A compositional proof theory for fault tolerant real-time distributed systems


_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](http://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.
A Compositional Proof Theory for Fault Tolerant Real-Time Distributed Systems

Henk Schepers† Rob Gerth§

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

Abstract

In this report we present a compositional network proof theory to specify and verify fault tolerant real-time distributed systems. Important in such systems is the failure hypothesis that stipulates the class of failures that must be tolerated. In the formalism presented in this report, the failure hypothesis of a system is represented by a predicate which expresses how faults might transform the observable input and output behaviour of the system. A proof of correctness of a triple modular redundant system is given to illustrate our approach.

Key words: Compositional proof theory, distributed system, failure hypothesis, fault tolerance, real-time system, relative network completeness, 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. Yet, as distributed systems are employed in increasingly critical areas, e.g. to control aircraft and to monitor hospital patients, the inherently closely related fault tolerance and real-time requirements become stronger and stronger. In the Hoare style formalism of [6] 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 to deal with the occurrence of faults during execution (cf. [4, 10, 11, 15, 16, 23]) — of which only the approaches of [15] and to a smaller degree [4] provide support for reasoning about real-time issues — the occurrence of a fault is modeled explicitly as an observable action. In contrast, we suggest a more abstract approach where the effects of faults on the externally visible input and output behaviour are modeled while the syntactic interfaces of the processes remain unchanged. In particular, we propose 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 our approach is compositional, that is, it allows for the reasoning with the specifications of processes without considering their implementation and the precise nature and occurrence of faults in such an implementation.

In fault tolerant systems, three domains of behaviour are distinguished: normal, exceptional and catastrophic (see [14]). Normal behaviour is the behaviour when no faults occur. The discriminating factor between exceptional and catastrophic behaviour is the failure hypothesis which expresses how

†Supported by the Dutch STW under grant number NWI88.1517: ‘Fault Tolerance: Paradigms, Models, Logics, Construction’. E-mail: schepers@win.tue.nl.
§Currently working in ESPRIT project P6021: ‘Building Correct Reactive Systems (REACT)’. E-mail: robg@win.tue.nl.
faults affect the normal behaviour. Relative to the failure 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. [2, 14, 17]). Under a particular failure hypothesis for each of its components, a system is designed to tolerate (only) those anticipated component failures (see e.g. [19] for some design examples). In particular, the exceptional behaviour together with the normal behaviour constitutes the acceptable behaviour.

In [22] Schepers and Hooman developed a trace-based compositional proof theory for safety properties of fault tolerant distributed systems. In this theory, the failure hypothesis of a process is formalized as a relation between the normal and acceptable behaviour of that process providing a modular treatment of faults. Indeed, such a relation enables us to abstract from the precise nature of a fault and to focus on the abnormal behaviour it causes. Here, we extend this proof theory to reason about liveness, fairness, and real-time issues. To do so, we replace the underlying finite trace model by a model in which the timed, infinite traces of a process are decorated with timed refusal sets. The extended model enables deadlock to be taken into account. To exclude unrealistic behaviour, it incorporates finite variability [3], or non-Zenoness (cf. [1]), by guaranteeing that each action has a fixed minimal duration. However, the introduction of time causes the importance of liveness and fairness to decrease, since many interesting properties become safety properties [13].

The remainder of this report is organized as follows. Section 2 introduces the programming language. Section 3 introduces the model of computation and the denotational semantics. In Section 4 we present the assertion language and associated correctness formulae. In Section 5 we incorporate failure hypotheses into our formalism. Section 6 presents a compositional network proof theory for fault tolerant real-time distributed systems. We illustrate our method by applying it, in Section 7, to a triple modular redundant system. In Section 8 we show that the proof system of Section 6 is sound and relative network complete. A conclusion appears in Section 9. An extended abstract of this report will appear in [21].

2 Programming language

In this section we present an occam-like programming language [9] which is used to define networks of processes that communicate synchronously via directed channels. Let VAR be a nonempty set of program variables, CHAN a nonempty set of channel names, and VAL a denumerable domain of values. \( \mathbb{N} \) denotes the set of natural numbers (including 0), \( \mathbb{Q} \) the rationals, and \( \mathbb{R} \) the reals. Let \( \text{TIME} \) be some ordered time domain (\( \infty \in \text{TIME} \)). For the scope of this report it is immaterial whether time domain \( \text{TIME} \) is discrete, that is, \( \text{TIME} = \{ ur | r \in \mathbb{N} \} \) for some positive smallest time unit \( u \), dense, that is, \( \text{TIME} = \{ r \in \mathbb{Q} | r \geq 0 \} \), or continuous, that is, \( \text{TIME} = \{ r \in \mathbb{R} | r \geq 0 \} \). The syntax of our programming language is given in Table 1, with \( n \in \mathbb{N}, n \geq 1, x \in \text{VAR}, \mu \in \text{VAL}, c \in \text{CHAN}, d \in \text{TIME}, \text{and cset} \subseteq \text{CHAN} \).

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e )</td>
<td>( \mu</td>
</tr>
<tr>
<td>Boolean Expression</td>
<td>( b )</td>
</tr>
<tr>
<td>Guarded Command</td>
<td>( G )</td>
</tr>
<tr>
<td>Process</td>
<td>( P )</td>
</tr>
</tbody>
</table>

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

Atomic statements
- \( \text{skip} \) terminates after \( K_{\text{skip}} \) units of time, where constant \( K_{\text{skip}} > 0 \).
- 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 communication is synchronous, 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.

• Boolean guarded command `[ \[ i=1^n b_i \rightarrow P_i \] ]`. If none of the `b_i` evaluate to true then this 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`.

• Communication guarded command `[\[ i=1^n c_i?x_i \rightarrow P_i \] \[ delay d \rightarrow P \] ]`. Wait at most `d` time units for some input `c_i?x_i` to become enabled. As soon as one of the `c_i` communications is possible (before `d` time units have elapsed), it is performed and thereafter the corresponding `P_i` is executed. If two or more inputs become enabled at the same time, then one of these is non-deterministically chosen. If none of the inputs becomes enabled within `d` time units after the start of the execution of the communication guarded command, then `P` is executed.

• Iteration `*G` indicates repeated execution of guarded command `G` as long as at least one of the guards is open. When none of the guards is open `*G` terminates.

• `P_1 || P_2` indicates the parallel execution of the processes `P_1` and `P_2`.

• `P \setminus \text{cset}` hides the channels from `\text{cset}`.

Definition 1 (Variables occurring in a process) The set `var(P)` of variables occurring in process `P` is inductively defined as follows:

• `var(\mu) = \emptyset`
• `var(x) = \{x\}`
• `var(e_1 + e_2) = var(e_1) \cup var(e_2)`
• `var(e_1 \times e_2) = var(e_1) \cap var(e_2)`
• `var(e_1 < e_2) = \varnothing`
• `var((\neg b)) = var(b)`
• `var(b_1 \lor b_2) = var(b_1) \cup var(b_2)`
• `var(skip) = \emptyset`
• `var(x := e) = \{x\} \cup var(e)`
• `var(c!e) = var(e)`
• `var(c?x) = \{x\}`
• `var(P_1 ; P_2) = var(P_1) \cup var(P_2)`
• `var([ i=1^n (\neg b_i) \rightarrow P_i ]) = \bigcup_{i=1}^n var(b_i) \cup \bigcup_{i=1}^n var(P_i)`
• `var([ i=1^n c_i?x_i \rightarrow P_i \[ delay d \rightarrow P_0 \] ]) = \bigcup_{i=1}^n \{x_i\} \cup \bigcup_{i=1}^n var(P_i)`
• `var(*G) = var(G)`
• \( \text{var}(P_1 \parallel P_2) = \text{var}(P_1) \cup \text{var}(P_2) \)
• \( \text{var}(P \setminus \text{cset}) = \text{var}(P) \)

Definition 2 (Observable input channels of a process) The set of visible, or observable, input channels of process \( P \), notation \( \text{in}(P) \), is obtained as follows by structural induction:

- \( \text{in}({\text{skip}}) = \text{in}(x := e) = \text{in}(c!e) = \emptyset \)
- \( \text{in}(c?x) = \{c\} \)
- \( \text{in}(P_1 ; P_2) = \text{in}(P_1) \cup \text{in}(P_2) \)
- \( \text{in}(\prod_{i=1}^n b_i \rightarrow P_1) = \bigcup_{i=1}^n \text{in}(P_i) \)
- \( \text{in}(\prod_{i=1}^n c_i?x_i \rightarrow P_1 \parallel \text{delay } d \rightarrow P_0) = \bigcup_{i=1}^n \{c_i\} \cup \bigcup_{i=0}^n \text{in}(P_i) \)
- \( \text{in}(\star G) = \text{in}(G) \)
- \( \text{in}(P_1 \parallel P_2) = \text{in}(P_1) \cup \text{in}(P_2) \)
- \( \text{in}(P \setminus \text{cset}) = \text{in}(P) - \text{cset} \)

Definition 3 (Observable output channels of a process) The set of observable output channels of process \( P \), notation \( \text{out}(P) \), is defined inductively as follows:

- \( \text{out}({\text{skip}}) = \text{out}(x := e) = \text{out}(c!e) = \emptyset \)
- \( \text{out}(c?x) = \emptyset \)
- \( \text{out}(P_1 ; P_2) = \text{out}(P_1) \cup \text{out}(P_2) \)
- \( \text{out}(\prod_{i=1}^n b_i \rightarrow P_1) = \bigcup_{i=1}^n \text{out}(P_i) \)
- \( \text{out}(\prod_{i=1}^n c_i?x_i \rightarrow P_1 \parallel \text{delay } d \rightarrow P_0) = \bigcup_{i=1}^n \text{out}(P_i) \)
- \( \text{out}(\star G) = \text{out}(G) \)
- \( \text{out}(P_1 \parallel P_2) = \text{out}(P_1) \cup \text{out}(P_2) \)
- \( \text{out}(P \setminus \text{cset}) = \text{out}(P) - \text{cset} \)

Definition 4 (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 \( n \in \mathbb{N}, d \in \text{TIME}, c_1, \ldots, c_n \in \text{CHAN}, \) and \( x_1, \ldots, x_n \in \text{VAR} \)):

- For \( P_1 ; P_2 \) we require that \( \text{in}(P_1) \cap \text{out}(P_2) = \emptyset \) and \( \text{out}(P_1) \cap \text{in}(P_2) = \emptyset \).
- For \( \prod_{i=1}^n b_i \rightarrow P_1 \) we require that, for all \( i, j \in \{1, \ldots, n\}, i \neq j, \text{out}(P_i) \cap \text{in}(P_j) = \emptyset \).
- For \( \prod_{i=1}^n c_i?x_i \rightarrow P_1 \parallel \text{delay } d \rightarrow P_0 \) we require that
  - \( \bigcup_{i=1}^n \{c_i\} \cap \bigcup_{i=0}^n \text{out}(P_i) = \emptyset \), and
  - for all \( i, j \in \{0, \ldots, n\}, i \neq j, \text{out}(P_i) \cap \text{in}(P_j) = \emptyset \).
• For $P_1 || P_2$ we require that $\text{in}(P_1) \cap \text{in}(P_2) = \emptyset$ 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 that only internal channels are hidden:

• 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 $P_1 || P_2$ we require that $\text{var}(P_1) \cap \text{var}(P_2) = \emptyset$.

