A trace-based compositional proof theory for fault tolerant distributed systems

Eindhoven University of Technology

Department of Mathematics and Computing Science

A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems

by

H. Schepers and J. Hooman

93/16

Computing Science Note 93/16
Eindhoven, June 1993

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology. Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review. Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. M. Philips
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems

Henk Schepers[‡] *Jozef Hooman*[§]

Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

We present a compositional network proof theory to specify and verify safety properties of fault tolerant distributed systems. In this proof theory we abstract from the precise nature and occurrence of faults, but model their effect on the externally visible input and output behaviour. To this end we formalize a fault hypothesis as a reflexive relation between the normal behaviour (i.e. the behaviour when no faults occur) of a system and its acceptable behaviour, that is, the normal behaviour together with the exceptional behaviour (i.e. the behaviour whose abnormality should be tolerated). The method is compositional to allow for the reasoning with the specifications of processes while ignoring their implementation details. This compositionality is achieved by starting from a SAT formalism to reason about the normal behaviour and extending it with a single rule to obtain a specification of the acceptable behaviour from the specification of the normal behaviour and a predicate characterizing the fault hypothesis. We prove soundness and relative network completeness of the method. Our approach is illustrated by applying it to a triple modular redundant component and the alternating bit protocol.

Key words: Compositional proof theory, fault hypothesis, fault tolerance, relative network completeness, safety, soundness, specification, verification.

1 Introduction

It is difficult to prove the properties of a distributed system composed of failure prone processes, as such proofs must take into account the effects of faults occurring at any point in the execution of the individual processes. In the Hoare style formalism of [5] Cristian deals with the effects of faults that have occurred by partitioning the initial state space into disjoint subspaces, and providing a separate specification for each part. In the formalisms for fault tolerance that have been proposed in the more recent literature (cf. [3], [9], [14], [18]) the occurrence of a fault is modeled explicitly, typically using the designated symbol '†'. In contrast, we want to model the effects of faults on the externally visible input and output behaviour and let the alphabet of a process remain unchanged. In particular, we aim at a formalism which abstracts from the internal states of the processes and concentrates on the input and output behaviour that is observable at their interface. As a consequence, in our proof theory we do not deal with the sequential aspects of processes. To support top-down program design we want to reason with the specifications of processes without considering their implementation and the precise nature and occurrence of faults in such an implementation. This means that we aim at a *compositional* proof theory for fault tolerant distributed systems.

In fault tolerant systems, three domains of behaviour are distinguished: normal, exceptional and catastrophic (see [12]). Normal behaviour is the behaviour when no faults occur. The discriminating

[‡]Supported by the Dutch STW under grant number NWI88.1517: 'Fault Tolerance: Paradigms, Models, Logics, Construction'. E-mail: schepers@win.tue.nl.

[§]E-mail: wsinjh@win.tue.nl.

factor between exceptional and catastrophic behaviour is the *fault hypothesis* which stipulates how faults affect the normal behaviour. Relative to the fault hypothesis an exceptional behaviour exhibits an abnormality which should be tolerated (to an extent that remains to be specified). A catastrophic behaviour has an abnormality that was not anticipated (cf. [1], [11], [12], and [15]). In general, the catastrophic behaviour of a component cannot be tolerated by a system. Under a particular fault hypothesis, the system is designed as if the hypothetical faults are the only faults it can experience and measures are taken to tolerate (only) those *anticipated* faults (see, e.g., [16] for some design examples). In particular, the exceptional behaviour together with the normal behaviour constitutes the *acceptable* behaviour.

Given this classification of behaviour, we investigate whether an existing compositional proof theory for reasoning about the normal behaviour of a system can be adapted to deal with its acceptable behaviour. To do so, we formalize a fault hypothesis as a relation between the normal and the acceptable behaviour of a system. Indeed, such a relation enables one to abstract from the precise nature and occurrence of a fault and to focus on the abnormal behaviour it causes, if any.

As a starting point of the development of the proof theory, along the lines described above, we consider a simple SAT formalism to specify and verify safety properties of networks of processes that communicate synchronously via directed channels. Safety properties are properties that can be falsified by finite observations [20]. They are important for reliability because, in the characterization by Lamport [10], they express that 'nothing bad will happen'. We express a property of a process P by means of trace logic, using a special variable h to denote the trace, also called history, of P. Such a history describes the observable behaviour of a process by recording the communications along the visible channels of the process. For instance, a possible history h of 1-place buffer B which alternately inputs an integer via the observable channel *in* and outputs it via the observable channel *out*, may be $\langle(in,1),(out,1),(in,3),(out,3)\rangle$. To express that a process P satisfies a safety property ϕ we use a correctness formula of the form P **sat** ϕ. Typical safety properties of buffer B are 'if there is a communication on *out* then the communicated value is equal to the most recently communicated value on *in*' and 'the number of *out* communications is equal to or one less than the number of *in* communications'.

Based on a particular fault hypothesis, the set of behaviours that characterize a process is expanded. To keep such an expansion manageable, the fault hypothesis χ of a process P is formalized as a predicate, whose only free variables are h and h_{old}, representing a reflexive relation between the normal and acceptable histories of P. The interpretation is such that h_{old} represents a normal history of process P, whereas h is an *acceptable* history of P with respect to χ. For a predicate χ, representing a fault hypothesis, we introduce the construct $(P \wr \chi)$ to indicate execution of process P under the assumption of χ. This construct enables one to specify *failure prone processes.* Consider again buffer B. Under the hypothesis that, due to faults, values in the buffer are corrupted, which is formalized by some fault hypothesis predicate *Cor*, history $\langle(in,1),(out,1),(in,3),(out,3)\rangle$ may be transformed into history $\langle(in,1),(out,1),(in,3),(out,5)\rangle$. Then, we would like to prove that failure prone process $(B \wr Cor)$ still satisfies the property that 'the number of *out* communications is equal to or one less than the number of *in* communications'.

We define the trace semantics of a failure prone process *FP*, and define when correctness formulae of the form *FP* **sat** ϕ are valid. We present a proof theory to verify that a system tolerates the exceptional behaviour of its components to the desired extent. The proof theory is compositional in the sense that it allows for the reasoning with the specifications satisfied by failure prone processes while ignoring their implementation details. The usefulness of our method is illustrated by applying it to a triple modular redundant system and the alternating bit protocol, where, indeed, we only use the specifications of the components. Finally, we show that our proof theory is sound and obtain a completeness result by establishing preciseness preservation (see [19]).

The remainder of this report is organized as follows. Section 2 introduces the programming language. Section 3 defines the denotational semantics. In Section 4 we present the assertion language and associated correctness formulae. In Section 5 we incorporate fault hypotheses into our formalism. Section 6 presents a compositional network proof theory for fault tolerant distributed systems. We illustrate our method by applying it, in Section 7, to a triple modular redundant component, and, in Section 8, to the alternating bit protocol. In Section 9 we prove that the proof theory of Section 6 is sound and complete. A conclusion and suggestions for future research can be found in Section 10.

2 Programming Language

In this section we present an OCCAM-like programming language which is used to define networks of processes. Let VAR be a nonempty set of program variables, $CHAN$ a nonempty set of channel names, and let VAL be a denumerable domain of values. $\mathbb{N}$ denotes the set of natural numbers (including 0). We consider a concurrent programming language in which processes communicate synchronously via directed channels. The syntax of our programming language is given in Table 1, with $n \in \mathbb{N}$, $n \geq 1$, $x \in VAR$, $\mu \in VAL$, $c \in CHAN$, and $cset \subseteq CHAN$.

Table 1: Syntax of the Programming Language

Expression	$e ::=$	$\mu \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2$
Boolean Expression	$b ::=$	$e_1 = e_2 \mid e_1 < e_2 \mid \neg b \mid b_1 \vee b_2$
Guarded Command	$G ::=$	$[\lbrack\rbrack_{i=1}^{n} b_i \rightarrow P_i]$
Process	$P ::=$	$\mathbf{skip} \mid x := e \mid c!e \mid c?x \mid P_1 \,;\, P_2 \mid G \mid *G \mid P_1 \,\|\, P_2 \mid P \backslash cset$

Informally, the statements of our programming language have the following meaning:

Atomic statements

- **skip** terminates without any effect.
- Assignment $x := e$ assigns the value of expression e to the variable x.
- Output statement $c!e$ is used to send the value of expression e on channel c as soon as a corresponding input command is available. Since we assume synchronous communication, such an output statement is suspended until a parallel process executes an input statement $c?x$.
- Input statement $c?x$ is used to receive a value via channel c and assign this value to the variable x. As for the output command, such an input statement has to wait for a corresponding partner before a (synchronous) communication can take place.

Compound statements

- $P_1 \,;\, P_2$ indicates sequential composition: first execute P_1, and continue with the execution of P_2 if and when P_1 terminates.
- Guarded command $[\lbrack\rbrack_{i=1}^{n} b_i \rightarrow P_i]$. If none of the b_i evaluate to true then this guarded command terminates after evaluation of the booleans. Otherwise, non-deterministically select one of the b_i that evaluates to true and execute the corresponding statement P_i.
- Iteration $*G$ indicates repeated execution of guarded command G as long as at least one of the guards is open. When none of the guards is open $*G$ terminates.
- $P_1 \,\|\, P_2$ indicates the parallel execution of the processes P_1 and P_2.
- $P \backslash cset$ hides the channels from $cset$.

For a guarded command $G = [\lbrack\rbrack_{i=1}^{n} b_i \rightarrow P_i]$ we define $b_G = b_1 \vee \ldots \vee b_n$. Define $var(P)$ as the set of variables occurring in P.

Definition 1 (Observable input channels of a process) The set of visible, or observable, input channels of process P, notation $in(P)$, is defined inductively as follows:

- $in(\mathbf{skip}) = in(x := e) = in(c!e) = \emptyset$
- $in(c?x) = \{c\}$
- $in(P_1 \ ; \ P_2) = in(P_1) \cup in(P_2)$
- $in([\|_{i=1}^{n} b_i \rightarrow P_i]) = \cup_i in(P_i)$
- $in(*G) = in(G)$
- $in(P_1 \,\|\, P_2) = in(P_1) \cup in(P_2)$
- $in(P \setminus cset) = in(P) - cset$ $\diamond$

Definition 2 (Observable output channels of a process) The set of visible, or observable, output channels of process P, notation $out(P)$, is defined inductively as follows:

- $out(\mathbf{skip}) = out(x := e) = \emptyset$
- $out(c!e) = \{c\}$
- $out(c?x) = \emptyset$
- $out(P_1 \ ; \ P_2) = out(P_1) \cup out(P_2)$
- $out([\|_{i=1}^{n} b_i \rightarrow P_i]) = \cup_i out(P_i)$
- $out(*G) = out(G)$
- $out(P_1 \,\|\, P_2) = out(P_1) \cup out(P_2)$
- $out(P \setminus cset) = out(P) - cset$ $\diamond$

Definition 3 (Observable channels of a process) The set of observable channels of a process P, notation $chan(P)$, is defined by $chan(P) = in(P) \cup out(P)$. $\diamond$

2.1 Syntactic Restrictions

To guarantee that channels are unidirectional and point-to-point, we have the following syntactic constraints (for any $c \in CHAN$, $x \in VAR$, expression e, etc.):

- For $P_1 \ ; \ P_2$ we require that if P_1 contains $c!e$ then P_2 does not contain $c?x$, and if P_1 contains $c?x$ then P_2 does not contain $c!e$. In other words, $in(P_1) \cap out(P_2) = \emptyset$ and $out(P_1) \cap in(P_2) = \emptyset$.
- For $[\|_{i=1}^{n} b_i \rightarrow P_i]$ we require that, for all $i, j \in \{1, \ldots, n\}$, $i \neq j$, if P_i contains $c!e$ then P_j does not contain $c?x$, that is, $out(P_i) \cap in(P_j) = \emptyset$.
- For $P_1 \| P_2$ we require that if P_1 contains $c!e_1$ then P_2 does not contain $c!e_2$, and if P_1 contains $c?x_1$ then P_2 does not contain $c?x_2$. Equivalently, $in(P_1) \cap in(P_2) = \emptyset$ and $out(P_1) \cap out(P_2) = \emptyset$.

To avoid programs such as $(c?x) \setminus \{c\}$, which would be equivalent to a *random assignment* to x, we require:

- For $P \setminus cset$ we require that $cset \subseteq in(P) \cap out(P)$.

Furthermore, we do not allow parallel processes to share program variables.

- For $P_1 \| P_2$ we require that $var(P_1) \cap var(P_2) = \emptyset$.

3 Denotational Semantics

In this section we define a denotational semantics for the programming language of the previous section. The semantics of a process P, denoted by $\mathcal{O}[\![P]\!]$, associates with P a set of triples consisting of the initial state, the sequence of communications, and the final state characterizing a possible execution of the process.

Define the set $STATE$ of states as the set of mappings from VAR to VAL:

$$STATE = \{\sigma \mid \sigma : VAR \rightarrow VAL\}$$

Thus a state σ assigns to each program variable x a value $\sigma(x)$. For simplicity we do not make a distinction between the semantic and the syntactic domain of values.

Definition 4 (Variant of a state) The *variant* of a state σ with respect to a variable x and a value ϑ, denoted $(\sigma : x \mapsto \vartheta)$, is given by

$$(\sigma : x \mapsto \vartheta)(y) = \begin{cases} \vartheta & \text{if } y \equiv x \\ \sigma(y) & \text{if } y \not\equiv x \end{cases}$$

using '$\equiv$' to denote *syntactic* equality. $\diamond$

In the sequel we assume that we have the standard arithmetical operators $+$, $-$, and $\times$ on VAL. Define the value of an expression e in a state σ, denoted by $\mathcal{E}[\![e]\!](\sigma)$, inductively as follows:

- $\mathcal{E}[\![\mu]\!](\sigma) = \mu$,
- $\mathcal{E}[\![x]\!](\sigma) = \sigma(x)$,
- $\mathcal{E}[\![e_1 + e_2]\!](\sigma) = \mathcal{E}[\![e_1]\!](\sigma) + \mathcal{E}[\![e_2]\!](\sigma)$,
- $\mathcal{E}[\![e_1 - e_2]\!](\sigma) = \mathcal{E}[\![e_1]\!](\sigma) - \mathcal{E}[\![e_2]\!](\sigma)$, and
- $\mathcal{E}[\![e_1 \times e_2]\!](\sigma) = \mathcal{E}[\![e_1]\!](\sigma) \times \mathcal{E}[\![e_2]\!](\sigma)$.

