rangle} \quad \text{for} \quad X \in dom(\Delta)$$

$$\frac{\langle [\![\Psi]\!](X), \Delta\rangle \overset{\alpha}{\underset{\Psi}{\longmapsto}} \langle E, \Delta\rangle}{\langle X, \Delta\rangle \overset{\alpha}{\underset{\Psi}{\longmapsto}} \langle E, \Delta\rangle} \quad \text{for} \quad X \in dom(\Psi)$$

Figure 4. Operational semantics of processes $\mathcal{P}$ affected by Ψ.

Observe that each of the three inference rules in Figure 4 determines a set of rules: one for each rule in Figure 1 and one for each process identifier $X \in dom(\Delta)$ and $X \in dom(\Psi)$ respectively. It is easy to see that $\longrightarrow \subseteq \underset{\Psi}{\longmapsto}$. We will write:

$$P \overset{\alpha}{\underset{\Psi}{\dashrightarrow}} P' \quad \text{iff} \quad P \overset{\alpha}{\underset{\Psi}{\longmapsto}} P' \quad \text{and} \quad P \overset{\alpha}{\not\longrightarrow} P'$$

Example 3 Consider the following declarations which specify various communication faults of the bounded buffer Buf_m, creation (Ψ_e), corruption (Ψ_c), omission (Ψ_o), replication (Ψ_r) and permutation (Ψ_p) of messages:

$$\begin{array}{lll}
\Psi_e & =_{def} & [X(s) \mathrel{\widehat{=}} \tau.\overline{out}(\surd).X(s) \quad \mid \; 0 \leq \sharp s \leq m] \\
\Psi_c & =_{def} & [X(s) \mathrel{\widehat{=}} \tau.\overline{out}(\surd).X(s') \quad \mid \; 0 < \sharp s \leq m] \\
\Psi_o & =_{def} & [X(s) \mathrel{\widehat{=}} \tau.X(s') \quad \mid \; 0 < \sharp s \leq m] \\
\Psi_r & =_{def} & [X(s) \mathrel{\widehat{=}} \tau.\overline{out}(s_0).X(s) \quad \mid \; 0 < \sharp s \leq m] \\
\Psi_p & =_{def} & [X(s) \mathrel{\widehat{=}} \tau.\overline{out}((s')_0).X(s_0 : s'') \quad \mid \; 1 < \sharp s \leq m]
\end{array}$$

Because we are not interested in the particular value of the corrupted or created messages, we represent them all by the same, distinguished message $\surd$. An implicit assumption is that corruption and creation can be easily detected (although not easily distinguished). We also assume that when permuted, only one message is delayed. $\square$

Consider assumptions Ψ and Φ about faults which affect the semantics of $\langle X, \Delta\rangle$. Suppose that $X \in dom(\Psi) \cap dom(\Phi)$. In addition to the 'normal' transitions of (X, Δ), following transitions of $\langle [\![\Delta]\!](X), \Delta\rangle$, two kinds of faulty transitions are also possible for $\langle X, \Delta\rangle$, according to the 'faulty' declarations $[\![\Psi]\!](X)$ and $[\![\Phi]\!](X)$ of X. We use transition relation $\underset{\Psi,\Phi}{\longmapsto}$ for the semantics of $\mathcal{P}$ affected by both Ψ and Φ (provided all process identifiers in right-side expressions of Ψ and Φ are declared). Such a relation is defined by inference rules in Figure 5.

$$\frac{\langle E_i, \Delta\rangle \overset{\alpha_i}{\underset{\Psi,\Phi}{\longmapsto}} \langle E_i', \Delta\rangle, \quad i \in I}{\langle E, \Delta\rangle \overset{\alpha}{\underset{\Psi,\Phi}{\longmapsto}} \langle E', \Delta\rangle} \quad \text{for} \quad \frac{E_i \overset{\alpha_i}{\longrightarrow} E_i', \quad i \in I}{E \overset{\alpha}{\longrightarrow} E'} \quad \text{in Figure 1}$$

$$\frac{\langle [\![\Delta]\!](X), \Delta\rangle \overset{\alpha}{\underset{\Psi,\Phi}{\longmapsto}} \langle E, \Delta\rangle}{\langle X, \Delta\rangle \overset{\alpha}{\underset{\Psi,\Phi}{\longmapsto}} \langle E, \Delta\rangle} \quad \text{for} \quad X \in dom(\Delta)$$

$$\frac{\langle [\![\Psi]\!](X), \Delta\rangle \overset{\alpha}{\underset{\Psi,\Phi}{\longmapsto}} \langle E, \Delta\rangle}{\langle X, \Delta\rangle \overset{\alpha}{\underset{\Psi,\Phi}{\longmapsto}} \langle E, \Delta\rangle} \quad \text{for} \quad X \in dom(\Psi)$$

$$\frac{\langle [\![\Phi]\!](X), \Delta\rangle \overset{\alpha}{\underset{\Psi,\Phi}{\longmapsto}} \langle E, \Delta\rangle}{\langle X, \Delta\rangle \overset{\alpha}{\underset{\Psi,\Phi}{\longmapsto}} \langle E, \Delta\rangle} \quad \text{for} \quad X \in dom(\Phi)$$

Figure 5. Operational semantics of processes $\mathcal{P}$ affected by both Ψ and Φ.

It is easy to show that the joint semantic effect of faults Ψ and Φ (on processes $\mathcal{P}$) is the same as the effect of the combined fault $\Psi \oplus \Phi$:

Proposition 1
If $\Psi, \Phi \in \mathcal{D}$ and $Q \in \mathcal{P}_{\Psi \oplus \Phi}$
then for all $\alpha \in \mathcal{A}$ we have: $Q \overset{\alpha}{\underset{\Psi,\Phi}{\mapsto}} Q'$ iff $Q \overset{\alpha}{\underset{\Psi \oplus \Phi}{\mapsto}} Q'$.