### 2.2 Basic timing assumptions

To determine the timed behaviour of programs we have to make assumptions about the time needed to execute atomic statements and how the execution time of compound constructs can be obtained from the timing of the components. In our proof system the correctness of a program with respect to a specification, which may include timing constraints, is verified relative to these assumptions.

In this report we assume that the execution time of atomic statements, except for communication statements, is given by fixed constants. By assumption, communication takes no time. The execution time of a (synchronous) communication statement consists of, besides an assumed fixed constant overhead before and after the actual communication, the time spent waiting for a partner.

In this report we assume maximal parallelism, that is, we assume that each process has its own processor. Hence, a process executes a local, non-communication, command immediately. Since communication is synchronous, a process is forced to wait until a communication partner is available. In case of maximal parallelism the communication occurs as soon as such a partner indeed comes forward: it is never the case that one process waits to perform $c!e$ while another process waits to execute $c?x$. Thus, maximal parallelism implies minimal waiting.

For simplicity, we assume that there is no overhead for compound statements and that execution of a delay $d$ statement takes exactly $d$ time units. Besides constant $K_{\text{skip}}$, we assume given a constant $K_{\mu}$ such that execution of each assignment statement takes $K_{\mu}$ time units, a constant $K_{\nu}$ denoting the overhead preceding a communication, a constant $K_{\omega}$ denoting the overhead following a communication, and a constant $K_{\xi}$ capturing the time required to evaluate the guards of a boolean guarded command and non-deterministically select one of the open guards.

### 3 Model of Computation and Denotational Semantics

The events in the various processes of a distributed system are related to each other by means of a conceptual global clock (as is done in [12, 18]). This global notion of time is introduced at a metalevel of reasoning and is not incorporated in the distributed system itself. We use a special symbol $T$ ($T \notin \text{VAR}$) to denote the global time.

**Definition 5 (States)** Define the set $\text{STATE}$ of states as the set of mappings $\sigma$ which map a variable $x \in \text{VAR}$ to a value $\sigma(x) \in \text{VAL}$ and which map $T$ to an instant $\sigma(T) \in \text{TIME}$.

Thus, besides assigning to each program variable $x$ a value $\sigma(x)$, a state $\sigma$ records the global time. For simplicity we do not make a distinction between the semantic and the syntactic domain of values and instants. In the sequel we assume that we have the standard arithmetical operators $+,-,$ and $\times$ on $\text{TIME}$ and $\text{VAL}$.

Define the value of an expression $e$ in a state $\sigma$, denoted by $E[e](\sigma)$, inductively as follows:

- $E[\mu](\sigma) = \mu$,
- $E[x](\sigma) = \sigma(x)$,
\[ E[e_1 + e_2](\sigma) = E[e_1](\sigma) + E[e_2](\sigma), \]
\[ E[e_1 - e_2](\sigma) = E[e_1](\sigma) - E[e_2](\sigma), \]
\[ E[e_1 \times e_2](\sigma) = E[e_1](\sigma) \times E[e_2](\sigma). \]

We define when a boolean expression \( b \) holds in a state \( \sigma \), denoted by \( B[b](\sigma) \), as

\[ B[e_1 = e_2](\sigma) \text{ iff } E[e_1](\sigma) = E[e_2](\sigma), \]
\[ B[e_1 < e_2](\sigma) \text{ iff } E[e_1](\sigma) < E[e_2](\sigma), \]
\[ B[\neg b](\sigma) \text{ iff } \neg B[b](\sigma), \]
\[ B[b \lor b'](\sigma) \text{ iff } B[b](\sigma) \lor B[b'](\sigma). \]

We represent a synchronous communication of value \( \mu \in Val \) on channel \( c \in Chan \) at time \( \tau \in Time \) by a triple \((\tau, c, \mu)\), and define

\[ (Timestamp) \quad ts((\tau, c, \mu)) = \tau \]
\[ (Channel) \quad ch((\tau, c, \mu)) = c \]
\[ (Value) \quad val((\tau, c, \mu)) = \mu \]

To denote the observable input and output behaviour of a process \( P \) we use a timed trace \( \theta \) which is a possibly infinite sequence of the form \(((\tau_1, c_1, \mu_1), (\tau_2, c_2, \mu_2), \ldots)\), where \( \tau_i \geq \tau_{i-1}, c_i \in chan(P), \) and \( \mu_i \in Val, \) for \( i \geq 1; \) for all \( i \) and \( j \) such that \( \tau_i = \tau_j \) we require \( c_i \neq c_j \). Such a history denotes the communications performed by \( P \) during an execution, and the times at which they occurred.

**Definition 6 (Timed traces)** Let, for \( OBS = Time \times Chan \times Val \), \( TRACE \) be the set of timed traces, that is,

\[
TRACE = \{ \theta \in OBS^* \cup OBS^\omega \mid \forall i \cdot ts(\theta(i)) \leq ts(\theta(i+1)) \land \forall j \cdot ts(\theta(i)) = ts(\theta(j)) \rightarrow ch(\theta(i)) \neq ch(\theta(j)) \}
\]

Let \( () \) denote the empty trace, i.e. the sequence of length 0. The concatenation of two traces \( \theta_1 \) and \( \theta_2 \) is denoted \( \theta_1 \theta_2 \) (and equals \( \theta_1 \) if \( \theta_1 \) is infinite). We use first(\( \theta \)) and, if \( \theta \) is finite, last(\( \theta \)) to refer to the first and last record of \( \theta \), respectively.

However, a model based on merely timed traces is too abstract to define a compositional semantics, as has been argued in [18] and [8]. The model proposed there is the timed failures model; a confusing name for researchers in the fault tolerant systems community. The 'failure' refers to the fact that in this model one not only records the communications that take place but also the failed or refused attempts due to the absence of a communication partner. Henceforth, we will refer to this notion as timed observation.

A timed observation is a timed (trace, refusal) pair. A timed refusal is a set of (channel, instant) pairs.

**Definition 7 (Timed refusals)** Let \( REF \) be the set of timed refusal sets, that is,

\[
REF = \{ \mathcal{R} \mid \mathcal{R} \subseteq Chan \times [0, \infty) \}
\]

\[ \diamond \]
We usually define a timed refusal by a cartesian product $cset \times \text{INT}$, where $cset \subseteq \text{CHAN}$ is a set of channels and $\text{INT}$ an interval from $\text{TIME}$.

Let $\text{STATE}_\perp = \text{STATE} \cup \{\bot\}$. The semantic function $\mathcal{M}$ assigns to a process $P$ a set of triples $(\sigma_0, (\theta, \mathcal{R}), \sigma)$ with $\sigma_0 \in \text{STATE}$, $\theta \in \text{TRACE}$, $\mathcal{R} \in \text{REF}$, and $\sigma \in \text{STATE}_\perp$. A triple $(\sigma_0, (\theta, \mathcal{R}), \sigma) \in \mathcal{M}[P]$ denotes a maximal observation of process $P$ with the following informal meaning:

- if $\sigma \neq \bot$ then it represents a terminating computation which starts in state $\sigma_0$, performs the communications as described in $\theta$ while refusing those in $\mathcal{R}$, and terminates in state $\sigma$, and
- if $\sigma = \bot$ then it represents a computation which starts in state $\sigma_0$, performs the communications as described in $\theta$ while refusing those in $\mathcal{R}$, but never terminates. A computation does not terminate either because it is infinite or the process deadlocks.

**Definition 8 (Projection on traces)** For a trace $\theta \in \text{TRACE}$ and a set of channels $cset \subseteq \text{CHAN}$, we define the projection of $\theta$ onto $cset$, denoted by $\theta \upharpoonright cset$, as the sequence obtained from $\theta$ by deleting all records with channels not in $cset$. Formally,

$$
\theta \upharpoonright cset = \begin{cases} 
\{\} & \text{if } \theta = \{\} \\
\theta \upharpoonright cset & \text{if } \theta = (t, c, v)^n \theta_0 \text{ and } c \notin cset \\
(t, c, v)^n (\theta_0 \upharpoonright cset) & \text{if } \theta = (t, c, v)^n \theta_0 \text{ and } c \in cset
\end{cases}
$$

**Definition 9 (Hiding on traces)** Hiding is the complement of projection. Formally, the hiding of a set $cset$ of channels from a trace $\theta \in \text{TRACE}$, notation $\theta \setminus cset$, is defined as

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

**Definition 10 (Time shift on traces)** For timed trace $\theta$ such that $ts(first(\theta)) \geq \tau$ we define the time shift operation $\cdot$ as follows:

$$
\theta \cdot \tau = \begin{cases} 
\{\} & \text{iff } \theta = \{\} \\
(\tau - t, c, v)^n (\theta_0 \cdot \tau) & \text{iff } \theta = (t, c, v)^n \theta_0
\end{cases}
$$

**Definition 11 (Projection on refusals)** For a refusal $\mathcal{R} \in \text{REF}$ and a set of channels $cset \subseteq \text{CHAN}$, we define the projection of $\mathcal{R}$ onto $cset$, denoted by $\mathcal{R} \upharpoonright cset$ as follows:

$$
\mathcal{R} \upharpoonright cset = \mathcal{R} \cap (cset \times [0, \infty))
$$

**Definition 12 (Hiding on refusals)** Hiding is the complement of projection. Formally, the hiding of a set $cset$ of channels from a refusal $\mathcal{R} \in \text{REF}$, notation $\mathcal{R} \setminus cset$, is defined as

$$
\mathcal{R} \setminus cset = \mathcal{R} \cap ((\text{CHAN} - cset) \times [0, \infty))
$$

**Definition 13 (Time shift on refusals)** For $\mathcal{R} \in \text{REF}$ such that for all $(c, t) \in \mathcal{R}$ it is the case that $t \geq \tau$ the time shift operation $\mathcal{R} \cdot \tau$ is defined as follows:

$$
\mathcal{R} \cdot \tau = \{ (c, t - \tau) \mid (c, t) \in \mathcal{R} \}
$$
Definition 14 (Variant of a state) The variant of a state \( \sigma \) with respect to a variable \( x \) and a value \( \vartheta \), denoted \( (\sigma : x \mapsto \vartheta) \), is given by

- if \( \sigma = \bot \) then \( (\sigma : x \mapsto \vartheta) = \bot \)
- if \( \sigma \neq \bot \) then \( (\sigma : x \mapsto \vartheta)(y) = \begin{cases} \vartheta & \text{if } y \equiv x \\ \sigma(y) & \text{if } y \neq x \end{cases} \)

using `\equiv` to denote syntactic equality.