We define when a boolean expression b holds in a state σ, denoted by $\mathcal{B}[\![b]\!](\sigma)$, as

- $\mathcal{B}[\![e_1 = e_2]\!](\sigma)$ iff $\mathcal{E}[\![e_1]\!](\sigma) = \mathcal{E}[\![e_2]\!](\sigma)$,
- $\mathcal{B}[\![e_1 < e_2]\!](\sigma)$ iff $\mathcal{E}[\![e_1]\!](\sigma) < \mathcal{E}[\![e_2]\!](\sigma)$,
- $\mathcal{B}[\![\neg b]\!](\sigma)$ iff not $\mathcal{B}[\![b]\!](\sigma)$, and
- $\mathcal{B}[\![b_1 \vee b_2]\!](\sigma)$ iff $\mathcal{B}[\![b_1]\!](\sigma)$ or $\mathcal{B}[\![b_2]\!](\sigma)$.

We represent a synchronous communication of value $\mu \in VAL$ along channel $c \in CHAN$ by a pair (c, μ), such that $ch((c,\mu)) = c$ and $val((c,\mu)) = \mu$. To denote the behaviour of a process P we use a history θ which is a finite sequence (also called a trace) of the form $\langle (c_1,\mu_1), \ldots, (c_n,\mu_n)\rangle$ of length $len(\theta) = n$, where $n \in \mathbb{N}$, $c_i \in chan(P)$, and $\mu_i \in VAL$, for $1 \le i \le n$. Such a history denotes the communications of P along its observable channels up to some point in an execution. Let $\langle\rangle$ denote the empty history, i.e. the sequence of length 0. The concatenation of two histories $\theta_1 = \langle (c_1,\mu_1), \ldots, (c_k,\mu_k)\rangle$ and $\theta_2 = \langle (d_1,\nu_1), \ldots, (d_l,\nu_l)\rangle$, denoted $\theta_1{}^\wedge\theta_2$, is defined as $\langle (c_1,\mu_1), \ldots, (c_k,\mu_k), (d_1,\nu_1), \ldots, (d_l,\nu_l)\rangle$. We use $\theta^\wedge(c,\mu)$ as an abbreviation of $\theta^\wedge\langle (c,\mu)\rangle$.

Let $TRACE$ be the set of traces, that is, the smallest set such that

- $\langle\rangle \in TRACE$,
- if $\theta \in TRACE$, $c \in CHAN$, and $\mu \in VAL$ then $\theta^\wedge(c,\mu) \in TRACE$.

Definition 5 (Projection) For a trace $\theta \in TRACE$ and a set of channels $cset \subseteq CHAN$, we define the *projection* of θ onto $cset$, denoted by $\theta \uparrow cset$, as the sequence obtained from θ by deleting all records with channels not in $cset$. Formally,

$$\theta \uparrow cset = \begin{cases} \langle\rangle & \text{if } \theta = \langle\rangle \\ \theta_0 \uparrow cset & \text{if } \theta = \theta_0 {}^\wedge (c, \mu) \text{ and } c \notin cset \\ (\theta_0 \uparrow cset)^\wedge (c, \mu) & \text{if } \theta = \theta_0 {}^\wedge (c, \mu) \text{ and } c \in cset \end{cases}$$

◇

Definition 6 (Hiding) Hiding is the complement of projection. Formally, the hiding of a set $cset$ of channels from a trace $\theta \in TRACE$, notation $\theta \backslash cset$, is defined as

$$\theta \backslash cset = \theta \uparrow (CHAN - cset)$$

◇

Definition 7 (Channels occurring in a trace) The set of channels occurring in a trace θ, notation $chan(\theta)$, is defined by

$$chan(\theta) = \{c \in CHAN \mid \theta \uparrow \{c\} \neq \langle\rangle\}$$

◇

Notice that $\theta \uparrow cset = \theta$ iff $chan(\theta) \subseteq cset$, and that $\theta \uparrow \{c\} = \langle\rangle$ iff $c \notin chan(\theta)$.

Definition 8 (Length of a trace) The length of a trace θ, denoted by $len(\theta)$, is defined by

- $len(\langle\rangle) = 0$,
- $len(\theta^\wedge (c, \mu)) = len(\theta) + 1$. ◇

Definition 9 (Prefix) The trace θ_1 is a prefix of a trace θ_2, notation $\theta_1 \preceq \theta_2$, iff there exists a trace θ_3 such that $\theta_1 {}^\wedge \theta_3 = \theta_2$. ◇

Let $\text{STATE}_\perp = \text{STATE} \cup \{\perp\}$. The semantic function $\mathcal{O}$ assigns to a process P a set of triples $(\sigma_0, \theta, \sigma)$ with $\sigma_0 \in \text{STATE}$, $\theta \in \text{TRACE}$, and $\sigma \in \text{STATE}_\perp$. Informally, a triple $(\sigma_0, \theta, \sigma) \in \mathcal{O}[\![P]\!]$ has the following meaning:

- if $\sigma \neq \perp$ then it represents a terminating computation which has performed the communications as described in θ and terminates in state σ, and
- if $\sigma = \perp$ then it represents a point in a computation of P at which P has performed the communications as described in θ but has not yet terminated.

To define the semantic function $\mathcal{O}$ we use the operator PC which yields the prefix closure of a set O of triples:

$$PC(O) \;=\; O \cup \{(\sigma_0, \theta, \perp) \mid \text{there exists a } (\sigma_0, \theta_1, \sigma) \in O \text{ such that } \theta \preceq \theta_1\}$$

For instance, $PC(\{(\sigma_0, \langle(c,1)\rangle, \sigma)\}) = \{(\sigma_0, \langle\rangle, \perp), (\sigma_0, \langle(c,1)\rangle, \perp), (\sigma_0, \langle(c,1)\rangle, \sigma)\}$. Thus, for infinite executions of a process P the set $\mathcal{O}[\![P]\!]$ contains all finite approximations, which is justified since we only deal with safety properties [20].

The semantics of a process P can now inductively be defined as follows:

- $\mathcal{O}[\![\textbf{skip}]\!] = PC(\{(\sigma_0, \langle\rangle, \sigma_0)\})$
- $\mathcal{O}[\![x := e]\!] = PC(\{(\sigma_0, \langle\rangle, (\sigma_0 : x \mapsto \mathcal{E}[\![e]\!](\sigma_0)))\})$

- $\mathcal{O}[\![c!e]\!] = PC(\{(\sigma_0, \langle(c, \mathcal{E}[\![e]\!](\sigma_0))\rangle, \sigma_0)\})$
- $\mathcal{O}[\![c?x]\!] = PC(\{(\sigma_0, \theta, \sigma) \mid$ there exists a value $\mu \in VAL$ such that $\theta = \langle(c, \mu)\rangle$ and $\sigma = (\sigma_0 : x \mapsto \mu)\})$
- $\mathcal{O}[\![P_1 \,;\, P_2]\!] = \{(\sigma_0, \theta, \bot) \mid (\sigma_0, \theta, \bot) \in \mathcal{O}[\![P_1]\!]\}$
  $\cup\ \{(\sigma_0, \theta_1{}^\wedge\theta_2, \sigma) \mid$ there exists a $\sigma_1 \neq \bot$ such that $(\sigma_0, \theta_1, \sigma_1) \in \mathcal{O}[\![P_1]\!]$ and $(\sigma_1, \theta_2, \sigma) \in \mathcal{O}[\![P_2]\!]\}$
- $\mathcal{O}[\![\,[\,[\!]_{i=1}^{n} b_i \rightarrow P_i]\,]\!] = PC(\{(\sigma_0, \langle\rangle, \sigma_0) \mid \neg\mathcal{B}[\![b_1 \vee \ldots \vee b_n]\!](\sigma_0)\})$
  $\cup\ PC(\{(\sigma_0, \theta, \sigma) \mid$ there exists a $k \in \{1, \ldots, n\}$ such that $\mathcal{B}[\![b_k]\!](\sigma_0)$ and $(\sigma_0, \theta, \sigma) \in \mathcal{O}[\![P_k]\!]\})$
- $\mathcal{O}[\![*G]\!] = PC(\{(\sigma_0, \theta, \sigma) \mid$ there exists a $k \in \mathbb{N}$ and a list $(\sigma_0, \theta_1, \sigma_1), \ldots, (\sigma_{k-1}, \theta_k, \sigma_k)$ such that $\theta = \theta_1{}^\wedge \ldots {}^\wedge\theta_k$, $\sigma = \sigma_k$, and for all $i \in \{0, \ldots, k-1\}$: $\sigma_i \neq \bot$, $\mathcal{B}[\![b_G]\!](\sigma_i)$ and $(\sigma_i, \theta_{i+1}, \sigma_{i+1}) \in \mathcal{O}[\![G]\!]$, and if $\sigma_k \neq \bot$ then $\mathcal{B}[\![\neg b_G]\!](\sigma_k)\})$
- $\mathcal{O}[\![P_1 \,\|\, P_2]\!] = \{(\sigma_0, \theta, \sigma) \mid$ for $i = 1, 2$ there exist θ_i, σ_i such that $(\sigma_0, \theta_i, \sigma_i) \in \mathcal{O}[\![P_i]\!]$, and if $\sigma_1 = \bot$ or $\sigma_2 = \bot$ then $\sigma = \bot$ else for all $x \in VAR$
  $$\sigma(x) = \begin{cases} \sigma_i(x) & \text{if } x \in var(P_i) \\ \sigma_0(x) & \text{if } x \notin var(P_1 \,\|\, P_2) \end{cases}$$
  $\theta \uparrow chan(P_i) = \theta_i$, and $\theta \uparrow chan(P_1 \,\|\, P_2) = \theta\}$
- $\mathcal{O}[\![P \backslash cset]\!] = \{(\sigma_0, \theta \backslash cset, \sigma) \mid (\sigma_0, \theta, \sigma) \in \mathcal{O}[\![P]\!]\}$

We conclude this section by defining the set of traces of a process.

Definition 10 (Traces of a process) The traces of a process P, notation $\mathcal{H}[\![P]\!]$, follow from:

$$\mathcal{H}[\![P]\!] \;=\; \{\theta \mid \text{there exist } \sigma_0 \text{ and } \sigma \text{ such that } (\sigma_0, \theta, \sigma) \in \mathcal{O}[\![P]\!]\}$$

◇

4 Assertion Language and Correctness Formulae

As mentioned before, we use a correctness formula P **sat** ϕ to express that process P satisfies safety property ϕ. Informally, since we abstract from the internal states of the processes and focus on the pattern of communications, such a correctness formula expresses that any sequence of communications P may exhibit satisfies ϕ.

Conform the format of traces in the semantics of the previous section, we use communication record expressions such as (c, μ), with $c \in CHAN$ and $\mu \in VAL$, in assertions. We have channel expressions, e.g. using the operator ch which yields the channel of a communication record, and value expressions, including the operator val which yields the value of a communication record and the length operator len. Further, we use in assertions the empty trace, $\langle\rangle$, traces of one record, e.g. $\langle(c, \mu)\rangle$, as well as the concatenation operator $^\wedge$ and the projection operator $\uparrow$. To refer to the communication history of a process we use a special variable h. This variable is not updated explicitly by the process: it refers to a trace from the semantics, and consequently its value will in general change during the execution of the process. Then, we can write specifications like $c!2$ **sat** $h\uparrow\{c\} = \langle\rangle \vee h\uparrow\{c\} = \langle(c, 2)\rangle$. Let $VVAR$, with typical representative v, denote the set of logical value variables ranging over VAL, and let $TVAR$, with characteristic element t, be the set of logical trace variables ranging over $TRACE$. Assume that $VVAR \cap TVAR = \emptyset$.

Table 2 presents the assertion language, with $c \in CHAN$, $\mu \in VAL$, $v \in VVAR$, $t \in TVAR$, and $cset \subseteq CHAN$. Observe that an expression in the assertion language of Table 2 does not refer to program variables since we abstract from the internal state of a process in this report.

Table 2: Syntax of the Assertion Language

Channel expression	$cexp ::=$	$c \mid ch(rexp)$
Value expression	$vexp ::=$	$\mu \mid v \mid val(rexp) \mid len(texp)$
Record expression	$rexp ::=$	$(cexp, vexp) \mid texp(vexp)$
Trace expression	$texp ::=$	$t \mid h \mid \langle\rangle \mid \langle rexp \rangle \mid texp_1 {}^\wedge texp_2 \mid texp \uparrow cset$
Assertion	$\phi ::=$	$cexp_1 = cexp_2 \mid vexp_1 = vexp_2 \mid texp_1 = texp_2 \mid$ $\phi_1 \wedge \phi_2 \mid \neg\phi \mid \exists v : \phi \mid \exists t : \phi$

Definition 11 (Abbreviations) Henceforth we use the following abbreviations:

- $ch(cexp, vexp) \equiv ch((cexp, vexp))$
- $val(cexp, vexp) \equiv val((cexp, vexp))$
- $texp \uparrow cexp \equiv texp \uparrow \{cexp\}$
- $rexp_1 = rexp_2 \equiv ch(rexp_1) = ch(rexp_2) \wedge val(rexp_1) = val(rexp_2)$
- $texp \setminus cset \equiv texp \uparrow (CHAN - cset)$
- $last(texp) \equiv texp(len(texp))$
- $texp_1 \preceq texp_2 \equiv \exists t : texp_1{}^\wedge t = texp_2$
  This expresses that $texp_1$ is a prefix of $texp_2$.
- $texp_1 \preceq^n texp_2 \equiv \exists t : len(t) \leq n : texp_1{}^\wedge t = texp_2$
  To assert that $texp_1$ is a prefix of $texp_2$ which is at most n records shorter.
- $texp_1 \prec texp_2 \equiv texp_1 \preceq texp_2 \wedge texp_1 \neq texp_2$
  To denote that $texp_1$ is a *strict* prefix of $texp_2$.
- $texp_1 \prec^n texp_2 \equiv \exists t : 1 < len(t) \leq n : texp_1{}^\wedge t = texp_2$
  To express that $texp_1$ is a strict prefix of $texp_2$ which is at most n records shorter.
- $texp[vexp] \equiv texp(1)^\wedge \ldots {}^\wedge texp(vexp)$
  To refer to the prefix of $texp$ that has length $vexp$.
- $texp_1 \trianglelefteq texp_2 \equiv \begin{cases} \exists t : t^\wedge texp_1 \preceq texp_2 & \text{if } len(texp_1) \leq 1 \\ \forall i : \forall j > i : \exists t_1, t_2 : t_1{}^\wedge texp_1(i)^\wedge t_2{}^\wedge texp_1(j) \preceq texp_2 & \text{if } len(texp_1) > 1 \end{cases}$
  To denote that $texp_1$ is a (not necessarily contiguous) subsequence of $texp_2$. ◇

Furthermore, we use the standard abbreviations $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$, and $\phi_1 \rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$. Also, for natural numbers x and y, we use the relations $x \leq^n y$ and $x <^n y$ to denote that $0 \leq y - x \leq n$, respectively that $0 < y - x \leq n$.

