Abstracting from Failure Probabilities*

Jan Jürjens†
Computing Laboratory, University of Oxford, GB
jan@comlab.ox.ac.uk – http://www.jurjens.de/jan

Abstract

In fault-tolerant computing, dependability of systems is usually demonstrated by abstracting from failure probabilities (under simplifying assumptions on failure occurrences).

In the specification framework FOCUS*, we show under which conditions and to which extent this is sound: We use a specification language that is interpreted in the usual abstract model and in a probabilistic model. We give probability bounds showing the degree of faithfulness of the abstract model wrt. the probabilistic one. These include cases where the usual assumptions are not fulfilled.*

1. Introduction

Formal methods have been substantially applied in safety-critical systems to ensure a high degree of dependability by proving correctness of the system design.

To keep reasoning feasible, formal verification of safety-critical systems often abstracts from failure probabilities by assuming that failure is masked perfectly. Failure probabilities of underlying components (such as hardware) are considered (using Markov models) in isolation from the whole system. For this one usually makes simplifying assumptions, e.g. that the failure distributions of different components are not correlated, do not change over time or with use of the component, that failures do not remain latent for a period of time to become effective only later etc.

To show under which assumptions and to what extent this approach is sound, we use a specification framework that allows one to

- obtain concrete information on the degree of dependability of the overall system, given the dependability of the used fault-tolerance components and to
- do without some of the simplifying assumptions commonly made (e.g. independence of failure probabilities from previous history).

Concrete estimates are useful since one would like to employ no more redundancy than necessary, since increased replication may lead to increased cost, decreased performance, and can even itself cause failures due to the increase in complexity [25]. This applies especially where tight constraints on hardware or performance must be met, e.g. in embedded systems.

In the framework of FOCUS [5, 6] (supported by the tool AUTOFOCUS [16]) we define a specification language (where specifications are formulated as nondeterministic programs) that is interpreted at two levels of abstraction:

- in the abstract model one may reason about the system in a simple way by treating fault-tolerance components as "black boxes",
- in the probabilistic model, the failure probabilities are retained.

We give results on the degree of faithfulness of the abstract model wrt. the probabilistic one.

A common criticism of probabilistic formal models is that in practice it is not always possible to obtain precise probabilities to incorporate into the formal model. Another new aspect of our work is that in such a case one may reason qualitatively rather than quantitatively: The dependability of a replication mechanism can be specified as a function of a *safety parameter* (e.g. the number of replication copies) and can thus be defined asympotically rather than concretely (examples are given in Section 4).

In the next subsection we provide background and point out related work. In Section 2 we present the specification framework used. Section 3 interprets it in the abstract model. In Section 4 we give definitions regarding fault-tolerance components and provide examples and results on their degree of safety. Section 5 gives the probabilistic model and Section 6 the faithfulness of the abstract model wrt. the probabilistic one. In Section 7 we give as an example the specification of a fault-tolerant unbounded

*This is the web-version 4/7/01 of a paper at the IEEE International Conference on Application of Concurrency to System Design, 2001. Please refer to http://www.jurjens.de/jan for the current version.

†This work was partially supported by the Studienstiftung des deutschen Volkes and Lucent Technologies.

buffer. Section 8 gives a fault-tolerant ring storage as a final example, after that we conclude. The appendix gives proof sketches omitted from the main body.

1.1. Fault-tolerant systems and Formal Methods

A central requirement on safety-critical systems is high dependability. Since there are failures in any operational system, fault-tolerance is used at execution time "to provide, by redundancy, service complying with the specification in spite of faults occurred or occurring" [20].

Fault-tolerance usually involves fault detection and fault masking. Fault masking may require complex protocols whose correctness can be non-obvious [25]. Other measures such as assessing damage, error recovery and fault removal are not in the focus of our current approach.

Forms of redundancy commonly employed include space redundancy (physical copies of a resource), time redundancy (rerunning functions) and information redundancy (error-correcting codes). Generally, fault-tolerance using redundancy depends on statistical independence among the failure occurrences. This is reasonable to assume for hardware component failures but less clear wrt. hardware or software design flaws [7].

Reliability goals for safety-critical systems are often expressed quantitatively via the maximum allowed failure rate. For example, critical services of the Advanced Automation System (AAS, providing Air Traffic Control services) should be unavailable at most 3 seconds a year [9]. To prevent any single catastrophic failure in any aircraft of a given type during its entire life-time one estimates that the maximum admissible failure rate for each failure condition is about 10^{-9} per hour [21, p.37]. Since 10^9 hours amounts to over 100,000 years, one may not achieve confidence that a system has such a degree of dependability just by testing.

This motivates the use of formal methods. Faced with feasability aspects (such as the state explosion problem [27]), one often abstracts from probabilities by assuming that failures are masked perfectly, in order to keep the model as simple as possible. Then probabilistic behaviour is factored out into fault-tolerance components. To evaluate their dependability a range of models has been developed (with advanced tools), such as Reliability block diagrams, Markov models and stochastic Petri Nets (for an overview cf. [13, 28]). [2] shows how to decompose a fault-tolerant program into a fault-intolerant program and a set of fault-tolerance components (probabilistic aspects are not considered).

The "relevance [of formal methods] to the actual running of the program is only as good as the degree of faithfulness to which the model represents real executions of the program" [22]. Following this, formal models have been developed that include probabilistic information, e.g. [18, 23, 14, 26, 3, 11] (cf. also work on performance evaluation in [15, 10, 4]). Here we follow the alternative (and to our knowledge new) approach of showing, within a formal model, under which assumptions and to what degree non-probabilistic modeling of safety-critical systems is faithful to reality. In particular we treat cases where usual assumptions (such as history-independence of failures) are violated (note that Markov models may also be able to treat such cases, but so far this does not seem to be accounted for in the non-probabilistic formal models from which probabilistic behaviour is factored out as described above). Even though our application here are fault-tolerant systems, the general idea should be applicable to other situations where probabilities occur. For example, [1] gives similar work regarding the formal methods treatment of cryptography.

2. Specification language

The specifications in our framework are formulated as nondeterministic, concurrently executing processes [6]. Communication is asynchronous in the sense that transmission of a value cannot be prevented by the receiver (i. e. processes are input-total in the sense of [17] – one may model synchronous communication using handshake [6]). Systems can be composed using the operator $\otimes$ defined below.

A process P is a collection of programs that communicate synchronously (in rounds) through channels, with the constraint that for each of its output channels c, P contains exactly one program p_c that outputs on c. This program p_c may take input from any of P's input channels. Intuitively, the program is a description of a value to be output on the channel c in round $n+1$, computed from values found on channels in round n (this is made precise in Section 3). Local state can be maintained through the use of feedback channels, and used for iteration (for instance, for coding *while* loops).

