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

Citation for published version (APA):

Document status and date:
Published: 01/01/1993

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
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
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.
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: winjh@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 $((in,1),(out,1),(in,3),(out,3))$. To express that a process $P$ satisfies a safety property $\phi$ we use a correctness formula of the form $P \; sat \; \phi$. 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 $\chi$ of a process $P$ is formalized as a predicate, whose only free variables are $h$ and $h_{\text{old}}$, representing a reflexive relation between the normal and acceptable histories of $P$. The interpretation is such that $h_{\text{old}}$ represents a normal history of process $P$, whereas $h$ is an acceptable history of $P$ with respect to $\chi$. For a predicate $\chi$, representing a fault hypothesis, we introduce the construct $(P|h)$ to indicate execution of process $P$ under the assumption of $\chi$. 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 $((in,1),(out,1),(in,3),(out,3))$ may be transformed into history $((in,1),(out,1),(in,3),(out,5))$. Then, we would like to prove that failure prone process $(B|Cor)$ 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 $FP \; sat \; \phi$ 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 \( cset \subseteq CHAN \).

| Expression | \( e ::= \mu \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2 \) |
| Boolean Expression | \( b ::= e_1 = e_2 \mid e_1 < e_2 \mid \neg b \mid b_1 \lor b_2 \) |
| Guarded Command | \( G ::= [b_1 = 1 \rightarrow P_1] \) |
| Process | \( P ::= \text{skip} \mid x := e \mid c!e \mid c?x \mid P_1 ; P_2 \mid G \mid \ast G \mid P_1 || P_2 \mid P \setminus cset \) |

Table 1: Syntax of the Programming Language

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

Atomic statements

- \( \text{skip} \) 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 \( [b_1 = 1 \rightarrow P_1] \). 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 \( \ast G \) 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 \( \ast G \) terminates.
- \( P_1 || P_2 \) indicates the parallel execution of the processes \( P_1 \) and \( P_2 \).
- \( P \setminus cset \) hides the channels from \( cset \).

For a guarded command \( G = [b_1 = 1 \rightarrow P_1] \) we define \( b_G = b_1 \lor \ldots \lor b_n \). Define \( \text{var}(P) \) 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 $\text{in}(P)$, is defined inductively as follows:

- $\text{in}(\text{skip}) = \text{in}(x := e) = \text{in}(c!e) = \emptyset$
- $\text{in}(c?x) = \{c\}$
- $\text{in}(P_1 ; P_2) = \text{in}(P_1) \cup \text{in}(P_2)$
- $\text{in}([\prod_{i=1}^n b_i \rightarrow P_i]) = \bigcup_i \text{in}(P_i)$
- $\text{in}(e; G) = \text{in}(G)$
- $\text{in}(P \parallel P_2) = \text{in}(P_1) \cup \text{in}(P_2)$
- $\text{in}(P \setminus \text{cset}) = \text{in}(P) - \text{cset}$

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

- $\text{out}(\text{skip}) = \text{out}(x := e) = \emptyset$
- $\text{out}(c!e) = \{c\}$
- $\text{out}(e?x) = \emptyset$
- $\text{out}(P_1 ; P_2) = \text{out}(P_1) \cup \text{out}(P_2)$
- $\text{out}([\prod_{i=1}^n b_i \rightarrow P_i]) = \bigcup_i \text{out}(P_i)$
- $\text{out}(e; G) = \text{out}(G)$
- $\text{out}(P \parallel P_2) = \text{out}(P_1) \cup \text{out}(P_2)$
- $\text{out}(P \setminus \text{cset}) = \text{out}(P) - \text{cset}$

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

2.1 Syntactic Restrictions

To guarantee that channels are unidirectional and point-to-point, we have the following syntactic constraints (for any $c \in \text{CHAN}$, $x \in \text{VAR}$, 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, $\text{in}(P_1) \cap \text{out}(P_2) = \emptyset$ and $\text{out}(P_1) \cap \text{in}(P_2) = \emptyset$.
- For $[\prod_{i=1}^n b_i \rightarrow P_i]$ 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, $\text{out}(P_i) \cap \text{in}(P_j) = \emptyset$.
- For $P_1 \parallel P_2$ 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, $\text{in}(P_1) \cap \text{in}(P_2) = \emptyset$ and $\text{out}(P_1) \cap \text{out}(P_2) = \emptyset$.

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

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

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

- For $P_1 \parallel P_2$ we require that $\text{var}(P_1) \cap \text{var}(P_2) = \emptyset$.
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 $\text{STATE}$ of states as the set of mappings from $\text{VAR}$ to $\text{VAL}$:

$$\text{STATE} = \{ \sigma : \text{VAR} \to \text{VAL} \}$$

Thus a state $\sigma$ 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 $\sigma$ with respect to a variable $x$ and a value $\theta$, denoted $(\sigma : x \mapsto \theta)$, is given by

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

using ‘$\equiv$’ to denote syntactic equality.

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

- $\mathcal{E}[1](\sigma) = 1'$,
- $\mathcal{E}[x](\sigma) = \sigma(x)$,
- $\mathcal{E}[e_1 + e_2](\sigma) = \mathcal{E}[e_1](\sigma) + \mathcal{E}[e_2](\sigma)$,
- $\mathcal{E}[e_1 - e_2](\sigma) = \mathcal{E}[e_1](\sigma) - \mathcal{E}[e_2](\sigma)$, 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 $\sigma$, 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
- $\mathcal{B}[b_1 \lor b_2](\sigma)$ iff $\mathcal{B}[b_1](\sigma)$ or $\mathcal{B}[b_2](\sigma)$.

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

Let $\text{TRACE}$ be the set of traces, that is, the smallest set such that

- $\text{} \in \text{TRACE}$,
- if $\theta \in \text{TRACE}$, $c \in \text{CHAN}$, and $\mu \in \text{VAL}$ then $\theta^n(c, \mu) \in \text{TRACE}$.
Definition 5 (Projection) For a trace $\theta \in \text{TRACE}$ and a set of channels $\text{cset} \subseteq \text{CHAN}$, we define the projection of $\theta$ onto $\text{cset}$, denoted by $\theta \upharpoonright \text{cset}$, as the sequence obtained from $\theta$ by deleting all records with channels not in $\text{cset}$. Formally,

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

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

$$\theta \setminus \text{cset} = \theta \upharpoonright (\text{CHAN} - \text{cset})$$

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

$$\text{chan}(\theta) = \{ c \in \text{CHAN} \mid \theta \upharpoonright \{c\} \neq \{\} \}$$

Notice that $\theta \upharpoonright \text{cset} = \theta$ iff $\text{chan}(\theta) \subseteq \text{cset}$, and that $\theta \upharpoonright \{c\} = \{\}$ iff $c \notin \text{chan}(\theta)$.

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

- $\text{len}(\langle \rangle) = 0$,
- $\text{len}(\theta \wedge (c, \mu)) = \text{len}(\theta) + 1$.

Definition 9 (Prefix) The trace $\theta_1$ is a prefix of a trace $\theta_2$, notation $\theta_1 \preceq \theta_2$, iff there exists a trace $\theta_3$ 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 $\theta$ and terminates in state $\sigma$, and
- if $\sigma = \perp$ then it represents a point in a computation of $P$ at which $P$ has performed the communications as described in $\theta$ 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, $PC(\{(\sigma_0, \langle c, 1 \rangle, \sigma)\}) = \{(\sigma_0, \langle \rangle, \perp), (\sigma_0, \langle (c, 1), \perp \rangle), (\sigma_0, \langle (c, 1), \perp \rangle), (\sigma_0, \langle c, 1 \rangle, \perp)\}$. 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:

- $\mathcal{O}[\text{skip}] = PC(\{(\sigma_0, \langle \rangle, \sigma_0)\})$
- $\mathcal{O}[x := c] = PC(\{(\sigma_0, \langle \rangle, (\sigma_0 : x \mapsto E[c](\sigma_0)))\})$
• $O[c]z = PC((\{ (\sigma_0, \theta, \sigma) \mid \text{there exists a value } \mu \in VAL \text{ such that } \theta = ((c, \mu)) \text{ and } \sigma = (\sigma_0 : x \rightarrow \mu) \})$

• $O[P_1 ; P_2] = \{ (\sigma_0, \theta, \perp) \mid (\sigma_0, \theta, \perp) \in O[P_1] \}
\cup \{ (\sigma_0, \theta, 0_2, \sigma) \mid \text{there exists a } \sigma_1 \neq \perp \text{ such that } (\sigma_0, \theta, 0_1, \sigma_1) \in O[P_1] \text{ and } (\sigma_1, \theta, \sigma) \in O[P_2] \}$

• $O\prod_{i=1}^{n} (b_i \rightarrow P_i) = \{ (\sigma_0, \theta, \perp) \mid \text{there exists a } k \in \mathbb{N} \text{ and a list } (\sigma_0, \theta_1, \sigma_1), \ldots, (\sigma_k-1, \theta_k, \sigma_k) \text{ such that } \theta = \theta_1 \cdot \ldots \cdot \theta_k, \sigma = \sigma_k, \text{ and for all } i \in \{0, \ldots, k-1\}: \sigma_i \neq \perp, \mathcal{B}(b_k)(\sigma_i) \text{ and } (\sigma_i, \theta_i+1, \sigma_{i+1}) \in O[\mathcal{G}]; \text{ and } \mathcal{B}(b_k)(\sigma_k) \}$