Definition 12 (Sequence of values) For a trace $texp$,

$$Val(texp) = \begin{cases} \langle\rangle & \text{if } texp = \langle\rangle \\ Val(texp_0)^\wedge v & \text{if } texp = texp_0{}^\wedge(c, v) \end{cases}$$

◇

Example 1 (Medium) Consider a medium M that accepts messages via m_{in} and delivers them via m_{out} in first in-first out order. To specify that M has a capacity of one message, we use

$$M \textbf{ sat } Val(h\uparrow m_{out}) \preceq^1 Val(h\uparrow m_{in})$$

$\triangle$

For an assertion ϕ we define a set $chan(\phi)$ of channel names such that ϕ may only be invalidated by a communication on the channels of $chan(\phi)$.

Definition 13 (Channels in an assertion) For an assertion ϕ we inductively define the set $chan(\phi)$ of channels such that $c \in chan(\phi)$ iff a communication along c might affect the validity of ϕ.

- $chan(c) = \emptyset$
- $chan(ch(rexp)) = chan(rexp)$
- $chan(\mu) = chan(v) = \emptyset$
- $chan(val(rexp)) = chan(rexp)$
- $chan(len(texp)) = chan(texp)$
- $chan((cexp, vexp)) = chan(cexp) \cup chan(vexp)$
- $chan(texp(vexp)) = chan(texp) \cup chan(vexp)$
- $chan(t) = \emptyset$
- $chan(h) = CHAN$
- $chan(\langle\rangle) = \emptyset$
- $chan(\langle rexp\rangle) = chan(rexp)$
- $chan(texp_1{}^\wedge texp_2) = chan(texp_1) \cup chan(texp_2)$
- $chan(texp\uparrow cset) = chan(texp) \cap cset$
- $chan(cexp_1 = cexp_2) = chan(cexp_1) \cup chan(cexp_2)$
- $chan(vexp_1 = vexp_2) = chan(vexp_1) \cup chan(vexp_2)$
- $chan(texp_1 = texp_2) = chan(texp_1) \cup chan(texp_2)$
- $chan(\phi_1 \wedge \phi_2) = chan(\phi_1) \cup chan(\phi_2)$
- $chan(\neg\phi) = chan(\exists v : \phi) = chan(\exists t : \phi) = chan(\phi)$ $\diamond$

Next we define the meaning of assertions. An assertion is interpreted with respect to a pair (θ, γ). Trace θ gives h its value, and environment γ interprets the logical variables of $VVAR \cup TVAR$. We use the special symbol $\dagger$ to deal with the interpretation of $texp(vexp)$ where index $vexp$ is not a positive natural number, or if it is greater than the length of $texp$. The value of an expression is undefined whenever a subexpression yields $\dagger$. We define the value of a channel expression $cexp$ in the trace θ, and an environment γ, denoted by $\mathcal{C}[\![cexp]\!](\theta, \gamma)$, yielding a value in $CHAN \cup \{\dagger\}$, the value of a value expression $vexp$ in the trace θ, and an environment γ, denoted by $\mathcal{V}[\![vexp]\!](\theta, \gamma)$, yielding a value in $VAL \cup \{\dagger\}$, the value of a record expression $rexp$ in the trace θ, and an environment γ, denoted by $\mathcal{R}[\![rexp]\!](\theta, \gamma)$, yielding a value in $CHAN \times VAL \cup \{\dagger\}$, and the value of a trace expression $texp$ for trace θ, and an environment γ, denoted by $\mathcal{T}[\![texp]\!](\theta, \gamma)$, yielding a value in $TRACE \cup \{\dagger\}$.

- $\mathcal{C}[\![c]\!](\theta,\gamma) = c$

- $\mathcal{C}[\![ch(rexp)]\!](\theta,\gamma) = \begin{cases} \dagger & \text{iff } \mathcal{R}[\![rexp]\!](\theta,\gamma) = \dagger \\ c & \text{iff there exists a } \mu \text{ such that } \mathcal{R}[\![rexp]\!](\theta,\gamma) = (c,\mu) \end{cases}$

- $\mathcal{V}[\![\mu]\!](\theta,\gamma) = \mu$

- $\mathcal{V}[\![v]\!](\theta,\gamma) = \gamma(v)$

- $\mathcal{V}[\![val(rexp)]\!](\theta,\gamma) = \begin{cases} \dagger & \text{iff } \mathcal{R}[\![rexp]\!](\theta,\gamma) = \dagger \\ \mu & \text{iff there exists a } c \text{ such that } \mathcal{R}[\![rexp]\!](\theta,\gamma) = (c,\mu) \end{cases}$

- $\mathcal{V}[\![len(texp)]\!](\theta,\gamma) = \begin{cases} \dagger & \text{iff } \mathcal{T}[\![texp]\!](\theta,\gamma) = \dagger \\ len(\mathcal{T}[\![texp]\!](\theta,\gamma)) & \text{otherwise} \end{cases}$

- $\mathcal{R}[\![(cexp, vexp)]\!](\theta,\gamma) = \begin{cases} \dagger & \text{iff } \mathcal{C}[\![cexp]\!](\theta,\gamma) = \dagger \text{ or } \mathcal{V}[\![vexp]\!](\theta,\gamma) = \dagger \\ (\mathcal{C}[\![cexp]\!](\theta,\gamma), \mathcal{V}[\![vexp]\!](\theta,\gamma)) & \text{otherwise} \end{cases}$

- $\mathcal{R}[\![texp(vexp)]\!](\theta,\gamma) = \begin{cases} (c,\mu) & \text{iff there exist } \theta_1 \text{ and } \theta_2 \text{ such that } len(\theta_1) = \mathcal{V}[\![vexp]\!](\theta,\gamma) - 1 \\ & \quad \text{and } \mathcal{T}[\![texp]\!](\theta,\gamma) = \theta_1{}^\wedge(c,\mu)^\wedge\theta_2 \\ \dagger & \text{otherwise} \end{cases}$

- $\mathcal{T}[\![t]\!](\theta,\gamma) = \gamma(t)$

- $\mathcal{T}[\![h]\!](\theta,\gamma) = \theta$

- $\mathcal{T}[\![\langle\rangle]\!](\theta,\gamma) = \langle\rangle$

- $\mathcal{T}[\![\langle rexp\rangle]\!](\theta,\gamma) = \begin{cases} \dagger & \text{iff } \mathcal{R}[\![rexp]\!](\theta,\gamma) = \dagger \\ \langle(c,\mu)\rangle & \text{iff } \mathcal{R}[\![rexp]\!](\theta,\gamma) = (c,\mu) \end{cases}$

- $\mathcal{T}[\![texp_1{}^\wedge texp_2]\!](\theta,\gamma) = \begin{cases} \dagger & \text{iff } \mathcal{T}[\![texp_1]\!](\theta,\gamma) = \dagger \text{ or} \\ & \quad \mathcal{T}[\![texp_2]\!](\theta,\gamma) = \dagger \\ \mathcal{T}[\![texp_1]\!](\theta,\gamma)^\wedge\mathcal{T}[\![texp_2]\!](\theta,\gamma) & \text{otherwise} \end{cases}$

- $\mathcal{T}[\![texp \uparrow cset]\!](\theta,\gamma) = \begin{cases} \dagger & \text{iff } \mathcal{T}[\![texp]\!](\theta,\gamma) = \dagger \\ \mathcal{T}[\![texp]\!](\theta\uparrow cset,\gamma)\uparrow cset & \text{otherwise} \end{cases}$

Definition 14 (Variant of an environment) The *variant* of an environment γ with respect to a logical variable l and a value λ, denoted $(\gamma : l \mapsto \lambda)$, is given by

$$(\gamma : l \mapsto \lambda)(m) = \begin{cases} \lambda & \text{if } m \equiv l \\ \gamma(m) & \text{if } m \not\equiv l \end{cases}$$

◇

We inductively define when an assertion ϕ holds for a trace θ, and an environment γ, denoted by $(\theta,\gamma) \models \phi$. To avoid the complexity of a three-valued logic, an equality predicate is interpreted *strictly* with respect to $\dagger$, that is, it has truthvalue false if it contains some expression that has an undefined value.

- $(\theta,\gamma) \models cexp_1 = cexp_2$ iff $\mathcal{C}[\![cexp_1]\!](\theta,\gamma) = \mathcal{C}[\![cexp_2]\!](\theta,\gamma)$ and $\mathcal{C}[\![cexp_1]\!](\theta,\gamma) \neq \dagger$

- $(\theta,\gamma) \models vexp_1 = vexp_2$ iff $\mathcal{V}[\![vexp_1]\!](\theta,\gamma) = \mathcal{V}[\![vexp_2]\!](\theta,\gamma)$ and $\mathcal{V}[\![vexp_1]\!](\theta,\gamma) \neq \dagger$

- $(\theta,\gamma) \models texp_1 = texp_2$ iff $\mathcal{T}[\![texp_1]\!](\theta,\gamma) = \mathcal{T}[\![texp_2]\!](\theta,\gamma)$ and $\mathcal{T}[\![texp_1]\!](\theta,\gamma) \neq \dagger$

- $(\theta,\gamma) \models \phi_1 \wedge \phi_2$ iff $(\theta,\gamma) \models \phi_1$ and $(\theta,\gamma) \models \phi_2$

- $(\theta, \gamma) \models \neg\phi$ iff not $(\theta, \gamma) \models \phi$
- $(\theta, \gamma) \models \exists v : \phi$ iff there exists a value μ such that $(\theta, (\gamma : v \mapsto \mu)) \models \phi$
- $(\theta, \gamma) \models \exists t : \psi$ iff there exists a trace $\widehat{\theta}$ such that $(\theta, (\gamma : t \mapsto \widehat{\theta})) \models \phi$

Definition 15 (Validity of an assertion) An assertion ϕ is *valid*, which we denote by $\models \phi$, iff for all θ and γ : $(\theta, \gamma) \models \phi$ ◇

We conclude this section by defining when a correctness formula P **sat** ϕ is valid.

Definition 16 (Validity of a correctness formula) For a process P and an assertion ϕ a correctness formula P **sat** ϕ is *valid*, denoted by $\models P$ **sat** ϕ, iff

$$\text{for all } \gamma \text{ and all } \theta \in \mathcal{H}[\![P]\!] : (\theta, \gamma) \models \phi$$

◇

5 Incorporating Fault Hypotheses

Based on a particular fault hypothesis, the set of behaviours that characterize a process is expanded. To keep such an expansion manageable, the fault hypothesis χ of a process P is formalized as a predicate, expressed in a first order assertion language, whose only free variables are h and h_{old}, representing a reflexive relation between the normal and acceptable histories of a process. The interpretation is such that h_{old} represents a normal history of process P, whereas h is an *acceptable* history of P with respect to χ. Such relations enable one to abstract from the precise nature of a fault and to focus on the abnormal behaviour it causes. For instance, a possible history h of process *Square*, which alternately inputs an integer via the observable channel *in* and outputs its square via the observable channel *out*, may be $\langle(in, 1), (out, 1), (in, 3), (out, 9)\rangle$. The exceptional behaviour resulting from *Square*'s output channel becoming transiently stuck at zero can be defined using a predicate *StuckAtZero* asserting that h_{old} and h are equally long, if the ith element of h_{old} records an *in* communication then it is equal to the ith element of h, and if the ith element of h_{old} records an *out* communication then so does the ith element of h, but in the latter case the communicated value recorded in h is equal to the original value or it is equal to zero. Using, similar to [17], the construct $(Square \wr StuckAtZero)$ to indicate execution of process *Square* under the assumption of *StuckAtZero*, we still have $\langle(in, 1), (out, 1), (in, 3), (out, 9)\rangle \in \mathcal{H}[\![(Square \wr StuckAtZero)]\!]$, but also, for instance, $\langle(in, 1), (out, 1), (in, 3), (out, 0)\rangle \in \mathcal{H}[\![(Square \wr StuckAtZero)]\!]$. Our goal is to examine whether it is possible to develop a compositional proof theory based on the idea of transforming histories; for the time being it is not our aim to find a logic to express fault hypotheses as elegantly as possible.

Example 2 (Stuck at zero) The before mentioned predicate *StuckAtZero* can formally be defined as follows:

$$\begin{array}{lll} StuckAtZero \triangleq & len(h_{old}) = len(h) & \\ & \wedge\ \forall i : 1 \leq i \leq len(h) : & ch(h\uparrow\{in, out\}(i)) = ch(h_{old}\uparrow\{in, out\}(i)) \\ & & \wedge\ val(h\uparrow in(i)) = val(h_{old}\uparrow in(i)) \\ & & \wedge\quad val(h\uparrow out(i)) = val(h_{old}\uparrow out(i)) \\ & & \quad \vee\ val(h\uparrow out(i)) = 0 \end{array}$$

△

By not specifying the value part of an *out* record in h, allowing it to be any element of VAL, we can formalize corruption.

Example 3 (Corruption) We formalize corruption as follows:

$$\begin{array}{lll} Cor \triangleq & len(h_{old}) = len(h) & \\ & \wedge\ \forall i : 1 \leq i \leq len(h) : & ch(h\uparrow\{in, out\}(i)) = ch(h_{old}\uparrow\{in, out\}(i)) \\ & & \wedge\ val(h\uparrow in(i)) = val(h_{old}\uparrow in(i)) \end{array}$$

$\triangle$

Example 4 (Loss) Consider medium M of Example 1. To formalize the hypothesis that M may lose messages we define:

$$\begin{array}{ll} Loss \triangleq & h\uparrow\{m_{in}, m_{out}\} \trianglelefteq h_{old}\uparrow\{m_{in}, m_{out}\} \\ & \wedge\ h\uparrow m_{in} = h_{old}\uparrow m_{in} \end{array}$$

$\triangle$

We extend the assertion language with trace expression term h_{old}. Sentences of the extended language are called *transformation expressions*, with typical representative ψ. A transformation expression is interpreted with respect to a triple $(\theta_0, \theta, \gamma)$. Trace θ_0 gives h_{old} its value, and, in conformity with the foregoing, trace θ gives h its value, and environment γ interprets the logical variables of $VVAR \cup TVAR$. The meaning of assertions, as defined on page 9, can easily be adapted for transformation expressions; the only new clause is $\mathcal{T}[\![h_{old}]\!](\theta_0, \theta, \gamma) = \theta_0$. Similarly, the channels occurring in an transformation expression are defined as in Definition 13 with the extra clause $chan(h_{old}) = CHAN$.