To be able to reason inductively on syntax, we use a simple specification language from [19, 1]. We assume disjoint sets $\mathbf{Channels}$ of channels, $\mathbf{Var}$ of variables and $\mathcal{D}$ of data values. The values communicated over channels are formal *expressions* built from the error value $\perp$, variables, values on input channels, and data values using concatenation. Precisely, the set $\mathbf{Exp}$ of expressions contains the empty expression ε and the non-empty expressions generated by the grammar

$E ::=$	expression
$\perp$	error value
x	variable ($x \in \mathbf{Var}$)
c	input value ($c \in \mathbf{Channels}$)
d	data value ($d \in \mathcal{D}$)
$E_1 :: E_2$	concatenation

An occurrence of a channel name c refers to the value found on c at the previous instant. The empty expression ε denotes absence of output on a channel at a given point in time. The error value $\perp$ signals a failure. We write $\mathbf{CExp}$ for the set of *closed* expressions (those containing no subterms in $\mathbf{Var} \cup \mathbf{Channels}$).

Definition 1 (Nondeterministic) *programs* are defined inductively by:

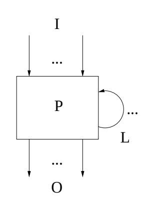
$p ::=$	programs
E	output of expression
$\mathit{either}\ p\ \mathit{or}\ p'$	nondeterminism
$\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p'$	conditional
$\mathit{case}\ E\ \mathit{of}\ x :: y\ \mathit{do}\ p\ \mathit{else}\ p'$	break up list

Here $E, E' \in \mathbf{Exp}$ are expressions. Variables are introduced in case constructs, which determine their values. The case construct tests whether E is a list with head x and tail y; if so, p is evaluated, using the actual values of x, y; if not, p' is evaluated (again this is made precise in Section 3). In the case constructs, x and y are bound variables. A program is *closed* if it contains no unbound variables.

Definition 2 Any program obtained from a program p by substituting each subconstruct $\mathit{either}\ p_1\ \mathit{or}\ p_2$ either by p_1 or by p_2 is called a *deterministic component* of p. We write $\mathcal{C}(p)$ for the set of deterministic components of the program p.

Example The program
$p \stackrel{\text{def}}{=} \mathit{if}\ E = \perp\ \mathit{then}\ (\mathit{either}\ 0\ \mathit{or}\ 1)\ \mathit{else}\ (\mathit{either}\ 0\ \mathit{or}\ 1)$
has four different deterministic components, each of the form $\mathit{if}\ E = \perp\ \mathit{then}\ i\ \mathit{else}\ j$ with $i, j \in \{0, 1\}$.

Definition 3 A *process* is of the form $P = (I, O, L, (p_c)_{c \in \tilde{O}})$ where



- $I \subseteq \mathbf{Channels}$ is the set of its input channels,
- $O \subseteq \mathbf{Channels}$ is the set of its output channels,
- $L \subseteq \mathbf{Channels}$ is the set of its local channels, and
- each p_c ($c \in \tilde{O} \stackrel{\text{def}}{=} O \cup L$) is a closed program with input channels in $\tilde{I} \stackrel{\text{def}}{=} I \cup L$. From inputs on $\tilde{I}$ at a given point in time, p_c computes the output on c at the following point in time.

We usually write I_P, O_P and L_P for the sets of input, output and local channels of the process P (which are assumed to be mutually disjoint, as well as the sets of local channels of different processes). They are used to store local state between the execution rounds which can be shared among the programs of a process. We write $\mathcal{C}(P)$ for the set of deterministic components of the process P, defined analogously to the case of programs.

Definition 4 (Data access bounds) For any program p, we define a bound $n(p)$ on the number of data accesses during one iteration of the execution of p:

- $n(E) = 1$ for an expression $E \in \mathbf{Exp}$
- $n(\mathit{either}\ p\ \mathit{or}\ p') = \max(n(p), n(p'))$
- $n(\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p') = \max(n(p), n(p')) + 2$
- $n(\mathit{case}\ E\ \mathit{of}\ x :: y\ \mathit{do}\ p\ \mathit{else}\ p') = \max(n(p), n(p'))$ $+1$

For a process $P = (I, O, L, (p_c)_{c \in \tilde{O}})$ we define $n(P) = \sum_{c \in \tilde{O}} n(p_c)$.

We note that this definition actually captures the intended meaning.

Fact 1 *Given any process P, to compute one output tuple from an input tuple, P has access to data at most $n(P)$ times.*

3. Abstract Model

We interpret processes in the abstract model of stream-processing functions, after recalling definitions on streams and stream-processing functions from [6].

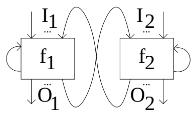
3.1. Stream-processing functions

We write $\mathbf{Stream}_C \stackrel{\text{def}}{=} (\mathbf{CExp}^{\omega})^C$ (where $C \subseteq \mathbf{Channels}$) for the set of C-indexed tuples of (finite or infinite) sequences of closed expressions, called (timed) *streams* [6]. Each stream $\vec{s} \in \mathbf{Stream}_C$ is a tuple consisting of components $\vec{s}(c)$ (for each $c \in C$) that denote the sequence of expressions appearing at the channel c at execution time. The n^{th} element in this sequence is the expression appearing at time $t = n$ (time is assumed to be discrete).

A process $P = (I, O, L, (p_c)_{c \in \tilde{O}})$ is interpreted by a total *stream-processing function* $[\![P]\!] : \mathbf{Stream}_I \rightarrow (\mathbf{Stream}_O)^{\mathcal{C}(P)}$ from input streams to families of output streams indexed by the deterministic components of P (we use families of streams instead of sets of streams since this simplifies the presentation of later definitions). Thus

$[\![P]\!](\vec{s})_u$ denotes the output stream computed by the component u of P from the input stream $\vec{s}$.

Any function $[\![P]\!]$ arising from a process in our language is *causal*, which means that the $n+1$st expression in any output sequence depends only on the first n input expressions. Thus, we restrict our attention to causal stream-processing functions.



