Reasoning about Failure Probabilities - formally and feasibly

Jan Jürjens *
Computing Laboratory, University of Oxford, GB

August 21, 2000

Abstract

It is often not feasible to formally reason about fault-tolerant systems while keeping track of the probabilities of all possible faults that may occur during their execution. However, one may not just neglect these probabilities if one would like to show that the employed fault-tolerance components are sufficient for the required level of dependability.

In the specification framework of FOCUS, we present a specification language that is interpreted firstly in an abstract model allowing formal reasoning in a feasible way. Secondly, it is interpreted in a probabilistic model. We give probability bounds that show the degree of faithfulness of the abstract model wrt. the probabilistic one. Further results are given for refinement and composition of systems.

1 Introduction

Formal methods have been substantially applied in safety-critical systems to ensure the needed high degree of dependability by proving correctness of the system design and in particular of the fault tolerance components. In spite of very successful work in probabilistic verification [SPH84, PZ93, dAKN+00], formal verification of safety-critical systems usually abstracts away from probabilities to make reasoning more feasible. Failure probabilities of underlying components (such as hardware) are considered (using Markov models) in isolation from the whole system, which is verified formally under the assumption that all failure probabilities are small enough. By doing this, one usually needs to make 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 absence of failure is composable (in the sense that if two components in isolation exhibit acceptable failure behaviour, then they do so when composed), that failures do not remain latent for a period of time to become effective only later etc.

*jan@dcs.ed.ac.uk - This work was partially supported by the Studienstiftung des deutschen Volkes and Lucent Technologies.

In practice, these assumptions are not always guaranteed. Thus it would be very desirable to have a complementary approach that allows to formally reason about safety-critical systems in a feasible way while keeping track of failure probabilities.

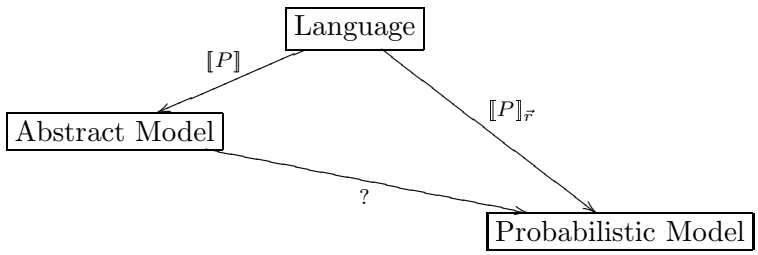
We propose a specification framework that allows to

- obtain concrete information on the degree of dependability of the overall system, depending on 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).

This is useful also since fault-tolerance components can only provide a relative degree of dependability, especially under performance and cost constraints.

In the framework of FOCUS [Bro99, BS00] we define a specification language (where specifications are formulated as nondeterministic programs) that is interpreted in 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. For this we clearly identify the assumptions on the fault tolerance components and give concrete estimates for the failure probabilities of a system depending on the dependability of the fault tolerance components and the execution time. 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 [Rus94].

Since formal methods are especially beneficial when applied in the early stages of the software development process [FL98], we give results regarding refinement and composition to allow reuse of results obtained early in the development cycle.

In the next subsection we provide background and point out related work. In Section 2 we present the specification framework used and explain some design decisions. 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 buffer. In Sections 8 we provide results on the preservation of failure probability bounds by refinement and composition. Section 9 gives a fault-tolerant ring storage as a final example, and we conclude in Section 10.

The proofs of most results have to be omitted due to space restrictions; they are to be found in the long version of this paper [Jür00].[1]

1.1 Background and Related Work

1.1.1 Fault-tolerant systems

A central requirement on safe-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" [Lap92].

Fault tolerance usually involves fault detection (determining that a fault has occurred) and fault masking (insuring that in spite of the failure of a component only correct values are passed to the environment of the system). Fault masking may require complex protocols whose correctness can be non-obvious [Rus94]. 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, time redundancy and information redundancy. Space redundancy provides physical copies of a resource and is especially effective for persisting faults. Time redundancy is realised by rerunning functions and information redundancy e. g. by using error-correcting codes; both are used to handle transient faults. Generally, fault-tolerance using redundancy depends on statistical independence among the failure occurrences.

Failures may arise in various ways: e. g. as hardware component failures or as hardware or software design flaws. While it seems generally reasonable to assume (though it is not at all guaranteed) that hardware components replicated in space (isolated from each other and possibly differently designed) fail independently, this assumption seems less clear wrt. hardware or software design flaws [BF93].

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 [CDD90]. 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 [LT82, 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.

[1] These proofs are included in this version in an appendix for the purpose of the submission procedure and can be read at discretion of the program committee.

A range of models have been developed (with advanced tools) to evaluate dependability of safety-critical systems, such as Reliability block diagrams, Markov models [BJ95] and stochastic Petri Nets (for an overview cf. [GT98]). Additionally, formal models have been developed that include probabilistic information, e. g. [SPH84, PZ93, HJ94, Sei95, dAKN$^+$00].

In [AK99] it is shown that a fault-tolerant program can be decomposed into a fault-intolerant program and a set of fault-tolerance components (probabilistic aspects are not considered).

The faithfulness of formal modelling in general has been considered in [MP91].

This work is inspired by results in relating the formal methods and computational complexity approaches to cryptography [AR00, AJ00]. Even though our application here are fault-tolerant systems, the general idea should be applicable to other situations where probabilities occur.

2 The Specification framework

The specifications in our framework are formulated as nondeterministic processes. Specifically, we consider concurrently executing processes interacting by transmitting sequences of data values over unidirectional FIFO communication channels [BS00]. Communication is asynchronous in the sense that transmission of a value cannot be prevented by the receiver (note that one may model synchronous communication using handshake [BS00]).