Since the term h_{old} does not occur in assertions, the following lemma is trivial.

Lemma 1 (Correspondence) For assertion ϕ for all θ_0 $(\theta_0, \theta, \gamma) \models \phi$ iff $(\theta, \gamma) \models \phi$. $\circ$

In this section we define the trace semantics $\mathcal{H}[\![(FP \wr \chi)]\!]$ of failure prone process $(FP \wr \chi)$, that is, process FP under assumption of fault hypothesis χ. A fault hypothesis χ is a fault assertion which, since it formalizes a relation between the normal and the acceptable behaviour of a process, represents a reflexive relation between h and h_{old}. Formally,

- $\models \chi[h_{old}/h]$

Furthermore, we require a fault hypothesis χ to be prefix closedness preserving, that is, we require

- $\models \chi \wedge t \prec h \rightarrow \exists t_{old} \preceq h_{old} : \chi[t/h, t_{old}/h_{old}]$

Using P to denote a process expressed in the programming language of Section 2, we define the syntax of our extended programming language in Table 3.

Table 3: Extended Syntax of the Programming Language

Failure Prone Process	$FP ::=$	P	$FP_1 \parallel FP_2$	$FP \backslash cset$	$(FP \wr \chi)$

We require, in $(FP \wr \chi)$, that $chan(\chi) \subseteq chan(FP)$. Hence, $chan((FP \wr \chi)) = chan(FP)$, and, as before, $chan(FP_1 \parallel FP_2) = chan(FP_1) \cup chan(FP_2)$, and $chan(FP \backslash cset) = chan(FP) - cset$.

As we are only interested in the traces of a process, the semantics of a failure prone process FP is inductively defined as follows:

- $\mathcal{H}[\![FP_1 \,\|\, FP_2]\!] = \{\ \theta \mid \text{for } i = 1,2,\ \theta{\uparrow} chan(FP_i) \in \mathcal{H}[\![FP_i]\!], \text{ and } \theta{\uparrow} chan(FP_1 \,\|\, FP_2) = \theta\ \}$
- $\mathcal{H}[\![FP \backslash cset]\!] = \{\ \theta \backslash cset \mid \theta \in \mathcal{H}[\![FP]\!]\ \}$
- $\mathcal{H}[\![(FP \wr \chi)]\!] = \{\ \theta \mid \text{there exists a } \theta_0 \in \mathcal{H}[\![FP]\!] \text{ such that, for all } \gamma,\ (\theta_0, \theta, \gamma) \models \chi, \text{ and } \theta{\uparrow} chan(FP) = \theta\ \}$

The set $\mathcal{H}[\![(FP \wr \chi)]\!]$ represents the *acceptable* behaviour of FP with respect to fault hypothesis χ. Notice that, $\mathcal{H}[\![FP]\!] = \mathcal{H}[\![(FP \wr h{\uparrow} chan(FP) = h_{old}{\uparrow} chan(FP))]\!]$, and that, because of the reflexivity of χ, $\mathcal{H}[\![FP]\!] \subseteq \mathcal{H}[\![(FP \wr \chi)]\!]$. Also, observe that the semantics is defined such that if $\theta \in \mathcal{H}[\![FP]\!]$ then $chan(\theta) \subseteq chan(FP)$.

Lemma 2 (Prefix closedness) If $\theta \in \mathcal{H}[\![FP]\!]$ and $\theta' \preceq \theta$ then $\theta' \in \mathcal{H}[\![FP]\!]$.

Proof. See Appendix A. □

Definition 17 (Composite transformation expression) For transformation expressions ψ_1 and ψ_2, the composite transformation expression $(\psi_1 \wr \psi_2)$ is defined as follows

$$(\psi_1 \wr \psi_2) \ \triangleq\ \exists t : \psi_1[t/h] \ \wedge\ \psi_2[t/h_{old}]$$

where t must be fresh. ◇

From this definition we easily obtain the following lemma.

Lemma 3 (Composite fault hypothesis)

$$\mathcal{H}[\![(FP \wr (\chi_1 \wr \chi_2))]\!] \ =\ \mathcal{H}[\![((FP \wr \chi_1) \wr \chi_2)]\!]$$

Proof. See Appendix B. □

The following lemmas are easy to prove by structural induction.

Lemma 4 (Projection) Consider $cset \subseteq CHAN$ and transformation expression ψ. If $chan(\psi) \subseteq cset$ then, for all θ_0, θ, and γ

(a) $(\theta_0, \theta, \gamma) \models \psi$ iff $(\theta_0, \theta{\uparrow} cset, \gamma) \models \psi$

(b) $(\theta_0, \theta, \gamma) \models \psi$ iff $(\theta_0{\uparrow} cset, \theta, \gamma) \models \psi$ ○

Lemma 5 (Substitution) Consider transformation expression ψ.

(a) $(\theta_0, \theta, \gamma) \models \psi[texp/h]$ iff $(\theta_0, \mathcal{T}[\![texp]\!](\theta_0, \theta, \gamma), \gamma) \models \psi$

(b) $(\theta_0, \theta, \gamma) \models \psi[texp/h_{old}]$ iff $(\mathcal{T}[\![texp]\!](\theta_0, \theta, \gamma), \theta, \gamma) \models \psi$ ○

Since the interpretation of assertions has not changed, the validity of correctness formula FP **sat** ϕ is defined as in Definition 16, with P replaced by FP.

Definition 18 (Validity of a correctness formula) For a failure prone process FP and an assertion ϕ a correctness formula FP **sat** ϕ is *valid*, denoted by $\models FP$ **sat** ϕ, iff

$$\text{for all } \gamma \text{ and all } \theta \in \mathcal{H}[\![FP]\!] : (\theta, \gamma) \models \phi$$

◇

6 A Compositional Network Proof Theory

In this section we present a compositional proof theory to prove safety properties of networks of processes. Since we focus on the relation between fault hypotheses and concurrency, we have abstracted from the internal states of the processes and do not give rules for atomic statements, nor sequential composition. Such rules could be formulated by using an extended assertion language which includes program variables and a denotation to indicate termination (e.g. [20]).

The following rules are standard:

Rule 6.1 (Consequence)

$$\frac{FP \textbf{ sat } \phi_1,\ \phi_1 \rightarrow \phi_2}{FP \textbf{ sat } \phi_2}$$

Rule 6.2 (Conjunction)

$$\frac{FP \textbf{ sat } \phi_1,\ FP \textbf{ sat } \phi_2}{FP \textbf{ sat } \phi_1 \wedge \phi_2}$$

Rule 6.3 (Invariance)

$$\frac{cset \cap chan(FP) = \emptyset}{FP \textbf{ sat } h \uparrow cset = \langle\rangle}$$

From this rule we can derive the following lemma.

Lemma 6 (Invariance)

$$P \textbf{ sat } h \setminus chan(P) = \langle\rangle$$

○

Rule 6.4 (Parallel composition)

$$\frac{FP_1 \textbf{ sat } \phi_1,\ FP_2 \textbf{ sat } \phi_2}{FP_1 \| FP_2 \textbf{ sat } \phi_1 \wedge \phi_2}$$

provided that $chan(\phi_1) \cap chan(FP_2) \subseteq chan(FP_1)$ and $chan(\phi_2) \cap chan(FP_1) \subseteq chan(FP_2)$, i.e. if the assertion that holds for one process refers to channels of the other process then this concerns channels connecting the two processes (cf. [8], [20]). Note that, as a consequence of this restriction, any occurrence of h in specification ϕ_i of process FP_i should be projected onto a subset of $chan(FP_i)$. Recall that we do not allow shared variables.

Rule 6.5 (Hiding)

$$\frac{FP \textbf{ sat } \phi,\ chan(\phi) \cap cset = \emptyset}{FP \setminus cset \textbf{ sat } \phi}$$

Next, we formulate the rule for the introduction of a fault hypothesis.

Rule 6.6 (Fault hypothesis introduction)

$$\frac{FP \textbf{ sat } \phi}{(FP \wr \chi) \textbf{ sat } (\phi \wr \chi)}$$

Observe that since ϕ is an assertion, h_{old} does not occur in ϕ, and hence also $(\phi \wr \chi)$ is an assertion.

Example 5 (Loss) Consider the medium of Example 4. By (Fault hypothesis introduction),

$$\begin{aligned}(M \wr Loss) \textbf{ sat } \exists t :\ & (Val(h\uparrow m_{out}) \preceq^1 Val(h\uparrow m_{in}))[t/h] \\ & \wedge\ (h\uparrow\{m_{in}, m_{out}\} \trianglelefteq h_{old}\uparrow\{m_{in}, m_{out}\}\ \wedge\ h\uparrow m_{in} = h_{old}\uparrow m_{in})[t/h_{old}]\end{aligned}$$

which reduces to

$$(M \wr Loss) \textbf{ sat } \exists t: \quad Val(t \uparrow m_{out}) \preceq^1 Val(t \uparrow m_{in}) \\ \wedge\ h \uparrow \{m_{in}, m_{out}\} \trianglelefteq t \uparrow \{m_{in}, m_{out}\} \ \wedge\ h \uparrow m_{in} = t \uparrow m_{in}$$

Now, for instance, by $h \uparrow \{m_{in}, m_{out}\} \trianglelefteq t \uparrow \{m_{in}, m_{out}\}$, we have, obviously, $h \uparrow m_{out} \trianglelefteq t \uparrow m_{out}$, which, since $Val(t \uparrow m_{out}) \preceq^1 Val(t \uparrow m_{in})$, implies $Val(h \uparrow m_{out}) \trianglelefteq Val(t \uparrow m_{in})$. Then, by $t \uparrow m_{in} = h \uparrow m_{in}$, we obtain

$$(M \wr Loss) \textbf{ sat } Val(h \uparrow m_{out}) \trianglelefteq Val(h \uparrow m_{in})$$

Also, since $Val(t \uparrow m_{out}) \preceq^1 Val(t \uparrow m_{in})$, we have $\forall i : ch(t'(i)) = m_{out} : val(t'(i)) = val(last(t'[i] \uparrow m_{in}))$, with $t' = t \uparrow \{m_{in}, m_{out}\}$. Because $h \uparrow \{m_{in}, m_{out}\} \trianglelefteq t \uparrow \{m_{in}, m_{out}\}$ whilst $h \uparrow m_{in} = t \uparrow m_{in}$, this leads to

$$(M \wr Loss) \textbf{ sat } \forall i: \ ch(h \uparrow \{m_{in}, m_{out}\}(i)) = m_{out}: \\ val(h \uparrow \{m_{in}, m_{out}\}(i)) = val(last(h \uparrow \{m_{in}, m_{out}\}[i] \uparrow m_{in}))$$

$\triangle$

Finally, since the semantics is prefix closed we have the following rule.

Rule 6.7 (Prefixing)

$$\frac{FP \textbf{ sat } \phi}{FP \textbf{ sat } \forall t \preceq h : \phi[t/h]}$$

7 Example I: Triple Modular Redundancy

Consider the triple modular redundant component of Figure 1. It consists of three identical components C_j, $j = 1, 2, 3$, an input triplicating component *In*, and a component *Voter* that determines the ultimate output. The intuition of the triple modular redundancy paradigm is that 3 identical components operate on the same input and send their output to a voter which outputs the result of a majority vote. Clearly, the failure of one component can be masked, and the failure of two or all three components can be detected, as long as they do not fail identically.

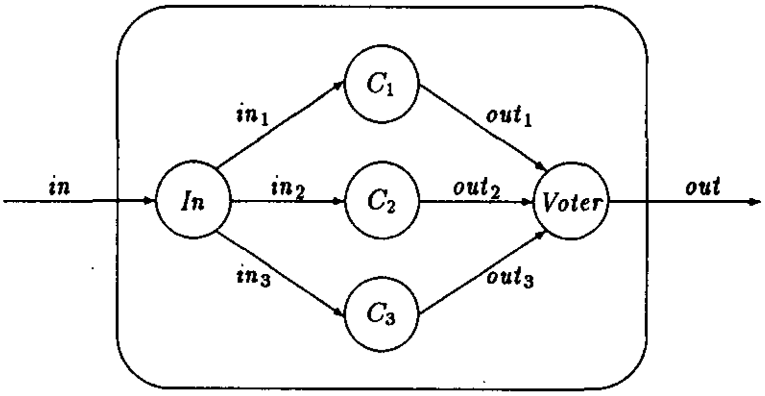

Figure 1: Triple modular redundant component

Definition 19 (Abbreviations) Throughout this section we use the following abbreviations:

- $c(i) \;\equiv\; val((h \uparrow c)(i))$
- $c^{old}(i) \;\equiv\; val((h_{old} \uparrow c)(i))$
- $c^{t}(i) \;\equiv\; val((t \uparrow c)(i))$ $\diamond$

Each component alternately awaits an input message from in, performs some computation f, and produces an output message on out. We abstract from the implementation details of a component; we only consider the following specification.