The proof proceeds by induction on the inference of transitions $Q \overset{\alpha}{\underset{\Psi,\Phi}{\mapsto}} Q'$ and $Q \overset{\alpha}{\underset{\Psi \oplus \Phi}{\mapsto}} Q'$.

Example 4 According to Proposition 1, the joint effect of the three communication faults, creation, omission and permutation of messages, can be specified as $\Psi_e \oplus \Psi_o \oplus \Psi_p$. Following denotational semantics of $\mathcal{D}$ in Figure 2, $\Psi_e \oplus \Psi_o \oplus \Psi_p$ equals:

$$
\begin{array}{l}
[X(s) \mathrel{\hat{=}} \tau.\overline{out}(\surd).X(s) \mid \sharp s = 0] \\
[X(s) \mathrel{\hat{=}} \tau.\overline{out}(\surd).X(s) + \tau.X(s') \mid \sharp s = 1] \\
[X(s) \mathrel{\hat{=}} \tau.\overline{out}(\surd).X(s) + \tau.X(s') + \tau.\overline{out}((s')_0).X(s_0 : s'') \mid 1 < \sharp s \leq m]
\end{array}
\qquad \square
$$

3.2. Quantitative Assumptions

Consider $\Psi \in \mathcal{D}$ and suppose that transitions $\underset{\Psi}{\mapsto}$ are assigned types: $\longrightarrow$ having type 0 and $\underset{\Psi}{\dashrightarrow}$ having type 1. In order to specify the quantity of faulty transitions we will use sets $H \subseteq \{0,1\}^*$ of all admissible sequences of transition types. Because H is intended to constrain transitions $\underset{\Psi}{\dashrightarrow}$ only, we assume that $\varepsilon \in H$ and if $h \in H$ then $h : 0 \in H$ (H is closed with respect to the concatenation of 0).

Example 5 Suppose that $n \in \mathbb{N} \cup \{\infty\}$ denotes the maximal number of times transitions $\underset{\Psi}{\dashrightarrow}$ can occur successively (if $n = \infty$ then they can occur at any time; if $n = 0$ then not at all). The set H_n, of all admissible sequences of 0's and 1's, under assumption n, equals:

$$
H_n =_{def} \begin{cases} \{0,1\}^* & \text{if } \; n = \infty \\ \{0,1\}^* - \{h : 1^{n+1} \mid h \in \{0,1\}^*\} & \text{if } \; n \in [0, \infty) \end{cases} \qquad \square
$$

Recall that $Q \overset{s}{\underset{\Psi}{\mapsto}} Q'$ if Q evolves into Q' performing the sequence s of transitions $\longrightarrow$ and $\underset{\Psi}{\dashrightarrow}$. This may be no longer the case if transitions $\underset{\Psi}{\dashrightarrow}$ can only occur under assumption H about admissible sequences of transition types. When this is the case then we use the family $\{ \underset{\Psi}{\mapsto}{}^{h}_{h'} \}_{h,h' \in H}$ of transition relations $\underset{\Psi}{\mapsto}{}^{h}_{h'} \subseteq \mathcal{P}_\Psi \times \mathcal{A}^* \times \mathcal{P}_\Psi$. If $(Q, s, Q') \in \underset{\Psi}{\mapsto}{}^{h}_{h'}$ then we write $Q \overset{s}{\underset{\Psi}{\mapsto}}{}^{h}_{h'} Q'$ and say that Q evolves into Q' by the sequence $s \in \mathcal{A}^*$ of transitions $\underset{\Psi}{\mapsto}$, under assumption H and given that h is the history of transition types before and h' after transition. Formally, $\underset{\Psi}{\mapsto}{}^{h}_{h'}$ is defined by the following inductive rules:

$$
\begin{array}{lll}
Q \overset{\varepsilon}{\underset{\Psi}{\mapsto}}{}^{h}_{h} Q & & \\
Q \overset{\alpha:s}{\underset{\Psi}{\mapsto}}{}^{h}_{h'} Q'' & \text{iff} \;\; \exists_{Q'} & Q \overset{\alpha}{\longrightarrow} Q' \overset{s}{\underset{\Psi}{\mapsto}}{}^{h:0}_{h'} Q'' \;\; \vee \\
& & Q \overset{\alpha}{\underset{\Psi}{\dashrightarrow}} Q' \overset{s}{\underset{\Psi}{\mapsto}}{}^{h:1}_{h'} Q'' \;\; \wedge \;\; h : 1 \in H
\end{array}
\tag{8}
$$

Observe that the induction is well-defined: the first rule provides the base, for the empty sequence ε, and the second rule decreases the length of the action sequence by one. The family $\{ \underset{\Psi}{\mapsto}{}^{h}_{h'} \}_{h,h' \in H}$ denotes the semantics of processes $\mathcal{P}_\Psi$ when they are affected by Ψ and under assumption H about the quantity of transitions $\underset{\Psi}{\dashrightarrow}$.

4. VERIFICATION: BISIMULATION AND TRANSFORMATIONS

Fault-tolerance is a crucial property for safety-critical systems. Such a system is said to tolerate anticipated faults when its behaviour is 'correct' in an operating environment which contains these faults. As such, fault tolerance depends on the chosen notion of correctness. For verifying correctness we provide the choice of the three relations: trace equivalence, bisimulation equivalence and bisimulation preorder. They give rise to different notions of fault-tolerance which is verified using transformations $\mathcal{T}(\cdot, \Psi)$ to model effects of faults on the semantics of processes. Transformations are also used to verify conditional fault-tolerance, under assumption H about the quantity of faults.