2.1 Specification Language

We assume disjoint sets **Channels** of channels, **Var** of variables and $\mathcal{D}$ of data values. The set **Exp** of expressions contains the empty expression ε and the nonempty expressions defined inductively by

$E ::=$	expression
$\bot$	error value
x	$x \in$ **Var**
c	$c \in$ **Channels**
d	$d \in \mathcal{D}$
$E_1 :: E_2$	concatenation

Here $c \in$ **Channels** represents the input on the channel c. The empty expression ε denotes absence of output on a channel at a given point in time. The error value $\bot$ signals a failure (similar to its use in [VBC92]). We write $\mathbf{Exp}_c$ for the set of *closed* expressions (those containing no subterms in **Var** $\cup$ **Channels**). As usual, we assume $(E_1 :: E_2) :: E_3 = E_1 :: (E_2 :: E_3)$ and $E_1 :: \varepsilon = E_1 = \varepsilon :: E_1$ (for all $E_1, E_2, E_3 \in$ **Exp**).

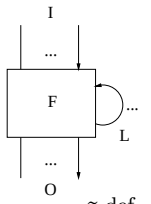
(Nondeterministic) *programs* are defined inductively by:

$p ::=$	programs
E	output of expression
either p *or* p'	nondeterministic branching
if $E = E'$ *then* p *else* p'	conditional
case E *of* $x :: y$ *do* p *else* p'	head::tail

In *case* E *of* $x :: y$ *do* p *else* p', x and y are bound in p. We regard programs as equal up to renaming of variables.

Any program p computes an output expression from the inputs given (at evaluation time) at the channels $c \in$ **Channels** occurring in p (which are called the input channels of p).

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



- $I \subseteq$ **Channels** is the set of its input channels,
- $O \subseteq$ **Channels** is the set of its output channels,
- each p_c ($c \in \tilde{O} \stackrel{\text{def}}{=} O \cup L$) is a closed program with input channels in $\tilde{I} \stackrel{\text{def}}{=} I \cup L$ (where $L \subseteq$ **Channels** is a set of local channels). From inputs on $\tilde{I}$ 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. We assume that different processes have disjoint sets of local channels. They are used to store local state between the execution rounds which can be shared among the programs of a process.

Refinement on programs Since our results necessarily refer to the syntax of the specifications (by giving error bounds depending on the number of expression evaluations), refinement has to be defined on the level of syntax as well:

The relation $p \rightsquigarrow p'$ (pronounced "p refines to p'") between programs p, p' is defined to be the precongruence generated by (*either* p *or* p') $\rightsquigarrow p$ and (*either* p *or* p') $\rightsquigarrow p'$. By precongruence we mean a reflexive and transitive relation that fulfils

$$p \rightsquigarrow p' \wedge q \rightsquigarrow q' \Rightarrow op(p, p') \rightsquigarrow op(q, q')$$

for

$$op(p, p') \in \{(\textit{either } p \textit{ or } p'), (\textit{if } E = E' \textit{ then } p \textit{ else } p'), (\textit{case } E \textit{ of } x :: y \textit{ do } p \textit{ else } p')\}.$$

A program p' is a *deterministic component* of a program p if $p \rightsquigarrow p'$ and p' is minimal wrt. $\rightsquigarrow$ (i. e. there is no p'' with $p' \rightsquigarrow p''$). We write $\mathcal{C}(p)$ for the set of deterministic components of the program p.

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

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

3 Abstract Model

3.1 Stream-processing functions

In this subsection we recall the definitions of streams and stream-processing functions from [Bro99, BS00].

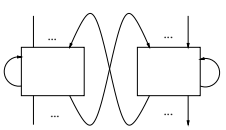
We write $\mathbf{Stream}_C \stackrel{\text{def}}{=} (\mathbf{Exp}_c^*)^C$ (where $C \subseteq \mathbf{Channels}$) for the set of C-indexed tuples of (finite or infinite) sequences of closed expressions. The elements of this set are called *streams*, specifically *input streams* (resp. *output streams*) if C denotes the set of non-local input (resp. output) channels of a process P. Each stream $\vec{s} \in \mathbf{Stream}_C$ consists 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 $P = (I, O, (p_c)_{c \in O})$ is modelled by a total *stream-processing function* $[\![P]\!] : \mathbf{Stream}_I \to (\mathbf{Stream}_O)^{\mathcal{C}(P)}$ 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 presentation of later definitions). Thus $[\![P]\!](\vec{s})_d$ denotes the output stream computed by the component d 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 $f(\vec{s})$ (for $(d_1, d_2) \in D_1 \times D_2$) is defined to be $\vec{t}\,\lfloor_{\mathbf{Stream}_O}$, where $\vec{t} \in \mathbf{Stream}_{I \cup O}$ is the unique stream with $\vec{t}\,\lfloor_{\mathbf{Stream}_I} = \vec{s}\,\lfloor_{\mathbf{Stream}_I}$ and $\vec{t}\,\lfloor_{\mathbf{Stream}_{O_i}} = f_i(\vec{s}\,\lfloor_{\mathbf{Stream}_{I_i}})_{d_i}$ $(i = 1, 2)$.

Example If $f : \mathbf{Stream}_{\{a\}} \to \mathbf{Stream}^2_{\{b\}},\ f(\vec{s}) = \langle 0.\vec{s}, 1.\vec{s}\rangle$ is the stream-processing function with input channel a and output channel b 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, then 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 either 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 to a process

We give the interpretation of processes in the abstract model.