The semantic function \( \mathcal{M} \) is inductively defined as follows. Notice that a terminated process will indefinitely refuse to communicate on its channels.

- Execution of `skip` terminates after \( K_{\text{skip}} \) time units, all the while refusing no communication.
  \[
  \mathcal{M}[\text{skip}] = \{ (\sigma_0, (\emptyset), (\sigma_0 : T \mapsto K_{\text{skip}})), | \mathcal{E}[T](\sigma_0) = 0 \} 
  \]

- Execution of assignment \( x := e \) terminates after \( K_a \) time units, all the while refusing no communication. In the final state \( x \) has the value of \( e \) in the initial state.
  \[
  \mathcal{M}[x := e] = \{ (\sigma_0, (\emptyset), ((\sigma_0 : x \mapsto e), T \mapsto K_a)), | \mathcal{E}[T](\sigma_0) = 0 \} 
  \]

- In the execution of a synchronous io-statement there comes, after an initial period of \( K_a \) time units during which the communication are refused, a waiting period for a communication partner to become available. Execution of output statement \( c!e \) either never terminates (in case no communication partner ever shows up) or terminates \( K_w \) time units after the \( c \) communication has occurred.
  \[
  \mathcal{M}[c!e] = 
  \begin{cases}
    \{ (\sigma_0, (\emptyset, \emptyset), (\sigma_0 : T \mapsto K_a), | \mathcal{E}[T](\sigma_0) = 0 \} 
    & {\text{if } \mathcal{R} = \{ e \} \times [0, K_a) } \\
    \cup \{ (\sigma_0, (\emptyset, \emptyset, \emptyset), (\sigma_0 : T \mapsto r + K_a), | \mathcal{E}[T](\sigma_0) = 0 \} 
    & {\text{if } \mathcal{R} = \{ e \} \times ([0, K_a) \cup (r, \infty)) } 
  \end{cases}
  \]

Recall that we allow at most one \( c \) communication at time \( T = r \).

- Execution of input statement \( c?x \) either never terminates (in case no communication partner ever shows up) or terminates when the \( c \) communication has occurred and the received value is assigned to \( x \).
  \[
  \mathcal{M}[c?x] = 
  \begin{cases}
    \{ (\sigma_0, (\emptyset, \emptyset, \emptyset), (\sigma_0 : T \mapsto K_a), | \mathcal{E}[T](\sigma_0) = 0 \} 
    & {\text{if } \mathcal{R} = \{ e \} \times [0, K_a) } \\
    \cup \{ (\sigma_0, (\emptyset, \emptyset, \emptyset, \emptyset), (\sigma_0 : T \mapsto r + K_a), | \mathcal{E}[T](\sigma_0) = 0 \} 
    & {\text{if } \mathcal{R} = \{ e \} \times ([0, K_a) \cup (r, \infty)) } 
  \end{cases}
  \]

- An execution of \( P_1 ; P_2 \) is either a non-terminating execution of \( P_1 \) or a terminating execution of \( P_1 \) followed by some execution of \( P_2 \). Under the convention that a process can only refuse communications on its own channels we must, in case of sequential and suchlike composition, expand the refusal sets of the respective components to the union of the channels of those components.
\[ \mathcal{M}[P_1 ; P_2] = \\
\{ (\sigma_0 , (\theta, \tau) \cup (\tau , \text{chan}(P_2 ) - \text{chan}(P_1 )) \times [0, \infty) ) , \perp ) | (\sigma_0 , (\theta, \tau)) \in \mathcal{M}[P_1] \} \\
\cup \{ (\sigma_0 , (\theta_1 \wedge \theta_2 , \tau), \pi ) \mid \\
\text{there exist a } \tau_1 , \tau_2 , \text{a } \sigma_1 \neq \perp \text{ and a } \pi > 0 \text{ such that } E[\tau](\sigma_1 ) = \pi , \\
(\sigma_0 , (\theta_1 , \tau_1 ) , \sigma_1 ) \in \mathcal{M}[P_1], \\
(\sigma_1 : T \rightarrow 0 , (\theta_2 , \mathcal{R}_2 ) \wedge \pi , (\sigma : T \rightarrow T - \pi )) \in \mathcal{M}[P_2], \\
\text{and } \mathcal{R} = \mathcal{R}_1 \cup (\text{chan}(P_2 ) - \text{chan}(P_1 )) \times [0, \pi) \cup \mathcal{R}_2 \cup (\text{chan}(P_1 ) - \text{chan}(P_2 )) \times [\pi, \infty) \} \]

where \((\theta, \tau) \wedge t\) equals \((\theta \wedge t, \mathcal{R} \wedge t)\).

- If no guard is open, that is, evaluates to true, the boolean guarded command terminates after evaluating the guards which takes \(K_g\) time units. Otherwise, the process corresponding to one of the open guards (non-deterministically chosen) is executed. While evaluating the guards, communications on \(U_i \text{chan}(P_i )\) are refused.

\[ \mathcal{M}[\bigcup_{i=1}^n \; b_i \rightarrow P_i | \text{ delay } d \rightarrow P] = \\
\{ (\sigma_0 , (\perp, \text{chan}(P_i ) \times [0, \infty) ) , \sigma_0 : T \rightarrow K_g ) | E[\tau](\sigma_0 ) = 0 \wedge \neg B[b_1 \vee \ldots \vee b_n ](\sigma_0 ) \} \\
\cup \{ (\sigma_0 , (\theta, \mathcal{R}), \sigma ) | E[\tau](\sigma_0 ) = 0 , \text{ and there exist a } k \in \{1, \ldots, n\} \text{ and a } \mathcal{R} \text{ such that} \\
B[b_k ](\sigma_0 ) , (\sigma_0 , (\theta, \mathcal{R}) \wedge K_g , (\sigma : T \rightarrow T - K_g )) \in \mathcal{M}[P_k] , \text{ and} \\
\mathcal{R} = U_i \text{chan}(P_i ) \times [0, K_g ] \cup \mathcal{R} \cup (U_i \text{chan}(P_i ) - \text{chan}(P_1 )) \times [K_g , \infty) \} \]

- In case of a communication guarded command the first communication that occurs resolves the choice of which process to execute. If no communication occurs before \(d\) time units \((0 \leq d \leq \infty)\) have elapsed, process \(P\) is executed.

\[ \mathcal{M}[\bigcup_{i=1}^n \; c_i ? x \rightarrow P_i | \text{ delay } d \rightarrow P] = \\
\cup_i \{ (\sigma_0 , (\pi, c_i , v)^\wedge \theta, \mathcal{R}) , \sigma ) \mid \\
E[\tau](\sigma_0 ) = 0 , K_\perp \leq \tau < d , v \in \text{VAL} , \text{ and there exists a } \mathcal{R} \text{ such that} \\
\mathcal{R} = (U_j \text{chan}(P_j ) \cup \text{chan}(P)) - U_j \{c_j \} \times [0, \tau) - \{c_i , \tau\} \\
\cup \mathcal{R} \\
U_i (U_j \text{chan}(P_j ) \cup \text{chan}(P)) - \text{chan}(P_i ) \times [\tau, \infty) , \\
\text{and } (\sigma_0 , (\theta, \mathcal{R}) \wedge (\tau + K_\perp + K_a ) , (\sigma : T \rightarrow T - \tau - K_\perp + K_a )) \in \mathcal{M}[P_k] \} \\
\cup \{ (\sigma_0 , (\theta, \mathcal{R}), \sigma ) \mid \\
E[\tau](\sigma_0 ) = 0 , \text{ and there is a } \mathcal{R} \text{ with } (\sigma_0 , (\theta, \mathcal{R}) \wedge d , (\sigma : T \rightarrow T - d)) \in \mathcal{M}[P] , \text{ and} \\
\mathcal{R} = (U_j \text{chan}(P_j ) \cup \text{chan}(P)) - U_j \{c_j \} \times [0, d] \cup \mathcal{R} \cup (U_j \text{chan}(P_j ) - \text{chan}(P)) \times [d, \infty) \} \]

- An execution of \( \ast G \) consists of either an infinite number of executions of \( G \) that terminate in a state in which at least one of the guards is open, or a finite number of executions of \( G \) such that the last execution does not terminate or terminates in a state in which no guard is open.
\[ \mathcal{M}[G] = \{ (\sigma_0, (\theta, \mathcal{R}), \sigma) \mid \mathcal{E}[T](\sigma_0) = 0 \text{ and there exists a } k \in \mathbb{N} \cup \{\infty\}, \text{ and for every } i, 0 \leq i < k, \text{ there exists a triple } (\sigma_i, (\theta_{i+1}, \mathcal{R}_{i+1}), \sigma_{i+1}) \text{ such that} \]

\[ \begin{align*}
&\sigma_i \neq \bot, \\
&B[\theta_G](\sigma_i), \\
&( (\sigma_i : T \rightarrow 0), \\
&(\theta_{i+1}, \mathcal{R}_{i+1}) \cap \mathcal{E}[T](\sigma_i), \\
&(\sigma_{i+1} : T \rightarrow T - \mathcal{E}[T](\sigma_i)) \in \mathcal{M}[G], \text{ and} \\
&k = \infty \text{ then} \\
&\text{for all } j, 1 \leq j < k, \theta_1 \ldots \theta_j \leq \theta \text{ and } \cap_{j=1}^{i} \mathcal{R}_i \supseteq \mathcal{R}, \text{ and } \sigma = \bot, \\
&\text{else if } k < \infty \text{ then} \\
&\theta' = \theta_1 \ldots \theta_k, \mathcal{R}' = \cap_{i=1}^{k} \mathcal{R}_i, \sigma' = \sigma_k, \text{ and if } \sigma_k \neq \bot \text{ then } B[\theta_G](\sigma_k) \}
\end{align*} \]