C_j **sat** $\forall i : out(i) = f(in(i))$

The voter awaits the output of each of the 3 components, takes a majority vote, and outputs the result of that vote. Formally,

$Voter$ **sat** $\forall i, v : out(i) = v \leftrightarrow (\exists k, l : k \neq l : out_k(i) = out_l(i) = v)$

Finally, component In conforms to

In **sat** $\forall i, j : in_j(i) = in(i)$

The voter produces the desired output if at least two of the values output by C_1, C_2, and C_3 are correct. Hence, to mask the failure of one component, at most one of the values output by C_1, C_2, and C_3 may be corrupted for each vote. This assumption is formalized by the following fault hypothesis.

$$\begin{array}{lll} Cor^{\leq 1} & \triangleq & \forall i : \exists k, l : k \neq l : out_k(i) = out_k^{old}(i) \wedge out_l(i) = out_l^{old}(i) \\ & & \wedge\ h \uparrow \{in_1, in_2, in_3\} = h_{old} \uparrow \{in_1, in_2, in_3\} \end{array}$$

We show that, given this assumption, the system $In \| ((C_1 \| C_2 \| C_3) \wr Cor^{\leq 1}) \| Voter$ produces the desired output, that is, hiding internal channels we prove

$$(In \| ((C_1 \| C_2 \| C_3) \wr Cor^{\leq 1}) \| Voter) \setminus \{in_1, in_2, in_3, out_1, out_2, out_3\} \ \ \textbf{sat} \ \ \forall i : out(i) = f(in(i))$$

Proof. By (Parallel Composition)

$$C_1 \| C_2 \| C_3 \ \ \textbf{sat} \ \ \bigwedge_{j=1}^{3} \forall i : out_j(i) = f(in_j(i))$$

By (Fault Hypothesis Introduction)

$$((C_1 \| C_2 \| C_3) \wr Cor^{\leq 1}) \ \ \textbf{sat} \ \ \exists t : (\bigwedge_{j=1}^{3} \forall i : out_j(i) = f(in_j(i)))[t/h] \wedge Cor^{\leq 1}[t/h_{old}]$$

which is, by definition, equivalent to

$$\begin{array}{lll} ((C_1 \| C_2 \| C_3) \wr Cor^{\leq 1}) \ \ \textbf{sat} \ \ \exists t : & & \bigwedge_{j=1}^{3} \forall i : out_j^t(i) = f(in_j^t(i)) \\ & \wedge & \forall i : \exists k, l : k \neq l : out_k(i) = out_k^t(i) \wedge out_l(i) = out_l^t(i) \\ & \wedge & h \uparrow \{in_1, in_2, in_3\} = t \uparrow \{in_1, in_2, in_3\} \end{array}$$

and, thus, by (Consequence),

$$((C_1\|C_2\|C_3)\wr Cor^{\leq 1}) \textbf{ sat } \exists t: \begin{array}{l} \forall i: \exists k,l: k \neq l: out_k(i) = f(in_k^t(i)) \wedge out_l(i) = f(in_l^t(i)) \\ \wedge\; h\uparrow\{in_1, in_2, in_3\} = t\uparrow\{in_1, in_2, in_3\} \end{array}$$

Using $h\uparrow\{in_1, in_2, in_3\} = t\uparrow\{in_1, in_2, in_3\}$, we have that $\bigwedge_{j=1}^{3} \forall i: t\uparrow in_j(i) = h\uparrow in_j(i)$. Hence

$$((C_1\|C_2\|C_3)\wr Cor^{\leq 1}) \textbf{ sat } \forall i: \exists k,l: k \neq l: out_k(i) = f(in_k(i)) \wedge out_l(i) = f(in_l(i))$$

By (Parallel Composition), we get

$$In\|((C_1\|C_2\|C_3)\wr Cor^{\leq 1}) \textbf{ sat } \begin{array}{l} \forall i: \exists k,l: k \neq l: out_k(i) = f(in_k(i)) \wedge out_l(i) = f(in_l(i)) \\ \wedge\; \forall i,j: in_j(i) = in(i) \end{array}$$

Hence, by (Consequence),

$$In\|((C_1\|C_2\|C_3)\wr Cor^{\leq 1}) \textbf{ sat } \forall i: \exists k,l: k \neq l: out_k(i) = f(in(i)) \wedge out_l(i) = f(in(i))$$

and thus

$$In\|((C_1\|C_2\|C_3)\wr Cor^{\leq 1}) \textbf{ sat } \forall i: \exists k,l: k \neq l: out_k(i) = out_l(i) = f(in(i))$$

By (Parallel Composition) and (Consequence), we add the voter and obtain the relation between in and out.

$$In\|((C_1\|C_2\|C_3)\wr Cor^{\leq 1})\|Voter \textbf{ sat } \forall i: out(i) = f(in(i))$$

Finally, by (Hiding), we obtain

$$(In\|((C_1\|C_2\|C_3)\wr Cor^{\leq 1})\|Voter)\backslash\{in_1, in_2, in_3, out_1, out_2, out_3\} \textbf{ sat } \forall i: out(i) = f(in(i))$$

□

8 Example II: The Alternating Bit Protocol

The alternating bit protocol [2], extended with timers, is a simple way to achieve communication over a medium that may lose messages. Consider the duplex communication medium of Figure 2, where A and M are media with fault hypothesis *loss* as already discussed in Example 5.

Sender S accepts via in data from the environment, appends a bit to it, and sends it via m_{in}; the value of the bit alternates for successive messages, starting with 1. Receiver R awaits a message via m_{out}, and sends the bit via a_{in} as an acknowledgement; R only passes the data via out to the environment if the value of the message's bit differs from the value of the previous message's bit, or if it is the first message. Consequently, messages along M consist of data-bit pairs (d, b), such that $dat((d, b)) = d$ and $bit((d, b)) = b$. Medium A transmits bits. Under the alternating bit protocol, S keeps sending a message via m_{in} until its acknowledgement arrives via a_{out}. The alternating bit ensures that R can identify duplicates.

In this section we will prove that $ABP \triangleq S \,\|\, (M \wr Loss) \,\|\, (A \wr Loss) \,\|\, R$ satisfies the safety property that $Val(h\uparrow out) \preceq Val(h\uparrow in)$. We use the following functions:

Definition 20 (Removal of duplicate messages) For a trace $texp$ of $chan(M) \times \alpha M$ records,

$$RDMsg(texp) = \begin{cases} \langle\rangle & \text{if } texp = \langle\rangle \\ RDMsg(texp_0) & \text{if } texp = texp_0\,\hat{}\,(c,(d,b)) \text{ and } b = bit(val(last(texp_0))) \\ RDMsg(texp_0)\,\hat{}\,(c,(d,b)) & \text{if } texp = texp_0\,\hat{}\,(c,(d,b)) \text{ and } b \neq bit(val(last(texp_0))) \end{cases}$$

◇

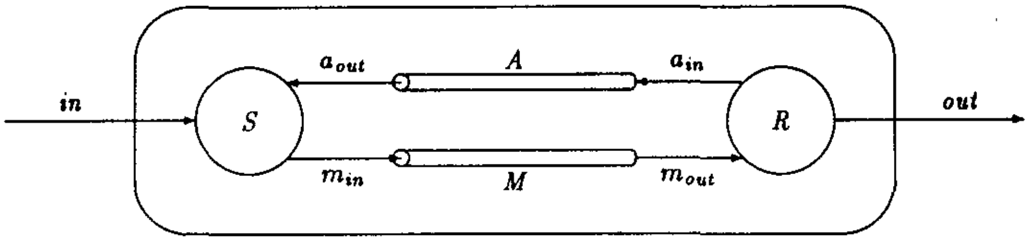

Figure 2: Duplex communication medium

Definition 21 (Removal of duplicate acknowledgements) For a trace *texp* that consists only of $chan(A) \times \alpha A$ records,

$$RDAck(texp) = \begin{cases} \langle\rangle & \text{if } texp = \langle\rangle \\ RDAck(texp_0) & \text{if } texp = texp_0{}^\wedge(c,b) \text{ and } b = val(last(texp_0)) \\ RDAck(texp_0)^\wedge(c,b) & \text{if } texp = texp_0{}^\wedge(c,b) \text{ and } b \neq val(last(texp_0)) \end{cases}$$

◇

Definition 22 (Sequence of data) For a trace *texp* of $chan(M) \times \alpha M$ records,

$$Dat(texp) = \begin{cases} \langle\rangle & \text{if } texp = \langle\rangle \\ Msg(texp_0)^\wedge d & \text{if } texp = texp_0{}^\wedge(c,(d,b)) \end{cases}$$

◇

Definition 23 (Sequence of bits) For a trace *texp* of $chan(M) \times \alpha M$ records,

$$Bit(texp) = \begin{cases} \langle\rangle & \text{if } texp = \langle\rangle \\ Bit(texp_0)^\wedge b & \text{if } texp = texp_0{}^\wedge(c,(d,b)) \end{cases}$$

◇

In the sequel we write h where we mean $h \uparrow chan(ABP)$.

The informal description of sender S given above can be formalized as follows.

$$\begin{array}{ll} S\ \textbf{sat} & Dat(RDMsg(h\uparrow m_{in})) \preceq^1 Val(h\uparrow in) \\ \wedge & Val(RDAck(h\uparrow a_{out})) \preceq^1 Bit(RDMsg(h\uparrow m_{in})) \end{array}$$

Similarly, we obtain the following specification for receiver R.

$$\begin{array}{ll} R\ \textbf{sat} & Val(h\uparrow out) \preceq^1 Dat(RDMsg(h\uparrow m_{out})) \\ \wedge & Val(RDAck(h\uparrow a_{in})) \preceq^1 Bit(RDMsg(h\uparrow m_{out})) \end{array}$$

Then, by (Consequence) and (Parallel composition), we obtain:

$$ABP\ \textbf{sat}\ Dat(RDMsg(h\uparrow m_{in})) \preceq^1 Val(h\uparrow in) \quad \text{(A1)}$$

$$ABP\ \textbf{sat}\ Val(RDAck(h\uparrow a_{out})) \preceq^1 Bit(RDMsg(h\uparrow m_{in})) \quad \text{(A2)}$$

$$ABP\ \textbf{sat}\ Val(h\uparrow out) \preceq^1 Dat(RDMsg(h\uparrow m_{out})) \quad \text{(A3)}$$

$$ABP\ \textbf{sat}\ Val(RDAck(h\uparrow a_{in})) \preceq^1 Bit(RDMsg(h\uparrow m_{out})) \quad \text{(A4)}$$

From Example 5 we learned that $(M \wr Loss)$ **sat** $Val(h \uparrow m_{out}) \trianglelefteq Val(h \uparrow m_{in})$ which implies that

$$ABP \textbf{ sat } len(RDMsg(h \uparrow m_{out})) \leq len(RDMsg(h \uparrow m_{in})) \tag{A5}$$

Also recall from Example 5 that $(M \wr Loss)$ **sat** $\forall i : ch(h'(i)) = m_{out} : val(h'(i)) = val(last(h'[i] \uparrow m_{in}))$, with $h' = h \uparrow \{m_{in}, m_{out}\}$. Since this property can only be invalidated by communications on m_{in} and m_{out}, we conclude

$$ABP \textbf{ sat } \forall i : ch(h(i)) = m_{out} : val(h(i)) = val(last(h[i] \uparrow m_{in})) \tag{A6}$$

For medium A we obtain similarly

$$ABP \textbf{ sat } len(RDAck(h \uparrow a_{out})) \leq len(RDAck(h \uparrow a_{in})) \tag{A7}$$

$$ABP \textbf{ sat } \forall i : ch(h(i)) = a_{out} : val(h(i)) = val(last(h[i] \uparrow a_{in})) \tag{A8}$$

The crucial property of the alternating bit protocol is the following.

Lemma 7 (Persistency)

$$\begin{array}{lll} ABP \textbf{ sat} & & Val(RDAck(h \uparrow a_{out})) \preceq^1 Val(RDAck(h \uparrow a_{in})) \\ & \wedge & Dat(RDMsg(h \uparrow m_{out})) \preceq^1 Dat(RDMsg(h \uparrow m_{in})) \end{array}$$

Proof. See Appendix C. □

Then, by (Consequence), we have

$$ABP \textbf{ sat } Dat(RDMsg(h \uparrow m_{out})) \preceq^1 Dat(RDMsg(h \uparrow m_{in}))$$

which, by (A1) and (A3), yields

$$ABP \textbf{ sat } Val(h \uparrow out) \preceq Val(h \uparrow in)$$

which shows that the alternating bit protocol tolerates loss of messages and acknowledgements.

9 Soundness and Relative Network Completeness

In this section we prove that the proof theory of Section 6 is sound, that is, we prove that, if a correctness formula FP **sat** ϕ is derivable, then it is valid. Furthermore, we prove the proof system to be complete, that is, we prove that, if a correctness formula FP **sat** ϕ is valid, then it is derivable. In fact, the prefixing rule is not needed to establish completeness.

Theorem 1 (Soundness) The proof system of Section 6 is sound.

Proof. See Appendix D. □

As usual when proving completeness, we assume that we can prove any valid formula of the underlying (trace) logic (cf. [4]). Thus, using $\vdash \phi$ to denote that assertion ϕ is derivable, we add the following axiom to our proof theory.

Axiom 1 (Relative completeness assumption) For an assertion ϕ,

$$\vdash \phi \text{ if } \models \phi .$$

As in [19] we use the preciseness preservation property to achieve relative completeness. The intuition is that, as long as the specifications of the individual processes are precise, so are the deduced specifications of systems composed of such processes. Informally, a specification of a failure prone process is precise if it characterizes exactly the set of behaviours of the process.

Definition 24 (Preciseness) An assertion ϕ is precise for failure prone process FP iff

(i) $\models FP \textbf{ sat } \phi$.

(ii) if $chan(\theta) \subseteq chan(FP)$ and, for some γ, $(\theta, \gamma) \models \phi$ then $\theta \in \mathcal{H}[\![FP]\!]$.

(iii) $chan(\phi) \subseteq chan(FP)$. ◇

Let $\vdash P \textbf{ sat } \phi$ denote that correctness formula $P \textbf{ sat } \phi$ is derivable. Note that no proof rules were given for the sequential aspects of processes, so our notion of completeness is relative to the assumption that for a process P there exists a precise assertion ϕ. This leads to the definition of *network* completeness.

Definition 25 (Network completeness) Assume that for every process P there exists a precise assertion ϕ with $\vdash P \textbf{ sat } \phi$. Then, for any failure prone process FP and assertion ξ, $\models FP \textbf{ sat } \xi$ implies $\vdash FP \textbf{ sat } \xi$. ◇

The following lemma asserts that preciseness is preserved by the proof rules of Section 6.

Lemma 8 (Preciseness preservation) Assume that for any process P there exists an assertion ϕ which is precise for P and $\vdash P \textbf{ sat } \phi$. Then, for any failure prone process FP there exists an assertion ξ which is precise for FP and $\vdash FP \textbf{ sat } \xi$.

Proof. See Appendix E. □

The following lemma asserts that any specification satisfied by a failure prone process is implied by the precise specification of that process. Since a precise specification only refers to channels of the process, and a valid specification might refer to other channels, we have to add a clause expressing that the process does not communicate on those other channels.

Lemma 9 (Preciseness consequence) If ϕ_1 is precise for FP and $\models FP \textbf{ sat } \phi_2$ then

$$\models (\phi_1 \ \wedge \ h\uparrow(chan(\phi_2) - chan(FP)) = \langle\rangle) \rightarrow \phi_2$$

Proof. Assume that ϕ_1 is precise for FP, and that $\models FP \textbf{ sat } \phi_2$ (1).
Consider any θ and γ. Assume that $(\theta, \gamma) \models \phi_1 \wedge h\uparrow(chan(\phi_2) - chan(FP)) = \langle\rangle$ (2).

By (2), $(\theta, \gamma) \models \phi_1$. Since ϕ_1 is precise for FP, $chan(\phi_1) \subseteq chan(FP)$. Hence projection lemma (a) yields $(\theta\uparrow chan(FP), \gamma) \models \phi_1$, thus, once more by the preciseness of ϕ_1 for FP, $\theta\uparrow chan(FP) \in \mathcal{H}[\![FP]\!]$. By (1), $(\theta\uparrow chan(FP), \gamma) \models \phi_2$ (3).