Any closed deterministic program p with input channels in I defines a function $[p] : \mathbf{Exp}^I_c \to \mathbf{Exp}_c$ as follows:

- $[E](\vec{M}) = E(\vec{M})$ where E is an expression with input channels in I, $\vec{M} \in \mathbf{Exp}^I_c$, and $E(\vec{M})$ denotes the result of substituting each occurrence of c (for $c \in I$) in E by $\vec{M}(c)$
- $[\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p'](\vec{M}) = [p](\vec{M})$ 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 $h \in \{\bot\} \cup \mathbf{Var} \cup \mathcal{D} \cup I$
- $[\mathit{case}\ E\ \mathit{of}\ x :: y\ \mathit{do}\ p\ \mathit{else}\ p'](\vec{M}) = [p'](\vec{M})$ if $[E](\vec{M}) = \varepsilon$

We write $p[E/x]$ for the outcome of replacing each free occurrence of x in process P with the term E, renaming variables to avoid capture.

Then any nondeterministic program p (with output channel c) defines a stream-processing function $[p] : \mathbf{Stream}_{\tilde{I}} \to (\mathbf{Stream}_c)^{\mathcal{C}(p)}$ as follows. Given $\vec{s} \in \mathbf{Stream}_{\tilde{I}}$ and a deterministic p' with $p \rightsquigarrow p'$, 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
- $\vec{t}_{n+1} = [p'](\vec{s}_n)$.

Finally, a process $P = (I, O, (p_c)_{c\in\tilde{O}})$ 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

- $[\mathit{if}\ c = \varepsilon\ \mathit{then}\ \varepsilon\ \mathit{else}\ 1](\vec{s}) = \varepsilon.\vec{t}$ where for any $n \in \mathbb{N}$, $\vec{t}_n = \varepsilon$ if $\vec{s}_n = \varepsilon$ and $\vec{t}_n = 1$ otherwise (and $c \in \mathbf{Channels}$).

- $[\![either\ c\ or\ \varepsilon]\!](\vec{s}) = \langle \varepsilon.\vec{s}, (\varepsilon, \varepsilon, \varepsilon, \ldots)\rangle$ (for $c \in$ **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 $[\![P]\!](\vec{s}) = (\varepsilon, \vec{s}_0, \vec{s}_0 :: \vec{s}_1, \vec{s}_0 :: \vec{s}_1 :: \vec{s}_2, \ldots)$.

4 Fault Tolerance

Data may be corrupt due to failures e. g. of the underlying hardware. We assume that occurring failures are detected immediately (via "built-in self tests" (BIST) [Rus94] or "detectors" and "correctors" [AK99]) 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).

To achieve fault-tolerance, a distributed system architecture incorporates redundant processing components in order to mask failures [Cri94]. For the purpose of this exploratory note, we assume that the same failure distribution and replication mechanism, denoted by Π, applies to the whole system. Π may include mechanisms to resolve conflicting results from different replication copies. From the point of view of the processes, replication is transparent.

In terms of the model in [MS92], Π is part of the Fault-Tolerance Support Layer, the system it accesses is part of the Operating System Services, and the specifications written in our language are part of the Application Software Layer.

For each *safety parameter* $\eta \in \mathbb{N}$ (e. g. the number of replication copies), Π determines a distribution τ_η on the set **Failures** $\stackrel{\text{def}}{=} \{0,1\}^{\mathbb{N}}$ of failure occurrences as follows: For any $\vec{r} \in$ **Failures** and $n \in \mathbb{N}$, if $\vec{r}_n = 1$, this signifies that the n^{th} data access to the server via the replication mechanism Π during an execution of a process P (the client) raised a failure not masked by Π, while $\vec{r}_n = 0$ denotes absence of a failure. Thus the probability that the n^{th} access raises a failure is $\Pr\left[\vec{r} \stackrel{R}{\leftarrow} \tau_\eta : \vec{r}_n = 1\right]$ (the probability that a sample from τ_η drawn according to its probability distribution has 1 as its n^{th} entry).

The formal model can be seen as the special case where we assume that (for each η) the distribution τ_η assigns the probability 1 to the sequence of failure occurrences $(0, 0, 0, \ldots) \in$ **Failures** representing absence of any failure.

Definition 1 Let $p(\eta, t) : \mathbb{N} \times \mathbb{N} \rightharpoonup [0,1]$ be a (possibly partial) function that takes a safety parameter η and the number of access t and gives a probability. A replication mechanism Π is called $p(\eta,t)$-*safe* if for any (η, t) in the domain of p we have $\Pr\left[\vec{r} \stackrel{R}{\leftarrow} \tau_\eta : \vec{r}_t = 1\right] \leq p(\eta, t)$.

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, i. e. the failures that are likely to be observable by clients are crash/performance failures (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 number of previous accesses.

Proposition 1 *$\Pi_{c/p}$ is a $p(\eta, t)$-safe replication mechanism for $p(\eta, t) \stackrel{\text{def}}{=} p^\eta$.*

Thus for any failure probability $p < 1$ of the server S and any required bound on the failure probability per time unit of the replication mechanism $\Pi_{c/p}$ one can find a safety parameter η such that $\Pi(\eta)$ meets this bound.

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.

Proposition 2 *$\Pi'_{c/p}$ is a $p(\eta, t)$-safe replication mechanism for*

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

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. 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.

Proposition 3 *Π_v is a $p(\eta, t)$-safe replication mechanism for $p(\eta, t) \stackrel{\text{def}}{=} p^{\eta+1}$.*

Again for any $p < 1$ and any required bound on the failure probability per time unit of Π_v one can find a safety parameter η such that $\Pi_v(\eta)$ meets this bound.