4.1. Functional Correctness

Consider the 'weak' transition relation $\Longrightarrow\ \subseteq \mathcal{P} \times (\mathcal{L} \cup \{\varepsilon\}) \times \mathcal{P}$ where transitions $\stackrel{\tau}{\longrightarrow}$ are ignored and let β range over $\mathcal{L} \cup \{\varepsilon\}$. We define $\stackrel{\varepsilon}{\Longrightarrow}$ as the reflexive and transitive closure of $\stackrel{\tau}{\longrightarrow}$ and $\stackrel{a}{\Longrightarrow}$ as the composition $\stackrel{\varepsilon}{\Longrightarrow} \stackrel{a}{\longrightarrow} \stackrel{\varepsilon}{\Longrightarrow}$ of relations. If $(P, \beta, P') \in\ \Longrightarrow$ then we write $P \stackrel{\beta}{\Longrightarrow} P'$. We also write $P \stackrel{s}{\Longrightarrow}$, $s \in \mathcal{L}^*$, if there exists P' such that $P \stackrel{s}{\Longrightarrow} P'$.

Trace equivalence, denoted $\approx_1$, identifies a process with all sequences of (observable) actions that it can perform, like in the standard automata theory:

$$P \approx_1 Q \quad \text{iff} \quad \text{for all } s \in \mathcal{L}^* \;\; P \stackrel{s}{\Longrightarrow} \text{ iff } Q \stackrel{s}{\Longrightarrow} \tag{9}$$

Examples are processes P, Q, R and S in Figure 6 which can all perform the same sequences of actions a and b. Thus $P \approx_1 Q \approx_1 R \approx_1 S$. Trace equivalence however admits a linear-time approach to process executions, ignoring at which execution stages which choices are made. For example $P \approx_1 Q$ but action b is always possible for P and not always for Q. As a result, in the environment which continually demands b, P will always meet this demand and Q will sometimes deadlock. Thus $\approx_1$ is insensitive to deadlock.

This is not the case for bisimulation equivalence, $\approx$ [1, 4], which is defined in terms of relations B such that if $(P, Q) \in B$ then for all $\beta \in \mathcal{L} \cup \{\varepsilon\}$:

$$\begin{array}{llll} \text{whenever} & P \stackrel{\beta}{\Longrightarrow} P' & \text{then} & \exists_{Q'} \;\; Q \stackrel{\beta}{\Longrightarrow} Q' \;\wedge\; (P', Q') \in B \\ \text{whenever} & Q \stackrel{\beta}{\Longrightarrow} Q' & \text{then} & \exists_{P'} \;\; P \stackrel{\beta}{\Longrightarrow} P' \;\wedge\; (P', Q') \in B \end{array} \tag{10}$$

Such B is called a bisimulation and we have $P \approx Q$ iff there exists a bisimulation B which contains the pair (P, Q). For processes in Figure 6 we have $P \not\approx Q$ because $Q \stackrel{\varepsilon}{\Longrightarrow} Q'$ and $P \stackrel{\varepsilon}{\Longrightarrow} P$ only, however $P \stackrel{b}{\Longrightarrow} P'$ but $Q' \;\not\stackrel{b}{\Longrightarrow}$. Also $P \approx R \approx S$ because $\{(P, R), (P', R'), (P', R'')\}$ and $\{(P, S), (P', S'), (P', S'')\}$ are bisimulations. However, only S can engage (after a or b) in the infinite sequence of actions τ so bisimulation equivalence is insensitive to divergence.

The closest to $\approx$, divergence-sensitive relation is bisimulation preorder, $\sqsubseteq$ [6, 7]. Informally, $P \sqsubseteq Q$ iff P and Q are bisimulation equivalent, except perhaps when P diverges, and Q diverges no more than P does. Thus Q is at least as 'good' as P: whenever P converges, Q must converge as well, however if P diverges then Q need not diverge. Consider the predicate $\downarrow\ \subseteq \mathcal{P}$ where if $P \in\ \downarrow$ then we write $P \downarrow$ and say that the process P converges. Based on $\downarrow$, we define $P \Downarrow \beta$ if there is no process P' such that $P \stackrel{\beta}{\Longrightarrow} P'$ and

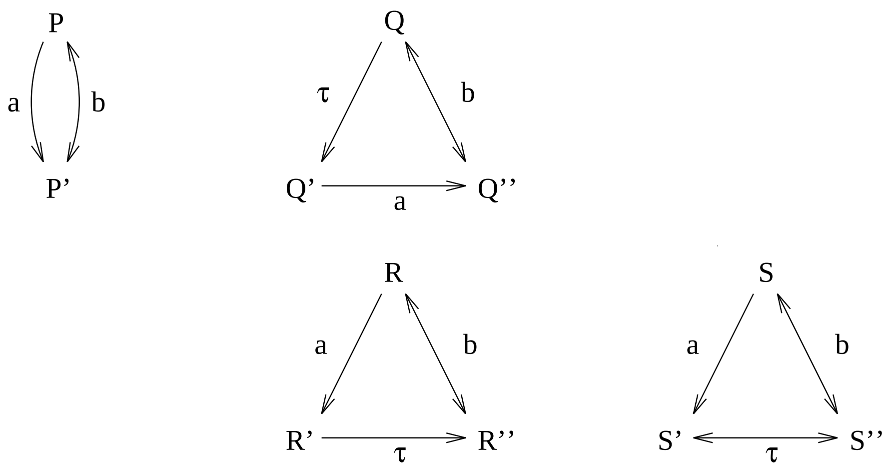

Figure 6. Transition diagrams of P, Q, R and S.