By (2), we have that $(\theta, \gamma) \models h\uparrow(chan(\phi_2) - chan(FP)) = \langle\rangle$. Hence, $\theta\uparrow(chan(\phi_2) - chan(FP)) = \langle\rangle$, and thus, $\theta\uparrow chan(FP) = \theta\uparrow(chan(FP) \cup (chan(\phi_2) - chan(FP))) = \theta\uparrow(chan(FP) \cup chan(\phi_2))$. Hence, we obtain from (3) that $(\theta\uparrow(chan(FP) \cup chan(\phi_2)), \gamma) \models \phi_2$, and consequently, by projection lemma (a), $(\theta, \gamma) \models \phi_2$. □

Now we can establish relative network completeness.

Theorem 2 (Relative network completeness) The proof system of Section 6 is relatively network complete.

Proof. Assume that for every process P there exists a precise specification ϕ with $\vdash P$ **sat** ϕ. Then, by the preciseness preservation lemma, for any failure prone process FP there exists an assertion ξ which is precise for FP and $\vdash FP$ **sat** ξ (1).

Assume $\models FP$ **sat** η. Since $(chan(\eta) - chan(FP)) \cap chan(FP) = \emptyset$, we obtain, by (Invariance), $\vdash FP$ **sat** $h\uparrow(chan(\eta) - chan(FP)) = \langle\rangle$ (2).

By (1) and (2), $\vdash FP$ **sat** $\xi \wedge h\uparrow(chan(\eta) - chan(FP)) = \langle\rangle$, and thus, by the preciseness consequence lemma, the relative completeness assumption, and (Consequence), $\vdash FP$ **sat** η. □

10 Conclusions and Future Research

Starting from a SAT formalism, we have defined a trace-based compositional proof theory for fault tolerant distributed systems. In this theory, the fault hypothesis of a process is formalized as a reflexive relation between the normal and acceptable observable input and output behaviour of that process. Such a relation enables one to abstract from the precise nature of a fault and to focus on the abnormal behaviour it causes. With respect to existing SAT formalisms, only one new rule, viz. the fault hypothesis introduction rule, is needed. We illustrated our method by proving safety of a triple modular redundant component and the alternating bit protocol, using only the specifications of the components. The proof of correctness of the alternating bit protocol that appears in [13] is also based on traces. There, a less natural specification of the receiver, which contains the requirement that non-duplicate input messages have alternating bits, evades the necessity to prove the property of persistency.

In this report we only considered safety properties, ignoring liveness issues. Since the underlying trace logic is based on finite approximations, the proof theory we presented is not appropriate to deal with liveness properties. To allow reasoning about liveness properties, trace logic can be replaced by a more expressive logic, e.g. temporal logic. Then, instead of relating normal and exceptional communication sequences, a fault hypothesis relates normal and exceptional sequences of states. Consider, for instance, a system S whose state consists of 2 integers x and y, that is, $STATE_S = \{\ \sigma \mid \sigma : \{x, y\} \rightarrow \mathbb{N}\ \}$. Assume that in a sequence s of states a new state is recorded whenever the value of x or y changes. If we allow transient memory faults to occur, then it is possible that, instead of some intended sequence $s_{old} = (0,0),(10,0),\ldots$, we observe $s = (0,0),(3,0),(10,0),\ldots$ because a fault affects the cell containing x before it is assigned the value 10. Notice that, since we only allow transient memory faults, assigning 10 to x undoes the effect of the preceding fault. In a description where each new state is related to its predecessor by stating which state variables have changed, transient memory faults can easily be formalized as the insertion of a state at an arbitrary position in the sequence.

We have also abstracted from the sequential aspects of processes. To reason about these aspects, often a proof system based on Hoare triples (see [6]) is more convenient. In such a proof system one reasons about correctness formulae of the form $\{p\}S\{q\}$ where S is a program, and p and q are assertions expressed in a first-order language. Informally, the triple $\{p\}S\{q\}$ means that if execution of S is started in a state satisfying p, and if S terminates, then the final state satisfies q.

Besides finding a logic to express fault hypotheses more elegantly, an obvious continuation of the research described in this report is the introduction of time to the formalism, to allow reasoning about properties of fault tolerant real-time systems. Then, the characterization that safety properties express that 'nothing bad will happen' and liveness properties express that 'eventually something good will happen' (see [10]) is, as indeed mentioned in [10], no longer appropriate. Consider, for instance, a communication medium that accepts messages via a channel *in* and relays them to a channel *out*. The property 'after a message is input to the medium via *in* it is output via *out* within 5 seconds' is a safety property, because it can be falsified after 5 seconds following an *in* communication, but it expresses that something must happen. Hence, by adding time, the class of safety properties is extended and, e.g., also includes progress properties.

References

[1] A. Avižienis and J.C. Laprie. Dependable computing: from concepts to design diversity, *Proceedings of the IEEE* **74**(5) (May 1986) 629–638.

[2] K.A. Bartlett, R.A. Scantlebury and P.T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links, *Communications of the ACM* **12**(5) (1969) 260–261.

[3] J. Coenen and J. Hooman. A compositional semantics for fault tolerant real-time systems, *Proc. Second Int. Symp. on Formal Techniques in Real-Time and Fault Tolerant Systems*, Lecture Notes in Computer Science **571** (Springer, 1991) 33–51.

[4] S.A. Cook. Soundness and completeness of an axiom system for program verification, *SIAM Journal on Computing* **7**(1) (February 1978) 70–90.

[5] F. Cristian. A rigorous approach to fault tolerant programming, *IEEE Trans. on Software Engineering* **SE–11**(1) (1985) 23 – 31.

[6] C.A.R. Hoare. An axiomatic basis for computer programming, *Communications of the ACM* **12**(10) (1969) 576–580,583.

[7] C.A.R. Hoare. *Communicating Sequential Processes* (Prentice-Hall International, 1985).

[8] J. Hooman. *Specification and Compositional Verification of Real-Time Systems*, Lecture Notes in Computer Science **558** (Springer, 1992).

[9] M. Joseph, A. Moitra and N. Soundararajan. Proof rules for fault tolerant distributed programs, *Science of Computer Programming* **8** (1987) 43–67.

[10] L. Lamport. What good is temporal logic, *in*: R.E. Manson, ed. *Information Processing* (North-Holland, 1983) 657–668.

[11] J.C. Laprie. Dependable computing and fault tolerance: concepts and terminology, *Proc. 15th Int. Symp. on Fault Tolerant Computing Systems*, (IEEE Computer Society Press, 1985) 2–11.

[12] P.A Lee and T. Anderson. *Fault Tolerance: Principles and Practice* (Springer, 1990).

[13] K. Paliwoda and J.W. Sanders. An incremental specification of the sliding window protocol, *Distributed Computing* **5** (1991) 83–94.

[14] J. Peleska. Design and verification of fault tolerant systems with CSP, *Distributed Computing* **5** (1991) 95–106.

[15] B. Randell, P.A. Lee and P.C. Treleaven. Reliability issues in computing system design, *ACM Computing Surveys* **10**(2) (June 1978) 123–165.

[16] H. Schepers. Terminology and paradigms for fault tolerance, *in*: J. Vytopil, ed. *Formal Techniques in Real-Time and Fault Tolerant Systems* (Kluwer Academic Publishers, 1993) 3–31.

[17] H. Schepers. Tracing fault tolerance, *Proc. 3rd IFIP Int. Working Conference on Dependable Computing for Critical Applications* (Springer, to appear).

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

[19] J. Widom, D. Gries and F. Schneider. Trace-based network proof systems: expressiveness and completeness, *ACM TOPLAS* **14**(3) (July 1992) 396–416.

[20] J. Zwiers. *Compositionality, Concurrency and Partial Correctness*, Lecture Notes in Computer Science **321** (Springer, 1989).

A Proof of the prefix closedness lemma

By induction on the structure of FP. (Base) Since the semantic function $\mathcal{O}$ generates prefix closed sets, the theorem holds trivially for $\mathcal{H}[\![P]\!]$. (Induction Step) Assume that the lemma holds for $\mathcal{H}[\![FP]\!]$:

(a) Assume $\theta \in \mathcal{H}[\![FP_1 \parallel FP_2]\!]$, that is, assume that, for $i = 1, 2$, $\theta \uparrow chan(FP_i) \in \mathcal{H}[\![FP_i]\!]$ (1) and $\theta \uparrow chan(FP_1 \parallel FP_2) = \theta$ (2). Consider any $\theta' \preceq \theta$. Since $\theta' \preceq \theta$, we have that, for $i = 1, 2$, $\theta' \uparrow chan(FP_i) \preceq \theta \uparrow chan(FP_i)$. By (1) and the induction hypothesis, we conclude that, for $i = 1, 2$, $\theta' \uparrow chan(FP_i) \in \mathcal{H}[\![FP_i]\!]$ (3). By (2), $chan(\theta) \subseteq chan(FP_1 \| FP_2)$. Since $\theta' \preceq \theta$, $chan(\theta') \subseteq chan(\theta)$. Consequently, $chan(\theta') \subseteq chan(FP_1 \parallel FP_2)$ which means that $\theta' \uparrow chan(FP_1 \parallel FP_2) = \theta'$ (4). From (3) and (4) we conclude that $\theta' \in \mathcal{H}[\![FP_1 \parallel FP_2]\!]$.

(b) Assume $\theta \in \mathcal{H}[\![FP \setminus cset]\!]$, that is, assume there exists a $\tau \in \mathcal{H}[\![FP]\!]$ such that $\tau \setminus cset = \theta$. Consider any $\theta' \preceq \theta$. There exists a $\tau' \preceq \tau$ such that $\tau' \setminus cset = \theta'$. By the induction hypothesis, $\tau' \in \mathcal{H}[\![FP]\!]$. Hence $\theta' \in \mathcal{H}[\![FP \setminus cset]\!]$.

(c) Assume $\theta \in \mathcal{H}[\![(FP \wr \chi)]\!]$, that is, assume that there exists a $\theta_0 \in \mathcal{H}[\![FP]\!]$ such that, for all γ, $(\theta_0, \theta, \gamma) \models \chi$. Consider $\theta' \preceq \theta$. Using $\widehat{\gamma} = (\gamma : t \mapsto \theta')$, t fresh, we have $(\theta_0, \theta, \widehat{\gamma}) \models \chi$. Since $\theta' \preceq \theta$, we have $(\theta_0, \theta, \widehat{\gamma}) \models t \preceq h$. Consequently, $(\theta_0, \theta, \widehat{\gamma}) \models \chi \wedge t \preceq h$. By the syntactic restriction on χ, we obtain that $(\theta_0, \theta, \widehat{\gamma}) \models \exists t_{old} \preceq h_{old} : \chi[t/h, t_{old}/h_{old}]$. Thus there exists a θ'' such that $(\theta_0, \theta, (\widehat{\gamma} : t_{old} \mapsto \theta'')) \models t_{old} \preceq h_{old} \wedge \chi[t/h, t_{old}/h_{old}]$. Consequently, we have that $\theta'' \preceq \theta_0$ and hence $(\theta_0, \theta, (\widehat{\gamma} : t_{old} \mapsto \theta'')) \models \chi[t/h, t_{old}/h_{old}]$. Then, by the substitution lemma, $(\theta'', \widehat{\gamma}(t), (\widehat{\gamma} : t_{old} \mapsto \theta'')) \models \chi$. Since $\widehat{\gamma}(t) = \theta'$ and t and t_{old} do not occur in χ, we obtain $(\theta'', \theta', \gamma) \models \chi$. Since $\theta_0 \in \mathcal{H}[\![FP]\!]$ and $\theta'' \preceq \theta_0$, the induction hypothesis yields $\theta'' \in \mathcal{H}[\![FP]\!]$, which proves $\theta' \in \mathcal{H}[\![(FP \wr \chi)]\!]$.

$\square$

B Proof of the composite fault hypothesis lemma

It is sufficient to prove that $\mathcal{H}[\![(FP \wr (\chi_1 \wr \chi_2))]\!] \subseteq \mathcal{H}[\![((FP \wr \chi_1) \wr \chi_2)]\!]$. We will even prove equality of these two two sets.

Assume $\theta \in \mathcal{H}[\![(FP \wr (\chi_1 \wr \chi_2))]\!]$, that is, assume that there exists a $\theta_0 \in \mathcal{H}[\![FP]\!]$ such that, for any γ, $(\theta_0, \theta, \gamma) \models (\chi_1 \wr \chi_2)$. By definition this equals $(\theta_0, \theta, \gamma) \models \exists t : \chi_1[t/h] \wedge \chi_2[t/h_{old}]$, i.e. there exists a θ_1 such that, for $\widehat{\gamma} = (\gamma : t \mapsto \theta_1)$, $(\theta_0, \theta, \widehat{\gamma}) \models \chi_1[t/h] \wedge \chi_2[t/h_{old}]$. Observe that $\mathcal{T}[\![t]\!](\theta_0, \theta, \widehat{\gamma}) = \theta_1$. By the substitution lemma, $(\theta_0, \theta, \widehat{\gamma}) \models \chi_1[t/h] \wedge \chi_2[t/h_{old}]$ iff $(\theta_0, \theta_1, \widehat{\gamma}) \models \chi_1$ and $(\theta_1, \theta, \widehat{\gamma}) \models \chi_2$. Hence, $\theta \in \mathcal{H}[\![(FP \wr (\chi_1 \wr \chi_2))]\!]$ iff there exists a $\theta_0 \in \mathcal{H}[\![FP]\!]$ such that, for any γ, there exists a θ_1 such that $(\theta_0, \theta_1, \gamma) \models \chi_1$ and $(\theta_1, \theta, \gamma) \models \chi_2$. Then, $\theta \in \mathcal{H}[\![(FP \wr (\chi_1 \wr \chi_2))]\!]$ iff there exists a $\theta_1 \in \mathcal{H}[\![(FP \wr \chi_1)]\!]$ such that $(\theta_1, \theta, \gamma) \models \chi_2$. Equivalently, $\theta \in \mathcal{H}[\![(FP \wr (\chi_1 \wr \chi_2))]\!]$ iff $\theta \in \mathcal{H}[\![((FP \wr \chi_1) \wr \chi_2)]\!]$. $\square$

C Proof of the persistency lemma

By induction on the length of h.

(*Base Step*) The case $h = \langle\rangle$ is trivial.

(*Induction Step*) Assume that the lemma holds for t, that is,

$$Val(RDAck(t \uparrow a_{out})) \preceq^1 Val(RDAck(t \uparrow a_{in})) \qquad (1),$$

and

$$Dat(RDMsg(t \uparrow m_{out})) \preceq^1 Dat(RDMsg(t \uparrow m_{in})) \qquad (2).$$

Four cases need examination:

1. $h = t^\wedge(m_{in}, (v, b))$, where $b \neq bit(val(last(t \uparrow m_{in})))$.