The composition of two stream-processing functions $f_i : \mathbf{Stream}_{I_i} \to (\mathbf{Stream}_{O_i})^{D_i}$ $(i = 1, 2)$ with $O_1 \cap O_2 = \emptyset$ is defined as

$$f_1 \otimes f_2 : \mathbf{Stream}_I \to (\mathbf{Stream}_O)^{D_1 \times D_2}$$

(with $I = (I_1 \cup I_2) \setminus (O_1 \cup O_2)$, $O = (O_1 \cup O_2) \setminus (I_1 \cup I_2)$). Here for any $\vec{s} \in \mathbf{Stream}_I$, the (d_1, d_2)-component of $f_1 \otimes f_2(\vec{s})$ (for $(d_1, d_2) \in D_1 \times D_2$) is defined to be $\vec{t} \restriction_O$, where $\vec{t} \in \mathbf{Stream}_{I \cup O}$ is the unique stream with $\vec{t} \restriction_I = \vec{s} \restriction_I$ and $\vec{t} \restriction_{O_i} = f_i(\vec{s} \restriction_{I_i})_{d_i}$ $(i = 1, 2)$. For $\vec{t} \in \mathbf{Stream}_C$ and $C' \subseteq C$, the restriction $\vec{t} \restriction_{C'} \in \mathbf{Stream}_{C'}$ is defined by $\vec{t} \restriction_{C'} (c) = \vec{t}(c)$ for each $c \in C'$. Since the operator $\otimes$ is associative and commutative [6], we can define a generalised composition operator $\bigotimes_{i \in I} f_i$ for a set $\{f_i : i \in I\}$ of stream-processing functions.

Example Suppose $f : \mathbf{Stream}_{\{a\}} \to \mathbf{Stream}^2_{\{b\}}$, $f(\vec{s}) = \langle 0 :: \vec{s}, 1 :: \vec{s} \rangle$ is the stream-processing function with input channel a, output channel b and two deterministic components that outputs the input stream prefixed with either 0 or 1, and $g : \mathbf{Stream}_{\{b\}} \to \mathbf{Stream}^2_{\{c\}}$, $g(\vec{s}) = \langle 0 :: \vec{s}, 1 :: \vec{s} \rangle$ the function with input (resp. output) channel b (resp. c) that does the same. The composition $f \otimes g : \mathbf{Stream}_{\{a\}} \to \mathbf{Stream}^4_{\{c\}}$, $f \otimes g(\vec{s}) = \langle 0 :: 0 :: \vec{s}, 0 :: 1 :: \vec{s}, 1 :: 0 :: \vec{s}, 1 :: 1 :: \vec{s} \rangle$ outputs the input stream prefixed with one of the 2-element streams $0 :: 0$, $0 :: 1$, $1 :: 0$ or $1 :: 1$. (We use $\langle\rangle$ to denote tuples arising from components of a nondeterministic function.)

3.2. Associating a stream-processing function with a process

We interpret any process $P = (I, O, L, (p_c)_{c \in \tilde{O}})$ as a stream-processing function $[\![P]\!] : \mathbf{Stream}_I \to (\mathbf{Stream}_O)^{\mathcal{C}(P)}$.

For any closed deterministic program p with input channels in $\tilde{I}$ and $\vec{M} \in \mathbf{CExp}^{\tilde{I}}$, we define $[p](\vec{M}) \in \mathbf{CExp}$ in Figure 1, so that $[p](\vec{M})$ is the expression that results from running p once, when the channels have the initial values given in $\vec{M}$. In the definition, $E(\vec{M})$ denotes the result of substituting each occurrence of c in E by $\vec{M}(c)$. We write $p[E/x]$ for the outcome of replacing each free occurrence of x in program p with the term E, renaming variables to avoid capture.

Then for any nondeterministic program p with input channels $\tilde{I}$ and output channel c and any $\vec{s} \in \mathbf{Stream}_{\tilde{I}}$ we define $[p](\vec{s}) \in (\mathbf{Stream}_c)^{\mathcal{C}(p)}$ as follows: For each deterministic component p' of p, let the p'-component of $[p](\vec{s})$ be the (unique) $\vec{t} \in \mathbf{Stream}_c$ with

- $\vec{t}_0 = [p'](\varepsilon, \ldots, \varepsilon)$ and
- $\vec{t}_{n+1} = [p'](\vec{s}_n)$

where $\vec{t}_n$ is the nth element of the stream $\vec{t}$.

Finally, a process $P = (I, O, L, (p_c)_{c \in \tilde{O}})$ is interpreted as the composition $[\![P]\!] \stackrel{\text{def}}{=} \bigotimes_{c \in \tilde{O}} [p_c] : \mathbf{Stream}_I \to (\mathbf{Stream}_O)^{\mathcal{C}(P)}$.

Examples

- $[\textit{if } c = \varepsilon \textit{ then } \varepsilon \textit{ else } 1](\vec{s}) = \langle \varepsilon.\vec{t} \rangle$ where for any $n \in \mathbb{N}$, $\vec{t}_n = \varepsilon$ if $\vec{s}_n = \varepsilon$ and $\vec{t}_n = 1$ otherwise (and $c \in \mathbf{Channels}$).
- $[\textit{either } c \textit{ or } \varepsilon](\vec{s}) = \langle \varepsilon.\vec{s}, (\varepsilon, \varepsilon, \varepsilon, \ldots) \rangle$ (for $c \in \mathbf{Channels}$). Thus the nondeterministic choices are resolved at compile-time.
- For the process P with $I_P = \{i\}$, $O_P = \{o\}$ and $L_P = \{l\}$ and with $p_l \stackrel{\text{def}}{=} l :: i$ and $p_o \stackrel{\text{def}}{=} l :: i$ we have $[\![P]\!](\vec{s}) = \langle (\varepsilon, \vec{s}_0, \vec{s}_0 :: \vec{s}_1, \vec{s}_0 :: \vec{s}_1 :: \vec{s}_2, \ldots) \rangle$.

4. Fault-Tolerance

We assume that occurring failures are detected immediately (via "built-in self tests" (BIST) [25] or "detectors" [2]) and the process accessing the data receives the error message $\perp$. The interpretation of $\perp$ may depend on the failure semantics of the system under consideration (for example, missing a deadline could be considered a failure, cf. below).

For simplicity, we assume that the same failure distribution and the same replication mechanism, denoted by Π, apply throughout the whole system. Whenever a process (in the role of the *client* in the terminology of [8]) requests a data value from the system, Π forwards the request to the replication copies (*servers* [8]; e.g. hardware components) in question and computes a result from the respective results of the servers (e.g. by majority vote). From the point of view of the clients modelled in our language, replication is transparent.