which performs action τ indefinitely:

$$\begin{array}{llll} P \Downarrow \varepsilon & \text{iff} \;\; P\downarrow & \text{and whenever} \;\; P \xrightarrow{\tau} P' & \text{then} \;\; P' \Downarrow \varepsilon \\ P \Downarrow a & \text{iff} \;\; P \Downarrow \varepsilon & \text{and whenever} \;\; P \stackrel{a}{\Longrightarrow} P' & \text{then} \;\; P' \Downarrow \varepsilon \end{array} \tag{11}$$

Bisimulation preorder $\sqsubseteq$ is defined in terms of relations B, called partial bisimulations, such that if $(P, Q) \in B$ then:

$$\begin{array}{llll} \text{whenever} \;\; P \stackrel{\beta}{\Longrightarrow} P' & \text{then} & \exists_{Q'} \; Q \stackrel{\beta}{\Longrightarrow} Q' \; \wedge \; (P', Q') \in B & \\ \text{whenever} \;\; P \Downarrow \beta & \text{then} & Q \Downarrow \beta \; \wedge & \\ & & \text{if} \;\; Q \stackrel{\beta}{\Longrightarrow} Q' \;\; \text{then} \;\; \exists_{P'} \; P \stackrel{\beta}{\Longrightarrow} P' \; \wedge \; (P', Q') \in B & \end{array} \tag{12}$$

We have $P \sqsubseteq Q$ iff $(P, Q) \in B$ for some partial bisimulation B. For processes in Figure 6 we have $P \sqsubseteq R$ because $\{(P, R), (P', R'), (P', R'')\}$ is a partial bisimulation and $P \not\sqsubseteq S$ because $P \Downarrow a$ and $S \not\Downarrow a$.

4.2. Fault-Tolerance

Suppose that $\preceq$ denotes any one of relations $\approx_1$, $\approx$ or $\sqsubseteq$, and consider the high-level process P. Such P determines the set of admissible implementations Q, $P \preceq Q$, where semantics of both P and Q is defined by transition relation $\longrightarrow$. The situation however is different if we want to ensure that Q is a fault-tolerant implementation of P, according to $\preceq$ and the specification Ψ of faults. Informally, such Q should behave ‘properly’ in any environment where faults are specified by Ψ. Such a Ψ-affected behaviour of Q is then defined by transition relation $\underset{\Psi}{\longmapsto}$, in contrast to P which still behaves according to $\longrightarrow$. This raises the problem of comparing two processes which behaviour is defined by different transition relations $\longrightarrow$ and $\underset{\Psi}{\longmapsto}$. In order to solve this problem consider the following transformation $\mathcal{T}$ of processes:

$$\mathcal{T}(\langle E, \Delta \rangle, \Psi) \;=_{def}\; \langle E, \Delta \oplus \Psi \rangle \tag{13}$$

It is easy to show that such $\mathcal{T}$ provides an equivalent to $\overset{}{\underset{\Psi}{\mapsto}}$, syntactic method of representing effect of faults on the behaviour of processes, i.e. we can show that:

Proposition 2
If $\Psi \in \mathcal{D}$ and $Q \in \mathcal{P}_\Psi$
then for all $\alpha \in \mathcal{A}$ we have: $Q \overset{\alpha}{\underset{\Psi}{\mapsto}} Q'$ iff $\mathcal{T}(Q,\Psi) \xrightarrow{\alpha} \mathcal{T}(Q',\Psi)$.

The proof proceeds by induction on the inference of transitions $\mathcal{T}(Q,\Psi) \xrightarrow{\alpha} \mathcal{T}(Q',\Psi)$ and $Q \overset{\alpha}{\underset{\Psi}{\mapsto}} Q'$. Thus the semantics of Q in the Ψ-affected environment is the same as the semantics of $\mathcal{T}(Q,\Psi)$ in the fault-free environment and in order to prove that Q is a Ψ-tolerant implementation of P (according to $\preceq$) it is enough to show that:

$$P \preceq \mathcal{T}(Q,\Psi) \tag{14}$$

Example 6 Consider the task to ensure a reliable communication (specified by the bounded buffer Buf_{m+1}, Example 2) over a medium of capacity m which creates messages (specified by Ψ_e, Example 3). To this end we can use a process Ret which ignores all $\surd$'s:

$$Ret \ =_{def} \ \langle Z, [Z \mathrel{\widehat{=}} in(x).\text{if } x = \surd \text{ then } Z \text{ else } \overline{out}(x).Z]\rangle$$

Given such Ret, it is easy to prove that $Buf_{m+1} \approx_1 \mathcal{T}(Buf_m \frown Ret, \Psi_e)$. However, we have $Buf_{m+1} \not\approx \mathcal{T}(Buf_m \frown Ret, \Psi_e)$ because after receiving $x \neq \surd$ and before its transmission, Ret does not accept any more $\surd$. The 'better' implementation is Re:

$$\begin{array}{lllll} Re & =_{def} & \langle Z, & [Z & \widehat{=} & in(x).\text{if } x = \surd \text{ then } Z \text{ else } Z(x)] \\ & & & [Z(x) & \widehat{=} & in(\surd).Z(x) + \overline{out}(x).Z]\rangle \end{array}$$

Then we have $Buf_{m+1} \approx \mathcal{T}(Buf_m \frown Re, \Psi_e)$ but $Buf_{m+1} \not\sqsubseteq \mathcal{T}(Buf_m \frown Re, \Psi_e)$ because $\mathcal{T}(Buf_m \frown Re, \Psi_e)$ but not Buf_{m+1} can diverge, due to arbitrary creation of messages. $\Box$