   By (A2), we have that $len(RDAck(h \uparrow a_{out})) \preceq^1 len(RDMsg(h \uparrow m_{in}))$. Since $t \prec h$, by (A2) and (Prefixing), we obtain $len(RDAck(t \uparrow a_{out})) \leq^1 len(RDMsg(t \uparrow m_{in}))$. Then, because $h = t^\wedge(m_{in}, (v, b))$, we conclude that $len(RDAck(t \uparrow a_{out})) = len(RDMsg(t \uparrow m_{in}))$ (3).

   Since $t \prec h$, we have, by (A4) and (Prefixing), $Val(RDAck(t \uparrow a_{in})) \preceq^1 Bit(RDMsg(t \uparrow m_{out}))$. Then, by (1), we obtain that $Val(RDAck(t \uparrow a_{out})) \preceq Bit(RDMsg(t \uparrow m_{out}))$. Consequently, we have $len(Val(RDAck(t \uparrow a_{out}))) \leq len(Bit(RDMsg(t \uparrow m_{out})))$, from which we conclude that $len(RDAck(t \uparrow a_{out})) \leq len(RDMsg(t \uparrow m_{out}))$ (4).

   By (2) we have that $len(RDMsg(t \uparrow m_{out})) \leq^1 len(RDMsg(t \uparrow m_{in}))$. Hence, by (4), we obtain $len(RDAck(t \uparrow a_{out})) \leq len(RDMsg(t \uparrow m_{out})) \leq^1 len(RDMsg(t \uparrow m_{in}))$. Finally, by (3), we have $len(RDMsg(t \uparrow m_{out})) = len(RDMsg(t \uparrow m_{in}))$, from which we conclude, by (2), that $Dat(RDMsg(t \uparrow m_{out})) = Dat(RDMsg(t \uparrow m_{in}))$. Then it is obviously the case that $Dat(RDMsg(h \uparrow m_{out})) \prec^1 Dat(RDMsg(h \uparrow m_{in}))$, from which the theorem follows.

2. $h = t^\wedge(m_{out}, (v, b))$, where $b \neq bit(val(last(t \uparrow m_{out})))$.

   By (A4), we have that $Val(RDAck(h \uparrow a_{in})) \preceq^1 Bit(RDMsg(h \uparrow m_{out}))$. Since $t \prec h$, we obtain, by (A4) and (Prefixing), that $Val(RDAck(t \uparrow a_{in})) \preceq^1 Bit(RDMsg(t \uparrow m_{out}))$. Hence, we conclude that $Val(RDAck(t \uparrow a_{in})) = Bit(RDMsg(t \uparrow m_{out}))$. Then, by (1), we obtain that $Val(RDAck(t \uparrow a_{out})) \preceq^1 Bit(RDMsg(t \uparrow m_{out}))$, from which we can easily conclude that $len(RDAck(t \uparrow a_{out})) \leq^1 len(RDMsg(t \uparrow m_{out}))$ (5).

   Since $t \prec h$, by (A2) and (Prefixing), $len(RDAck(t \uparrow a_{out})) \leq^1 len(RDMsg(t \uparrow m_{in}))$ (6).

   Since $t \prec h$, we have, by (A5) and (Prefixing), $len(RDMsg(t \uparrow m_{out})) \leq len(RDMsg(t \uparrow m_{in}))$. Then, by (5) and (6), $len(RDMsg(t \uparrow m_{out})) \leq^1 len(RDMsg(t \uparrow m_{in}))$ (7).

   Assume that $len(RDMsg(t \uparrow m_{out})) = len(RDMsg(t \uparrow m_{in}))$. Since $h = t^\wedge(m_{out}, (v, b))$, with $b \neq bit(val(last(t \uparrow m_{out})))$, we obtain $len(RDMsg(h \uparrow m_{out})) = len(RDMsg(h \uparrow m_{in})) + 1$, which is in conflict with (A5). Hence, by (7), $len(RDMsg(t \uparrow m_{out})) <^1 len(RDMsg(t \uparrow m_{in}))$, which, using (2), yields that $Dat(RDMsg(t \uparrow m_{out})) \prec^1 Dat(RDMsg(t \uparrow m_{in}))$. By (A6), $v = msg(val(last(h[len(h)] \uparrow m_{in})))$, or, equivalently, $v = msg(val(last(t \uparrow m_{in})))$. Then, $Dat(RDMsg(h \uparrow m_{out})) = Dat(RDMsg(h \uparrow m_{in}))$, from which we conclude that the theorem holds.

3. $h = t^\wedge(a_{in}, b)$, where $b \neq val(last(t \uparrow a_{in}))$.

   By (A4), we have that $len(RDAck(h \uparrow a_{in})) \leq^1 len(RDMsg(h \uparrow m_{out}))$. Since $t \prec h$, by (A4) and (Prefixing), we obtain $len(RDAck(t \uparrow a_{in})) \leq^1 len(RDMsg(t \uparrow m_{out}))$. Then, we conclude that $len(RDAck(t \uparrow a_{in})) <^1 len(RDMsg(t \uparrow m_{out}))$ (8).

   By (2), we have that $len(RDMsg(t \uparrow m_{out})) \leq^1 len(RDMsg(t \uparrow m_{in}))$. Then, by (8), we conclude that $len(RDAck(t \uparrow a_{in})) < len(RDMsg(t \uparrow m_{in}))$ (9).

   Since $t \prec h$, by (A7) and (Prefixing), $len(RDAck(t \uparrow a_{out})) \leq len(RDAck(t \uparrow a_{in}))$, which leads, by (9), to $len(RDAck(t \uparrow a_{out})) \leq len(RDAck(t \uparrow a_{in})) < len(RDMsg(t \uparrow m_{in}))$ (10).

   Since $t \prec h$, we have, by (A2) and (Prefixing), $len(RDAck(t \uparrow a_{out})) \leq^1 len(RDMsg(t \uparrow m_{in}))$, which, by (10), yields that $len(RDAck(t \uparrow a_{out})) = len(RDAck(t \uparrow a_{in}))$. Hence, by (1), we obtain that $Val(RDAck(t \uparrow a_{out})) = Val(RDAck(t \uparrow a_{in}))$. Then, it is obviously the case that $Val(RDAck(h \uparrow a_{out})) \prec^1 Val(RDAck(h \uparrow a_{in}))$, from which we conclude that the theorem holds.

4. $h = t^\wedge(a_{out}, b)$, where $b \neq val(last(t \uparrow a_{out}))$.

   By (A2), we have $Val(RDAck(h \uparrow a_{out})) \preceq^1 Bit(RDMsg(h \uparrow m_{in}))$. Since $t \prec h$, by (A2) and (Prefixing), we also have $Val(RDAck(t \uparrow a_{out})) \preceq^1 Bit(RDMsg(t \uparrow m_{in}))$. Hence, we conclude that $Val(RDAck(t \uparrow a_{out})) \prec^1 Bit(RDMsg(t \uparrow m_{in}))$, from which we can conclude that $len(RDAck(t \uparrow a_{out})) <^1 len(RDMsg(t \uparrow m_{in}))$. (11).

   By (2), we have that $len(RDMsg(t \uparrow m_{out})) \leq^1 len(RDMsg(t \uparrow m_{in}))$. Then, by (11), we conclude $len(RDAck(t \uparrow a_{out})) \leq^1 len(RDMsg(t \uparrow m_{out}))$ (12).

Since $t \prec h$, by (A4) and (Prefixing), $len(RDAck(t \uparrow a_{in})) \leq^1 len(RDMsg(t \uparrow m_{out}))$ (13).
Since $t \prec h$, we have, by (A7) and (Prefixing), $len(RDAck(t \uparrow a_{out})) \leq len(RDAck(t \uparrow a_{in}))$.
Then, by (12) and (13), we conclude $len(RDAck(t \uparrow a_{out})) \leq^1 len(RDAck(t \uparrow a_{in}))$ (14).
Assume that $len(RDAck(t \uparrow a_{out})) = len(RDAck(t \uparrow a_{in}))$. Then, since $h = t^\wedge(a_{out}, b)$, where $b \neq val(last(t \uparrow a_{out}))$, we obtain $len(RDAck(h \uparrow a_{out})) = len(RDAck(h \uparrow a_{in})) + 1$, which conflicts with (A7). Consequently, by (14), $len(RDAck(t \uparrow a_{out})) <^1 len(RDAck(t \uparrow a_{in}))$, which, combined with (1), yields $Val(RDAck(t \uparrow a_{out})) \prec^1 Val(RDAck(t \uparrow a_{in}))$. Finally, since, by (A8), we have that $b = val(last(h[len(h)] \uparrow a_{in}))$, or, equivalently, $b = val(last(t \uparrow a_{in}))$, we obtain $Val(RDAck(h \uparrow a_{out})) = Val(RDAck(h \uparrow a_{in}))$, from which we conclude that the theorem holds.

□

D Proof of the soundness theorem

D.1 Soundness of the consequence and conjunction rule

Trivial.

D.2 Soundness of the invariance rule

Follows from the fact that if $\theta \in \mathcal{H}[\![FP]\!]$ then $chan(\theta) \subseteq chan(FP)$. Thus, $cset \cap chan(FP) = \emptyset$ implies $chan(\theta) \cap cset = \emptyset$.

D.3 Soundness of the parallel composition rule

Suppose $chan(\phi_1) \cap chan(FP_2) \subseteq chan(FP_1)$, $chan(\phi_2) \cap chan(FP_1) \subseteq chan(FP_2)$ (1).
Assume $\models FP_1$ **sat** ϕ_1, $\models FP_2$ **sat** ϕ_2 (2).
We have to prove $\models FP_1 \| FP_2$ **sat** $\phi_1 \wedge \phi_2$. Consider any γ. Let $\theta \in \mathcal{H}[\![FP_1 \| FP_2]\!]$. By the definition of the semantics, we have, for $i = 1, 2$, $\theta \uparrow chan(FP_i) \in \mathcal{H}[\![FP_i]\!]$ and $\theta \uparrow chan(FP_1 \| FP_2) = \theta$. Since $\theta \uparrow chan(FP_i) \in \mathcal{H}[\![FP_i]\!]$, we obtain, by (2), $(\theta \uparrow chan(FP_i), \gamma) \models \phi_i$. By projection lemma (a) $((\theta \uparrow chan(FP_i)) \uparrow chan(\phi_i), \gamma) \models \phi_i$, thus $(\theta \uparrow (chan(FP_i) \cap chan(\phi_i)), \gamma) \models \phi_i$.
By (1), we obtain that $chan(FP_2) \cap chan(\phi_1) \subseteq chan(FP_1) \cap chan(\phi_1)$, from which we conclude that $(chan(FP_2) \cap chan(\phi_1)) \cup (chan(FP_1) \cap chan(\phi_1)) \subseteq chan(FP_1) \cap chan(\phi_1)$. Consequently, we have that $(chan(FP_2) \cap chan(\phi_2)) \cup (chan(FP_1) \cap chan(\phi_1)) = chan(FP_1) \cap chan(\phi_1)$, from which we deduce $chan(FP_1) \cap chan(\phi_1) = (chan(FP_1) \cup chan(FP_2)) \cap chan(\phi_1) = chan(FP_1 \| FP_2) \cap chan(\phi_1)$. By similar reasoning, $chan(FP_2) \cap chan(\phi_2) = chan(FP_1 \| FP_2) \cap chan(\phi_2)$. Consequently, for $i = 1, 2$, $(\theta \uparrow (chan(FP_1 \| FP_2) \cap chan(\phi_i)), \gamma) \models \phi_i$. Hence, $((\theta \uparrow (chan(FP_1 \| FP_2)) \uparrow chan(\phi_i), \gamma) \models \phi_i$, which leads to $(\theta \uparrow chan(\phi_i), \gamma) \models \phi_i$, and consequently, by projection lemma (a), $(\theta, \gamma) \models \phi_i$. This proves $\models FP_1 \| FP_2$ **sat** $\phi_1 \wedge \phi_2$.

D.4 Soundness of the hiding rule

Assume $\models FP$ **sat** ϕ (1),
and $chan(\phi) \cap cset = \emptyset$ (2).
We show $FP \backslash cset$ **sat** ϕ. Consider any γ. Let $\theta \in \mathcal{H}[\![FP \backslash cset]\!]$. Then there exists a $\theta_1 \in \mathcal{H}[\![FP]\!]$ with $\theta = \theta_1 \backslash cset$. By (1), $(\theta_1, \gamma) \models \phi$. Since, by (2), $chan(\phi) \subseteq CHAN - cset$, projection lemma (a) leads to $(\theta_1 \uparrow (CHAN - cset), \gamma) \models \phi$, and consequently, by definition, $(\theta_1 \backslash cset, \gamma) \models \phi$. Hence, $(\theta, \gamma) \models \phi$.

D.5 Soundness of the fault hypothesis introduction rule

Assume $\models FP$ **sat** ϕ (1). Consider any γ. Let $\theta \in \mathcal{H}[\![(FP \wr \chi)]\!]$. Then there exists a $\theta_0 \in \mathcal{H}[\![FP]\!]$ such that, for all γ, $(\theta_0, \theta, \gamma) \models \chi$. By (1), for any θ_0', $(\theta_0', \theta_0, \gamma) \models \phi$, thus also $(\theta_0, \theta_0, \gamma) \models \phi$. Let, for fresh t, $\widehat{\gamma} = (\gamma : t \mapsto \theta_0)$. Since t does not occur in ϕ, $(\theta_0, \theta_0, \widehat{\gamma}) \models \phi$. Observe that $\mathcal{T}[\![t]\!](\theta_0, \theta, \widehat{\gamma}) = \theta_0$, thus $(\theta_0, \mathcal{T}[\![t]\!](\theta_0, \theta, \widehat{\gamma}), \widehat{\gamma}) \models \phi$. By substitution lemma (a) we obtain $(\theta_0, \theta, \widehat{\gamma}) \models \phi[t/h]$, or, by the correspondence lemma, $(\theta, \widehat{\gamma}) \models \phi[t/h]$ (2).
Since $(\theta_0, \theta, \widehat{\gamma}) \models \chi$, we have $(\mathcal{T}[\![t]\!](\theta_0, \theta, \widehat{\gamma}), \theta, \widehat{\gamma}) \models \chi$. Applying substitution lemma (b) leads to $(\theta_0, \theta, \widehat{\gamma}) \models \chi[t/h_{old}]$. Since h_{old} does not occur in $\chi[t/h_{old}]$, the correspondence lemma leads to $(\theta, \widehat{\gamma}) \models \chi[t/h_{old}]$ (3).
From (2) and (3) we obtain $(\theta, (\gamma : t \mapsto \theta_0)) \models \phi[t/h] \wedge \chi[t/h_{old}]$, from which we may conclude that $(\theta, \gamma) \models \exists t : \phi[t/h] \wedge \chi[t/h_{old}]$.