- Since communication is synchronous a trace \( \theta \) of process \( P \parallel P \) has the property that \( \theta \upharpoonright \text{chan}(P_1) \) and \( \theta \upharpoonright \text{chan}(P_2) \) match traces of \( P_1 \) and \( P_2 \) respectively. Communications along the channels in \( \text{chan}(P_1) \cap \text{chan}(P_2) \) are refused if they are refused by \( P_1 \) or \( P_2 \). Since process \( P \) does not refuse to communicate on the channels in \( \text{CHAN} - \text{chan}(P) \), it is also the case that communications on the channels in \( \text{CHAN} - \text{chan}(P_1) \cap \text{chan}(P_2) \) are refused if they are refused by \( P_1 \) or \( P_2 \). Process \( P_1 \parallel P_2 \) terminates if and only if both \( P_1 \) and \( P_2 \) terminate.

\[ \mathcal{M}[P_1 \parallel P_2] = \{ (\sigma_0, (\theta, \mathcal{R}), \sigma) \mid \text{for } i = 1, 2 \text{ there exist } (\theta_i, \mathcal{R}_i), \sigma_i \text{ such that} \]

\[ \begin{align*}
&\sigma_0, (\theta_i, \mathcal{R}_i), \sigma_i \in \mathcal{M}[P_i], \\
&\text{and if } \sigma_i = \bot \text{ or } \sigma_i = \bot \text{ then } \sigma = \bot \text{ else, for all } x \in \text{VAR}, \\
&\sigma(x) = \{ \sigma_i(x) \mid x \in \text{var}(P_i) \}, \sigma(T) = \max_i(\sigma_i(T)), \\
&\theta \upharpoonright \text{chan}(P_i) = \theta_i, \theta \upharpoonright \text{chan}(P_1 \parallel P_2) = \theta, \text{ and } \mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2 \} \}
\]

- The observations of \( P \setminus \text{csel} \), where \( \text{csel} \subset \text{in}(P) \cap \text{out}(P) \), are characterized by the fact that the internal \( \text{csel} \) communications take place as soon as they become enabled. This means that such communications occur at the first instant they are no longer refused. Recall that we allow only one communication per channel to occur at a particular instant. Furthermore, by our definition of the semantics it takes a non-zero period before such a taken communication can become enabled again. Hence, an observation of \( P \setminus \text{csel} \) is characterized by the fact that \( \text{csel} \) communications are continuously refused, except on single instants.

**Definition 15 (As soon as possible)** For a timed refusal set \( \mathcal{R} \) and a set \( \text{csel} \) of channels:

\[ \text{ASAP}(\mathcal{R}, \text{csel}) \equiv \forall c \in \text{csel} : \forall t_1, t_2 : c \times [t_1, t_2] \cap \mathcal{R} = \emptyset \rightarrow t_1 = t_2 \]

Then,

\[ \mathcal{M}[P \setminus \text{csel}] = \{ (\sigma_0, (\theta \setminus \text{csel}, \mathcal{R} \setminus \text{csel}), \sigma) \mid (\sigma_0, (\theta, \mathcal{R}), \sigma) \in \mathcal{M}[P] \land \text{ASAP}(\mathcal{R}, \text{csel}) \} \]

Notice that this definition incorporates finite variability, or non-Zenoness.

**Definition 16 (Timed observations)** The **timed observations** of a process \( P \), notation \( \mathcal{O}[P] \), follow from:

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

Then, \( \mathcal{O}[P \setminus \text{csel}] \equiv \mathcal{O}[P] \land \text{ASAP}(\mathcal{R}, \text{csel}) \). The set \( \mathcal{O}[P] \) represents the normal behaviour of process \( P \). In Section 5 we determine the set \( \mathcal{O}[P \setminus \chi] \) representing the acceptable behaviour of \( P \) under the assumption of failure hypothesis \( \chi \). Besides the already mentioned finite variability, other important properties of the semantic function \( \mathcal{O} \) are that if \( (\theta, \mathcal{R}) \in \mathcal{O}[P] \) then \( \theta \upharpoonright \text{chan}(P) = \theta \) and \( \mathcal{R} \upharpoonright \text{chan}(P) = \mathcal{R} \).
4 Assertion Language and Correctness Formulae

Assertions are used to express the relevant program properties in terms of the observable quantities. Since we abstract from the internal state of a process, the primitives of our assertion language are similar to the denotations as used in the semantic function $O$. In this report we specify the relation between a program $P$ and an assertion $\phi$ by means of a so-called correctness formula of the form $P \text{ sat } \phi$. Intuitively, such a formula expresses that all executions of $P$ satisfy $\phi$.

Similar to the semantic denotation of traces in the previous section, we use record expressions such as $(\tau, c, \mu)$, with $\tau \in \text{TIME}$, $c \in \text{CHAN}$, and $\mu \in \text{VAL}$, in assertions. We use instant expressions, e.g. using the function $\text{ts}$ to obtain the timestamp of a record. We have channel expressions, e.g. using the operator $\text{ch}$ which yields the channel of a record, and value expressions, including the operator $\text{val}$ which yields the value of a communication record. We use the empty trace, $\text{()}$, traces of one record, e.g. $(\tau, c, \mu)$, as well as the concatenation operator $\wedge$ and the projection operator $\uparrow$ to create trace expressions. Further, for a trace expression $\text{texp}$ and a value expression $\text{vexp}$ we have $\text{texp}(\text{vexp})$ to refer to a particular record of $\text{texp}$, provided $\text{vexp}$ is a positive natural number less than or equal to the length of trace $\text{texp}$. We use expressions such as $\text{cset} \times \{\tau_1, \tau_2\}$ and the projection operator $\uparrow$ to form refusal expressions. To refer to the timed observation of a process we use the special variables $h$ and $R$ to denote the trace of the process and the refusal set of the process, respectively. These variables are not updated explicitly by the process: they refer to a timed observation from the semantics. Then, we can write specifications such as $\text{cl2 sat } h \{c\} = (\{} \wedge \exists t \geq 0 \cdot h \{c\} = ((t, c, 2))$. To reason about natural numbers, the assertion language includes, for value expression $\text{vexp}$, the predicate $\text{vexp} \in \mathbb{N}$ which is true if, and only if, the value of value expression $\text{vexp}$ is a natural number. Henceforth we use variables $i, j, k, l, m$ that range over $\mathbb{N}$. We use, for instance, $\forall i \cdot \phi$ as an abbreviation of $\forall i \in \mathbb{N} \rightarrow \phi$. For an assertion $\phi$ we also write $\phi(h, R)$ to indicate that $\phi$ has two free variables $h$ and $R$. We use $\phi(\text{texp}, \text{rfexp})$ to denote the assertion which is obtained from $\phi$ by replacing $h$ by trace expression $\text{texp}$, and $R$ by refusal expression $\text{rfexp}$. Let $\text{IVAR}$, with typical representative $t$, denote the set of logical time variables ranging over $\text{TIME}$, let $\text{VVAR}$, with typical representative $v$, denote the set of logical value variables ranging over $\text{VAL}$, let $\text{TVAR}$, with characteristic element $s$, be the set of logical trace variables ranging over $\text{TRACE}$, and let $\text{RVAR}$, with typical element $N$, be the set of logical refusal variables ranging over $\text{REF}$.

Table 2 presents the language we use to define assertions, with $\tau \in \text{TIME}$, $t \in \text{IVAR}$, $c \in \text{CHAN}$, $\mu \in \text{VAL}$, $v \in \text{VVAR}$, $s \in \text{TVAR}$, $N \in \text{RVAR}$, 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.

| Instant expression | $\text{ieexp} ::= \tau | t | \text{is}(\text{rexp}) | \text{ieexp}_1 \uparrow \text{ieexp}_2$ |
|--------------------|---------------------------------------------------------------|
| Channel expression | $\text{ccexp} ::= c | \text{ch}(\text{rexp})$ |
| Value expression   | $\text{vexp} ::= \mu | v | \text{val}(\text{rexp}) | \text{len}(\text{texp})$ |
| Record expression  | $\text{rexp} ::= (\text{ieexp}, \text{ccexp}, \text{vexp}) | \text{texp}(\text{vexp})$ |
| Trace expression   | $\text{texp} ::= s | h | (\text{vexp}) | \text{ieexp}_1 \uparrow \text{ieexp}_2 | \text{texp} \uparrow \text{cset}$ |
| Interval expression| $\text{inexp} ::= [\text{ieexp}_1, \text{ieexp}_2] | \{\text{ieexp}\}$ |
| Refusal expression | $\text{rfexp} ::= N | R | \text{O} | \text{cset} \times \text{inexp} | \text{rfexp}_1 \cup \text{rfexp}_2 | \text{rfexp} \uparrow \text{cset}$ |
| Assertion          | $\phi ::= \text{ieexp}_1 = \text{ieexp}_2 | \text{ccexp}_1 = \text{ccexp}_2 | \text{vexp}_1 = \text{vexp}_2 | \text{vexp} \in \mathbb{N} | \text{texp}_1 = \text{texp}_2 | \text{rfexp}_1 = \text{rfexp}_2 | \phi_1 \land \phi_2 | \neg \phi | \exists t \cdot \phi | \exists v \cdot \phi | \exists s \cdot \phi | \exists N \cdot \phi$ |

**Definition 17 (Abbreviation)** For record expression $\text{rexp}$ and trace expression $\text{texp}$, $\text{rexp} \in \text{texp}$ if there exists an $i$ such that $\text{texp}(i) = \text{rexp}$.  

11
Furthermore, we use the standard abbreviations $\phi_1 \lor \phi_2 \equiv \neg(\neg\phi_1 \land \neg\phi_2)$, $\phi_1 \rightarrow \phi_2 \equiv \neg\phi_1 \lor \phi_2$, and equivalences such as $\forall t \cdot \phi \equiv \neg(\exists! t \cdot \neg\phi)$.