Here we model Π as a family of distributions τ_η on the set $\mathbf{Failures} \stackrel{\text{def}}{=} \{0,1\}^{\mathbb{N}}$ of $\mathbb{N}$-indexed sequences. This means, for each *safety parameter* $\eta \in \mathbb{N}$ (e.g. the number of replication copies), τ_η is a probability distribution on $\mathbf{Failures}$. Here for any $\vec{r} \in \mathbf{Failures}$ and $n \in \mathbb{N}$,

$[E](\vec{M}) = E(\vec{M})$	$E \in \mathbf{CExp}$
$[\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p'](\vec{M}) = [p](\vec{M})$	if $[E](\vec{M}) = [E'](\vec{M})$
$[\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p'](\vec{M}) = [p'](\vec{M})$	if $[E](\vec{M}) \neq [E'](\vec{M})$
$[\mathit{case}\ E\ \mathit{of}\ x :: y\ \mathit{do}\ p\ \mathit{else}\ p'](\vec{M}) = [p[h/x, t/y]](\vec{M})$	if $[E](\vec{M}) = h :: t$ with $h \in \{\perp\} \cup \mathbf{Var} \cup \mathcal{D} \cup \tilde{I}$
$[\mathit{case}\ E\ \mathit{of}\ x :: y\ \mathit{do}\ p\ \mathit{else}\ p'](\vec{M}) = [p'](\vec{M})$	if $[E](\vec{M}) = \varepsilon$

Figure 1. Definition of $[p](\vec{M})$ in the abstract model.

$\vec{r}_n = 1$ means that the n^{th} data access to the server during an execution of a client raised an unmasked failure, while $\vec{r}_n = 0$ denotes successful masking (or absence) of a failure. Thus the probability that the n^{th} access raises a failure is $\Pr[\vec{r} \leftarrow \tau_\eta : \vec{r}_n = 1]$ (the probability that a sample from τ_η drawn according to its probability distribution has 1 as its n^{th} entry).

Definition 5 Let $p(\eta, t) : \mathbb{N} \times \mathbb{N} \to [0, 1]$ be a function that takes a safety parameter η and the number of accesses t and gives a probability. A replication mechanism $\Pi = (\tau_\eta)_\eta$ is called $p(\eta, t)$*-safe* if for any $(\eta, t) \in \mathbb{N} \times \mathbb{N}$ we have $\Pr[\vec{r} \leftarrow \tau_\eta : \vec{r}_t = 1] \leq p(\eta, t)$.

In the following subsections we recall two well-known examples of replication mechanisms and consider their degree of safety, depending on the failure semantics of the underlying server.

4.1. Crash/performance failure semantics

Suppose the servers whose possible faults have to be masked have crash/performance failure semantics (meaning that the server may crash or may deliver the requested data only after the specified time limit, but it is assumed to be partially correct). Suppose that the replication mechanism $\Pi_{c/p}$ consists of a group of η replicated servers (where $\eta \in \mathbb{N}$ is the safety parameter) and the group output is determined by the fastest member. Suppose that for each server S the probability that S fails (wrt. to the crash/performance failure semantics) is p, and the failure events of the different copies are independent from each other, and also independent from the previous history.

Fact 2 $\Pi_{c/p}$ *is a* $p(\eta, t)$*-safe replication mechanism for* $p(\eta, t) \stackrel{\text{def}}{=} p^\eta$.

Now suppose that the probability that S fails at the t^{th} access is $p \cdot t/(t + 1)$ (e.g., S may depend on mechanical parts whose failure rate increases with time) and denote the replication mechanism accessing it as above by $\Pi'_{c/p}$. Then we obtain the following result.

Fact 3 $\Pi'_{c/p}$ *is a* $p(\eta, t)$*-safe replication mechanism for*

$$p(\eta, t) \stackrel{\text{def}}{=} (p \cdot t/(t + 1))^\eta.$$

Note that in this example the usual assumption that failure rates are time-independent is violated.

4.2. Value failure semantics

Suppose now that the server has value failure semantics, i. e. the server may deliver incorrect values (represented by the error message $\perp$). Suppose that the replication mechanism Π_v consists of a group of $2 \cdot \eta + 1$ replicated servers and the group output is determined by majority vote (thus the result is correct unless at least $\eta + 1$ servers are corrupt). Suppose again that for each server S the probability that S fails is p, and that the failure events of the different copies are independent from each other, and from the previous history.

Fact 4 Π_v *is a* $p(\eta, t)$*-safe replication mechanism for* $p(\eta, t) \stackrel{\text{def}}{=} p^{\eta+1}$.

Now suppose again that the probability that S fails at the t^{th} access is $p \cdot t/(t + 1)$ and denote the replication mechanism accessing it as above by Π'_v.

Fact 5 Π'_v *is a* $p(\eta, t)$*-safe replication mechanism for*

$$p(\eta, t) \stackrel{\text{def}}{=} (p \cdot t/(t + 1))^{\eta+1}.$$

5. Probabilistic Model

We give a second interpretation of processes, which takes probability into account. A process $P = (I, O, L, (p_c)_{c \in \tilde{O}})$ defines a distribution $[\![P]\!]_\Pi$ on the set of stream-processing functions $f : \mathbf{Stream}_I \to (\mathbf{Stream}_O)^{\mathcal{C}(P)}$.

For any closed deterministic program p with input channels in $\tilde{I}$, any $\vec{M} \in \mathbf{CExp}^{\tilde{I}}$ and any $\vec{r} \in \mathbf{Failures}$, we define $[p]_{\vec{r}}(\vec{M}) \in \mathbf{CExp}$ in Figure 2, so that $[p]_{\vec{r}}(\vec{M})$ is the expression that results from running p once, when the

$$
\begin{array}{ll}
[E]_{0.\vec{r}}(\vec{M}) = E(\vec{M}) & \text{for } E \in \mathbf{CExp} \\
[E]_{1.\vec{r}}(\vec{M}) = \perp & \\
[\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p']_{r.\vec{r}}(\vec{M}) = [p]_{\vec{r}}(\vec{M}) & \text{if } [E]_{r.\vec{r}}(\vec{M}) = [E']_{r.\vec{r}}(\vec{M}) \\
[\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p']_{r.\vec{r}}(\vec{M}) = [p']_{\vec{r}}(\vec{M}) & \text{if } [E]_{r.\vec{r}}(\vec{M}) \neq [E']_{r.\vec{r}}(\vec{M}) \\
[\mathit{case}\ E\ \mathit{of}\ x :: y\ \mathit{do}\ p\ \mathit{else}\ p']_{r.\vec{r}}(\vec{M}) = [p']_{\vec{r}}(\vec{M}) & \text{if } [E]_{r.\vec{r}}(\vec{M}) = \varepsilon \\
[\mathit{case}\ E\ \mathit{of}\ x :: y\ \mathit{do}\ p\ \mathit{else}\ p']_{r.\vec{r}}(\vec{M}) = [p[h/x, t/y]]_{\vec{r}}(\vec{M}) & \text{if } [E]_{r.\vec{r}}(\vec{M}) = h :: t \text{ where } h \in \{\perp\} \cup \mathbf{Var} \cup \mathcal{D} \cup \tilde{I}.
\end{array}
$$

Figure 2. Definition of $[p]_{\vec{r}}(\vec{M})$ in the probabilistic model.

channels have the initial values given in $\vec{M}$ and given a sequence of failure occurences $\vec{r}$. In the definition, $0.\vec{r}$ is the sequence arising from prefixing $\vec{r}$ with 0.