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.

Proposition 4 *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 the probabilistic interpretation of processes. The basic idea is that any expression $E \in \mathbf{Exp}$ is interpreted probabilistically either as E or as the error value $\perp$.

Given a sequence of failure occurrences $\vec{r} \in \mathbf{Failures}$, any deterministic closed program p with input channels in I defines a function $[p]_{\vec{r}} : \mathbf{Exp}^I \to \mathbf{Exp}$ as follows:

- $[E]_{0.\vec{r}}(\vec{M}) = E(\vec{M})$ where E is an expression with input channels in I, $\vec{M} \in \mathbf{Exp}^I$ and $0.\vec{r}$ is the sequence arising from prefixing $\vec{r}$ with 0.
- $[E]_{1.\vec{r}}(\vec{M}) = \perp$
- $[\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p']_{r.\vec{r}}(\vec{M}) = [p]_{\vec{r}}(\vec{M})$ if $[E]_{r.\vec{r}}(\vec{M}) = [E']_{r.\vec{r}}(\vec{M})$
- $[\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p']_{r.\vec{r}}(\vec{M}) = [p']_{\vec{r}}(\vec{M})$ if $[E]_{r.\vec{r}}(\vec{M}) \neq [E']_{r.\vec{r}}(\vec{M})$
- $[\mathit{case}\ E\ \mathit{of}\ x :: y\ \mathit{do}\ p\ \mathit{else}\ p']_{r.\vec{r}}(\vec{M}) = [p']_{\vec{r}}(\vec{M})$ if $[E]_{r.\vec{r}}(\vec{M}) = \varepsilon$
- $[\mathit{case}\ E\ \mathit{of}\ x :: y\ \mathit{do}\ p\ \mathit{else}\ p']_{r.\vec{r}}(\vec{M}) = [p[h/x, t/y]]_{\vec{r}}(\vec{M})$ if $[E]_{r.\vec{r}}(\vec{M}) = h :: t$ where $h \in \{\perp\} \cup \mathbf{Var} \cup \mathcal{D}$ or $h = c$ for $c \in I$

We write $p[E/x]$ for the outcome of replacing each free occurrence of x in process P with the term E, renaming variables to avoid capture.

Then, given $\vec{r}$, any non-deterministic program p defines a stream-processing function $[p]_{\vec{r}} : \mathbf{Stream}_{\tilde{I}} \to \mathbf{Stream}_{\mathcal{D}(p)}$ as follows. Given $\vec{s} \in \mathbf{Stream}_{\tilde{I}}$ and a deterministic p' with $p \rightsquigarrow p'$, 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 r^n (for $n \in \mathbb{N}$) denote disjoint subsequences of $\vec{r}$ defined by $r^n_m \stackrel{\text{def}}{=} \vec{r}_{n \cdot n(p)+m}$. Thus, depending on $\vec{r}$, failures may or may not be independent in different iterations of the program.

Given $\vec{r}$, a process $P = (I, O, (p_c)_{c \in \tilde{O}})$ is interpreted as the composition $[\![P]\!]_{\vec{r}} \stackrel{\text{def}}{=} \bigotimes_{c \in \tilde{O}} [p_c]_{r^c}$ (here r^{c_n} is the subsequence of $\vec{r}$ defined by $r^{c_n}_m \stackrel{\text{def}}{=} \vec{r}_{m \cdot |\tilde{O}|+n}$ where c_n is the n^{th} channel in $\tilde{O}$ according to a fixed linear order on **Channels** and $|\tilde{O}|$ is the number of channels in $\tilde{O}$).

Finally, $[\![P]\!]_\Pi = \{[\![P]\!]_{\vec{r}} : \vec{r} \stackrel{R}{\leftarrow} \tau_\eta\}$ for replication mechanism Π with failure distributions τ_η (as a function of the safety parameter η).

Examples

- $[c]_{\vec{r}}(\vec{s}) = \vec{r}$ (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

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

6 Faithfulness of the Abstract Model

Definition 2 Let $\delta(\eta, l) : \mathbb{N} \times \mathbb{N} \rightharpoonup [0,1]$ be a (possibly partial) function that takes a safety parameter η and a stream length l and gives a probability.

A process P with replication mechanism Π (having failure distributions τ_η) is called $\delta(\eta, l)$-*safe* if for any (η, l) in the domain of δ the following equivalence holds for any input stream $\vec{s} \in \mathbf{Stream}_I$ of length up to l and any $T \in (\mathbf{Stream}_O)^{\mathcal{C}(P)}$:

$$[\![P]\!](\vec{s}) = T \text{ iff } \Pr\left[\vec{r} \stackrel{R}{\leftarrow} \tau_\eta : [\![P]\!]_{\vec{r}}(\vec{s}) \neq T\right] \leq \delta(\eta, l).$$

Theorem 1 (Faithfulness) *Suppose that* Π *is a* $p(\eta, t)$*-safe replication mechanism employed by the process* P*. Define* $\delta(\eta, l) \stackrel{\text{def}}{=} 1 - (1 - p(\eta, 1)) \cdot \ldots \cdot (1 - p(\eta, l \cdot n(P)))$*. Then* P *is* $\delta(\eta, l)$*-safe for each* l, η *with* $\delta(\eta, l) < 1/2$*.*

The proof proceeds by induction on the stucture of the programs p_c. The condition $\delta < 1/2$ is only used for the implication $\Leftarrow$ in the definition of $\delta(\eta, l)$-safety.