**Definition 18 (Primitive predicates I)** Primitive predicates have a free variable $t$, the 'base time'. For a set $cset$ of channels and an instant expression $iexp$, a few typical examples are:

- **enable $cset$ at $iexp$** $\equiv (cset \times iexp) \cap R = \emptyset$
- **enable $cset$ for $iexp$** $\equiv (cset \times [t, t + iexp]) \cap R = \emptyset$
- **refuse $cset$ upto $iexp$** $\equiv cset \times [t, t + iexp] \subseteq R$
- **refuse $cset$ precisely upto $iexp$** $\equiv \forall \bar{t} \cdot (\text{refuse } cset \text{ upto } \bar{t} \iff \bar{t} \leq iexp )$
- **after $iexp : \psi$** $\equiv \psi[t + iexp/t]$

where $t + iexp/t$ denotes syntactic substitution of $t + iexp$ for $t$.

plus obvious combinations, e.g. using the connective 'and'.

It is sometimes convenient to refer to the willingness of the environment to communicate. For instance, as a communication does not occur until the environment stops refusing it, we can specify precisely for how long a communication must be enabled by taking the willingness mentioned before into account. In particular, consider the case that due to faults messages are lost. The fact that, after an input to a transmission medium, output fails to occur may indicate either that the message was lost, or that no communication partner has come forward yet. Using assumptions about the readiness of the environment to receive a message elegantly resolves such issues.

Suppose $(\theta, \mathcal{R}) \in \mathcal{O}[P]$. If $P$ did not refuse a $c$ communication at time $t$, that is, $(c,t) \not\in \mathcal{R}$, then the fact that no $c$ communication occurred at $t$, that is, $\neg(\exists! \bar{t} \cdot (t,c,v) \in \theta)$, implies that the environment was not prepared to engage in such a $c$ communication at time $t$. On the other hand, a $c$ communication that did occur at time $t$ could not have been refused by the environment. Thus, we can define possible refusal sets of the environment:

**Definition 19 (Match)** A timed refusal set $N$ matches timed trace $h$ and timed refusal set $R$, notation $\mathsf{Match}(h, R, N)$, iff

$$\forall c, t \cdot ((c,t) \not\in R \land \neg(\exists! \bar{t} \cdot (t,c,v) \in h)) \rightarrow (c,t) \in N$$

$$\land \forall c, t, \bar{v} \cdot (t,c,\bar{v}) \in h \rightarrow (c,t) \not\in N$$

**Definition 20 (Primitive predicates II)** We use a second category of primitive predicates tailored to the refusal set of the environment. For a set $cset$ of channels and an instant expression $iexp$, a few typical examples are:

- **$cset$ enabled at $iexp$** $\equiv$
  $$\forall N \cdot \mathsf{Match}(h, R, N) \rightarrow (cset \times iexp) \cap N = \emptyset$$
- **$cset$ refused upto $iexp$** $\equiv$
  $$\forall N \cdot \mathsf{Match}(h, R, N) \rightarrow cset \times [t, t + iexp] \subseteq N$$
- **$cset$ refused precisely upto $iexp$** $\equiv$
  $$\forall \bar{t} \cdot (\text{refused } cset \text{ upto } \bar{t} \iff \bar{t} \leq iexp )$$

Observe that we use the present tense to refer to refusals of the process, and the past tense to refer to refusals of the environment.
Example 1 (Calculator) Consider process $C$ that accepts a value via $in$, applies a function $f$ to it and produces the result via $out$. After an input it takes $K_C$ time units before the corresponding output becomes enabled. Once an output has occurred, a next input becomes enabled after $\varepsilon$ time units. We specify $C$ as follows:

\[
C \text{ sat } \forall i \cdot 1 \leq i \leq \text{len}(h \uparrow out) \rightarrow \text{val}(h \uparrow out(i)) = f(\text{val}(h \uparrow in(i)))
\]

\[
\land h = \emptyset \rightarrow \text{enable in and refuse out upto } \infty
\]

\[
\land \forall t, v \cdot (t, in, v) \in h \rightarrow \\
\text{refuse } \{in, out\} \text{ upto } K_C
\]

\[
\land \text{ after } K_C : \forall t \cdot \text{out refused precisely upto } \hat{t} \\
\rightarrow \text{enable out and refuse in for } \hat{t}
\]

\[
\land \forall t, v \cdot (t, out, v) \in h \rightarrow \\
\text{refuse } \{in, out\} \text{ upto } \varepsilon
\]

\[
\land \text{ after } \varepsilon : \forall t \cdot \text{in refused precisely upto } \hat{t} \\
\rightarrow \text{enable in and refuse out for } \hat{t}
\]

Notice how references to the readiness of the environment to communicate are used to determine, for instance, the time $K_C + \hat{t}$ at which an $out$ communication occurs after an input.

For an assertion $\phi$ we define the set $\text{chan}(\phi)$ of channels such that $c \in \text{chan}(\phi)$ if, and only if, a communication along $c$ might affect the validity of $\phi$. For instance, the validity of assertion $h = \emptyset$ is affected by any communication and thus we should have $\text{chan}(h = \emptyset) = \text{CHAN}$. Since, by the definition of the semantics, communications on a channel are refused for some time after a communication on that channel did occur, assertion $R \upharpoonright \{c\} = \emptyset$, like assertion $R \upharpoonright \{c\} = \{c\} \times [0, \infty)$, is invalidated by a communication along $c$, and by a communication along $c$ only. On the other hand, also the validity of assertion $(h \upharpoonright \{c\})'(\hat{5}, d, \hat{7}) = ((\hat{5}, d, \hat{7}))$ can only be changed by a communication along channel $c$, although $d$ occurs in the assertion as well. Hence, $\text{chan}(\phi)$ consists of the channels to which references to $h$ and $R$ in $\phi$ are restricted rather than the channels occurring syntactically in $\phi$. Note that the value of a logical variable is not affected by any communication.

Definition 21 (Channels in an assertion) For an assertion $\phi$ we inductively define the set $\text{chan}(\phi)$ as the smallest set of channels such that the validity of $\phi$ may only be affected by communications on the channels of $\text{chan}(\phi)$.