Then for any non-deterministic program p with input channels $\tilde{I}$ and output channel c, any $\vec{s} \in \mathbf{Stream}_{\tilde{I}}$ and any $\vec{r} \in \mathbf{Failures}$ we define $[p]_{\vec{r}}(\vec{s}) \in (\mathbf{Stream}_c)^{\mathcal{C}(p)}$ as follows. For any deterministic component p' of p, let the p'-component of $[p]_{\vec{r}}(\vec{s})$ be the (unique) $\vec{t} \in \mathbf{Stream}_c$ with

- $\vec{t}_0 = [p']_{r^0}(\varepsilon, \ldots, \varepsilon)$
- $\vec{t}_{n+1} = [p']_{r^{n+1}}(\vec{s}_n)$.

Here the r^n (for $n \in \mathbb{N}$) denote disjoint subsequences of $\vec{r}$. Thus, depending on $\vec{r}$, failures may or may not be independent in different iterations of the program (so we need not make the usual assumption on history-independence of failure).

Given $\vec{r}$, a process $P = (I, O, L, (p_c)_{c \in \tilde{O}})$ is interpreted as the composition $[\![P]\!]_{\vec{r}} \stackrel{\text{def}}{=} \bigotimes_{c \in \tilde{O}} [p_c]_{r^c}$ (again the r^c are disjoint subsequences of $\vec{r}$).

Finally, $[\![P]\!]_\Pi = \{[\![P]\!]_{\vec{r}} : \vec{r} \leftarrow \tau_\eta\}$ for a replication mechanism $\Pi = (\tau_\eta)_\eta$.

Example

- $[c]_{\vec{r}}(\vec{s}) = \vec{r}$ (where $c \in \mathbf{Channels}$) for any $\vec{s}$ with $\vec{s}_n \neq \perp$ for each n.
- For the process P with $I_P = \emptyset$, $O_P = \{o\}$ and $L_P = \{l\}$ where $p_o \stackrel{\text{def}}{=} l$ and $p_l \stackrel{\text{def}}{=} l$ we have

$$
\begin{aligned}
&\Pr\left[\vec{t} \leftarrow [\![P]\!]_{\vec{r}}(\emptyset) : \vec{t}_n = \perp\right] \\
&= \sum_{i=0,\ldots,n} \Pr\left[\vec{r} \leftarrow \tau_\eta : \vec{r}_i = 1\right]
\end{aligned}
$$

6. Faithfulness of the Abstract Model

Definition 6 Let $\delta(\eta, l) : \mathbb{N} \times \mathbb{N} \to [0,1]$ be a function that takes a safety parameter η and a stream length l and gives a probability. A process P with replication mechanism $\Pi = (\tau_\eta)_\eta$ is called $\delta(\eta, l)$-*safe* if for any deterministic component P' of P, any $(\eta, l) \in \mathbb{N} \times \mathbb{N}$, and any input stream $\vec{s} \in \mathbf{Stream}_I$ of length up to l we have $\Pr\left[\vec{r} \leftarrow \tau_\eta : [\![P']\!]_{\vec{r}}(\vec{s}) \neq [\![P']\!](\vec{s})\right] \leq \delta(\eta, l)$.

We give two theorems regarding the faithfulness of the abstract model wrt. the probabilistic one. More precisely, we give a bound $\delta(\eta, l)$ such that, intuitively, the probability that the failures not masked by Π cause a process P to deviate from its specified behaviour, given an input history of length l and a safety parameter η, is bounded by $\delta(\eta, l)$ (making use of the data access bound $n(P)$ defined above). The proofs proceed by induction on the syntactic structure of the processes.

The first result does not make any additional assumptions on Π.

Theorem 1 *Suppose that Π is a $p(\eta, t)$-safe replication mechanism employed by the process P. Define $\delta(\eta, l) \stackrel{\text{def}}{=} \sum_{i=1,\ldots,l \cdot n(P)} p(\eta, i)$ (with $i \in \mathbb{N}$). Then P is $\delta(\eta, l)$-safe.*

To see that this bound is optimal when no additional assumptions on Π are given, consider the following replication mechanism $\Pi = (\tau_\eta)_\eta$: For each η, τ_η assigns probability $p > 0$ each to the two sequences in $\mathbf{Failures}$ where exactly the even (resp. exactly the odd) elements are 1 (and the others 0). Also τ_η assigns probability $1 - 2 \cdot p$ to the sequence in $\mathbf{Failures}$ that contains no 1. Then Π is a $p(\eta, t)$-safe replication mechanism (for each η, t) and e.g. for the process P that simply outputs 0 we have $\Pr\left[\vec{r} \leftarrow \tau_\eta : [\![P]\!]_{\vec{r}}(\vec{s}) \neq [\![P]\!](\vec{s})\right] = \delta(\eta, l)$ for the δ defined in the theorem.

Definition 7 A replication mechanism $\Pi = (\tau_\eta)_\eta$ is *history-independent* if for all bit-sequences $\vec{r}_1, \vec{r}_2 \in \{0,1\}^l$ of length l and each bit $b \in \{0,1\}$ we have

$$
\frac{\Pr\left[\vec{r} \leftarrow \tau_\eta : \vec{r}\lfloor_{l+1} = \vec{r}_1.b\right]}{\Pr\left[\vec{r} \leftarrow \tau_\eta : \vec{r}\lfloor_l = \vec{r}_1\right]} = \frac{\Pr\left[\vec{r} \leftarrow \tau_\eta : \vec{r}\lfloor_{l+1} = \vec{r}_2.b\right]}{\Pr\left[\vec{r} \leftarrow \tau_\eta : \vec{r}\lfloor_l = \vec{r}_2\right]}
$$

where $\vec{r}\lfloor_n$ is the prefix of length n of $\vec{r}$.

Note that a history-independent replication mechanism $\Pi = (\tau_\eta)_\eta$ may still depend on *time* (e.g. the probability that a sample from τ_η has 0 as its first entry may be different from that having 0 as its second entry).

Theorem 2 *Suppose that* Π *is a* $p(\eta,t)$*-safe history-independent replication mechanism employed by the process* P. *Define* $\delta(\eta,l) \stackrel{\text{def}}{=} 1 - \Pi_{i=1,\dots,l\cdot n(P)}(1-p(\eta,i))$ *(with* $i \in \mathbb{N}$*). Then* P *is* $\delta(\eta,l)$*-safe.*

7. Example: Unbounded buffer

Here we consider an unbounded FIFO buffer P. P receives either a request from its environment and then outputs the first value of its current content (and deletes this value from its content) or outputs ε if it is empty, or P receives a data value which it stores.