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 delete 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 $P = (\{c, l\}, \{d, l\}, (p_d, p_l))$, 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

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

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 5 *P is $\delta(\eta, l)$-safe where $\delta(\eta, l) \stackrel{\text{def}}{=} 1 - (1 - p^\eta)^{l \cdot n(P)} = 1 - (1 - p^\eta)^{4 \cdot l}$ for l with $\delta(\eta, l) < 1/2$.*

Proof This follows from Proposition 1 and Theorem 1. □

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

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

Proposition 6 *P is $\delta(\eta, l)$-safe (for η, l with $\delta(\eta, l) < 1/2$), where*

$$\delta(\eta, l) \stackrel{\text{def}}{=} 1 - (1 - p/2) \cdot \ldots \cdot (1 - p \cdot l \cdot n(P)/(l \cdot n(P) + 1)).$$

Proof This follows from Proposition 2 and Theorem 1. □

8 Refinement and Composition

We define refinement and composition on processes and give a result on failure bounds.

Definition 3 The relation $P \rightsquigarrow P'$ (pronounced "*P refines* to P'") between processes $P = (I, O, (p_c)_{c \in \tilde{O}})$ and $P' = (I, O, (p'_c)_{c \in \tilde{O}})$ is defined to hold if $p_c \rightsquigarrow p'_c$ for each $c \in \tilde{O}$.

Theorem 2 *Given a replication mechanism Π with failure distributions τ_η, processes P, P' with $P \rightsquigarrow P'$, a finite input stream $\vec{s} \in \mathbf{Stream}_I$ and a safety parameter η, we have*

$$\Pr\left[\vec{r} \stackrel{R}{\leftarrow} \tau_\eta : [\![P']\!]_{\vec{r}}(\vec{s}) \neq [\![P']\!](\vec{s})\right] \leq \Pr\left[\vec{r} \stackrel{R}{\leftarrow} \tau_\eta : [\![P]\!]_{\vec{r}}(\vec{s}) \neq [\![P]\!](\vec{s})\right].$$

Corollary 1 *Given a replication mechanism Π and a bound $\delta(\eta, l)$, if a process P is $\delta(\eta, l)$-safe and P refines to P' then P' is $\delta(\eta, l)$-safe (for η, l with $\delta(\eta, l) < 1/2$).*

Since our results necessarily refer to the syntax of programs we define composition on the level of syntax:

Definition 4 Suppose two processes $P_1 = (I_1, O_1, (p_c)_{c \in \tilde{O}_1})$ and $P_2 = (I_2, O_2, (p_c)_{c \in \tilde{O}_2})$ have disjoint sets of output channels (in particular also disjoint sets of local channels): $\tilde{O}_1 \cap \tilde{O}_2 = \emptyset$. We define their *composition* $P = P_1 \otimes P_2$ to be the process $P = (I_1 \cup I_2, O_1 \cup O_2, (p_c)_{c \in \tilde{O}_1 \cup \tilde{O}_2})$.

Composition of processes is translated to composition of stream-processing functions in both models:

Proposition 7 *For all processes P_1, P_2 we have*

- $[\![P_1 \otimes P_2]\!] = [\![P_1]\!] \otimes [\![P_2]\!]$ *and*
- $[\![P_1 \otimes P_2]\!]_\Pi = [\![P_1]\!]_\Pi \otimes [\![P_2]\!]_\Pi$ *for any* Π.

Theorem 3 *Suppose that* Π *is a replication mechanism with failure distributions* τ_η *and* $P_1 = (I_1, O_1, (p_c)_{c\in\tilde{O}_1})$ *and* $P_2 = (I_2, O_2, (p_c)_{c\in\tilde{O}_2})$ *are processes. Let an input stream* $\vec{s} \in \mathbf{Stream}_I$ *of length* l *and a safety parameter* η *be given. Then*

$$\Pr\left[\vec{r} \xleftarrow{R} \tau_\eta : [\![P_1 \otimes P_2]\!]_{\vec{r}}(\vec{s}) \neq [\![P_1 \otimes P_2]\!](\vec{s})\right] \quad \leq \quad 1 - \pi_1 \cdot \pi_2$$

where for $i = 1, 2$, $\pi_i \stackrel{\text{def}}{=} 1 - \max_{\vec{t}}\Big(\Pr\left[\vec{r} \xleftarrow{R} \tau_\eta : [\![P_i]\!]_{\vec{r}}(\vec{t}) \neq [\![P_i]\!](\vec{t})\right]\Big)$ *where* $\vec{t} \in \mathbf{Stream}_{I_i}$ *ranges over input streams of length* l.

Corollary 2 *Suppose we have a replication mechanism* Π *and bounds* $\delta_1(\eta, l)$ *and* $\delta_2(\eta, l)$ *such that* $\delta(\eta, l) \stackrel{\text{def}}{=} 1-(1-\delta_1(\eta, l))\cdot(1-\delta_2(\eta, l)) < 1/2$ *and processes* P_i *that are* $\delta_i(\eta, l)$*-safe (for* $i = 1, 2$*). Then* $P_1 \otimes P_2$ *is* $\delta(\eta, l)$*-safe (for* η, l *with* $\delta(\eta, l) < 1/2$*).*

9 Example: Ring storage

Here we consider a storage mechanism realised by processes $P_1, \ldots, P_m$ grouped in a ring. The environment can either submit a request r_n to P_1 and consequently receives 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 $n = 1, \ldots, m$).