• $O[\bullet \mathcal{G}] = PC((\{ (\sigma_0, \theta, \sigma) \mid \text{there exists a } k \in \mathbb{N} \text{ and a list } (\sigma_0, \theta_1, \sigma_1), \ldots, (\sigma_k-1, \theta_k, \sigma_k) \text{ such that } \theta = \theta_1 \cdot \ldots \cdot \theta_k, \sigma = \sigma_k, \text{ and for all } i \in \{0, \ldots, k-1\}: \sigma_i \neq \perp, \mathcal{B}(b_k)(\sigma_i) \text{ and } (\sigma_i, \theta_i+1, \sigma_{i+1}) \in O[\mathcal{G}]; \text{ and } \mathcal{B}(b_k)(\sigma_k) \}$

• $O[P_i \parallel P_2] = \{ (\sigma_0, \theta, \sigma) \mid \text{for } i = 1, 2 \text{ there exist } \theta_i, \sigma_i \text{ such that } (\sigma_0, \theta_i, \sigma_i) \in O[P_i] \text{, and } \}
\text{if } \sigma_i = \perp \text{ or } \sigma_2 = \perp \text{ then } \sigma = \perp \text{ else for all } x \in VAR
\{ \sigma(x) = \left\{ \begin{array}{ll} \sigma_i(x) & \text{if } x \in \text{var}(P_i) \\ \sigma_0(x) & \text{if } x \notin \text{var}(P_1 \parallel P_2) \end{array} \right. \\ \theta_i \uparrow \text{chan}(P_i) = \theta_i, \text{ and } \theta \uparrow \text{chan}(P_1 \parallel P_2) = \theta \}$

• $O[P \setminus \text{cset}] = \{ (\sigma_0, \theta \setminus \text{cset}, \sigma) \mid (\sigma_0, \theta, \sigma) \in O[P] \}$

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:

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

\[\square\]

4 Assertion Language and Correctness Formulae

As mentioned before, we use a correctness formula $P \text{ sat } \phi$ to express that process $P$ satisfies safety property $\phi$. 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 $\phi$.

Conform the format of traces in the semantics of the previous section, we use communication record expressions such as $(c, \mu)$, with $c \in \text{CHAN}$ and $\mu \in \text{VAL}$, in assertions. We have channel expressions, e.g. using the operator chan 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, $\emptyset$, traces of one record, e.g. $((c, \mu))$, as well as the concatenation operator $\cdot$ 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 $c!2 \text{ sat } h \uparrow \{ c \} = \emptyset \lor h \uparrow \{ c, 2 \}$. Let $\text{VVAR}$, with typical representative $v$, denote the set of logical value variables ranging over $\text{VAL}$, and let $\text{TVAR}$, with characteristic element $t$, be the set of logical trace variables ranging over $\text{TRACE}$. Assume that $\text{VVAR} \cap \text{TVAR} = \emptyset$.

Table 2 presents the assertion language, with $c \in \text{CHAN}$, $\mu \in \text{VAL}$, $v \in \text{VVAR}$, $t \in \text{TVAR}$, and $\text{cset} \subseteq \text{CHAN}$. 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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Expression Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel expression</td>
<td>`ceexp ::= c</td>
</tr>
<tr>
<td>Value expression</td>
<td>`veexp ::= µ</td>
</tr>
<tr>
<td>Record expression</td>
<td>`rexp ::= (ceexp, veexp)</td>
</tr>
<tr>
<td>Trace expression</td>
<td>`teexp ::= t</td>
</tr>
</tbody>
</table>
| Assertion             | `φ ::= ceexp1 = ceexp2 | veexp1 = veexp2 | terp1 = terp2 |`

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

- `ch(cexp, veexp) ≡ ch((cexp, veexp))`
- `val(cexp, veexp) ≡ val((cexp, veexp))`
- `terp | ceexp ≡ terp(ceexp)`
- `rexp1 = rexp2 ≡ ch(rexp1) = ch(rexp2) ∧ val(rexp1) = val(rexp2)`
- `terp \ cset ≡ terp(CHAN - cset)`
- `last(terp) ≡ terp(len(terp))`
- `terp1 ≤ terp2 ≡ ∃ t : terp1 ▷ t = terp2`
  This expresses that `terp1` is a prefix of `terp2`.
- `terp1 ≤^n terp2 ≡ ∃ t : len(t) ≤ n : terp1 ▷ t = terp2`
  To assert that `terp1` is a prefix of `terp2` which is at most `n` records shorter.
- `terp1 < terp2 ≡ terp1 ≤ terp2 ∧ terp1 ≠ terp2`
  To denote that `terp1` is a strict prefix of `terp2`.
- `terp1 <^n terp2 ≡ ∃ t : 1 < len(t) ≤ n : terp1 ▷ t = terp2`
  To express that `terp1` is a strict prefix of `terp2` which is at most `n` records shorter.
- `teexp[veexp] ≡ teexp(1)^ ... ^ teexp(veexp)`
  To refer to the prefix of `teexp` that has length `veexp`.
- `teexp1 ≤ teexp2 ≡ { ∃ t : t : terp1 ≤ terp2`  
  \( ∀ i : ∃ j > i : ∃ t_1, t_2 : t_1 ▷ terp1(i)^ t_2 ▷ terp1(j) ≤ terp2 \) if `len(terp1) ≤ 1`
  \( ∃ t : t' : terp1 ≤ terp2`  
  \( ∀ i : ∃ j > i : ∃ t_1, t_2 : t_1 ▷ terp1(i)^ t_2 ▷ terp1(j) ≤ terp2 \) if `len(terp1) > 1`
  To denote that `terp1` is a (not necessarily contiguous) subsequence of `terp2`.

Furthermore, we use the standard abbreviations \( ϕ_1 ∨ ϕ_2 \equiv ¬ (¬ ϕ_1 ∧ ¬ ϕ_2) \), and \( ϕ_1 → ϕ_2 \equiv ¬ ϕ_1 ∨ ϕ_2 \). Also, for natural numbers \( x \) and \( y \), we use the relations \( x ≤^n y \) and \( x <^n y \) to denote that \( 0 ≤ y - x ≤ n \), respectively that \( 0 < y - x ≤ n \).

**Definition 12 (Sequence of values)** For a trace `teexp`,

\[
Val(teexp) = \begin{cases} 
{()} & \text{if } teexp = () \\
Val(teexp_0)^{v} & \text{if } teexp = teexp_0^v(c, v)
\end{cases}
\]
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

$$M \text{ sat } Val(h \uparrow m_{out}) \leq 1 \text{ Val}(h \uparrow m_{in})$$

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

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

- $\text{chan}(c) = \emptyset$
- $\text{chan}(ch(rexp)) = \text{chan}(rexp)$
- $\text{chan}(\mu) = \text{chan}(\nu) = \emptyset$
- $\text{chan}(\text{val}(rexp)) = \text{chan}(rexp)$
- $\text{chan}(\text{len}(texp)) = \text{chan}(texp)$
- $\text{chan}(cexp \cdot vexp) = \text{chan}(cexp) \cup \text{chan}(vexp)$
- $\text{chan}(texp(vexp)) = \text{chan}(texp) \cup \text{chan}(vexp)$
- $\text{chan}(t) = \emptyset$
- $\text{chan}(h) = \text{CHAN}$
- $\text{chan}() = \emptyset$
- $\text{chan}(\text{exp}) = \text{chan}(rexp)$
- $\text{chan}(texp_1 \cdot texp_2) = \text{chan}(texp_1) \cup \text{chan}(texp_2)$
- $\text{chan}(texp | cset) = \text{chan}(texp) \cap cset$
- $\text{chan}(cexp_1 \cdot cexp_2) = \text{chan}(cexp_1) \cup \text{chan}(cexp_2)$
- $\text{chan}(vexp_1 \cdot vexp_2) = \text{chan}(vexp_1) \cup \text{chan}(vexp_2)$
- $\text{chan}(texp_1 \cdot texp_2) = \text{chan}(texp_1) \cup \text{chan}(texp_2)$
- $\text{chan}(\phi_1 \land \phi_2) = \text{chan}(\phi_1) \cup \text{chan}(\phi_2)$
- $\text{chan}(\neg \phi) = \text{chan}(\exists v : \phi) = \text{chan}(\exists t : \phi) = \text{chan}(\phi)$