The corresponding process is $P = (\{c\},\{d\},\{l\},(p_d,p_l))$, i. e. it has an input channel c, output channel d and local feedback channel l. We assume that $request \in \mathcal{D}$ is a value that represents a data request from the environment of the buffer. Then

$$\begin{aligned} p_l &\stackrel{\text{def}}{=} \textit{if } c = \text{request } \textit{then } (\textit{case } l \textit{ of } h :: t \textit{ do } t \textit{ else } \varepsilon) \\ &\qquad \textit{else } l :: c \\ p_d &\stackrel{\text{def}}{=} \textit{if } c = \text{request } \textit{then } (\textit{case } l \textit{ of } h :: t \textit{ do } h \textit{ else } \varepsilon) \\ &\qquad \textit{else } \varepsilon \end{aligned}$$

Suppose we would like to provide a dependable implementation of the buffer using hardware storage with crash/performance failure semantics as in section 4.1. We can do this using the replication mechanism $\Pi_{c/p}$ defined there which leads to the following degree of safety of P:

Proposition 1 P *is* $\delta(\eta,l)$*-safe where* $\delta(\eta,l) \stackrel{\text{def}}{=} 1-(1-p^\eta)^{l\cdot n(P)} = 1-(1-p^\eta)^{4\cdot l}$.

Proof This follows from Fact 2, Theorem 2 and $n(P) = 4$. □

Note that for concrete systems the constants (such as $n(P) = 4$) depend on implementation details.

Time-dependent failures Now we assume that P is implemented using $\Pi'_{c/p}$ and get the following result:

Proposition 2 P *is* $\delta(\eta,l)$*-safe, where* $\delta(\eta,l) \stackrel{\text{def}}{=} 1 - \Pi_{i=1,\dots,l\cdot n(P)}(1-(p\cdot i/(i+1))^\eta)$.

Proof This follows from Fact 3 and Theorem 2. □

Here the usual assumption of time-independence (but not of history-independence) is violated.

8. Example: Ring storage

Here we consider a storage mechanism realised by processes $P_1,\dots,P_m$ grouped in a ring. The environment can either submit a request r_n to P_1 and consequently receive the data stored at P_n from P_m, or submit $s_n :: d$ to P_1 so that the value d will be stored at P_n and acknowledged by ok (in either case for any $n = 1,\dots,m$).

Define $P_n = (\{c_n\},\{c_{n+1}\},\{l_n\},(p_{c_{n+1}},p_{l_n}))$ for $n = 1,\dots,m$ and assume that $\{ok,r_1,\dots,r_m,s_1,\dots,s_m\} \subseteq \mathcal{D}$. Define

$$\begin{aligned} p_{c_{n+1}} &\stackrel{\text{def}}{=} \textit{case } c_n \textit{ of } h :: t \textit{ do} \\ &\qquad (\textit{if } h = r_n \textit{ then } l_n \\ &\qquad\quad \textit{else if } h = s_n \textit{ then ok else } h :: t) \\ &\qquad \textit{else } \varepsilon \\ p_{l_n} &\stackrel{\text{def}}{=} \textit{case } c_n \textit{ of } h :: t \textit{ do } (\textit{if } h = s_n \textit{ then } t \\ &\qquad \textit{else } (\textit{either } l_n \\ &\qquad\quad \textit{or if } h = r_n \textit{ then } \varepsilon \\ &\qquad\quad \textit{else } l_n)) \\ &\qquad \textit{else } l_n. \end{aligned}$$

Each process P_n takes an input on c_n and checks its header. If it is a request r_n addressed to it, it outputs the content of its local channel l_n on the channel c_{n+1} (and nondeterministically either leaves the local channel unchanged or sets it to ε). If the header is s_n it outputs ok on c_{n+1} and updates the local channel with the tail of the expression. Otherwise it simply passes on the input from c_n to c_{n+1} (and leaves the local storage unchanged).

Suppose the implementation of the processes relies on hardware storage with value failure semantics as in section 4.2 and suppose Π_v is the replication mechanism defined there.

Proposition 3 *For each* n, P_n *is* $\delta(\eta,l)$*-safe (for* η, l *with* $\delta(\eta,l) < 1/2$*), where*

$$\delta(\eta,l) \stackrel{\text{def}}{=} 1-(1-p^{\eta+1})^{l\cdot n(P)} = 1-(1-p^{\eta+1})^{6\cdot l}.$$

Proof This follows from Fact 4 and Theorem 2. □

General results on composition (which have to be left for the extended version of this paper for space reasons) give the following result:

Proposition 4 $P_1\otimes\dots\otimes P_m$ *is* $\delta(\eta,l)$*-safe where* $\delta(\eta,l) \stackrel{\text{def}}{=} 1-(1-p^{\eta+1})^{m\cdot l\cdot n(P)}$.

Suppose the processes P'_n are obtained (for each n) from P_n by substituting p_{l_n} by

$$p_{l_n} \stackrel{\text{def}}{=} \textit{case } c_n \textit{ of } h :: t \textit{ do}$$

$$(\mathit{if}\ h = s_n\ \mathit{then}\ t\ \mathit{else}\ l_n)$$
$$\mathit{else}\ l_n$$

and leaving the rest unchanged.

General results on refinement (again to be given in the extended version) give the following result:

Proposition 5 $P'_1 \otimes \ldots \otimes P'_m$ *is* $\delta(\eta, l)$*-safe where* $\delta(\eta, l) \stackrel{\mathrm{def}}{=} 1 - (1 - p^{\eta+1})^{m \cdot l \cdot n(P)}$.

Analogous results can be obtained for the replication mechanism Π'_v.

9. Conclusion and Future Work

In a formal framework, we showed under which assumptions and to what extent the usual approach to abstract from failure probabilities when verifying safety-critical systems is sound. To our knowledge, this is the first formal justification of this kind. Within the specification framework FOCUS, we obtained probability bounds on failure of the overall system, given the dependability of the used fault-tolerance components. Our approach can do without some of the simplifying assumptions commonly made (e.g. independence of failure probabilities from previous history). The concrete estimates obtained are useful to estimate the level of redundancy needed, in order to avoid overly high levels of redundancy with associated cost, complexity, hardware need and performance penalties (especially where tight constraints on hardware or performance must be met, e.g. in embedded systems). On the other hand, one can also derive results if knowledge on the dependability of the used components is available only qualitatively.

In summary, this work on the one hand gives a first formal justification of the usual divide-and-conquer approach to reasoning about dependability of systems that keeps reasoning feasible, and additionally it allows for a more fine-grained analysis (while making less assumptions) where necessary, at least for small parts of a system.