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

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

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 8 *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 Proposition 3 and Corollary 1. □

Proposition 9 *$P_1 \otimes \ldots \otimes P_m$ is $\delta(\eta, l)$-safe where $\delta(\eta, l) \stackrel{\text{def}}{=} 1 - (1 - p^{\eta+1})^{m \cdot l \cdot n(P)}$ for l with $\delta(\eta, l) < 1/2$.*

Proof This follows from Corollary 2 (note that all P_n are equal up to renaming of channels). □

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

$$p_{l_n} \stackrel{\text{def}}{=} \textit{case } c_n \textit{ of } h :: t \textit{ do } (\textit{if } h = s_n \textit{ then } t \textit{ else } l_n) \textit{ else}$$

and leaving the rest unchanged.

Proposition 10 *$P'_1 \otimes \ldots \otimes P'_m$ is $\delta(\eta, l)$-safe where $\delta(\eta, l) \stackrel{\text{def}}{=} 1 - (1 - p^{\eta+1})^{m \cdot l \cdot n(P)}$ for l with $\delta(\eta, l) < 1/2$.*

Proof This follows from Corollary 1. □

Analogous results can be obtained for the case Π'_v.

10 Conclusion and Further Work

We extended the specification framework FOCUS to allow formal reasoning about safety-critical systems in a feasible way while keeping track of failure probabilities. Our specification language is interpreted in 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. For this we clearly identify the assumptions on the fault tolerance components and give concrete estimates for the failure probabilities of a system depending on the dependability of the fault tolerance components and the execution time. We also give results regarding refinement and composition to allow reuse of results obtained early in the development cycle.

This work should be seen as complementary to the usual approach of treating fault tolerance components and the systems employing them separately. An advantage of our framework is that it allows to

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

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 should be straightforward to allow for more flexibility.

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 (Palumbo, Butler 1985 [CDD90]) 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.

It would be desirable to have a tool that from a specification and failure bounds on the used fault tolerance components computes failure bounds of the overall system, by making use of the results presented here.

11 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 for useful comments on the draft.

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

References

[AJ00] M. Abadi and J. Jürjens. Soundness of eavesdropping on protocol executions, 2000. in preparation.

[AK99] A. Arora and S. Kulkarni. Detectors and correctors: A theory of fault-tolerance components. *to appear in IEEE Transactions on Software Engineering*, 1999.

[AR00] M. Abadi and P. Rogaway. Reconciling two views of cryptography (invited lecture). In *TCS 2000*, August 2000.

[BF93] Ricky W. Butler and George B. Finelli. The infeasibility of quantifying the reliability of life-critical real-time software. *IEEE Transactions on Software Engineering*, 19(1):3–12, January 1993.

[BJ95] Ricky W. Butler and Sally C. Johnson. Techniques for modeling the reliability of fault-tolerant systems using the markov state-space approach. Technical Report RP-1348, NASA, September 1995.

[Bro99] M. Broy. A logical basis for component-based systems engineering. In M. Broy and R. Steinbrüggen, editors, *Calculational System Design*, number 173 in Series F: Computer and Systems Sciences, 1999.

[BS00] M. Broy and K. Stølen. *Specification and Development of Interactive Systems*. Springer Verlag, 2000. (to be published).

[CDD90] 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.

[Cri94] 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.

[dAKN+00] 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*, Lecture Notes in Computer Science, January 2000.

[Fel00] Massimo Felici. Dependability perspectives in requirements engineering. In *Proceedings of International Conference on Dependable Systems and Networks*, 2000.

[FL98] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems: Practical Tools and Techniques for Software Development*. Cambridge University Press, 1998.

[GT98] Swapna S. Gokhale and Kishor S. Trivedi. Analytical modeling. In *Encyclopedia of Distributed Systems*. Kluwer Academic Publishers, 1998.

[HJ94] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994.

[Jür00] J. Jürjens. Reasoning about failure probabilities - formally and feasibly (long version), 2000. to be submitted.

[Lap92] J.C. Laprie. Dependability: basic concepts and terminology. *Dependable Computing and Fault-Tolerant Systems*, 5, 1992.

[LT82] E. Lloyd and W. Tye. Systematic safety: Safety assessment of aircraft systems, 1982. Reprinted 1992.

[MP91] Z. Manna and A. Pnueli. On the faithfulness of formal models. In *Mathematical Foundation of Computer Science*, volume 520 of *Lecture Notes in Computer Science*, pages 28–42. Springer Verlag, 1991.

[MS92] Shivakant Mishra and Richard D. Schlichting. Abstractions for constructing dependable distributed systems. Technical Report Technical Report TR92-19, Department of Computer Science, University of Arizona, 1992.

[PZ93] A. Pnueli and L.D. Zuck. Probabilistic verification. *Information and Computation*, 103(1):1–29, 1993.

[Rus94] J. Rushby. Critical system properties: Survey and taxonomy. *Reliability Engineering and System Safety*, 43(2):189–219, 1994.

[Sei95] K. Seidel. Probabilistic communicating processes. *Theoretical Computer Science*, 152:219–249, 1995.

[SPH84] M. Sharir, A. Pnueli, and S. Hart. Verification of probabilistic programs. *SIAM Journal Comp.*, 13:292–314, 1984.

[VBC92] B. Di Vito, R. Butler, and J. Caldwell. High level design proof of a reliable computing platform. In *Dependable Computing for Critical Applications 2, Dependable Computing and Fault-Tolerant Systems*, pages 279–306. Springer Verlag, 1992.

Appendix - not part of the paper !

This appendix contains sketches of the proofs that had to be omitted from the actual paper due to space restrictions. Full proofs are intended to be published in a long version of this paper, the sketches are included in this appendix for the purpose of the submission procedure and can be read at discretion of the program committee.