Next we define the meaning of assertions. An assertion is interpreted with respect to a pair $(\theta, \gamma)$. Trace $\theta$ gives $h$ its value, and environment $\gamma$ 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 $\theta$, and an environment $\gamma$, denoted by $C[cexp][\theta, \gamma]$, yielding a value in $\text{CHAN} \cup \{\dagger\}$, the value of a value expression $vexp$ in the trace $\theta$, and an environment $\gamma$, denoted by $V[vexp][\theta, \gamma]$, yielding a value in $\text{VAL} \cup \{\dagger\}$, the value of a record expression $rexp$ in the trace $\theta$, and an environment $\gamma$, denoted by $R[rexp][\theta, \gamma]$, yielding a value in $\text{CHAN} \times \text{VAL} \cup \{\dagger\}$, and the value of a trace expression $texp$ for trace $\theta$, and an environment $\gamma$, denoted by $T[texp][\theta, \gamma]$, yielding a value in $\text{TRACE} \cup \{\dagger\}$.
\begin{itemize}
  \item $C[c](\theta, \gamma) = c$
  \item $C[ch(rexp)](\theta, \gamma) = \begin{cases} 
    \top & \text{iff } R[rexp](\theta, \gamma) = \top \\
    c & \text{iff there exists a } \mu \text{ such that } R[rexp](\theta, \gamma) = (c, \mu)
  \end{cases}$
  \item $V[\mu](\theta, \gamma) = \mu$
  \item $V[v](\theta, \gamma) = \gamma(v)$
  \item $V[val(rexp)](\theta, \gamma) = \begin{cases} 
    \top & \text{iff } R[rexp](\theta, \gamma) = \top \\
    \mu & \text{iff there exists a } c \text{ such that } R[rexp](\theta, \gamma) = (c, \mu)
  \end{cases}$
  \item $V[len(texp)](\theta, \gamma) = \begin{cases} 
    \top & \text{iff } T[texp](\theta, \gamma) = \top \\
    \text{len}(T[texp](\theta, \gamma)) & \text{otherwise}
  \end{cases}$
  \item $R[(cexp, vexp)](\theta, \gamma) = \begin{cases} 
    \top & \text{iff } C[cexp](\theta, \gamma) = \top \text{ or } V[vexp](\theta, \gamma) = \top \\
    (C[cexp](\theta, \gamma), V[vexp](\theta, \gamma)) & \text{otherwise}
  \end{cases}$
  \item $R[texp(vexp)](\theta, \gamma) = \begin{cases} 
    (c, \mu) & \text{iff there exist } \theta_1 \text{ and } \theta_2 \text{ such that } len(\theta_1) = V[vexp](\theta, \gamma) - 1 \\
    \top & \text{otherwise}
  \end{cases}$
  \item $T[l](\theta, \gamma) = \gamma(t)$
  \item $T[h](\theta, \gamma) = \emptyset$
  \item $T[()](\theta, \gamma) = \emptyset$
  \item $T[rexp](\theta, \gamma) = \begin{cases} 
    \top & \text{iff } R[rexp](\theta, \gamma) = \top \\
    ((c, \mu)) & \text{iff } R[rexp](\theta, \gamma) = (c, \mu)
  \end{cases}$
  \item $T[texp_1; texp_2](\theta, \gamma) = \begin{cases} 
    \top & \text{iff } T[texp_1](\theta, \gamma) = \top \text{ or } T[texp_2](\theta, \gamma) = \top \\
    T[texp_1](\theta, \gamma); T[texp_2](\theta, \gamma) & \text{otherwise}
  \end{cases}$
  \item $T[texp; \text{set}](\theta, \gamma) = \begin{cases} 
    \top & \text{iff } T[texp](\theta, \gamma) = \top \\
    T[texp](\theta; \text{set}, \gamma) & \text{otherwise}
  \end{cases}$
\end{itemize}

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

\[
(\gamma : I \mapsto \lambda)(m) = \begin{cases}
  \lambda & \text{if } m \equiv I \\
  \gamma(m) & \text{if } m \not\equiv I
\end{cases}
\]

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

\begin{itemize}
  \item $(\theta, \gamma) \models cexp_1 = cexp_2$ iff $C[cexp_1](\theta, \gamma) = C[cexp_2](\theta, \gamma)$ and $C[cexp_1](\theta, \gamma) \neq \top$
  \item $(\theta, \gamma) \models vexp_1 = vexp_2$ iff $V[vexp_1](\theta, \gamma) = V[vexp_2](\theta, \gamma)$ and $V[vexp_1](\theta, \gamma) \neq \top$
  \item $(\theta, \gamma) \models texp_1 = texp_2$ iff $T[texp_1](\theta, \gamma) = T[texp_2](\theta, \gamma)$ and $T[texp_1](\theta, \gamma) \neq \top$
  \item $(\theta, \gamma) \models \phi_1 \land \phi_2$ iff $(\theta, \gamma) \models \phi_1$ and $(\theta, \gamma) \models \phi_2$
\end{itemize}
• \((\theta, \gamma) \models \neg \phi \text{ iff not } (\theta, \gamma) \models \phi\)
• \((\theta, \gamma) \models \exists v : \phi \text{ iff there exists a value } \mu \text{ such that } (\theta, (\gamma : v \mapsto \mu)) \models \phi\)
• \((\theta, \gamma) \models \exists t : \psi \text{ iff there exists a trace } \hat{\theta} \text{ such that } (\theta, (\gamma : t \mapsto \hat{\theta})) \models \phi\)

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

We conclude this section by defining when a correctness formula \(P \text{ sat } \phi\) is valid.

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

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

\[\Diamond\]

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 \(\chi\) 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 \(\chi\). 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 \text{in} and outputs its square via the observable channel \text{out}, may be \(((\text{in}, 1), (\text{out}, 1), (\text{in}, 3), (\text{out}, 9))\). The exceptional behaviour resulting from Square's output channel becoming transiently stuck at zero can be defined using a predicate \(\text{StuckAtZero}\) asserting that \(h_{old}\) and \(h\) are equally long, if the \(i\)th element of \(h_{old}\) records an in communication then it is equal to the \(i\)th element of \(h\), and if the \(i\)th element of \(h_{old}\) records an out communication then so does the \(i\)th 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 \((\text{Square} \mid \text{StuckAtZero})\) to indicate execution of process Square under the assumption of \(\text{StuckAtZero}\), we still have \(((\text{in}, 1), (\text{out}, 1), (\text{in}, 3), (\text{out}, 9)) \in H[(\text{Square} \mid \text{StuckAtZero})]\), but also, for instance, \(((\text{in}, 1), (\text{out}, 1), (\text{in}, 3), (\text{out}, 0)) \in H[(\text{Square} \mid \text{StuckAtZero})]\). 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 \(\text{StuckAtZero}\) can formally be defined as follows:

\[
\text{StuckAtZero} \triangleq \begin{align*}
\text{len}(h_{old}) &= \text{len}(h) \\
\land \forall i : 1 \leq i \leq \text{len}(h) : \\
& \quad \text{ch}(h \upharpoonright \{\text{in, out}\}(i)) = \text{ch}(h_{old} \upharpoonright \{\text{in, out}\}(i)) \\
& \quad \land \text{val}(h \upharpoonright \text{in}(i)) = \text{val}(h_{old} \upharpoonright \text{in}(i)) \\
& \quad \land \text{val}(h \upharpoonright \text{out}(i)) = \text{val}(h_{old} \upharpoonright \text{out}(i)) \\
& \quad \lor \text{val}(h \upharpoonright \text{out}(i)) = 0
\end{align*}
\]

\[\Diamond\]

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:

\[
\begin{align*}
\text{Cor} & \triangleq \quad \text{len}(h_{\text{old}}) = \text{len}(h) \\
& \quad \land \forall i : 1 \leq i \leq \text{len}(h) : \quad \text{ch}(h \uparrow \{\text{in}, \text{out}\}(i)) = \text{ch}(h_{\text{old}} \uparrow \{\text{in}, \text{out}\}(i)) \\
& \quad \land \text{val}(h \uparrow \text{in}(i)) = \text{val}(h_{\text{old}} \uparrow \text{in}(i)) 
\end{align*}
\]

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

\[
\begin{align*}
\text{Loss} & \triangleq \quad h \uparrow \{m_{\text{in}}, m_{\text{out}}\} \subseteq h_{\text{old}} \uparrow \{m_{\text{in}}, m_{\text{out}}\} \\
& \quad \land h \uparrow m_{\text{in}} = h_{\text{old}} \uparrow m_{\text{in}} 
\end{align*}
\]

We extend the assertion language with trace expression term \(h_{\text{old}}\). Sentences of the extended language are called transformation expressions, with typical representative \(\psi\). A transformation expression is interpreted with respect to a triple \((\theta_0, \theta, \gamma)\). Trace \(\theta_0\) gives \(h_{\text{old}}\) its value, and, in conformity with the foregoing, trace \(\theta\) gives \(h\) its value, and environment \(\gamma\) interprets the logical variables of \(\text{VVAR} \cup \text{TVAR}\).

The meaning of assertions, as defined on page 9, can easily be adapted for transformation expressions; the only new clause is \(T[h_{\text{old}}](\theta_0, \theta, \gamma) = \theta_0\). Similarly, the channels occurring in a transformation expression are defined as in Definition 13 with the extra clause \(\text{chan}(h_{\text{old}}) = \text{CHAN}\).