The same approach can be used to verify that Q tolerates multiple faults, say faults specified by Ψ and Φ. Applying Proposition 1 which shows that the joint effect of Ψ and Φ, $\underset{\Psi,\Phi}{\mapsto}$, is the same as the effect of the combined fault $\Psi \oplus \Phi$, $\underset{\Psi\oplus\Phi}{\mapsto}$, and Proposition 2 which allows to express $\underset{\Psi\oplus\Phi}{\mapsto}$ in terms of $\longrightarrow$ and $\mathcal{T}(\cdot, \Psi \oplus \Phi)$, it is enough to prove that:

$$P \preceq \mathcal{T}(Q, \Psi \oplus \Phi) \tag{15}$$

4.3. Conditional Fault-Tolerance

Suppose now that we want to verify $\langle E, \Delta\rangle$ in the presence of faulty transitions $\underset{\Psi}{\dashrightarrow}$ and under assumption $H \subseteq \{0,1\}^*$ about admissible transition type histories. To this end, like before, we will use process transformations. The idea is to use a family $\{X_h\}_{h\in H}$ of the process identifiers, for each identifier X of Δ or Ψ. The index h of X_h denotes the history of transition types. Consider $h \in H$ and the following transformation $\widetilde{\mathcal{T}}_h(\cdot, \Psi, H)$:

$$\widetilde{\mathcal{T}}_h(\langle E, \Delta\rangle, \Psi, H) =_{def} \mathcal{T}(\langle E_h, \Delta_H\rangle, \Psi_H) \tag{16}$$

$$\begin{array}{lll} \text{where } E_h & =_{def} & E\{X_h/X \mid X \in \mathcal{X}(E)\} \\ \Delta_H & =_{def} & [X_h \mathrel{\widehat{=}} F\{Y_{h:0}/Y \mid Y \in \mathcal{X}(F)\} \mid [\![\Delta]\!](X) = F \wedge h \in H] \\ \Psi_H & =_{def} & [X_h \mathrel{\widehat{=}} F\{Y_{h:1}/Y \mid Y \in \mathcal{X}(F)\} \mid [\![\Psi]\!](X) = F \wedge h \in H \wedge h:1 \in H] \end{array}$$

Thus E_h is obtained from E by replacing all process identifiers $X \in \mathcal{X}(E)$ by X_h; Δ_H is obtained from Δ by replacing each declaration $X \mathrel{\widehat{=}} F$ by the family of declarations $X_h \mathrel{\widehat{=}} F\{Y_{h:0}/Y \mid Y \in \mathcal{X}(F)\}$ for all $h \in H$, and Ψ_H is similar like Δ_H but only taking $h \in H$ such that $h:1 \in H$. Finally, we define $\widetilde{\mathcal{T}}(\langle E, \Delta\rangle, \Psi, H)$ as $\widetilde{\mathcal{T}}_\varepsilon(\langle E, \Delta\rangle, \Psi, H)$.

Recall that the effect of Ψ on the semantics of $\mathcal{P}_\Psi$, under assumption H about the quantity of transitions $\underset{\Psi}{\dashrightarrow}$, is defined by the family $\{\underset{\Psi}{\longmapsto}{}^{h}_{h'}\}_{h,h' \in H}$ of transition relations. There are two problems to obtain the same effect using transformations:

1. Consider $X \in dom(\Delta) \cap dom(\Psi)$ and transition $\langle X, \Delta\rangle \overset{\alpha}{\underset{\Psi}{\longmapsto}} \langle E, \Delta\rangle$ which can be either inferred from $\langle [\![\Delta]\!](X), \Delta\rangle \overset{\alpha}{\underset{\Psi}{\longmapsto}} \langle E, \Delta\rangle$ or from $\langle [\![\Psi]\!](X), \Delta\rangle \overset{\alpha}{\underset{\Psi}{\longmapsto}} \langle E, \Delta\rangle$. The problem appears when $\langle X, \Delta\rangle \overset{\alpha}{\underset{\Psi}{\longmapsto}} \langle E, \Delta\rangle$ can be inferred from both of them. Then it is regarded as 'normal' by $\underset{\Psi}{\longmapsto}{}^{h}_{h'}$ but either as 'normal' or 'faulty' by $\widetilde{\mathcal{T}}_h(\cdot, \Psi, H)$. If no such E and α exists then we say that Ψ *has the proper effect on* Δ.

2. The second problem is that in case of $\widetilde{\mathcal{T}}_h(\cdot, \Psi, H)$ (but not $\underset{\Psi}{\longmapsto}{}^{h}_{h'}$), some transitions do not contribute to the history h of transition types. Suppose that $\Psi =_{def} [X \mathrel{\widehat{=}} \tau.a.X]$ and $\Delta =_{def} [X \mathrel{\widehat{=}} b.X]$. Then we have: $\langle X, \Delta\rangle \overset{\tau}{\underset{\Psi}{\dashrightarrow}} \langle a.X, \Delta\rangle \xrightarrow{a} \langle X, \Delta\rangle \xrightarrow{b} \langle X, \Delta\rangle$ and thus $\langle X, \Delta\rangle \overset{\tau ab}{\underset{\Psi}{\longmapsto}}{}^{h}_{h:100} \langle X, \Delta\rangle$, however $\widetilde{\mathcal{T}}_h(\langle X, \Delta\rangle, \Psi, H) \xrightarrow{\tau ab} \widetilde{\mathcal{T}}_{h:10}(\langle X, \Delta\rangle, \Psi, H)$. We can solve this problem assuming that all expressions involved are *linear* i.e. they are of the form $\sum_{i=1}^{k} \alpha_i.X_i$ (Δ is linear if all $F \in ran(\Delta)$ are linear).