- $\text{chan}(\tau) = \text{chan}(t) = \emptyset$
- $\text{chan}(\text{ts}(\text{rexp})) = \text{chan}(\text{rexp})$
- $\text{chan}(\text{iexp}_1 + \text{iexp}_2) = \text{chan}(\text{ie xp}_1) \cup \text{chan}(\text{ie xp}_2)$
- $\text{chan}(c) = \emptyset$
- $\text{chan}(\text{ch}(\text{rexp})) = \text{chan}(\text{rexp})$
- $\text{chan}(\mu) = \text{chan}(v) = \emptyset$
- $\text{chan}(\text{val}(\text{rexp})) = \text{chan}(\text{rexp})$
- $\text{chan}(\text{len}(\text{te xp})) = \text{chan}(\text{te xp})$
- $\text{chan}(\text{iexp}, \text{ce xp}, \text{ve xp})) = \text{chan}(\text{ie xp}) \cup \text{chan}(\text{ce xp}) \cup \text{chan}(\text{ve xp})$
- $\text{chan}(\text{te xp}(\text{ve xp})) = \text{chan}(\text{te xp}) \cup \text{chan}(\text{ve xp})$
- $\text{chan}(s) = \emptyset$
- $\text{chan}(h) = \text{CHAN}$
Next we define the meaning of assertions. We use an environment $\gamma$ to interpret the logical variables of IVAR U VVAR U TVAR U TVAR. This environment maps a logical time variable $t$ to a value $\gamma(t) \in TIME$, a logical value variable $v$ to a value $\gamma(v) \in VAL$, a logical trace variable $s$ to a trace $\gamma(s) \in TRACE$, and a logical refusal variable $N$ to a refusal set $\gamma(N) \in REF$. An assertion is interpreted with respect to a triple $(\theta, \mathcal{R}, \gamma)$. Trace $\theta$ gives $h$ its value, refusal set $\mathcal{R}$ gives $R$ its value, and, as said before, environment $\gamma$ interprets the logical variables of IVAR U VVAR U TVAR U TVAR. We use the special symbol $\dagger$ to deal with the interpretation of $texp(vezp)$ where index $vezp$ 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 an instant expression $iexp$ in the trace $\theta$, refusal $\mathcal{R}$, and an environment $\gamma$, denoted by $I[iexp](\theta, \mathcal{R}, \gamma)$, yielding a value in $TIME \cup \{\dagger\}$, the value of a channel expression $cexp$ in the trace $\theta$, refusal $\mathcal{R}$, and an environment $\gamma$, denoted by $C[cexp](\theta, \mathcal{R}, \gamma)$, yielding a value in $CHAN \cup \{\dagger\}$, the value of a value expression $vexp$ in the trace $\theta$, refusal $\mathcal{R}$, and an environment $\gamma$, denoted by $V[vexp](\theta, \mathcal{R}, \gamma)$, yielding a value in $VAL \cup \{\dagger\}$, the value of a record expression $rexp$ in the trace $\theta$, refusal $\mathcal{R}$, and an environment $\gamma$, denoted by $R[rexp](\theta, \mathcal{R}, \gamma)$, yielding a value in $(CHAN \times VAL) \cup \{\dagger\}$, and the value of a refusal expression $rfxp$ in the trace $\theta$, refusal $\mathcal{R}$, and an environment $\gamma$, denoted by $RF[rfxp](\theta, \mathcal{R}, \gamma)$, yielding a value in $REF \cup \{\dagger\}$.
\[ I[\tau](\theta, \mathcal{R}, \gamma) = \tau \]
\[ I[\iota](\theta, \mathcal{R}, \gamma) = \gamma(t) \]
\[ I[i{%exp}(\theta, \mathcal{R}, \gamma)] = \begin{cases} \{ \tau \} \quad \text{iff } \mathcal{R}[{%exp}](\theta, \mathcal{R}, \gamma) = \tau \\
\{ \iota \} \quad \text{iff } \mathcal{R}[{%exp}](\theta, \mathcal{R}, \gamma) = \iota \\
\{ c \} \quad \text{iff there exist } c \text{ and } \mu \text{ such that } \mathcal{R}[{%exp}](\theta, \mathcal{R}, \gamma) = (\tau, c, \mu) \end{cases} \]
\[ I[i{exp}_1 + i{exp}_2](\theta, \mathcal{R}, \gamma) = I[i{exp}_1](\theta, \mathcal{R}, \gamma) + I[i{exp}_2](\theta, \mathcal{R}, \gamma) \]
\[ C[c](\theta, \mathcal{R}, \gamma) = c \]
\[ C[\text{ch}({%exp})](\theta, \mathcal{R}, \gamma) = \begin{cases} \{ \tau \} \quad \text{iff } \mathcal{R}[{%exp}](\theta, \mathcal{R}, \gamma) = \tau \\
\{ c \} \quad \text{iff there exist } \tau \text{ and } c \text{ such that } \mathcal{R}[{%exp}](\theta, \mathcal{R}, \gamma) = (\tau, c, \mu) \end{cases} \]
\[ \mathcal{V}[\mu](\theta, \mathcal{R}, \gamma) = \mu \]
\[ \mathcal{V}[\nu](\theta, \mathcal{R}, \gamma) = \gamma(\nu) \]
\[ \mathcal{V}[\text{val}({%exp})](\theta, \mathcal{R}, \gamma) = \begin{cases} \{ \tau \} \quad \text{iff } \mathcal{R}[{%exp}](\theta, \mathcal{R}, \gamma) = \tau \\
\{ c \} \quad \text{iff there exist } \tau \text{ and } c \text{ such that } \mathcal{R}[{%exp}](\theta, \mathcal{R}, \gamma) = (\tau, c, \mu) \end{cases} \]
\[ \mathcal{V}[\text{len}({%exp})](\theta, \mathcal{R}, \gamma) = \begin{cases} \{ \tau \} \quad \text{iff } \mathcal{R}[{%exp}](\theta, \mathcal{R}, \gamma) = \tau \\
\{ \text{len}(\mathcal{T}[{%exp}](\theta, \mathcal{R}, \gamma)) \} \quad \text{otherwise} \end{cases} \]
\[ \mathcal{R}[\text{ee{exp},ve{exp}}](\theta, \mathcal{R}, \gamma) = \begin{cases} \{ \tau \} \quad \text{iff } \mathcal{C}[\text{ee{exp}}](\theta, \mathcal{R}, \gamma) = \tau \text{ or } \mathcal{V}[\text{ve{exp}}](\theta, \mathcal{R}, \gamma) = \tau \\
\{ \mathcal{R}[\text{ee{exp}}](\theta, \mathcal{R}, \gamma), \mathcal{V}[\text{ve{exp}}](\theta, \mathcal{R}, \gamma) \} \quad \text{otherwise} \end{cases} \]
\[ \mathcal{R}[\text{te{exp}(ve{exp})}](\theta, \mathcal{R}, \gamma) = \begin{cases} \{ \tau \} \quad \text{iff there exist } \theta_1 \text{ and } \theta_2 \text{ such that } \text{len}(\theta_1) = \mathcal{V}[\text{ve{exp}}](\theta, \mathcal{R}, \gamma) - 1 \\
\{ \mathcal{T}[{%exp}](\theta, \mathcal{R}, \gamma) = \theta_1 \} \quad \text{iff } \mathcal{T}[{%exp}](\theta, \mathcal{R}, \gamma) = \theta_1 \end{cases} \]
\[ \mathcal{T}[s](\theta, \mathcal{R}, \gamma) = \gamma(s) \]
\[ \mathcal{T}[h](\theta, \mathcal{R}, \gamma) = \theta \]
\[ \mathcal{T}[\emptyset](\theta, \mathcal{R}, \gamma) = \emptyset \]
\[ \mathcal{T}[\gamma](\theta, \mathcal{R}, \gamma) = \gamma(\tau) \]
\[ \mathcal{T}[\{\}](\theta, \mathcal{R}, \gamma) = \emptyset \]
\[ \mathcal{T}[\{\tau\}](\theta, \mathcal{R}, \gamma) = \{ \tau \} \]
\[ \mathcal{T}[\{c, \mu\}](\theta, \mathcal{R}, \gamma) = \{ (c, \mu) \} \]
\[ \mathcal{T}[\text{exp}_1 \uparrow \text{exp}_2](\theta, \mathcal{R}, \gamma) = \begin{cases} \{ \iota \} \quad \text{iff } \mathcal{T}[\text{exp}_1](\theta, \mathcal{R}, \gamma) = \emptyset \text{ or } \mathcal{T}[\text{exp}_2](\theta, \mathcal{R}, \gamma) = \emptyset \\
\{ \mathcal{T}[\text{exp}_1](\theta, \mathcal{R}, \gamma) \} \quad \text{otherwise} \end{cases} \]
\[ \mathcal{T}[\text{exp} \uparrow \text{cset}](\theta, \mathcal{R}, \gamma) = \begin{cases} \{ \mathcal{T}[\text{exp}](\theta, \mathcal{R}, \gamma) \} \quad \text{iff } \mathcal{T}[\text{exp}](\theta, \mathcal{R}, \gamma) = \emptyset \text{ or } \mathcal{T}[\text{cset}](\theta, \mathcal{R}, \gamma) = \emptyset \\
\{ \mathcal{T}[\text{exp}](\theta, \mathcal{R}, \gamma) \} \quad \text{otherwise} \end{cases} \]
\[ \mathcal{N}[\text{ie{exp}_1,ie{exp}_2}](\theta, \mathcal{R}, \gamma) = \mathcal{I}[\text{ie{exp}_1}](\theta, \mathcal{R}, \gamma) \cup \mathcal{I}[\text{ie{exp}_2}](\theta, \mathcal{R}, \gamma) \]
\[ \mathcal{N}[\text{ie{exp}}](\theta, \mathcal{R}, \gamma) = \mathcal{I}[\text{ie{exp}}](\theta, \mathcal{R}, \gamma) \]
\[ \mathcal{R}[N](\theta, \mathcal{R}, \gamma) = \gamma(N) \]
\[ \mathcal{R}[\mathcal{R}](\theta, \mathcal{R}, \gamma) = \mathcal{R} \]
\[ \mathcal{R}[\emptyset](\theta, \mathcal{R}, \gamma) = \emptyset \]
Definition 22 (Variant of an environment) The variant of an environment \( \gamma \) with respect to a logical variable \( u \) (either in IVAR, VVAR, TVAR, or RVAR) and a \( v \) (resp. in TIME, VAL, TRACE, or REF), denoted \( (\gamma : u \mapsto v) \), is given by
\[
(\gamma : u \mapsto v)(w) = \begin{cases} 
  v & \text{if } w \equiv u \\
  \gamma(w) & \text{if } w \not\equiv u 
\end{cases}
\]

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

- \((\theta, \mathcal{R}, \gamma) \models vexp_1 = vexp_2 \) iff 
  \( \forall vexp_1, vexp_2 \in \text{Value} \) \( (\theta, \mathcal{R}, \gamma) \models vexp_1(\theta, \mathcal{R}, \gamma) = vexp_2(\theta, \mathcal{R}, \gamma) \) and \( vexp_1(\theta, \mathcal{R}, \gamma) \not\equiv vexp_2(\theta, \mathcal{R}, \gamma) \)
- \((\theta, \mathcal{R}, \gamma) \models cexp_1 = cexp_2 \) iff 
  \( \forall cexp_1, cexp_2 \in \text{Constant} \) \( (\theta, \mathcal{R}, \gamma) \models cexp_1(\theta, \mathcal{R}, \gamma) = cexp_2(\theta, \mathcal{R}, \gamma) \) and \( cexp_1(\theta, \mathcal{R}, \gamma) \not\equiv cexp_2(\theta, \mathcal{R}, \gamma) \)
- \((\theta, \mathcal{R}, \gamma) \models texP_1 = texP_2 \) iff 
  \( \forall texP_1, texP_2 \in \text{Term} \) \( (\theta, \mathcal{R}, \gamma) \models texP_1(\theta, \mathcal{R}, \gamma) = texP_2(\theta, \mathcal{R}, \gamma) \) and \( texP_1(\theta, \mathcal{R}, \gamma) \not\equiv texP_2(\theta, \mathcal{R}, \gamma) \)
- \((\theta, \mathcal{R}, \gamma) \models \phi_1 \land \phi_2 \) iff \( (\theta, \mathcal{R}, \gamma) \models \phi_1 \) and \( (\theta, \mathcal{R}, \gamma) \models \phi_2 \)
- \((\theta, \mathcal{R}, \gamma) \models \neg \phi \) iff not \((\theta, \mathcal{R}, \gamma) \models \phi \)
- \((\theta, \mathcal{R}, \gamma) \models \exists v \cdot \phi \) iff there exists a value \( \mu \) such that \( (\theta, \mathcal{R}, (\gamma : v \mapsto \mu)) \models \phi \)
- \((\theta, \mathcal{R}, \gamma) \models \exists s \cdot \phi \) iff there exists a trace \( \tilde{\theta} \) such that \( (\theta, \mathcal{R}, (\gamma : s \mapsto \tilde{\theta})) \models \phi \)
- \((\theta, \mathcal{R}, \gamma) \models \exists N \cdot \phi \) iff there exists a refusal \( \mathcal{R} \) such that \( (\theta, \mathcal{R}, (\gamma : N \mapsto \mathcal{R})) \models \phi \)

Example 2 (Satisfaction) In Example 1 we came across assertion 
\[ \forall t, v \cdot (t, in, v) \in h \rightarrow \text{refuse } \{in, out\} \text{ upto } K_C \]
which is an abbreviation of 
\[ \forall t, v \cdot (t, in, v) \in h \rightarrow \{in, out\} \times [t, t + K_C] \subseteq R \]
This assertion holds for the triple \((\theta, \mathcal{R}, \gamma) \) if, and only if, for any instant \( \tau \) and value \( \mu \) we have, for environment \( \tilde{\gamma} = (\gamma : t \mapsto \tau, v \mapsto \mu) \) which gives logical variables \( t \) and \( v \) the value of \( \tau \) and \( \mu \) respectively,
\[ (\theta, \mathcal{R}, \tilde{\gamma}) \models (t, in, v) \in h \rightarrow \{in, out\} \times [t, t + K_C] \subseteq R \]
Since \( h \) and \( R \) obtain their value from \( \theta \) and \( \mathcal{R} \), respectively, this implication holds for those traces \( \theta \) and refusals \( \mathcal{R} \) such that if \( \theta \) contains a record \((\tau, in, \mu)\) then \( \mathcal{R} \) contains \{in, out\} \times [\tau, \tau + K_C]. \]

Definition 23 (Validity of an assertion) An assertion is valid, notation \( \models \phi \), iff for all \( \theta, \mathcal{R}, \gamma \), \( (\theta, \mathcal{R}, \gamma) \models \phi \).