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

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

In this section we define the trace semantics \(H[(FP|\chi)]\) of failure prone process \((FP|\chi)\), that is, process \(FP\) under assumption of fault hypothesis \(\chi\). A fault hypothesis \(\chi\) 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 \(h_{\text{old}}\). Formally,

\[
\models \chi[h_{\text{old}}/h] 
\]

Furthermore, we require a fault hypothesis \(\chi\) to be prefix closedness preserving, that is, we require

\[
\models \chi \land t < h \rightarrow \exists h_{\text{old}} \subseteq h_{\text{old}} : \chi[t/h, h_{\text{old}}/h_{\text{old}}] 
\]

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 \((FP)\) ::= | \(P\) | \((FP_1 | FP_2)\) | \((FP \cset)\) | \((FP|\chi)\) |
|---|---|---|---|---|

We require, in \((FP|\chi)\), that \(\text{chan}(\chi) \subseteq \text{chan}(FP)\). Hence, \(\text{chan}((FP|\chi)) = \text{chan}(FP)\), and, as before, \(\text{chan}(FP_1 | FP_2) = \text{chan}(FP_1) \cup \text{chan}(FP_2)\), and \(\text{chan}(FP \cset) = \text{chan}(FP) - \cset\).

As we are only interested in the traces of a process, the semantics of a failure prone process \(FP\) is inductively defined as follows:
• \( \mathcal{H}[FP_1 \parallel FP_2] = \{ \theta \mid \text{for } i = 1, 2, \theta \uparrow \text{chan}(FP_i) \in \mathcal{H}[FP_i], \text{ and } \theta \uparrow \text{chan}(FP_1 \parallel FP_2) = \theta \} \)

• \( \mathcal{H}[FP \setminus \text{cset}] = \{ \theta \setminus \text{cset} \mid \theta \in \mathcal{H}[FP] \} \)

• \( \mathcal{H}[(FP\chi)] = \{ \theta \mid \text{there exists a } \theta_0 \in \mathcal{H}[FP] \text{ such that, for all } \gamma, (\theta_0, \theta, \gamma) \models \chi, \text{ and } \theta_0 \uparrow \text{chan}(FP) = \theta \} \)

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

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

Proof. See Appendix A. \( \Box \)

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

\[
(\psi_1 \psi_2) \triangleq \exists t : \psi_1[t/h] \land \psi_2[t/h_{\omega\omega}] 
\]

where \( t \) must be fresh. \( \Diamond \)

From this definition we easily obtain the following lemma.

Lemma 3 (Composite fault hypothesis)

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

Proof. See Appendix B. \( \Box \)

The following lemmas are easy to prove by structural induction.

Lemma 4 (Projection) Consider \( \text{cset} \subseteq \text{CHAN} \) and transformation expression \( \psi \). If \( \text{chan}(\psi) \subseteq \text{cset} \) then, for all \( \theta_0, \theta, \) and \( \gamma \)

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

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

Lemma 5 (Substitution) Consider transformation expression \( \psi \).

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

(b) \( (\theta_0, \theta, \gamma) \models \psi[\text{texp}/h_{\omega\omega}] \) iff \( (T[\text{texp}]\{(\theta_0, \theta, \gamma), \theta, \omega\omega\}) \models \psi \)

Since the interpretation of assertions has not changed, the validity of correctness formula \( FP \text{ sat } \phi \) 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 \( \phi \) a correctness formula \( FP \text{ sat } \phi \) is valid, denoted by \( \models FP \text{ sat } \phi \), 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)

\[
\frac{FP \text{ sat } \phi_1, \phi_1 \rightarrow \phi_2}{FP \text{ sat } \phi_2}
\]

Rule 6.2 (Conjunction)

\[
\frac{FP \text{ sat } \phi_1, FP \text{ sat } \phi_2}{FP \text{ sat } \phi_1 \land \phi_2}
\]

Rule 6.3 (Invariance)

\[
cset \cap \text{chan}(FP) = \emptyset \\
\frac{FP \text{ sat } h \upharpoonright cset = \emptyset}{cset \cap \text{chan}(FP) = \emptyset}
\]

From this rule we can derive the following lemma.

Lemma 6 (Invariance)

\[
P \text{ sat } h \setminus \text{chan}(P) = \emptyset
\]

Rule 6.4 (Parallel composition)

\[
\frac{FP_1 \text{ sat } \phi_1, FP_2 \text{ sat } \phi_2}{FP_1 \parallel FP_2 \text{ sat } \phi_1 \land \phi_2}
\]

provided that \(\text{chan}(\phi_1) \cap \text{chan}(FP_2) \subseteq \text{chan}(FP_1)\) and \(\text{chan}(\phi_2) \cap \text{chan}(FP_1) \subseteq \text{chan}(FP_2)\), 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 \(\phi_2\) of process \(FP_1\) should be projected onto a subset of \(\text{chan}(FP_1)\). Recall that we do not allow shared variables.

Rule 6.5 (Hiding)

\[
\frac{FP \text{ sat } \phi, \text{chan}(\phi) \cap cset = \emptyset}{FP \setminus cset \text{ sat } \phi}
\]

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

Rule 6.6 (Fault hypothesis introduction)

\[
\frac{FP \text{ sat } \phi}{(FP\backslash x) \text{ sat } (\phi \backslash x)}
\]

Observe that since \(\phi\) is an assertion, \(h_{old}\) does not occur in \(\phi\), and hence also \((\phi \backslash x)\) is an assertion.

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

\[
(M \parallel Loss) \text{ sat } \exists t : (\text{Val}(h \uparrow m_{out}) \leq \text{Val}(h \uparrow m_{in}))[t/h] \\
\wedge (h \uparrow \{m_{in},m_{out}\} \ni h_{old} \uparrow \{m_{in},m_{out}\} \wedge h \uparrow m_{in} = h_{old} \uparrow m_{in})[t/h_{old}]
\]
which reduces to

\[(M1Loss) \text{ sat } \exists t : Val(t[m_{\text{out}}]) \leq Val(t[m_{\text{in}}])
\]
\[\land h \{m_{\text{in}}, m_{\text{out}}\} \leq t \{m_{\text{in}}, m_{\text{out}}\} \land h[m_{\text{in}}] = t[m_{\text{in}}]\]

Now, for instance, by \(h \{m_{\text{in}}, m_{\text{out}}\} \leq t \{m_{\text{in}}, m_{\text{out}}\}\), we have, obviously, \(h[m_{\text{out}}] \leq t[m_{\text{out}}]\), which, since \(Val(t[m_{\text{out}}]) \leq Val(t[m_{\text{in}}])\), implies \(Val(h[m_{\text{out}}]) \leq Val(t[m_{\text{in}}])\). Then, by \(t[m_{\text{in}}] = h[m_{\text{in}}]\), we obtain

\[(M1Loss) \text{ sat } Val(h[m_{\text{out}}]) \leq Val(h[m_{\text{in}}])\]

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

\[(M1Loss) \text{ sat } \forall i : ch(h \{m_{\text{in}}, m_{\text{out}}\}(i)) = m_{\text{out}} :
\]
\[val(h \{m_{\text{in}}, m_{\text{out}}\}(i)) = val(last(h \{m_{\text{in}}, m_{\text{out}}\}(i)m_{\text{in}}))\]

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

**Rule 6.7 (Prefixing)**

\[
\frac{FP \text{ sat } \phi}{FP \text{ sat } \forall t \leq h : \phi[t/h]}\]

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](image-url)
Definition 19 (Abbreviations) Throughout this section we use the following abbreviations:

- $c(i) \equiv \text{val}(hct(i))$
- $c^{old}(i) \equiv \text{val}(h_{old}ct(i))$
- $c'(i) \equiv \text{val}(t^c(i))$

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.

\[ C_j \text{ sat } \forall i: out(i) = f(in(i)) \]

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

\[ \text{Voter sat } \forall i, v : out(i) = v \iff (\exists k, l : k \neq l : out_k(i) = out_l(i) = v) \]

Finally, component $ln$ conforms to

\[ \text{ln sat } \forall i, j : in_j(i) = in(i) \]

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.

\[ \text{Cor}^{51} \triangleq \forall i : \exists k, l : k \neq l : out_k(i) = out^{old}_k(i) \land out_l(i) = out^{old}_l(i) \land h \uparrow \{in_1, in_2, in_3\} = h_{old} \uparrow \{in_1, in_2, in_3\} \]

We show that, given this assumption, the system $\text{ln} \parallel ((C_1 || C_2 || C_3) \parallel \text{Cor}^{51}) \parallel \text{Voter}$ produces the desired output, that is, hiding internal channels we prove

\[ ((\text{ln} || ((C_1 || C_2 || C_3) \parallel \text{Cor}^{51}) \parallel \text{Voter}) \setminus \{in_1, in_2, in_3, out_1, out_2, out_3\} \text{ sat } \forall i: out(i) = f(in(i)) \]

**Proof.** By (Parallel Composition)

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

By (Fault Hypothesis Introduction)

\[ ((C_1 || C_2 || C_3) \parallel \text{Cor}^{51}) \text{ sat } \exists t : (\bigwedge_{j=1}^{3} \forall i : out_j(i) = f(in_j(i))[t/h] \land Cor^{51}[t/h_{old}] \]

which is, by definition, equivalent to

\[ ((C_1 || C_2 || C_3) \parallel \text{Cor}^{51}) \text{ sat } \exists t : \bigwedge_{j=1}^{3} \forall i : out_j(i) = f(in_j(i)) \land \forall i : \exists k, l : k \neq l : out_k(i) = out_l(i) = out_{t'}(i) \land h \uparrow \{in_1, in_2, in_3\} = t \uparrow \{in_1, in_2, in_3\} \]