As this is the first step for work in this direction, the results given here are mathematically rather straightforward, since the main aim was to set up a general formal framework that allows to reason about the considered issues. In particular we only considered discrete probabilities (as most commonly done).

Since error handling on the application layer can be specified explicitly, one may also analyse the interplay between failures on lower levels that are not masked by the fault-tolerance components and their handling in different parts of the system. This is left for further work.

For simplicity we assumed the same failure semantics and replication mechanism throughout the whole system. It is straightforward to allow for more flexibility. In that context, one can also extend this approach to be able to model correlation of failure distribution between different components.

It is more expensive to provide highly dependable components, but it is cheaper to handle the failure behaviours when accessing these components. Since group management mechanisms can use up to 80% of the total throughput of a system [9] one needs to consider the performance of fault-tolerance mechanisms. Therefore it would be very interesting to extend our framework to include information on performance of a system depending on the performance of the fault-tolerance components.

Also one might consider extending this approach to other formalisms, such as synchronous languages (e.g. Lustre, Esterel, Signal) or asynchronous models (such as CSP [24]).

10. Acknowledgements

Many thanks go to Martín Abadi for interesting discussions and comments on the draft. Also many thanks to Bruno Dutertre for the opportunity to present this work at the Formal Methods and Dependable Systems group at SRI (Menlo Park) and to John Rushby, Jon Millen and the other group members for interesting comments. Finally many thanks to Massimo Felici and the anonymous referees for useful comments on the draft.

This work was partially performed during a visit at Bell Labs Research at Silicon Valley, Lucent Technologies (Palo Alto) whose hospitality is gratefully acknowledged.

References

[1] M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation, 2001. Submitted.

[2] A. Arora and S. Kulkarni. Detectors and correctors: A theory of fault-tolerance components. *IEEE Transactions on Software Engineering*, 2000. To appear.

[3] M. Bozga and O. Maler. On the representation of probabilities over structured domains. In *Computer-Aided Verification*, volume 1633 of *LNCS*. Springer, 1999.

[4] M. Bravetti and R. Gorrieri. The theory of interactive generalized semi-markov processes. *Theoretical Comput. Sci.* To appear.

[5] M. Broy. A logical basis for component-based systems engineering. In M. Broy and R. Steinbrüggen, editors, *Calculational System Design*. IOS Press, 1999.

[6] M. Broy and K. Stølen. *Specification and Development of Interactive Systems*. Springer, 2001.

[7] R. W. Butler and G. B. Finelli. The infeasibility of quantifying the reliability of life-critical real-time software. *IEEE Transactions on Software Engineering*, 19(1):3–12, Jan. 1993.

[8] F. Cristian. Abstractions for fault-tolerance. In K. Duncan and K. Krueger, editors, *Information Processing '94, Proceedings of the IFIP 13th World Computer Congress*, 1994.

[9] F. Cristian, R. Dancey, and J. Dehn. High availability in the advanced automation system. In *Digest of Papers, The 20th International Symposium on FaultTolerant Computing*, Newcastle-UK, June 1990. IEEE.

[10] P. D'Argenio, J.-P. Katoen, and E. Brinksma. An algebraic approach to the specification of stochastic systems. In *IFIP Working conference on Programming Concepts and Methods, PROCOMET'98*. Chapman & Hall, 1998.

[11] L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic processes using MTBDDs and the Kronecker representation. In *TACAS'2000*, LNCS. Springer, 2000.

[12] M. Felici, A. Pasquini, and M. Sujan. Applicability limits of software reliability growth models. In *International Conference on Mathematical Methods in Reliability*, 2000.

[13] S. S. Gokhale and K. S. Trivedi. Analytical modeling. In *Encyclopedia of Distributed Systems*. Kluwer Academic Publishers, 1998.

[14] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994.

[15] J. Hillston and M. Ribaudo. Stochastic process algebras: a new approach to performance modeling. In K. Bagchi and G. Zobrist, editors, *Modeling and Simulation of Advanced Computer Systems*. Gordon Breach, 1998.

[16] F. Huber, S. Molterer, A. Rausch, B. Schätz, M. Sihling, and O. Slotosch. Tool supported Specification and Simulation of Distributed Systems. In *International Symposium on Software Engineering for Parallel and Distributed Systems*, pages 155–164, 1998.

[17] B. Jonsson. *Compositional Verification of Distributed Systems*. PhD thesis, Department of Computer Systems, Uppsala University, 1987. Tech. Rep. DoCS 87/09.

[18] B. Jonsson and K. Larsen. Specification and refinement of probabilistic processes. In *IEEE Symposium on Logic in Computer Science (LICS '91)*, 1991.

[19] J. Jürjens. Secrecy-preserving refinement. In *Formal Methods Europe (International Symposium)*, volume 2021 of *LNCS*. Springer, 2001.

[20] J. Laprie. Dependability: basic concepts and terminology. *Dependable Computing and Fault-Tolerant Systems*, 5, 1992.

[21] E. Lloyd and W. Tye. Systematic safety: Safety assessment of aircraft systems, 1982. Reprinted 1992.

[22] Z. Manna and A. Pnueli. On the faithfulness of formal models. In *Mathematical Foundation of Computer Science*, volume 520 of *LNCS*, pages 28–42. Springer, 1991.

[23] A. Pnueli and L. Zuck. Probabilistic verification. *Information and Computation*, 103(1):1–29, 1993.

[24] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998.

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

[26] K. Seidel. Probabilistic communicating processes. *Theoretical Comput. Sci.*, 152:219–249, 1995.

[27] A. Valmari. The state explosion problem. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*, pages 429–528. Springer, 1998.

[28] A. Yakovlev and A. Koelmans. Petri nets and digital hardware design. In *Lectures on Petri Nets II: Applications*, volume 1492 of *LNCS*. Springer, 1998.

Appendix

This appendix contains proof sketches of the results given in the main body.

A. Fault-Tolerance

A.1. Crash/performance failure semantics

Fact 2 $\Pi_{c/p}$ *is a* p^η*-safe replication mechanism.*

Proof We have to show that (for all t, η) $\Pr[\vec{r} \leftarrow \tau_\eta : \vec{r}_t = 1] \leq p^\eta$. Since by assumption the failure probability does not depend on the number of previous accesses, it is sufficient to show that $\Pr[\vec{r} \leftarrow \tau_\eta : \vec{r}_0 = 1] \leq p^\eta$. But $\Pr[\vec{r} \leftarrow \tau_\eta : \vec{r}_0 = 1] = p^\eta$ since all replicated copies must fail for the replication mechanism to fail, and these failures were assumed to be independent. □

Fact 3 $\Pi'_{c/p}$ *is a* $p(\eta, t)$*-safe replication mechanism for* $p(\eta, t) \stackrel{\text{def}}{=} (p \cdot t/(t+1))^\eta$.