As mentioned before, we use a correctness formula \( P \text{ sat } \phi \) to express that process \( P \) satisfies property \( \phi \). Informally, since we abstract from the internal states of the processes and focus on communication, such a correctness formula expresses that any observation of \( P \) satisfies \( \phi \).

Definition 24 (Validity of a correctness formula) For process \( P \) and assertion \( \phi \) correctness formula \( P \text{ sat } \phi \) is valid, notation \( \models P \text{ sat } \phi \), iff for all \( \gamma \), and all \( (\theta, \mathcal{R}) \in C[P], (\theta, \mathcal{R}, \gamma) \models \phi \).
5 Incorporating Failure Hypotheses

Based on a particular failure hypothesis, the set of observations that characterize a process is expanded. To keep such an expansion manageable, failure hypothesis $\chi$ of process $P$ is formalized as a predicate whose only free variables are $h, h_{old}, R, R_{old}$ representing a relation between the normal and acceptable behaviours of $P$. The interpretation is such that $(h_{old}, R_{old})$ represents a normal observation of process $P$, whereas $(h, R)$ is an acceptable observation of $P$ with respect to $\chi$. Such relations enable us to abstract from the precise nature of a fault and to focus on the abnormal behaviour it causes. Notice that the faults that affect a process do not influence the enabledness of its environment to communicate. If, for instance, due to a failure process $C$ is sooner than usual willing to receive new input, then still this input will not occur before the environment is able to provide it.

We extend the assertion language with the trace expression term $h_{old}$ and refusal expression term $R_{old}$. Sentences of the extended language are called transformation expressions, with typical representatives $\psi$. To indicate that transformation expression $\psi$ has free variables $h_{old}, h, R_{old}$ and $R$, we also write $\psi(h_{old}, h, R_{old}, R)$. Then, $\psi(texp_1, texp_2, rfxp_1, rfxp_2)$ denotes the expression which is obtained from $\psi$ by substituting $texp_1$ for $h_{old}, texp_2$ for $h, rfxp_1$ for $R_{old},$ and $rfxp_2$ for $R$. A transformation expression is interpreted with respect to a quintet $(\theta_0, \theta, \mathcal{A}_0, \mathcal{A}, \gamma)$. Trace $\theta_0$ gives $h_{old}$ its value, refusal $\mathcal{A}_0$ gives $h$ and $R_{old}$ their value, and environment $\gamma$ interprets the logical variables of IVAR $\cup$ VVAR $\cup$ TVAR $\cup$ RVAR. The meaning of assertions, as defined on page 16, can easily be adapted for transformation expressions; the only new clauses are

- $T[h_{old}][\theta_0, \theta, \mathcal{A}_0, \mathcal{A}, \gamma] = \theta_0$
- $\mathcal{R}F[R_{old}][\theta_0, \theta, \mathcal{A}_0, \mathcal{A}, \gamma] = \mathcal{A}_0$

The channels occurring in a transformation expression are defined as in Definition 21 with extra clauses

- $\text{chan}(h_{old}) = \text{CHAN}$
- $\text{chan}(R_{old}) = \text{CHAN}$

Since the terms $h_{old}$ and $R_{old}$ do not occur in assertions, the following lemma is trivial.

**Lemma 1 (Correspondence)** For assertion $\phi$ it is for all $\theta_0$ and $\mathcal{A}_0$ the case that $(\theta_0, \theta, \mathcal{A}_0, \mathcal{A}, \gamma) \models \phi$ iff $(\theta, \mathcal{A}, \gamma) \models \phi$.

**Definition 25 (Failure hypothesis)** A failure hypothesis $\chi$ is a transformation expression which, to guarantee that the normal behaviour is part of the acceptable behaviour, represents a reflexive relation on the normal behaviour. Formally, we require that $\models \chi(h_{old}, h_{old}, R_{old}, R_{old})$. Furthermore, a failure hypothesis of failure prone process $FP$ does not impose restrictions on communications along channels not in $\text{chan}(FP)$, that is, $\models \chi(h_{old} \upharpoonright \text{chan}(FP), h_{old} \upharpoonright \text{chan}(FP), R_{old} \upharpoontright \text{chan}(FP), R \upharpoontright \text{chan}(FP))$.

Care has to be taken that a failure hypothesis upholds the principle that communications cannot occur while being refused. Also, a failure hypothesis may not allow communications via one and the same channel to succeed or even occur at all.

**Example 3 (Corruption)** Consider process $C$ as already defined in Example 1. Assuming that corruption does not influence the real-time behaviour of $C$, we formalize corruption by asserting that $h \upharpoontright \{\text{in, out}\}$ and $h_{old} \upharpoontright \{\text{in, out}\}$ are equally long, if the $i$th element of $h_{old} \upharpoontright \{\text{in, out}\}$ records an $\text{in}$ communication then it is equal to the $i$th element of $h \upharpoontright \{\text{in, out}\}$, if the $i$th element of $h_{old} \upharpoontright \{\text{in, out}\}$ records an $\text{out}$ communication then so does the $i$th element of $h \upharpoontright \{\text{in, out}\}$ and with equal timestamp. In the
latter case the communicated value recorded in $h$ is not specified allowing it to be any element of $VAL$.

$$\text{Cor} \equiv \begin{aligned} & \text{len}(h\{\text{in}, \text{out}\}) = \text{len}(h_{old}\{\text{in}, \text{out}\}) \\
& \wedge \forall i. 1 \leq i \leq \text{len}(h\{\text{in}, \text{out}\}) \rightarrow ch(h\{\text{in}, \text{out}\}(i)) = ch(h_{old}\{\text{in}, \text{out}\}(i)) \\
& \wedge \forall i. 1 \leq i \leq \text{len}(h) \rightarrow h\{\text{in}\}(i) = h_{old}\{\text{in}\}(i) \\
& \wedge \forall i. 1 \leq i \leq \text{len}(h) \rightarrow ts(h\{\text{out}\}(i)) = ts(h_{old}\{\text{out}\}(i)) \\
& \wedge R = R_{old} \end{aligned}$$

For a failure hypothesis $\chi$ we introduce, similar to [20], the construct $P|\chi$ to indicate execution of process $P$ under the assumption of $\chi$. This construct enables us to specify failure prone processes, with typical representative $FP$. 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.

| Failure Prone Process $FP$ | $P$ | $FP_1 \parallel FP_2$ | $FP \setminus \text{cset}$ | $FP|\chi$ |
|---------------------------|-----|-----------------------|-------------------------|---------|

From Definition 25 we obtain that $\text{chan}(\chi) \subseteq \text{chan}(FP)$. Hence, $\text{chan}(FP|\chi) = \text{chan}(FP) \cup \text{chan}(\chi) = \text{chan}(FP)$. As before, define $\text{chan}(FP_1 \parallel FP_2) = \text{chan}(FP_1) \cup \text{chan}(FP_2)$, and $\text{chan}(FP \setminus \text{cset}) = \text{chan}(FP) - \text{cset}$.

The timed observations of a failure prone process $FP$ are inductively defined as follows:

- $FP_1$ and $FP_2$ synchronize on communications on the channels in $\text{chan}(FP_1) \cap \text{chan}(FP_2)$. Hence, if $\theta$ is a trace of $FP_1 \parallel FP_2$ then $\theta \upharpoonright \text{chan}(FP_1)$ and $\theta \upharpoonright \text{chan}(FP_2)$ are the corresponding traces of $FP_1$ and $FP_2$, respectively. As we already saw in Section 3, a communication is refused by $FP_1 \parallel FP_2$ if, and only if, it is refused by $FP_1$ or $FP_2$. $O[FP_1 \parallel FP_2] = \{(\theta, \mathcal{R}) \mid \text{there exist } (\theta_1, \mathcal{R}_1) \in O[FP_1] \text{ and } (\theta_2, \mathcal{R}_2) \in O[FP_2] \text{ such that } \theta_1 \upharpoonright \text{chan}(FP_1) \equiv \theta_i, \text{ for } i = 1, 2, \theta_1 \upharpoonright \text{chan}(FP_1 \parallel FP_2) = \theta, \text{ and } \mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2 \}$

- The observations of $FP \setminus \text{cset}$ are, as before, characterized by the fact that cset communications are continuously refused, except on single instants. $O[FP \setminus \text{cset}] = \{(\theta, \mathcal{R} \setminus \text{cset}) \mid (\theta, \mathcal{R}) \in O[FP] \wedge \text{ASAP}(\mathcal{R}, \text{cset}) \}$

- The observations of failure prone process $FP|\chi$ are those observations that are related, according to $\chi$, to the observations of $FP$. $O[FP|\chi] = \{(\theta, \mathcal{R}) \mid \text{there exists } (\theta_0, \mathcal{R}_0) \in O[FP] \text{ such that, for all } \gamma, (\theta_0, \theta, \mathcal{R}_0, \mathcal{R}, \gamma) \models \chi, \theta \upharpoonright \text{chan}(FP) = \theta, \text{ and } \mathcal{R} \upharpoonright \text{chan}(FP) = \mathcal{R} \}$

From this definition of the semantics:

**Rule 5.1 (Invariance 1)**

$$\text{cset} \cap \text{chan}(FP) = \emptyset$$

$$\text{FP sat } h \downarrow \text{cset} = \emptyset$$

**Rule 5.2 (Invariance 2)**

$$\text{cset} \cap \text{chan}(FP) = \emptyset$$

$$\text{FP sat } R \downarrow \text{cset} = \emptyset$$

**Definition 26 (Composite transformation expression)** For transformation expressions $\psi_1$ and $\psi_2$, the composite transformation expression $\psi_1 \psi_2$ is defined as follows

$$\psi_1 \psi_2 \equiv \exists s, N. \psi_1(h_{old}, s, R_{old}, N) \wedge \psi_2(s, h, N, R)$$

where $s$ and $N$ must be fresh.
Since the interpretation of assertions has not changed, the validity of correctness formula \( FP \) sat \( \phi \) is defined as in Definition 24, with \( P \) replaced by \( FP \).

The following lemma is easy to prove by structural induction.