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

$\left( ((C_1 || C_2 || C_3) || Cor^{\leq 1}) \right) \quad \text{sat} \quad \forall i : \exists k, l : k \neq l : out_k(i) = f(in_k(i)) \land out_l(i) = f(in_l(i))$

By (Parallel Composition), we get

$\left( In \left( ((C_1 || C_2 || C_3) || Cor^{\leq 1}) \right) \right) \quad \text{sat} \quad \forall i : \exists k, l : k \neq l : out_k(i) = f(in_k(i)) \land out_l(i) = f(in_l(i)) \land \forall i, j : in_j(i) = in(i)$

and thus

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

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

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

Finally, by (Hiding), we obtain

$\left( In \left( ((C_1 || C_2 || C_3) || Cor^{\leq 1}) \right) \right) \setminus \{in_1, in_2, in_3, out_1, out_2, out_3\} \quad \text{sat} \quad \forall i : out(i) = f(in(i)) \quad \square$

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 $ABP \models S \parallel (M \parallel Loss) \parallel (A \parallel Loss) \parallel R$ satisfies the safety property that $Val(h \parallel out) \leq Val(h \parallel in)$. We use the following functions:

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

$$RDM_{Msg}(texp) = \begin{cases} \emptyset & \text{if } texp = \emptyset \\ RDM_{Msg}(texp_0) & \text{if } texp = texp_0 \iff (c, (d, b)) \text{ and } b = bit(val(last(texp_0))) \\ RDM_{Msg}(texp_0) \cup (c, (d, b)) & \text{if } texp = texp_0 \cup (c, (d, b)) \text{ and } b \neq bit(val(last(texp_0))) \end{cases}$$

\(\square\)
Definition 21 (Removal of duplicate acknowledgements) For a trace $texp$ that consists only of $\text{chan}(A) \times A$ records,

$$RDAck(texp) = \begin{cases} \emptyset & \text{if } texp = \emptyset \\ RDAck(texp_0) & \text{if } texp = texp_0 \cup (c, b) \text{ and } b = \text{val(last}(texp_0)) \\ RDAck(texp_0) \cup (c, b) & \text{if } texp = texp_0 \cup (c, b) \text{ and } b \neq \text{val(last}(texp_0)) \end{cases}$$

Definition 22 (Sequence of data) For a trace $texp$ of $\text{chan}(M) \times M$ records,

$$Dat(texp) = \begin{cases} \emptyset & \text{if } texp = \emptyset \\ \text{Msg}(texp_0) \cup d & \text{if } texp = texp_0 \cup (d, b) \end{cases}$$

Definition 23 (Sequence of bits) For a trace $texp$ of $\text{chan}(M) \times M$ records,

$$Bit(texp) = \begin{cases} \emptyset & \text{if } texp = \emptyset \\ \text{Bit}(texp_0) \cup b & \text{if } texp = texp_0 \cup (c, (d, b)) \end{cases}$$

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

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

$$S \text{ sat } Dat(RDMsg(h \uparrow m_{\text{in}})) \preceq^1 Val(h \uparrow \text{in})$$

$$\land Val(RDAck(h \uparrow a_{\text{out}})) \preceq^1 Bit(RDMsg(h \uparrow m_{\text{in}}))$$

Similarly, we obtain the following specification for receiver $R$.

$$R \text{ sat } Val(h \uparrow \text{out}) \preceq^1 Dat(RDMsg(h \uparrow m_{\text{out}}))$$

$$\land Val(RDAck(h \uparrow a_{\text{in}})) \preceq^1 Bit(RDMsg(h \uparrow m_{\text{out}}))$$

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

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

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

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

$$ABP \text{ sat } Val(RDAck(h \uparrow a_{\text{in}})) \preceq^1 Bit(RDMsg(h \uparrow m_{\text{out}})) \quad (A4)$$
From Example 5 we learned that $(M\mid Loss) \models Val(h \uparrow m_{out}) \subseteq Val(h \uparrow m_{in})$ which implies that

\[ ABP \models \text{len}(RDM_{out}(h \uparrow m_{out})) \leq \text{len}(RDM_{in}(h \uparrow m_{in})) \]  

(A5)

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

\[ ABP \models \forall i : ch(h(i)) = m_{out} : val(h(i)) = val(last(h[i]\uparrow m_{in})) \]  

(A6)

For medium $A$ we obtain similarly

\[ ABP \models \text{len}(RDAck(h \uparrow a_{out})) \leq \text{len}(RDAck(h \uparrow a_{in})) \]  

(A7)

\[ ABP \models \forall i : ch(h(i)) = a_{out} : val(h(i)) = val(last(h[i]\uparrow a_{in})) \]  

(A8)

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

Lemma 7 (Persistency)

\[ ABP \models Val(RDAck(h \uparrow a_{out})) \leq Val(RDAck(h \uparrow a_{in})) \]

\[ \land \quad Dat(RDM_{out}(h \uparrow m_{out})) \leq Dat(RDM_{in}(h \uparrow m_{in})) \]

Proof. See Appendix C. □

Then, by (Consequence), we have

\[ ABP \models Dat(RDM_{out}(h \uparrow m_{out})) \leq Dat(RDM_{in}(h \uparrow m_{in})) \]

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

\[ ABP \models Val(h \uparrow m_{out}) \subseteq Val(h \uparrow m_{in}) \]

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 $FP \models \phi$ is derivable, then it is valid. Furthermore, we prove the proof system to be complete, that is, we prove that, if a correctness formula $FP \models \phi$ 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 $\phi$ is derivable, we add the following axiom to our proof theory.

Axiom 1 (Relative completeness assumption) For an assertion $\phi$, 

\[ \vdash \phi \quad \text{if} \quad \models \phi \]
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) \( \models FP \text{ sat } \phi \).

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

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

Let \( \vdash P \text{ sat } \phi \) denote that correctness formula \( P \text{ sat } \phi \) 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 \( \phi \). This leads to the definition of network completeness.

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

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 \( \phi \) which is precise for \( P \) and \( \vdash P \text{ sat } \phi \). Then, for any failure prone process \( FP \) there exists an assertion \( \xi \) which is precise for \( FP \) and \( \vdash FP \text{ sat } \xi \).

**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 \( \phi_1 \) is precise for \( FP \) and \( \vdash FP \text{ sat } \phi_2 \) then

\[ \vdash (\phi_1 \land h(\text{chan(}\phi_2)-\text{chan(}FP)) = () \rightarrow \phi_2 \]

**Proof.** Assume that \( \phi_1 \) is precise for \( FP \), and that \( \vdash FP \text{ sat } \phi_2 \).

Consider any \( \theta \) and \( \gamma \). Assume that \( (\theta, \gamma) \models \phi \land h(\text{chan(}\phi_2)-\text{chan(}FP)) = () \).

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

By (1), \( (\theta \upharpoonright \text{chan(}FP), \gamma) \models \phi_2 \).

By (2), we have that \( (\theta, \gamma) \models h(\text{chan(}\phi_2)-\text{chan(}FP)) = () \). Hence, \( \theta \upharpoonright (\text{chan(}\phi_2)-\text{chan(}FP)) = () \), and thus, \( \theta \upharpoonright \text{chan(}FP) = \theta \upharpoonright (\text{chan(}FP) \cup (\text{chan(}\phi_2)-\text{chan(}FP)) = \theta \upharpoonright (\text{chan(}FP) \cup \text{chan(}\phi_2)) \). Hence, we obtain from (3) that \( (\theta \upharpoonright (\text{chan(}FP) \cup \text{chan(}\phi_2)), \gamma) \models \phi_2 \), 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 \( \phi \) with \( \vdash P \models \phi \). Then, by the preciseness preservation lemma, for any failure prone process \( FP \) there exists an assertion \( \xi \) which is precise for \( FP \) and \( \vdash FP \models \xi \).