Proof We have to show that (for all t, η) $\Pr[\vec{r} \leftarrow \tau_\eta : \vec{r}_t = 1] \leq (p \cdot t/(t+1))^\eta$. But $\Pr[\vec{r} \leftarrow \tau_\eta : \vec{r}_t = 1] = (p \cdot t/(t+1))^\eta$. since all replicated copies must fail for the replication mechanism to fail, and these failures were assumed to be independent. □

A.2. Value failure semantics

Fact 4 Π_v *is a* $p^{\eta+1}$*-safe replication mechanism.*

Proof Again it is sufficient to show that $\Pr[\vec{r} \leftarrow \tau_\eta : \vec{r}_0 = 1] \leq p^{\eta+1}$, and this is the case since the majority of copies must fail for the replication mechanism to fail. □

Fact 5 Π'_v *is a* $p(\eta, t)$*-safe replication mechanism for* $p(\eta, t) \stackrel{\text{def}}{=} (p \cdot t/(t+1))^{\eta+1}$.

Proof Similarly, we have to show that (for all t, η) $\Pr[\vec{r} \leftarrow \tau_\eta : \vec{r}_t = 1] \leq (p \cdot t/(t+1))^{\eta+1}$. But $\Pr[\vec{r} \leftarrow \tau_\eta : \vec{r}_t = 1] = (p \cdot t/(t+1))^{\eta+1}$, since the majority of copies must fail for the replication mechanism to fail. □

B. Faithfulness of the Abstract Model

Fact 1 *Given any process P, to compute one output tuple from an input tuple, P has access to data at most $n(P)$ times.*

Proof The proof proceeds essentially by induction on the structure of the programs in P. □

Theorem 1 *Suppose that Π is a $p(\eta,t)$-safe replication mechanism employed by the process P. Define $\delta(\eta,l) \stackrel{\text{def}}{=} \sum_{i=1,\ldots,l\cdot n(P)} p(\eta,i)$ (with $i \in \mathbb{N}$). Then P is $\delta(\eta,l)$-safe.*

Proof Suppose that Π is a $p(\eta,t)$-safe replication mechanism. We need to show that for each deterministic component P' of P, all η, l, and any input stream $\vec{s} \in \mathbf{Stream}_I$ of length up to l we have

$$\Pr[\vec{r} \leftarrow \tau_\eta : [\![P']\!]_{\vec{r}}(\vec{s}) \neq [\![P']\!](\vec{s})] \quad \leq \sum_{i=1,\ldots,l\cdot n(P)} p(\eta,i).$$

Fix a deterministic component P' of P and a safety parameter η. By Fact 1 we know that the number ν of data accesses of P is bounded by $l \cdot n(P)$.

The inequality $[\![P']\!]_{\vec{r}}(\vec{s}) \neq [\![P']\!](\vec{s})$ implies that at least one of the data accesses leads to a fault. Thus it is sufficient to show that the probability $\tilde{p}_\nu$ that at least one of the first ν data accesses leads to a fault is bounded by $\sum_{i=1,\ldots,\nu} p(\eta,i)$ and thus by $\sum_{i=1,\ldots,l\cdot n(P)} p(\eta,i)$.

We do this by induction on ν.

For $\nu = 0$ this bound is obviously valid since no data access is made.

Suppose we have $\tilde{p}_\nu \leq \sum_{i=1,\ldots,\nu} p(\eta,i)$ for some $\nu \geq 0$. By assumption, the probability that the $\nu+1$st data access raises a failure is bounded by $p(\eta,\nu+1)$. Thus we certainly have $\tilde{p}_{\nu+1} \leq \sum_{i=1,\ldots,\nu+1} p(\eta,i)$. □

Theorem 2 *Suppose that Π is a $p(\eta,t)$-safe history-independent replication mechanism employed by the process P. Define $\delta(\eta,l) \stackrel{\text{def}}{=} 1 - \Pi_{i=1,\ldots,l\cdot n(P)}(1-p(\eta,i))$ (with $i \in \mathbb{N}$). Then P is $\delta(\eta,l)$-safe.*

Proof (Sketch) Suppose that Π is a $p(\eta,t)$-safe replication mechanism. We need to show that for each deterministic component P' of P, all η, l, and any input stream $\vec{s} \in \mathbf{Stream}_I$ of length up to l we have

$$\Pr[\vec{r} \leftarrow \tau_\eta : [\![P']\!]_{\vec{r}}(\vec{s}) \neq [\![P']\!](\vec{s})]$$
$$\leq 1 - \Pi_{i=1,\ldots,l\cdot n(P)}(1-p(\eta,i))$$

Fix a deterministic component P' of P and a safety parameter η. Again we know that the number ν of data accesses of P is bounded by $l \cdot n(P)$.

As above, the inequality $[\![P']\!]_{\vec{r}}(\vec{s}) \neq [\![P']\!](\vec{s})$ implies that at least one of the data accesses leads to a fault. Thus it is sufficient to show that the probability $\tilde{p}_\nu$ that at least one of the first ν data accesses leads to a fault is bounded by $1-\Pi_{i=1,\ldots,\nu}(1-p(\eta,i))$ and thus by $1-\Pi_{i=1,\ldots,l\cdot n(P)}(1-p(\eta,i))$.

We do this by induction on ν.

For $\nu = 0$ this bound is obviously valid since no data access is made (note that here we follow the convention that a product indexed by an empty set gives 1).

Suppose we have $\tilde{p}_\nu \leq 1 - \Pi_{i=1,\ldots,\nu}(1-p(\eta,i))$ for some $\nu \geq 0$. We need to show that $\tilde{p}_{\nu+1} \leq 1 - \Pi_{i=1,\ldots,\nu+1}(1-p(\eta,i))$

By the homogeneity assumption, the event that the $\nu+1$st data access raises a failure is independent from the event that any of the first ν data accesses raises a failure. Thus we have

$$\begin{aligned}
\tilde{p}_{\nu+1} &= \tilde{p}_\nu + (1-\tilde{p}_\nu)\cdot p(\eta,\nu+1) \\
&= \tilde{p}_\nu \cdot (1-p(\eta,\nu+1)) + p(\eta,\nu+1) \\
&\leq (1-\Pi_{i=1,\ldots,\nu}(1-p(\eta,i)))\cdot(1-p(\eta,\nu+1)) \\
&\quad +p(\eta,\nu+1) \\
&= 1-\Pi_{i=1,\ldots,\nu+1}(1-p(\eta,i)
\end{aligned}$$

by the inductive assumption. □