A Fault Tolerance

A.1 Crash/performance failure semantics

Proposition 11 *$\Pi_{c/p}$ is a p^η-safe replication mechanism.*

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

Proposition 12 *$\Pi'_{c/p}$ 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) $\Pr\left[\vec{r} \stackrel{R}{\leftarrow} \tau_\eta : \vec{r}_t = 1\right] \leq (p \cdot t/(t+1))^\eta$. But $\Pr\left[\vec{r} \stackrel{R}{\leftarrow} \tau_\eta : \vec{r}_t = 1\right] = (p \cdot t/(t+1))^\eta$. 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

Proposition 13 *Π_v is a $p^{\eta+1}$-safe replication mechanism.*

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

Proposition 14 *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) $\Pr\left[\vec{r} \stackrel{R}{\leftarrow} \tau_\eta : \vec{r}_t = 1\right] \leq (p \cdot t/(t+1))^{\eta+1}$. But $\Pr\left[\vec{r} \stackrel{R}{\leftarrow} \tau_\eta : \vec{r}_t = 1\right] = (p \cdot t/(t+1))^{\eta+1}$, since the majority of copies must fail for the replication mechanism to fail. □

B Faithfulness of the Abstract Model

Theorem 4 (Faithfulness) *Suppose that* Π *is a* $p(\eta,t)$*-safe replication mechanism employed by the process* P*. Define* $\delta(\eta,l) \stackrel{\text{def}}{=} 1-(1-p(\eta,1))\cdot\ldots\cdot(1-p(\eta,l\cdot n(P)))$*. Then* P *is* $\delta(\eta,l)$*-safe for each* l,η *with* $\delta(\eta,l)<1/2$*.*

Proof (Sketch) Intuitively, we have to show that the following probability is bounded by $\delta(\eta,l)$: that it is *not* the case that at the first access no failure occurs *and* at the second access no failure occurs *and* ... *and* at the $l\cdot n(P)^{th}$ access no failure occurs. Intuitively, this is indeed the case since the probability that a failure occurs at the t^{th} access is bounded by $p(\eta,t)$.

Suppose that Π is a $p(\eta,t)$-safe replication mechanism with failure distributions τ_η and $P=(I,O,(p_c)_{c\in O})$ a process. Suppose we are given an input stream $\vec{s}\in\mathbf{Stream}_{\tilde{I}}$ of length l and a safety parameter η such that $\delta(\eta,l)\stackrel{\text{def}}{=}1-(1-p(\eta,1))\cdot\ldots\cdot(1-p(\eta,l\cdot n(P)))<1/2$. We have to show that for any T we have $[\![P]\!](\vec{s})=T$ iff $\Pr\left[\vec{r}\stackrel{R}{\leftarrow}\tau_\eta : [\![P]\!]_{\vec{r}}(\vec{s})\neq T\right]\leq\delta(\eta,l)$ Note that the direction $\Leftarrow$ of this equivalence follows from $\Rightarrow$ by the assumption that $\delta(\eta,l)<1/2$, so we only have to show $\Rightarrow$.

Claim 1 *For any finite input stream* $\vec{t}\in\mathbf{Stream}_{\tilde{I}}$ *of length* l *we have*

$$\Pr\left[\vec{r}\stackrel{R}{\leftarrow}\tau_\eta : [\![p]\!]_{\vec{r}}(\vec{t})\neq[p](\vec{t})\right]\leq\delta(\eta,l)$$

for any deterministic program p*.*

We first note that to prove the theorem it is sufficient to prove the claim: Suppose that for any finite input stream $\vec{t}\in\mathbf{Stream}_I$ of length l and any deterministic p we have $\Pr\left[\vec{r}\stackrel{R}{\leftarrow}\tau_\eta : [p]_{\vec{r}}(\vec{t})\neq[p](\vec{t})\right]\leq\delta(\eta,l)$. Then $\Pr\left[\vec{r}\stackrel{R}{\leftarrow}\tau_\eta : [\![P]\!]_{\vec{r}}(\vec{s})\neq[\![P]\!](\vec{s})\right]\leq\delta$: For any $\vec{r}$ such that $[p]_{\vec{r}}(\vec{t})=[p](\vec{t})$ it follows that $[\![P]\!]_{\vec{r}}(\vec{t}\restriction_{\tilde{I}})=[\![P]\!](\vec{t}\restriction_{\tilde{I}})$ (where $\vec{t}\restriction_{\tilde{I}}$ is the restriction of $\vec{t}$ to the channels in $\tilde{I}$) since the feedback and indexing by components certainly preserve equality. Thus the implication follows since by definition of $[\![P]\!]_{\vec{r}}$ there exists a $\vec{t}$ such that $\vec{s}=\vec{t}\restriction_{\tilde{I}}$.

Proof (of the claim) We show by induction on the structure of p that for any execution of a program p at time t_0 with input expression $\vec{M}\in\mathbf{Exp}^I$, we have

$$\Pr\left[\vec{r}\stackrel{R}{\leftarrow}\tau_\eta : [p]_{\vec{r}}(\vec{M})\neq[p](\vec{M})\right]\leq 1-(1-p(\eta,t_0))\cdot\ldots\cdot(1-p(\eta,t_0+n(p)-1)) :$$

- $E\in\mathbf{Exp}$: $\Pr\left[\vec{r}\stackrel{R}{\leftarrow}\tau_\eta : [E]_{\vec{r}}(\vec{M})\neq[E](\vec{M})\right]\leq p(\eta,t_0)$ since by assumption on Π, $\Pr\left[\vec{r}\stackrel{R}{\leftarrow}\tau_\eta : \vec{r}_{t_0}=1\right]\leq p(\eta,t_0)$.
- *either* p *or* p':

  $$\begin{aligned}&\Pr\left[\vec{r}\stackrel{R}{\leftarrow}\tau_\eta : [p]_{\vec{r}}(\vec{M})\neq[p](\vec{M})\right]\\&\leq 1-(1-p(\eta,t_0))\cdot\ldots\cdot(1-p(\eta,t_0+\max(n(p),n(p'))-1)) :\end{aligned}$$

  by inductive hypothesis on p and p'

- *if* $E = E'$ *then* p *else* p': Analogous to the previous case.
- *case* E *of* $x :: y$ *do* p *else* p': Analogous to the previous case.