Assume \( \models FP \models \eta \). Since \( (\text{chan}(\eta) - \text{chan}(FP)) \cap \text{chan}(FP) = \emptyset \), we obtain, by (Invariance),
\[
\vdash FP \models h \uparrow (\text{chan}(\eta) - \text{chan}(FP)) = \emptyset \tag{1}
\]

By (1) and (2), \( \vdash FP \models \xi \land h \uparrow (chan(\eta) - \text{chan}(FP)) = \emptyset \), and thus, by the preciseness consequence lemma, the relative completeness assumption, and (Consequence), \( \vdash FP \models \eta \).

\[ \square \]

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, \( \text{STATES} = \{ \sigma | \sigma : \{x,y\} \rightarrow \mathbb{N} \} \). 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 \( \langle p \rangle S[q] \) where \( S \) is a program, and \( p \) and \( q \) are assertions expressed in a first-order language. Informally, the triple \( \langle p \rangle S[q] \) 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


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

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

(c) Assume $\theta \in \mathcal{H}[(FP \setminus \text{set})]$, that is, assume that there exists a $\theta_0 \in \mathcal{H}[FP]$ such that, for all $\gamma$, $(\theta_0, \theta, \gamma) \models \chi$. Consider $\theta' \preceq \theta$. Using $\gamma = (\gamma : t \rightarrow \theta')$, $t$ fresh, we have $(\theta_0, \theta, \gamma) \models \chi$. Since $\theta' \preceq \theta$, we have $(\theta_0, \theta, \gamma) \models t \preceq h$. Consequently, $(\theta_0, \theta, \gamma) \models \chi \cap t \preceq h$. By the syntactic restriction on $\chi$, we obtain that $(\theta_0, \theta, \gamma) \models \exists t_\text{old} \preceq h_\text{old} : \chi \cap [t/h, t_\text{old}/h_\text{old}]$. Thus there exists a $\theta''$ such that $(\theta_0, \theta, (\gamma : t_\text{old} \rightarrow \theta'')) \models t_\text{old} \preceq h_\text{old} \land \chi \cap [t/h, t_\text{old}/h_\text{old}]$. Then, by the syntactic lemma, $(\theta'', \gamma(t), (\gamma : t_\text{old} \rightarrow \theta'')) \models \chi$. Since $\gamma(t) = \theta'$ and $t$ and $t_\text{old}$ do not occur in $\chi$, we obtain $\theta'' \preceq \theta$.

\[\square\]

B Proof of the composite fault hypothesis lemma

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

Assume $\theta \in \mathcal{H}[(FP \setminus (\chi_1 \cup \chi_2))]$, that is, assume that there exists a $\theta_0 \in \mathcal{H}[FP]$ such that, for any $\gamma$, $(\theta_0, \theta, \gamma) \models (\chi_1, \chi_2)$. By definition this equals $(\theta_0, \theta, \gamma) \models \exists t : x_1[t/h] \land x_2[t/h_\text{old}]$, i.e. there exists a $\theta_1$ such that, for $\gamma = (\gamma : t \rightarrow \theta_1)$, $(\theta_0, \theta, \gamma) \models x_1[t/h] \land x_2[t/h_\text{old}]$. Observe that $\mathcal{H}[(\theta_0, \theta, \gamma) \models \theta]$. By the substitution lemma, $(\theta_0, \theta, \gamma) \models x_1[t/h] \land x_2[t/h_\text{old}]$ if $(\theta_0, \theta, \gamma) \models x_1$ and $(\theta_0, \theta, \gamma) \models x_2$. Hence, $\theta \in \mathcal{H}[(FP \setminus (\chi_1 \cup \chi_2))]$ if there exists a $\theta_0 \in \mathcal{H}[FP]$ such that, for any $\gamma$, there exists a $\theta_1$ such that $(\theta_0, \theta_1, \gamma) \models x_1$ and $(\theta_1, \theta, \gamma) \models x_2$. Then, $\theta \in \mathcal{H}[(FP \setminus (\chi_1 \cup \chi_2))]$ if there exists a $\theta_1 \in \mathcal{H}[FP \setminus \chi_1]$ such that $(\theta_1, \theta, \gamma) \models x_2$. Equivalently, $\theta \in \mathcal{H}[(FP \setminus (\chi_1 \cup \chi_2))]$ if $\theta \in \mathcal{H}[(FP \setminus \chi_1) \parallel \chi_2]$. \[\square\]

C Proof of the persistency lemma

By induction on the length of $h$.

(Base Step) The case $h = \emptyset$ is trivial.

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

$$\text{Val}(\text{RDAck}(t \uparrow a_{\text{out}})) \preceq \text{Val}(\text{RD Ack}(t \uparrow a_{\text{in}}))$$ (1),

and

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

Four cases need examination:

\[23\]
1. \( h = t^*(m_{in}, (v, b)), \) where \( b \neq \text{val}(\text{last}(t^*[m_{in}])) \).

By (A2), we have that \( \text{len}(\text{RDAck}(h \mid a_{out})) \leq \text{len}(\text{RDMsg}(h \mid m_{in})) \). Since \( t < h \), by (A2) and (Prefixing), we obtain \( \text{len}(\text{RDAck}(t \mid a_{out})) \leq \text{len}(\text{RDMsg}(t \mid m_{in})) \). Then, because \( h = t^*(m_{in}, (v, b)) \), we conclude that \( \text{len}(\text{RDAck}(t \mid a_{out})) = \text{len}(\text{RDMsg}(t \mid m_{in})) \) (3).

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

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

2. \( h = t^*(m_{out}, (v, b)), \) where \( b \neq \text{val}(\text{last}(t \mid m_{out})) \).

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

Since \( t < h \), by (A2) and (Prefixing), \( \text{len}(\text{RDAck}(t \mid a_{out})) \leq \text{len}(\text{RDMsg}(t \mid m_{out})) \) (6).

Since \( t < h \), we have, by (A5) and (Prefixing), \( \text{len}(\text{RDMsg}(t \mid m_{out})) \leq \text{len}(\text{RDMsg}(t \mid m_{in})) \). Then, by (5) and (6), \( \text{len}(\text{RDMsg}(t \mid m_{out})) \leq \text{len}(\text{RDMsg}(t \mid m_{in})) \) (7).

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

3. \( h = t^*(a_{in}, b), \) where \( b \neq \text{val}(\text{last}(t \mid a_{in})) \).

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

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

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

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

4. \( h = t^*(a_{out}, b), \) where \( b \neq \text{val}(\text{last}(t \mid a_{out})) \).

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

By (2), we have that \( \text{len}(\text{RDMsg}(t \mid m_{out})) \leq \text{len}(\text{RDMsg}(t \mid m_{in})) \). Then, by (11), we conclude \( \text{len}(\text{RDAck}(t \mid a_{out})) \leq \text{len}(\text{RDMsg}(t \mid m_{in})) \) (12).
Since $t < h$, by (A4) and (Prefixing), $\text{len}(\text{RDAck}(\mathbf{a}_{\text{out}})) \leq \text{len}(\text{RDMsg}(\mathbf{a}_{\text{in}}))$ (13).

Then, by (12) and (13), we conclude $\text{len}(\text{RDAck}(\mathbf{a}_{\text{out}})) < \text{len}(\text{RDAck}(\mathbf{a}_{\text{in}}))$ (14).

Assume that $\text{len}(\text{RDAck}(\mathbf{a}_{\text{out}})) = \text{len}(\text{RDAck}(\mathbf{a}_{\text{in}}))$. Then, since $h = f(a_{\text{out}}, b)$, where $b \neq \text{val}((\text{last}(\mathbf{a}_{\text{out}})))$, we obtain $\text{len}(\text{RDAck}(h_{\text{out}})) = \text{len}(\text{RDAck}(h_{\text{in}})) + 1$, which conflicts with (A7). Consequently, by (14), $\text{len}(\text{RDAck}(\mathbf{a}_{\text{out}})) < \text{len}(\text{RDAck}(\mathbf{a}_{\text{in}}))$, which, combined with (1), yields $\text{Val}(\text{RDMsg}(\mathbf{a}_{\text{out}}))$. Finally, since, by (A8), we have that $b = \text{val}((\text{last}(h_{\mathbf{a}_{\text{in}}}))$, we obtain $\text{Val}(\text{RDAck}(h_{\mathbf{a}_{\text{out}}})) = \text{Val}(\text{RDAck}(h_{\mathbf{a}_{\text{in}}}))$, from which we conclude that the theorem holds.

\[\square\]

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 $\emptyset \in \mathcal{H}[FP]$ then chan($\emptyset$) $\subseteq$ chan($FP$). Thus, cset $\cap$ chan($FP$) $= \emptyset$ implies chan($\emptyset$) $\cap$ 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 $\phi_1, FP_2$ sat $\phi_2$ (2). We have to prove $\models FP_1 || FP_2$ sat $\phi_1 \land \phi_2$. Consider any $\gamma$. Let $\emptyset \in \mathcal{H}[FP_1 || FP_2]$. By the definition of the semantics, we have, for $i = 1, 2$, $\emptyset \upharpoonright \text{chan}(FP_i) \in \mathcal{H}[FP_i]$ and $\emptyset \upharpoonright \text{chan}(FP_1 || FP_2) = \emptyset$. Since $\emptyset \upharpoonright \text{chan}(FP_1) \in \mathcal{H}[FP_1]$, we obtain, by (2), $(\emptyset \upharpoonright \text{chan}(FP_1), \gamma) \models $ $\phi_1$. By projection lemma (a) $(\emptyset \upharpoonright \text{chan}(FP_1), \gamma) \models $ $\phi_1$, thus $(\emptyset \upharpoonright \text{chan}(FP_2), \gamma) \models $ $\phi_1$. By (1), we obtain that $\text{chan}(FP_2) \cap \text{chan}(\phi_1) \subseteq \text{chan}(FP_1) \cap \text{chan}(\phi_1)$, from which we conclude that $(\text{chan}(FP_2) \cap \text{chan}(\phi_1)) \cup (\text{chan}(FP_1) \cap \text{chan}(\phi_1)) \subseteq \text{chan}(FP_1) \cap \text{chan}(\phi_1)$. Consequently, we have that $(\text{chan}(FP_2) \cap \text{chan}(\phi_1)) \cup (\text{chan}(FP_1) \cap \text{chan}(\phi_1)) = \text{chan}(FP_1) \cap \text{chan}(\phi_1)$, from which we deduce $\text{chan}(FP_1) \cap \text{chan}(\phi_1) = (\text{chan}(FP_1) \cup \text{chan}(FP_2)) \cap \text{chan}(\phi_1) = \text{chan}(FP_1 || FP_2) \cap \text{chan}(\phi_1)$. By similar reasoning, $\text{chan}(FP_2) \cap \text{chan}(\phi_2) = \text{chan}(FP_1 || FP_2) \cap \text{chan}(\phi_2)$. Consequently, for $i = 1, 2$, $(\emptyset \upharpoonright \text{chan}(FP_2 || FP_2) \cap \text{chan}(\phi_i)), \gamma) \models $ $\phi_i$. Hence, $(\emptyset \upharpoonright \text{chan}(FP_1 || FP_2), \gamma) \models $ $\phi_1$, which leads to $(\emptyset \upharpoonright \text{chan}(\phi_1), \gamma) \models $ $\phi_1$, and consequently, by projection lemma (a), $(\emptyset, \gamma) \models $ $\phi_1$. This proves $\models FP_1 || FP_2$ sat $\phi_1 \land \phi_2$.