Under both conditions it is easy to prove the following proposition:

Proposition 3

If $\quad \Psi \in \mathcal{D}$ and $\langle X, \Delta\rangle \in \mathcal{P}_\Psi$ where
$\quad\quad \Psi$ has the proper effect on Δ and Ψ and Δ are linear
then $\quad \langle X, \Delta\rangle \overset{s}{\underset{\Psi}{\longmapsto}}{}^{h}_{h'} \langle Y, \Delta\rangle$ iff $\widetilde{\mathcal{T}}_h(\langle X, \Delta\rangle, \Psi, H) \xrightarrow{s} \widetilde{\mathcal{T}}_{h'}(\langle Y, \Delta\rangle, \Psi, H)$

The proof proceeds by induction on the length of s. Thus the 'normal' semantics of $\widetilde{\mathcal{T}}(\langle X, \Delta\rangle, \Psi, H)$ is the same as the Ψ-affected semantics of $\langle X, \Delta\rangle$, under assumption H. As a result, in order to prove that $\langle X, \Delta\rangle$ tolerates Ψ under assumption H (with respect to P and according to $\preceq$), it is enough to show that:

$$P \preceq \widetilde{\mathcal{T}}(\langle X, \Delta\rangle, \Psi, H) \tag{17}$$

Although the linear form of Δ and Ψ is necessary to prove Proposition 3, the meaning of $\widetilde{\mathcal{T}}(\langle E, \Delta\rangle, \Psi, H)$ for non-linear Δ and Ψ is also well-understood. While in the first case all transitions are significant, they all contribute to the history h, in the second case only chosen ones are significant. Thus the main reason for 'mismatch' between $\widetilde{\mathcal{T}}_h(\cdot, \Psi, H)$ and $\underset{\Psi}{\longmapsto}{}^{h}_{h'}$ for non-linear expressions lies in the restrictive form of the latter. In the sequel we will adopt (17) to verify conditional fault-tolerance for $\langle E, \Delta\rangle$ and Ψ where process expressions of Δ and Ψ are not necessarily linear.

For example we have $Buf_{m+1} \sqsubseteq \widetilde{\mathcal{T}}(Buf_m \frown Re, \Psi_e, H_n)$ for process Re (Example 6) which ignores all created messages $\surd$ and for H_n (Example 5) where $n \neq \infty$ is the bound on the number of successive occurrences of Ψ_e (Example 3).

Example 7 Consider $n, m > 0$ and the task to ensure a reliable communication (specified by Buf_{m+n+2} and according to $\sqsubseteq$) over a medium of capacity m which permutes messages. To this end we will use two processes: the sender Sp_n and the receiver Rp_n. In order to determine the proper transmission order, messages will be send by Sp_n with their sequence numbers modulo n. The value of n determines the number of parallel components St_i of Rp_n ($i = 0, \ldots, n-1$), each one used to store a message with the sequence value i, received out-of-order. The value of $\perp$ means that no message is stored. Suppose that the summation $i+1$ below is taken modulo n. Then we have:

$$
\begin{array}{lll}
Sp_n & =_{def} & \langle Zs(0), [Zs(i) \mathrel{\widehat{=}} in(x).\overline{out}(x,i).Zs(i+1) \mid 0 \le i < n]\rangle \\
Rp_n & =_{def} & (Ctr \mid St_0 \mid \cdots \mid St_{n-1}) \backslash \{st_0, \ldots, st_{n-1}\} \\
Ctr & =_{def} & \langle Zr(0),\ \ [Zr(i) \ \mathrel{\widehat{=}}\ in(x,j).\text{if } i=j \ \text{ then } \overline{out}(x).Zm(i+1) \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{else } \ \overline{st_j}(x).Zr(i) \qquad\quad \mid 0 \le i < n] \\
 & & \qquad\quad\ [Zm(i) \mathrel{\widehat{=}}\ st_i(x).\ \ \text{if } x = \perp \text{then } Zr(i) \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{else } \ \overline{out}(x).Zm(i+1) \mid 0 \le i < n]\rangle \\
St_i & =_{def} & \langle Zm_i, \quad [Zm_i \ \ \mathrel{\widehat{=}}\ \ st_i(x).\overline{st_i}(x).Zm_i + \overline{st_i}(\perp).Zm_i]\rangle,\ \ 0 \le i < n
\end{array}
$$

It can be shown that $Sp_n \frown Buf_m \frown Rp_n$ tolerates Ψ_p provided the number of successive permutations is not greater than n: $Buf_{m+n+2} \sqsubseteq \widetilde{\mathcal{T}}(Sp_n \frown Buf_m \frown Rp_n, \Psi_p, H_n)$. □

5. DEVELOPMENT: STEPWISE TRANSFORMATIONS

So far we were only concerned with how to verify that given a high-level process P, a fault hypothesis Ψ and a low-level, Ψ-affected process Q, Q is an implementation of P (according to some $\preceq$) which tolerates Ψ (perhaps under a certain assumption H about its quantity). The problem of how to design such Q has been completely ignored. This problem is the topic of the current section.

In most cases, Q can be obtained by the transformation $\mathcal{R}(P)$ of P. For example we have: $\mathcal{R}(Buf_m) = Buf_{m-1} \frown Re$ to tolerate Ψ_e according to $\approx$ (Example 6, $m > 1$) and $\mathcal{R}(Buf_m) = Sp_n \frown Buf_{m-n-2} \frown Rp_n$ to tolerate Ψ_p according to $\sqsubseteq$ (Example 7, $m > n+2$). Thus, when no assumption about the quantity of faults is made, our problem is to find a transformation $\mathcal{R}$ such that:

$$P \preceq \mathcal{T}(\mathcal{R}(P), \Psi) \tag{18}$$

To this end, $\mathcal{R}(P)$ will employ various techniques to detect, confine and to recover from erroneous states of P, by introducing some additional, recovery processes. Our task is easier when the recovery processes are assumed not to be affected by Ψ, i.e. when they do not share any process identifiers with Ψ (like Re, Sp_n and Rp_n). It this case, $\mathcal{T}(\mathcal{R}(P), \Psi)$ is identical with $\mathcal{R}(\mathcal{T}(P, \Psi))$ and it is enough to prove that:

$$P \preceq \mathcal{R}(\mathcal{T}(P, \Psi)) \tag{19}$$

For example, in order to verify $Buf_{m+1} \approx \mathcal{T}(Buf_m \frown Re, \Psi_e)$, it is enough to prove that $Buf_{m+1} \approx \mathcal{T}(Buf_m, \Psi_e) \frown Re$. It is out of scope of this paper to investigate any particular technique to design such $\mathcal{R}(P)$. Instead, when $\Psi = \Psi_1 \oplus \cdots \oplus \Psi_n$, we would like to propose that $\mathcal{R}(P)$ is obtained from P by the sequence of transformations, one for each component Ψ_i of Ψ. Then, we can expect that the task to tolerate each Ψ_i is easier than the task to tolerate them altogether. This suggests the following development process:

$$P_0 \preceq \mathcal{T}(P_1, \Psi_1) \preceq \mathcal{T}(P_2, \Psi_1 \oplus \Psi_2) \preceq \cdots \preceq \mathcal{T}(P_n, \Psi_1 \oplus \cdots \oplus \Psi_n) \tag{20}$$

where $P_0 \equiv P$ and $P_i \equiv \mathcal{R}_i(P_{i-1})$ for $i = 1 \ldots n$. Such P_i aims to tolerate Ψ_i in the presence of faults specified by $\Psi_1, \ldots, \Psi_{i-1}$ and in general depends on the recovery processes used in the earlier stages of the design. The final transformation $\mathcal{R}(P) = \mathcal{R}_n(\ldots \mathcal{R}_1(P) \ldots)$. Our task can be largely simplified if $\preceq$ is preserved by $\mathcal{R}_i$ i.e. whenever $P \preceq Q$ then $\mathcal{R}_i(P) \preceq \mathcal{R}_i(Q)$ and if recovery processes of $\mathcal{R}_i$ are not affected by $\Psi_1 \oplus \ldots \oplus \Psi_i$ i.e. $\mathcal{T}(\mathcal{R}_i(P_{i-1}), \Psi_1 \oplus \ldots \oplus \Psi_i)$ is identical with $\mathcal{R}_i(\mathcal{T}(P_{i-1}, \Psi_1 \oplus \ldots \oplus \Psi_i))$. Then, in the $i+1$-st stage, it is enough to find $\mathcal{R}_{i+1}$ which transforms P_{i-1}, not P_i:

$$\mathcal{T}(P_{i-1}, \Psi_1 \oplus \ldots \oplus \Psi_i) \preceq \mathcal{T}(\mathcal{R}_{i+1}(P_{i-1}), \Psi_1 \oplus \ldots \oplus \Psi_{i+1}) \tag{21}$$

This means that P_{i+1} does not depend on the recovery processes used in the i-th step, i.e. that both steps are independent. When $\preceq$ is preserved by all transformations $\mathcal{R}_i$ and when none of the recovery processes is affected by faults then all stages are mutually independent and the final transformation $\mathcal{R}(P) = \mathcal{R}_1(\ldots \mathcal{R}_n(P) \ldots)$.

Example 8 Consider the task to ensure a reliable communication (specified by Buf_m and according to $\approx$) over a medium of capacity m which corrupts, creates, omits and replicates messages, i.e. to find a transformation $\mathcal{R}(Buf_m)$ of Buf_m such that:

$$Buf_m \approx \mathcal{T}(\mathcal{R}(Buf_m), \Psi_e \oplus \Psi_c \oplus \Psi_o \oplus \Psi_r)$$

Applying development procedure (20) we will design such $\mathcal{R}(Buf_m)$ in two steps, by first tolerating $\Psi_o \oplus \Psi_r$ and then $\Psi_e \oplus \Psi_c$. To tolerate $\Psi_o \oplus \Psi_r$ we will use a version of the sliding window protocol with the window size m. The protocol consists of two processes, the sender So_m and the receiver Ro_m such that at most m messages are sent by So_m without being acknowledged by Ro_m (we acknowledge messages by actions ack). So_m uses s as the sequence of messages sent but not acknowledged (we have $\sharp s \leq m$) and repeatedly retransmits s_0 until acknowledgement for this message is received. Taking all arithmetic operations below modulo $m+1$, we have:

$$\begin{array}{llllll}
So_m & =_{def} & \langle Z(0,\varepsilon), & [Z(i,s) & \widehat{=}\ in(x).Z(i,s,x) & \mid 0 \leq \sharp s < m \wedge 0 \leq i \leq m]\oplus \\
& & & [Z(i,s) & \widehat{=}\ ack.Z(i,s') & \mid 0 < \sharp s \leq m \wedge 0 \leq i \leq m]\oplus \\
& & & [Z(i,s) & \widehat{=}\ \overline{out}(i-\sharp s, s_0).Z(i,s) & \mid 0 < \sharp s \leq m \wedge 0 \leq i \leq m] \\
& & & [Z(i,s,x) & \widehat{=}\ \overline{out}(i,x).Z(i+1, s:x) & \mid 0 \leq \sharp s < m \wedge 0 \leq i \leq m]\oplus \\
& & & [Z(i,s,x) & \widehat{=}\ ack.Z(i,s',x) & \mid 0 < \sharp s < m \wedge 0 \leq i \leq m]\oplus \\
& & & [Z(i,s,x) & \widehat{=}\ \overline{out}(i-1,s_0).Z(i,s,x) & \mid 0 < \sharp s < m \wedge 0 \leq i \leq m]\rangle \\
Ro_m & =_{def} & \langle Z(0), & [Z(i) & \widehat{=}\ in(j,x).\text{if } i=j & \text{then } \overline{out}(x).\overline{ack}.Z(i+1) \\
& & & & & \text{else } Z(i)]\rangle
\end{array}$$