□

□

C Refinement

Theorem 5 *Given a replication mechanism* Π *with failure distributions* τ_η*, processes* P, P' *with* $P \rightsquigarrow P'$*, a finite input stream* $\vec{s} \in \mathbf{Stream}_I$ *and a safety parameter* η*, we have*

$$\Pr\left[\vec{r} \xleftarrow{R} \tau_\eta : [\![P']\!]_{\vec{r}}(\vec{s}) \neq [\![P']\!](\vec{s})\right] \leq \Pr\left[\vec{r} \xleftarrow{R} \tau_\eta : [\![P]\!]_{\vec{r}}(\vec{s}) \neq [\![P]\!](\vec{s})\right].$$

Proof The proof is a straightforward induction on the structure of programs. The interesting case is *either* p *or* q and it follows since if $[p] \neq [p]_{\vec{r}}$ or $[q] \neq [q]_{\vec{r}}$ then $[\textit{either } p \textit{ or } q] \neq [\textit{either } p \textit{ or } q]_{\vec{r}}$, because stream-processing functions are functions from streams to tuples of streams indexed by the components of a program and two tuples are equal iff they are component-wise equal. □

Corollary 3 *Given a replication mechanism* Π *and a bound* $\delta(\eta, l)$*, if a process* P *is* $\delta(\eta, l)$*-safe and* P *refines to* P' *then* P' *is* $\delta(\eta, l)$*-safe (for* η, l *with* $\delta(\eta, l) < 1/2$*).*

Proof This follows from Theorem 2, using the assumption $\delta(\eta, l) < 1/2$). □

Proposition 15 *For all processes* P_1, P_2 *we have*

- $[\![P_1 \otimes P_2]\!] = [\![P_1]\!] \otimes [\![P_2]\!]$ *and*
- $[\![P_1 \otimes P_2]\!]_\Pi = [\![P_1]\!]_\Pi \otimes [\![P_2]\!]_\Pi$ *for any* Π.

Proof This is clear by definition of $[\![P]\!]$ and $[\![P]\!]_\Pi$. □

Theorem 6 *Suppose that* Π *is a replication mechanism with failure distributions* τ_η *and* $P_1 = (I_1, O_1, (p_c)_{c \in \tilde{O}_1})$ *and* $P_2 = (I_2, O_2, (p_c)_{c \in \tilde{O}_2})$ *are processes. Let an input stream* $\vec{s} \in \mathbf{Stream}_I$ *of length* l *and a safety parameter* η *be given. Then*

$$\Pr\left[\vec{r} \xleftarrow{R} \tau_\eta : [\![P_1 \otimes P_2]\!]_{\vec{r}}(\vec{s}) \neq [\![P_1 \otimes P_2]\!](\vec{s})\right] \quad \leq \quad 1 - \pi_1 \cdot \pi_2$$

where for $i = 1, 2$*,* $\pi_i \stackrel{\text{def}}{=} 1 - \max_{\vec{t}}\left(\Pr\left[\vec{r} \xleftarrow{R} \tau_\eta : [\![P_i]\!]_{\vec{r}}(\vec{t}) \neq [\![P_i]\!](\vec{t})\right]\right)$ *where* $\vec{t} \in \mathbf{Stream}_{I_i}$ *ranges over input streams of length* l*.*

Proof Intuitively, this is the case since $[\![P_1 \otimes P_2]\!]_{\vec{r}}(\vec{s}) \neq [\![P_1 \otimes P_2]\!](\vec{s})$ only if either $[\![P_1]\!]_{\vec{r}}(\vec{s}) \neq [\![P_1]\!](\vec{s})$ or $[\![P_2]\!]_{\vec{r}}(\vec{s}) \neq [\![P_2]\!](\vec{s})$, i. e. only if it is *not* the case that $[\![P_1]\!]_{\vec{r}}(\vec{s}) = [\![P_1]\!](\vec{s})$ *and* $[\![P_2]\!]_{\vec{r}}(\vec{s}) = [\![P_2]\!](\vec{s})$.

The formal proof is a tedious but straightforward induction on the structures of the programs in P_1 and P_2. □

Corollary 4 *Suppose we have a replication mechanism* Π *and bounds* $\delta_1(\eta, l)$ *and* $\delta_2(\eta, l)$ *such that* $\delta(\eta, l) \stackrel{\text{def}}{=} 1 - (1 - \delta_1(\eta, l)) \cdot (1 - \delta_2(\eta, l)) < 1/2$ *and processes* P_i *that are* $\delta_i(\eta, l)$*-safe (for* $i = 1, 2$*). Then* $P_1 \otimes P_2$ *is* $\delta(\eta, l)$*-safe (for* η, l *with* $\delta(\eta, l) < 1/2$*).*

Proof This follows from Theorem 3, using the assumption $\delta(\eta, l) < 1/2$). □