D.4 Soundness of the hiding rule

Assume $\models FP$ sat $\phi$ and chan($\emptyset$) $\cap$ cset $= \emptyset$ (1), (2). We show $FP \setminus \text{cset}$ sat $\phi$. Consider any $\gamma$. Let $\emptyset \in \mathcal{H}[FP \setminus \text{cset}]$. Then there exists a $\emptyset \in \mathcal{H}[FP]$ with $\emptyset = \emptyset_1 \setminus \text{cset}$. By (1), $(\emptyset_1, \gamma) \models $ $\phi$. Since, by (2), chan($\emptyset$) $\subseteq$ chan($FP \setminus \text{cset}$), projection lemma (a) leads to $(\emptyset_1 \setminus (\text{CHAN} - \text{cset}), \gamma) \models $ $\phi$, and consequently, by definition, $(\emptyset_1 \setminus \text{cset}, \gamma) \models $ $\phi$. Hence, $(\emptyset, \gamma) \models $ $\phi$. 

25
D.5 Soundness of the fault hypothesis introduction rule

Assume \( \vdash FP \) sat \( \phi \) (1). Consider any \( \gamma \). Let \( \theta \in H[\{FP \mid \chi\}] \). Then there exists a \( \theta_0 \in H[\{FP \}] \) such that, for all \( \gamma \), \( (\theta_0, \theta, \gamma) \models \chi \). By (1), for any \( \theta_0 \), \( (\theta_0, \theta, \gamma) \models \phi \), 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 \( \phi \), \( (\theta_0, \theta_0, \widehat{\gamma}) \models \phi \). Observe that \( T[\theta][\theta_0, \theta, \widehat{\gamma}] = \theta_0 \), thus \( (\theta_0, \theta_0, \widehat{\gamma}) \models \phi \). 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 \( (T[\theta][\theta_0, \theta, \widehat{\gamma}], \theta, \widehat{\gamma}) \models \chi \). 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 \( (\theta, (\gamma : t \mapsto \theta_0)) \models \phi[t/h] \land \chi[t/h_{old}] \), from which we may conclude that \( (\theta, \gamma) \models \exists t : \phi[t/h] \land \chi[t/h_{old}] \).

D.6 Soundness of the prefixing rule

Assume \( \vdash FP \) sat \( \phi \) (1). Consider any \( \gamma \). Let \( \theta \in H[\{FP \}] \). By (1), \( (\theta, \gamma) \models \phi \). For all \( \theta' \leq \theta \) we have, by the prefix closedness lemma, that \( \theta' \in H[\{FP \}] \), and thus, by (1), \( (\theta', \gamma) \models \phi \). Let \( t \) be a fresh logical variable. Then, as \( t \) does not occur in \( \phi \), for all \( \theta' \leq \theta \), \( (\theta', (\gamma : t \mapsto \theta')) \models \phi \). Equivalently, \( (T[\theta][\theta_0, \theta', (\gamma : t \mapsto \theta')) \models \phi \). By substitution lemma (a) \( (\theta', (\gamma : t \mapsto \theta')) \models \phi[t/h] \), for all \( \theta' \leq \theta \), and thus, as \( h \) obviously does not occur in \( \phi[t/h] \), for all \( \theta' \leq \theta \), \( (\theta, (\gamma : t \mapsto \theta')) \models \phi[t/h] \), and consequently, for all \( \theta' \), \( (\theta, (\gamma : t \mapsto \theta')) \models t \leq h \rightarrow \phi[t/h] \). Hence, \( (\theta, \gamma) \models \forall t : t \leq h \rightarrow \phi[t/h] \), i.e. \( (\theta, \gamma) \models \exists t : \phi[t/h] \land \chi[t/h_{old}] \).

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

(i) By (3) and soundness, we obtain \( \vdash FP_1 \parallel FP_2 \) sat \( \phi_1 \land \phi_2 \).

(ii) Let \( chan(\theta) \subseteq chan(FP_1) \parallel FP_2 \) (4) and assume \( (\theta, \gamma) \models \phi_1 \land \phi_2 \). Then, by (1) and projection lemma (a), \( \theta \models chan(FP_1), \gamma \models \phi_1 \). Consequently, by the preciseness of \( \phi_1 \) for \( FP_1 \), we conclude \( \theta \models chan(FP_1) \subseteq chan(FP_1) \parallel FP_2 \) (5). Similarly, \( \theta \models chan(FP_2) \subseteq chan(FP_2) \parallel FP_2 \) (6). Finally, by (4), \( \theta \models chan(FP_1) \parallel FP_2 \) \( = \theta(7) \). Then, by (5), (6), and (7), we conclude that \( \theta \models chan(FP_1) \parallel FP_2 \).

(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 \( chan(\phi_1 \land \phi_2) \subseteq chan(FP_1) \parallel FP_2 \).

(b) Assume \( \vdash FP \) sat \( \phi \) (1) with \( \phi \) precise for \( FP \). Define

\[
\hat{\phi} \equiv \exists t : \phi[t/h] \land h \upharpoonright (chan(FP) - cset) = t \upharpoonright (chan(FP) - cset)
\]

We show that \( \vdash FP \setminus cset \) sat \( \hat{\phi} \), and, furthermore, that \( \hat{\phi} \) is precise for \( FP \setminus cset \).

Lemma 10 \( \models \phi \rightarrow \hat{\phi} \)

Proof: Assume \( (\theta, \gamma) \models \phi \). Let, for fresh \( t \), \( \widehat{\gamma} = (\gamma : t \mapsto \theta) \). Then, \( (\theta, \widehat{\gamma}) \models \phi \), and, trivially, \( (\theta, \widehat{\gamma}) \models \phi[t/h] \land h \upharpoonright (chan(FP) - cset) = t \upharpoonright (chan(FP) - cset) \). Hence, \( (\theta, \gamma) \models \exists t : \phi[t/h] \land h \upharpoonright (chan(FP) - cset) = t \upharpoonright (chan(FP) - cset) \).
By Lemma 10 and the relative completeness assumption, we obtain \( \models \phi \rightarrow \widehat{\phi} \). By (1) and the consequence rule, \( \models FP \models \widehat{\phi} \). Note that, by definition, \( \text{chan}(\exists t : \phi[t/h]) = \emptyset \), thus \( \text{chan}(\widehat{\phi}) = \text{chan}(FP) - \text{cset} \), and hence \( \text{chan}(\widehat{\phi}) \cap \text{cset} = \emptyset \). Then the hiding rule leads to \( \models FP \models \text{cset} \models \widehat{\phi} \) (2). It remains to be shown that \( \widehat{\phi} \) is precise for \( FP \models \text{cset} \).

(i) By (2) and soundness, we have \( \models FP \models \text{cset} \models \widehat{\phi} \).

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

\[
(\theta, (\gamma : t \mapsto \widehat{\theta})) \models \phi[t/h] \land h \upharpoonright (\text{chan}(FP) - \text{cset}) = t \upharpoonright (\text{chan}(FP) - \text{cset})
\]  

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 \( (\widehat{\theta} \upharpoonright \text{chan}(\phi), \gamma) \models \phi \). Since, by the preciseness of \( \phi \) for \( FP \), \( \text{chan}(\phi) \subseteq \text{chan}(FP) \), we obtain \( (\widehat{\theta} \upharpoonright \text{chan}(FP), \gamma) \models \phi \). Obviously, \( \text{chan}(\widehat{\theta} \upharpoonright \text{chan}(FP)) \subseteq \text{chan}(FP) \), so, by the preciseness of \( \phi \) for \( FP \), we have that \( \widehat{\theta} \upharpoonright \text{chan}(FP) \in \mathcal{H}[FP] \). Since, by (3), \( \text{chan}(\theta) \subseteq \text{chan}(FP) - \text{cset} \) and, by (4), \( \theta \upharpoonright (\text{chan}(FP) - \text{cset}) = \widehat{\theta} \upharpoonright (\text{chan}(FP) - \text{cset}) \), we obtain \( \theta = \widehat{\theta} \upharpoonright \text{chan}(FP \models \text{cset}) \), and hence \( \theta \in \mathcal{H}[FP \models \text{cset}] \).