For such So_m and Ro_m we can prove the following equivalence:

$$Buf_m \approx (So_m \frown (\mathcal{T}(Buf_m, \Psi_o \oplus \Psi_r) \frown Buf_1) \frown Ro_m)\backslash\{ack\}$$

Recall a process Re (Example 6) to tolerate Ψ_e by ignoring all messages $\surd$. If we apply Re to the medium which creates, corrupts, omits and replicates messages $(\Psi_e \oplus \Psi_c \oplus \Psi_o \oplus \Psi_r)$ then the resulting medium is only affected by the last two faults $(\Psi_o \oplus \Psi_r)$:

$$\mathcal{T}(Buf_m, \Psi_o \oplus \Psi_r) \frown Buf_1 \approx \mathcal{T}(Buf_m, \Psi_e \oplus \Psi_c \oplus \Psi_o \oplus \Psi_r) \frown Re$$

Finally, because none of the processes So_m, Ro_m or Re is affected by faults and because $\approx$ is preserved by $\frown$ and $\backslash$, we get the desired transformation:

$$\begin{aligned} \mathcal{R}(Buf_m) &= \mathcal{R}_1(\mathcal{R}_2(Buf_m)) \\ &= \mathcal{R}_1(Buf_m \frown Re) \\ &= (So_m \frown (Buf_m \frown Re) \frown Ro_m)\backslash\{ack\} \end{aligned}$$

The resulting process is a two-layered protocol where the lower layer tolerates $\Psi_e \oplus \Psi_c$ and the higher one tolerates $\Psi_o \oplus \Psi_r$. It is not possible, using only bounded sequence numbers, to extend this protocol to tolerate permutation Ψ_p [8]. □

It is often the case that recovery processes are affected by faults themselves. Even worse, that they introduce new faults (not specified by Ψ), as in case of the sliding window protocol and acknowledgements which are not exchanged by simple synchronisations but using a medium which itself may be faulty. Suppose that Φ specifies 'new' faults, introduced by recovery processes of $\mathcal{R}(P)$. In this case we have to prove that:

$$P \preceq \mathcal{T}(\mathcal{R}(P), \Psi \oplus \Phi) \qquad (22)$$

So far, we were not concerned with assumptions H about the quantity of faults. However, such assumptions can be used to support the stepwise procedure (20). To this end, we can first design a process for strong assumptions about the quantity of faults (say H) and then to stepwise transform this process to ensure fault-tolerance for increasingly relaxed assumptions (H' where $H \subseteq H'$). Such a stepwise procedure will be described elsewhere.

6. CONCLUSIONS

Currently, there is a number of methods for specifying and proving correctness of fault-tolerant systems [3, 9–15]. In this paper, we did not aim to provide yet another formalism. Our purpose was to show how the well-established theory of CCS can be extended to reason about fault-tolerance, with emphasis placed on reasoning under weak assumptions about faults. This extension includes two languages (for specifying processes and faults), verification theory based on transformations of processes (and exemplified using bisimulations and partial bisimulations) and the development approach where multiple faults are proposed to be tolerated incrementally, by stepwise transformations.

We plan to continue this work in all three aspects: specification, verification and (first of all) development of fault-tolerant processes. When the last is concerned, we plan to provide some constructive proof rules for certain, well-known techniques for fault-tolerance, e.g. backward recovery and modular redundancy. Also, to utilise assumptions about the quantity of faults for the stepwise design of fault-tolerant processes.

ACKNOWLEDGMENTS

I am grateful to my supervisor, Mathai Joseph, for many valuable comments on draft versions of this paper, to Zhiming Liu for useful discussions on fault-tolerance, and to David Walker for helpful comments and for putting some literature to my attention. Thanks are also to the referees for their comments.

REFERENCES

1. R. Milner. *Communication and Concurrency.* Prentice-Hall International, 1989.
2. Z. Liu. *Fault-Tolerant Programming by Transformations.* PhD thesis, University of Warwick, 1991.
3. Z. Liu and M. Joseph. Transformations of programs for fault-tolerance. *Formal Aspects of Computing*, 4:442–469, 1991.
4. D. Park. Concurrency and automata on infinite sequences. *LNCS*, 104, 81.
5. G. Plotkin. A structural approach to operational semantics. Technical report, Computer Science Department, Aarhus University, 81.
6. R. Milner. A modal characterisation of observable machine-behaviour. *LNCS*, 112:25–34, 81.
7. D.J. Walker. Bisimulation and divergence. *Information and Computation*, 85:202–241, 90.
8. D. Wang and L. Zuck. Tight bounds for the sequence transmission problem. In *Proc. 8th ACM Symp. on Princ. of Distributed Computing*, pages 73–83, 89.
9. F. Cristian. A rigorous approach to fault-tolerant programming. *IEEE Transactions on Software Engineering*, 11(1):23–31, 1985.
10. He Jifeng and C.A.R. Hoare. Algebraic specification and proof of a distributed recovery algorithm. *Distributed Computing*, 2:1–12, 1987.
11. J. Nordahl. *Specification and Design of Dependable Communicating Systems.* PhD thesis, Technical University of Denmark, 1992.
12. J. Peleska. Design and verification of fault tolerant systems with CSP. *Distributed Computing*, 5:95–106, 1991.
13. D. Peled and M. Joseph. A compositional approach for fault-tolerance using specification transformation. *LNCS*, 694, 1993.
14. K.V.S. Prasad. *Combinators and Bisimulation Proofs for Restartable Systems.* PhD thesis, Department of Computer Science, University of Edinburgh, 1987.
15. H. Schepers. Tracing fault-tolerance. In *Proc. 3rd IFIP Working Conference on Dependable Computing for Critical Applications.* Springer-Verlag, 1993.