D.6 Soundness of the prefixing rule

Assume $\models FP$ **sat** ϕ (1). Consider any γ. Let $\theta \in \mathcal{H}[\![FP]\!]$. By (1), $(\theta, \gamma) \models \phi$. For all $\theta' \preceq \theta$ we have, by the prefix closedness lemma, that $\theta' \in \mathcal{H}[\![FP]\!]$, and thus, by (1), $(\theta', \gamma) \models \phi$. Let t be a fresh logical variable. Then, as t does not occur in ϕ, for all $\theta' \preceq \theta$, $(\theta', (\gamma : t \mapsto \theta')) \models \phi$. Equivalently, $(\mathcal{T}[\![t]\!](\theta_0, \theta', (\gamma : t \mapsto \theta')), (\gamma : t \mapsto \theta')) \models \phi$. By substitution lemma (a) $(\theta', (\gamma : t \mapsto \theta')) \models \phi[t/h]$, for all $\theta' \preceq \theta$, and thus, as h obviously does not occur in $\phi[t/h]$, for all $\theta' \preceq \theta$, $(\theta, (\gamma : t \mapsto \theta')) \models \phi[t/h]$, and consequently, for all θ', $(\theta, (\gamma : t \mapsto \theta')) \models t \preceq h \rightarrow \phi[t/h]$. Hence, $(\theta, \gamma) \models \forall t : t \preceq h \rightarrow \phi[t/h]$, i.e. $(\theta, \gamma) \models \forall t \preceq h : \phi[t/h]$. Thus, $\models FP$ **sat** $\forall t \preceq h : \phi[t/h]$.

E Proof of the preciseness preservation lemma

By induction on the structure of *FP*. (Base) By assumption, the lemma holds for P. (Induction Step) Assume that the lemma holds for *FP*:

(a) Assume $\vdash FP_1$ **sat** ϕ_1 and $\vdash FP_2$ **sat** ϕ_2, with ϕ_1 and ϕ_2 precise for FP_1 and FP_2, respectively. Since, by the preciseness of ϕ_1 for FP_1, we have $chan(\phi_1) \subseteq chan(FP_1)$ (1), we conclude $chan(\phi_1) \cap chan(FP_2) \subseteq chan(FP_1) \cap chan(FP_2) \subseteq chan(FP_1)$. Similarly, using $chan(\phi_2) \subseteq chan(FP_2)$ (2), we obtain $chan(\phi_2) \cap chan(FP_1) \subseteq chan(FP_2)$. Thus, by applying (Parallel Composition), we obtain $\vdash FP_1 \parallel FP_2$ **sat** $\phi_1 \wedge \phi_2$ (3). We show that $\phi_1 \wedge \phi_2$ is precise for $FP_1 \parallel FP_2$.

  (*i*) By (3) and soundness, we obtain $\models FP_1 \parallel FP_2$ **sat** $\phi_1 \wedge \phi_2$.

  (*ii*) Let $chan(\theta) \subseteq chan(FP_1 \parallel FP_2)$ (4) and assume $(\theta, \gamma) \models \phi_1 \wedge \phi_2$. Then, by (1) and projection lemma (a), $(\theta \uparrow chan(FP_1), \gamma) \models \phi_1$. Consequently, by the preciseness of ϕ_1 for FP_1, we conclude $\theta \uparrow chan(FP_1) \in \mathcal{H}[\![FP_1]\!]$ (5). Similarly, $\theta \uparrow chan(FP_2) \in \mathcal{H}[\![FP_2]\!]$ (6). Finally, by (4), $\theta \uparrow chan(FP_1 \parallel FP_2) = \theta$ (7). Then, by (5), (6), and (7), we conclude that $\theta \in \mathcal{H}[\![FP_1 \parallel FP_2]\!]$.

  (*iii*) By (1) and (2), we conclude $chan(\phi_1) \cup chan(\phi_2) \subseteq chan(FP_1) \cup chan(FP_2)$. Hence, by definition, we have $chan(\phi_1 \wedge \phi_2) \subseteq chan(FP_1 \parallel FP_2)$.

(b) Assume $\vdash FP$ **sat** ϕ (1) with ϕ precise for *FP*. Define

$$\widehat{\phi} \equiv \exists t : \phi[t/h] \wedge h \uparrow (chan(FP) - cset) = t \uparrow (chan(FP) - cset)$$

We show that $\vdash FP \setminus cset$ **sat** $\widehat{\phi}$, and, furthermore, that $\widehat{\phi}$ is precise for $FP \setminus cset$.

Lemma 10 $\models \phi \rightarrow \widehat{\phi}$

Proof: Assume $(\theta, \gamma) \models \phi$. Let, for fresh t, $\widehat{\gamma} = (\gamma : t \mapsto \theta)$. Then, $(\theta, \widehat{\gamma}) \models \phi$, and, trivially, $(\theta, \widehat{\gamma}) \models \phi[t/h] \wedge h \uparrow (chan(FP) - cset) = t \uparrow (chan(FP) - cset)$. Hence, $(\theta, \gamma) \models \exists t : \phi[t/h] \wedge h \uparrow (chan(FP) - cset) = t \uparrow (chan(FP) - cset)$. □

By Lemma 10 and the relative completeness assumption, we obtain $\vdash \phi \rightarrow \widehat{\phi}$. By (1) and the consequence rule, $\vdash$ *FP* **sat** $\widehat{\phi}$. Note that, by definition, $chan(\exists t : \phi[t/h]) = \emptyset$, thus $chan(\widehat{\phi}) = chan(FP) - cset$, and hence $chan(\widehat{\phi}) \cap cset = \emptyset$. Then the hiding rule leads to $\vdash FP \backslash cset$ **sat** $\widehat{\phi}$ (2). It remains to be shown that $\widehat{\phi}$ is precise for $FP \backslash cset$.

(*i*) By (2) and soundness, we have $\models FP \backslash cset$ **sat** $\widehat{\phi}$.

(*ii*) Let $chan(\theta) \subseteq chan(FP \backslash cset)$ (3) and, for some γ, $(\theta, \gamma) \models \widehat{\phi}$. There exists a $\widehat{\gamma}$ with

$$(\theta, (\gamma : t \mapsto \widehat{\theta})) \models \phi[t/h] \wedge h \uparrow (chan(FP) - cset) = t \uparrow (chan(FP) - cset) \qquad (4)$$

Then, by substitution lemma (a), $(\widehat{\theta}, (\gamma : t \mapsto \widehat{\theta})) \models \phi$, and thus $(\widehat{\theta}, \gamma) \models \phi$. Hence, by projection lemma (a), we have $(\widehat{\theta} \uparrow chan(\phi), \gamma) \models \phi$. Since, by the preciseness of ϕ for FP, $chan(\phi) \subseteq chan(FP)$, we obtain $(\widehat{\theta} \uparrow chan(FP), \gamma) \models \phi$. Obviously, $chan(\widehat{\theta} \uparrow chan(FP)) \subseteq chan(FP)$, so, by the preciseness of ϕ for FP, we have that $\widehat{\theta} \uparrow chan(FP) \in \mathcal{H}[\![FP]\!]$. Since, by (3), $chan(\theta) \subseteq chan(FP) - cset$ and, by (4), $\theta \uparrow (chan(FP) - cset) = \widehat{\theta} \uparrow (chan(FP) - cset)$, we obtain $\theta = \widehat{\theta} \uparrow chan(FP \backslash cset)$, and thus $\theta = (\widehat{\theta} \uparrow chan(FP)) \backslash cset$. Hence, $\theta \in \mathcal{H}[\![FP \backslash cset]\!]$.

(*iii*) Since $chan(\widehat{\phi}) = chan(FP) - cset$, we have, by definition, $chan(\widehat{\phi}) = chan(FP \backslash cset)$.

(c) Assume $\vdash$ *FP* **sat** ϕ (1) with ϕ precise for *FP*. Define $\widehat{\phi} \equiv (\phi \wr \chi)$, that is

$$\widehat{\phi} \;\equiv\; \exists t : \phi[t/h] \;\wedge\; \chi[t/h_{old}]$$

Then, by (Fault Hypothesis Introduction), $\vdash (FP \wr \chi)$ **sat** $\widehat{\phi}$ (2). We show that $\widehat{\phi}$ is precise for $(FP \wr \chi)$.

(*i*) By (2) and soundness, we have $\models (FP \wr \chi)$ **sat** $\widehat{\phi}$.

(*ii*) Let $chan(\theta) \subseteq chan(FP \wr \chi)$ (3) and assume, for some γ, $(\theta, \gamma) \models \widehat{\phi}$. Consequently, there exists a $\widehat{\theta}$ such that $(\theta, (\gamma : t \mapsto \widehat{\theta})) \models \phi[t/h] \wedge \chi[t/h_{old}]$ (4). Then, by substitution lemma (a), $(\widehat{\theta}, (\gamma : t \mapsto \widehat{\theta})) \models \phi$, and thus, since t does not occur free in ϕ, $(\widehat{\theta}, \gamma) \models \phi$. Since we have, by the preciseness of ϕ for FP, $chan(\phi) \subseteq chan(FP)$, we obtain, by projection lemma (a), $(\widehat{\theta} \uparrow chan(FP), \gamma) \models \phi$. Trivially, $chan(\widehat{\theta} \uparrow chan(FP)) \subseteq chan(FP)$, and hence, because of the preciseness of ϕ for FP, $\widehat{\theta} \uparrow chan(FP) \in \mathcal{H}[\![FP]\!]$ (5). By the correspondence lemma and substitution lemma (b), (4) leads to $(\widehat{\theta}, \theta, (\gamma : t \mapsto \widehat{\theta})) \models \chi$, thus, since t does not occur free in χ, $(\widehat{\theta}, \theta, \gamma) \models \chi$. Since $chan(\chi) \subseteq chan(FP)$, projection lemma (b) leads to $(\widehat{\theta} \uparrow chan(FP), \theta, \gamma) \models \chi$ (6).
Finally, by definition, (3) leads to $chan(\theta) \subseteq chan(FP)$ (7).
Consequently, by (5), (6), and (7), $\theta \in \mathcal{H}[\![(FP \wr \chi)]\!]$.

(*iii*) By definition, we have that $chan(\widehat{\phi}) = chan(\phi[t/h]) \cup chan(\chi[t/h_{old}])$ (1). Clearly, $chan(\chi[t/h_{old}]) \subseteq chan(\chi)$ (2). It is also obvious that $chan(\phi[t/h]) \subseteq chan(\phi)$, and, since, by the preciseness of ϕ for FP, we have that $chan(\phi) \subseteq chan(FP)$, we conclude $chan(\phi[t/h]) \subseteq chan(FP)$ (3). By (1), (2), and (3), $chan(\widehat{\phi}) \subseteq chan(FP) \cup chan(\chi)$, that is, $chan(\widehat{\phi}) \subseteq chan(FP \wr \chi)$.

□

In this series appeared:

91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.
91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.
91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.

91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. 25.
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.
91/29	H. Ten Eikelder R. van Geldrop	Correctness of Acceptor Schemes for Regular Languages, p. 31.
91/30	J.C.M. Baeten F.W. Vaandrager	An Algebra for Process Creation, p. 29.
91/31	H. ten Eikelder	Some algorithms to decide the equivalence of recursive types, p. 26.
91/32	P. Struik	Techniques for designing efficient parallel programs, p. 14.
91/33	W. v.d. Aalst	The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.
91/34	J. Coenen	Specifying fault tolerant programs in deontic logic, p. 15.
91/35	F.S. de Boer J.W. Klop C. Palamidessi	Asynchronous communication in process algebra, p. 20.

92/01	J. Coenen J. Zwiers W.-P. de Roever	A note on compositional refinement, p. 27.
92/02	J. Coenen J. Hooman	A compositional semantics for fault tolerant real-time systems, p. 18.
92/03	J.C.M. Baeten J.A. Bergstra	Real space process algebra, p. 42.
92/04	J.P.H.W.v.d.Eijnde	Program derivation in acyclic graphs and related problems, p. 90.
92/05	J.P.H.W.v.d.Eijnde	Conservative fixpoint functions on a graph, p. 25.
92/06	J.C.M. Baeten J.A. Bergstra	Discrete time process algebra, p.45.
92/07	R.P. Nederpelt	The fine-structure of lambda calculus, p. 110.
92/08	R.P. Nederpelt F. Kamareddine	On stepwise explicit substitution, p. 30.
92/09	R.C. Backhouse	Calculating the Warshall/Floyd path algorithm, p. 14.
92/10	P.M.P. Rambags	Composition and decomposition in a CPN model, p. 55.
92/11	R.C. Backhouse J.S.C.P.v.d.Woude	Demonic operators and monotype factors, p. 29.
92/12	F. Kamareddine	Set theory and nominalisation, Part I, p.26.
92/13	F. Kamareddine	Set theory and nominalisation, Part II, p.22.
92/14	J.C.M. Baeten	The total order assumption, p. 10.
92/15	F. Kamareddine	A system at the cross-roads of functional and logic programming, p.36.
92/16	R.R. Seljée	Integrity checking in deductive databases; an exposition, p.32.
92/17	W.M.P. van der Aalst	Interval timed coloured Petri nets and their analysis, p. 20.
92/18	R.Nederpelt F. Kamareddine	A unified approach to Type Theory through a refined lambda-calculus, p. 30.
92/19	J.C.M.Baeten J.A.Bergstra S.A.Smolka	Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, p. 36.
92/20	F.Kamareddine	Are Types for Natural Language? P. 32.
92/21	F.Kamareddine	Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.

92/22	R. Nederpelt F.Kamareddine	A useful lambda notation, p. 17.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for $F\omega$, p. 15.
92/26	T.H.W.Beelen W.J.J.Stut P.A.C.Verkoulen	A modelling method using MOVIE and SimCon/ExSpect, p. 15.
92/27	B. Watson G. Zwaan	A taxonomy of keyword pattern matching algorithms, p. 50.
93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.

92/22	R. Nederpelt F.Kamareddine	A useful lambda notation, p. 17.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for Fω, p. 15.
92/26	T.H.W.Beelen W.J.J.Stut P.A.C.Verkoulen	A modelling method using MOVIE and SimCon/ExSpect, p. 15.
92/27	B. Watson G. Zwaan	A taxonomy of keyword pattern matching algorithms, p. 50.
93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.
93/14	J.C.M. Baeten J.A. Bergstra	On Sequential Composition, Action Prefixes and Process Prefix, p. 21.

93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.