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

(c) Assume \( \models FP \models \phi \) (1) with \( \phi \) precise for \( FP \). Define \( \widehat{\phi} \equiv (\phi \chi) \), that is

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

Then, by (Fault Hypothesis Introduction), \( \models (FP \chi) \models \widehat{\phi} \) (2). We show that \( \widehat{\phi} \) is precise for \( FP \chi \).

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

(ii) Let \( \text{chan}(\theta) \subseteq \text{chan}(FP \chi) \) (3) and assume, for some \( \gamma, (\theta, \gamma) \models \widehat{\phi} \). Consequently, there exists a \( \theta \) such that \( (\theta, (\gamma : t \mapsto \widehat{\theta})) \models \phi[t/h] \land \chi[t/h_{old}] \) (4). Then, by substitution lemma (a), \( (\widehat{\theta}, (\gamma : t \mapsto \widehat{\theta})) \models \phi \), and thus, since \( t \) does not occur free in \( \phi \), \( (\widehat{\theta}, \gamma) \models \phi \). Since, by definition, \( \phi \) is precise for \( FP \), \( \text{chan}(\phi) \subseteq \text{chan}(FP) \), we obtain, by projection lemma (a), \( (\theta \upharpoonright \text{chan}(FP), \gamma) \models \phi \). Trivially, \( \text{chan}(\theta \upharpoonright \text{chan}(FP)) \subseteq \text{chan}(FP) \), and hence, because of the preciseness of \( \phi \) for \( FP \), \( \theta \upharpoonright \text{chan}(FP) \in \mathcal{H}[FP] \) (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 \( \chi \), \( (\widehat{\theta}, \theta, \gamma) \models \chi \). Since \( \text{chan}(\chi) \subseteq \text{chan}(FP) \), projection lemma (b) leads to \( (\widehat{\theta} \upharpoonright \text{chan}(FP), \theta, \gamma) \models \chi \) (6).

Finally, by definition, (3) leads to \( \text{chan}(\theta) \subseteq \text{chan}(FP) \) (7). Consequently, by (5), (6), and (7), \( \theta \in \mathcal{H}[FP \chi] \).

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


91/02 R.P. Nederpelt Implication. A survey of the different logical analyses "if..., then...", p. 26.
H.C.M. de Swart

91/03 J.P. Katoen Parallel Programs for the Recognition of P-invariant Segments, p. 16.
L.A.M. Schoenmakers

A.F. v.d. Stappen

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.


91/10 R.C. Backhouse POLYNOMIAL RELATORS, p. 52.
P.J. de Bruin
P. Hoogendijk
G. Malcolm
E. Voermans
J. v.d. Woude

91/11 R.C. Backhouse Relational Catamorphism, p. 31.
P.J. de Bruin
G. Malcolm
E. Voermans
J. van der Woude

91/12 E. van der Sluis A parallel local search algorithm for the travelling salesman problem, p. 12.


91/14 P. Lemmens The PDB Hypermedia Package. Why and how it was built, p. 63.

K.M. van Hee

91/16 A.J.J.M. Marcelis An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.

P.M.E. de Bra
K.M. van Hee
<table>
<thead>
<tr>
<th>ISBN</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>91/18</td>
<td>Rik van Geldrop</td>
<td>Transformational Query Solving, p. 35.</td>
</tr>
<tr>
<td>91/19</td>
<td>Erik Poll</td>
<td>Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.</td>
</tr>
<tr>
<td>91/23</td>
<td>K.M. van Hee L.J. Somers M. Voorhoeve</td>
<td>Z and high level Petri nets, p. 16.</td>
</tr>
<tr>
<td>91/24</td>
<td>A.T.M. Aerts D. de Reus</td>
<td>Formal semantics for BRM with examples, p. 25.</td>
</tr>
<tr>
<td>91/25</td>
<td>P. Zhou J. Hooman R. Kuiper</td>
<td>A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and completeness, p. 52.</td>
</tr>
<tr>
<td>91/27</td>
<td>F. de Boer C. Palamidessi</td>
<td>Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.</td>
</tr>
<tr>
<td>91/28</td>
<td>F. de Boer</td>
<td>A compositional proof system for dynamic processes creation, p. 24.</td>
</tr>
<tr>
<td>91/30</td>
<td>J.C.M. Baeten F.W. Vaandrager</td>
<td>An Algebra for Process Creation, p. 29.</td>
</tr>
<tr>
<td>91/31</td>
<td>H. ten Eikelder</td>
<td>Some algorithms to decide the equivalence of recursive types, p. 26.</td>
</tr>
<tr>
<td>91/33</td>
<td>W. v.d. Aalst</td>
<td>The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.</td>
</tr>
<tr>
<td>91/34</td>
<td>J. Coenen</td>
<td>Specifying fault tolerant programs in deontic logic, p. 15.</td>
</tr>
</tbody>
</table>
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

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

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
Axiomatizing Probabilistic Processes:
ACP with Generative Probabilities, p. 36.
S.A.Smolka

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.
<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>92/22</td>
<td>R. Nederpelt, F. Kamareddine</td>
<td>A useful lambda notation</td>
<td>17.</td>
</tr>
<tr>
<td>92/23</td>
<td>F. Kamareddine, E. Klein</td>
<td>Nominalization, Predication and Type Containment</td>
<td>40.</td>
</tr>
<tr>
<td>92/24</td>
<td>M. Codish, D. Dams, Eyal Yardeni</td>
<td>Bottom-up Abstract Interpretation of Logic Programs</td>
<td>33.</td>
</tr>
<tr>
<td>92/25</td>
<td>E. Poll</td>
<td>A Programming Logic for ( \mathcal{F} \omega )</td>
<td>15.</td>
</tr>
<tr>
<td>93/01</td>
<td>R. van Geldrop</td>
<td>Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods</td>
<td>36.</td>
</tr>
<tr>
<td>93/02</td>
<td>T. Verheoef</td>
<td>A continuous version of the Prisoner's Dilemma</td>
<td>17.</td>
</tr>
<tr>
<td>93/03</td>
<td>T. Verheoef</td>
<td>Quicksort for linked lists</td>
<td>8.</td>
</tr>
<tr>
<td>93/04</td>
<td>E. H. L. Aarts, J. H. M. Korst, P. J. Zwietering</td>
<td>Deterministic and randomized local search</td>
<td>78.</td>
</tr>
<tr>
<td>93/05</td>
<td>J. C. M. Baeten, C. Verhoef</td>
<td>A congruence theorem for structured operational semantics with predicates</td>
<td>18.</td>
</tr>
<tr>
<td>93/06</td>
<td>J. P. Veltkamp</td>
<td>On the unavoidability of metastable behaviour</td>
<td>29.</td>
</tr>
<tr>
<td>93/07</td>
<td>P. D. Moerland</td>
<td>Exercises in Multiprogramming</td>
<td>97.</td>
</tr>
<tr>
<td>93/08</td>
<td>J. Verhoosel</td>
<td>A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS</td>
<td>32.</td>
</tr>
<tr>
<td>93/10</td>
<td>K. M. van Hee</td>
<td>Systems Engineering: a Formal Approach Part II: Frameworks</td>
<td>44.</td>
</tr>
<tr>
<td>Year</td>
<td>Author(s)</td>
<td>Title</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>92/22</td>
<td>R. Nederpelt F.Kamareddine</td>
<td>A useful lambda notation, p. 17.</td>
<td></td>
</tr>
<tr>
<td>92/24</td>
<td>M.Codish D.Dams Eyal Yardeni</td>
<td>Bottom-up Abstract Interpretation of Logic Programs, p. 33.</td>
<td></td>
</tr>
<tr>
<td>92/25</td>
<td>E.Poll</td>
<td>A Programming Logic for Fσ, p. 15.</td>
<td></td>
</tr>
<tr>
<td>93/01</td>
<td>R. van Geldrop</td>
<td>Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.</td>
<td></td>
</tr>
<tr>
<td>93/02</td>
<td>T. Verhoeff</td>
<td>A continuous version of the Prisoner's Dilemma, p. 17</td>
<td></td>
</tr>
<tr>
<td>93/03</td>
<td>T. Verhoeff</td>
<td>Quicksort for linked lists, p. 8.</td>
<td></td>
</tr>
<tr>
<td>93/04</td>
<td>E.H.L. Aarts J.H.M. Korst P.J. Zwietering</td>
<td>Deterministic and randomized local search, p. 78.</td>
<td></td>
</tr>
<tr>
<td>93/05</td>
<td>J.C.M. Baeten C. Verhoeff</td>
<td>A congruence theorem for structured operational semantics with predicates, p. 18.</td>
<td></td>
</tr>
<tr>
<td>93/06</td>
<td>J.P. Veltkamp</td>
<td>On the unavoidability of metastable behaviour, p. 29</td>
<td></td>
</tr>
<tr>
<td>93/07</td>
<td>P.D. Moerland</td>
<td>Exercises in Multiprogramming, p. 97</td>
<td></td>
</tr>
<tr>
<td>93/08</td>
<td>J. Verhoosel</td>
<td>A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.</td>
<td></td>
</tr>
<tr>
<td>93/10</td>
<td>K.M. van Hee</td>
<td>Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.</td>
<td></td>
</tr>
</tbody>
</table>