**Lemma 2 (Substitution)** Consider transformation expression \( \psi(h_{old}, h, R_{old}, R) \).

a) \( (\theta_0, \theta, R_0, R, \gamma) \models \psi(texp, h, R_{old}, R) \) iff \( (T[texp]([\theta_0, \theta, R_0, R, \gamma], \theta, R_0, R, \gamma) \models \psi \)

b) \( (\theta_0, \theta, R_0, R, \gamma) \models \psi(h_{old}, texp, R_{old}, R) \) iff \( (\theta_0, T[texp]([\theta_0, \theta, R_0, R, \gamma], R_0, R, \gamma) \models \psi \)

c) \( (\theta_0, \theta, R_0, R, \gamma) \models \psi(h_{old}, h, rfzp, R) \) iff \( (\theta_0, \theta, R[rfzp](\theta_0, \theta, R_0, R, \gamma), R, \gamma) \models \psi \)

d) \( (\theta_0, \theta, R_0, R, \gamma) \models \psi(h_{old}, h, R_{old}, rfzp) \) iff \( (\theta_0, \theta, R[rfzp](\theta_0, \theta, R_0, R, \gamma), R_0, R, \gamma) \models \psi \)

\[ \square \]

6 A Compositional Network Proof Theory

In this section we give a compositional network proof system for the correctness formulae. Since we focus on the relation between fault tolerance and concurrency, we have abstracted from the internal states of the processes and do not give rules for atomic statements, nor sequential composition.

The proof system contains the following two general rules.

**Rule 6.1 (Consequence)**

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

**Rule 6.2 (Conjunction)**

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

If \( h \) is a timed history of process \( FP_1 \parallel FP_2 \) then we know that \( h \) restricted to \( chan(FP_1) \) is the timed trace of communications performed by process \( FP_1 \). Similarly, the restriction of \( h \) to \( chan(FP_2) \) is the trace of communications performed by process \( FP_2 \). We also know that a communication is refused by \( FP_1 \parallel FP_2 \) if, and only if, it is refused by \( FP_1 \) or \( FP_2 \). The following inference rule for parallel composition reflects this knowledge.

**Rule 6.3 (Parallel Composition)**

\[
\frac{\text{FP_1 sat } \phi_1(h, R), \quad \text{FP_2 sat } \phi_2(h, R)}{\text{FP_1 || FP_2 sat } \exists N_1, N_2 \cdot R = N_1 \cup N_2 \land \phi_1(h \uparrow chan(FP_1), N_1) \land \phi_2(h \uparrow chan(FP_2), N_2)}
\]

Observations of \( FP \setminus \text{cset} \) are characterized by the fact that \( \text{cset} \) communications occur as soon as possible. Then, the effect of hiding a set \( \text{cset} \) of channels is simply that records of communications via channels of that set disappear from the process's history as do records of refused attempts from the process's refusal set. Thus, \( FP \setminus \text{cset} \) satisfies an assertion \( \phi \) if \( FP \) satisfies \( ASAP(R, \text{cset}) \to \phi \), unless a reference to \( h \) or \( R \) in \( \phi \) includes one or more channels from \( \text{cset} \).

**Rule 6.4 (Hiding)**

\[
\frac{\text{FP sat } ASAP(R, \text{cset}) \to \phi(h \setminus \text{cset}, R \setminus \text{cset})}{\text{FP \setminus \text{cset sat } \phi(h, R)}}
\]

**Lemma 3 (Hiding)** With respect to hiding the following equalities are useful:

a) \( (FP_1 \setminus \text{cset}) \parallel FP_2 = (FP_1 \parallel FP_2) \setminus \text{cset} \) iff \( chan(FP_2) \cap \text{cset} = \emptyset \)

b) \( (FP \setminus \text{cset}_1) \setminus \text{cset}_2 = FP \setminus (\text{cset}_1 \cup \text{cset}_2) \)
Finally, for the introduction of a failure hypothesis we have

Rule 6.5 (Failure hypothesis introduction)

\[
\frac{FP \text{ sat } \phi}{FPX \text{ sat } \phiX}
\]

Observe that, since \( \phi \) is an assertion, \( h_{old} \) and \( R_{old} \) do not appear in \( \phi \), and hence also the composite expression \( \phiX \) is an assertion.

7 Example: Triple Modular Redundancy

Consider the triple modular redundant system of Figure 1. It consists of three identical components \( C_j \), \( j = 1, 2, 3 \), as already discussed in Example 1, 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. We assume that a component needs \( K_C \) time units to apply a function \( f \) to an input value. Further, we assume that a component may transiently fail to provide output. To guarantee that a failed component does not arbitrarily fast accept fresh input, and hence confuse \( Voter \), usually a synchronization channel \( sync \) is added. In this section we give the main steps of the proof that such failure of at most one component per round can be tolerated.

![Figure 1](image)

Figure 1: Triple modular redundant system

Definition 27 (Abbreviations) Throughout this section we use the following abbreviations:

- \( \text{until}(texp, t) = \begin{cases} () & \text{if } texp = () \text{ or } ts(first(texp)) > t \\ texp_1 & \text{if } texp = texp_1 \wedge texp_2 \text{ such that } ts(last(texp_1)) \leq t \text{ and } ts(first(texp_2)) > t \end{cases} \) to denote trace \( texp \)'s prefix up to and including \( t \).

- \( \text{from}(texp, t) = \begin{cases} () & \text{if } texp = () \text{ or } ts(last(texp)) < t \\ texp_2 & \text{if } texp = texp_1 \wedge texp_2 \text{ such that } ts(last(texp_1)) \leq t \text{ and } ts(first(texp_2)) > t \end{cases} \) to denote trace \( texp \)'s suffix starting at \( t \).
In accepts a value from the environment via channel \( \text{in} \) and distributes that value via channels \( \text{in}_1, \text{in}_2 \) and \( \text{in}_3 \) after \( K_{\text{in}} \) time units. When all three of them have occurred \( \text{In} \) tries to communicate via \( \text{sync} \). \( \varepsilon \) time units after this communication has been taken, it enables \( \text{in} \) again.

\[
\text{In sat } \forall i, j : 1 \leq i \leq \text{len(h}_i \text{in}_j) \rightarrow \text{val(h}_i \text{in}_j(i)) = \text{val(h}_i \text{in}(i)) \\
\wedge h = () \rightarrow \text{enable in and refuse } \bigcup_{j=1}^3 \{\text{in}_j\} \text{ upto } \infty \\
\wedge \forall t, v : (t, \text{in}, v) \in h \rightarrow \\
\text{refuse chan(} \text{In}) \text{ upto } K_{\text{in}} \\
\bigwedge_{j=1}^3 \text{ after } K_{\text{in}} : \forall t_1 \cdot \text{in}_j \text{ refused precisely upto } t_1 \\
\rightarrow \text{enable in}_j \text{ for } t_1 \\
\wedge \forall t, v : (\bigwedge_{j=1}^3 (t, \text{in}_j, v) \in h) \rightarrow \\
\text{refuse chan(} \text{In}) \text{ upto } \varepsilon \\
\wedge \text{after } \varepsilon : \forall t_1 \cdot \text{sync refused precisely upto } t_1 \\
\rightarrow \text{enable sync for } t_1 \\
\wedge \forall t, v : (t, \text{sync}, v) \in h \rightarrow \\
\text{refuse chan(} \text{In}) \text{ upto } \varepsilon \\
\wedge \text{after } \varepsilon : \forall t_1 \cdot \text{in refused precisely upto } t_1 \\
\rightarrow \text{enable in for } t_1
\]

\( \text{Voter} \) awaits a communication via any of the channels \( \text{out}_1, \text{out}_2 \) and \( \text{out}_3 \). Upon occurrence of such a communication it starts a timer and awaits the remaining communications. If those remaining communications do not occur within \( \Delta \) time units the timer expires, and \( K_{\text{Voter}} \) time units thereafter the tentative vote is communicated to the environment via \( \text{out} \). Thus, timing is essential as it ends the waiting for a value that got lost. \( \varepsilon \) time units after an output occurs, \( \text{Voter} \) tries to synchronize via \( \text{sync} \). When this communication is taken, it enables channels \( \text{out}_1, \text{out}_2 \) and \( \text{out}_3 \) again.

\[
\text{Voter sat } \quad h = () \rightarrow \text{enable } \{\text{in}_1, \text{in}_2, \text{in}_3\} \text{ upto } \infty \\
\wedge \forall k, l, m, t, v : k \neq l \wedge k \neq m \wedge l \neq m \rightarrow \\
\forall t_1 \cdot \text{out}_m \text{ refused precisely upto } t_1 \\
\rightarrow \text{refuse out upto } \min(t_1, \Delta) + K_{\text{Voter}} \\
\wedge \text{after } \min(t_1, \Delta) + K_{\text{Voter}} : \\
\forall t_2 \cdot \text{out refused precisely upto } t_2 \\
\rightarrow \text{enable out for } t_2 \\
\wedge \forall v_1 : (t_2, \text{out}, v_1) \in h \rightarrow v_1 = v \\
\wedge \forall t, v : (t, \text{out}, v) \in h \rightarrow \\
\text{refuse chan(} \text{Voter}) \text{ upto } \varepsilon \\
\wedge \text{after } \varepsilon : \forall t_1 \cdot \text{sync refused precisely upto } t_1 \\
\rightarrow \text{enable sync and refuse chan(} \text{Voter}) \rightarrow \{\text{sync}\} \text{ for } t_1 \\
\wedge \forall t, v : (t, \text{sync}, v) \in h \rightarrow \\
\text{refuse chan(} \text{Voter}) \text{ upto } \varepsilon \\
\bigwedge_{j=1}^3 \text{ after } \varepsilon : \forall t_1 \cdot \text{in}_j \text{ refused precisely upto } t_1 \\
\rightarrow \text{enable in}_j \text{ for } t_1
\]

Since \( C_1, C_2, \) and \( C_3 \) do not share a single channel, we easily obtain, by (Parallel composition) and (Consequence), that
C_1||C_2||C_3 \text{ sat } \forall i, j \cdot 1 \leq i \leq \text{len}(h \uparrow \text{out}_j) \rightarrow \text{val}(h \uparrow \text{out}_j(i)) = f(\text{val}(h \uparrow \text{in}_j(i))) \\
\land h = () \rightarrow \text{enable } \bigcup_{j=1}^{3} \{\text{in}_j\} \text{ upto } \infty \\
\land \forall t, v \cdot (\bigcup_{j=1}^{3} (t, \text{in}_j, v) \in h) \\
\rightarrow \text{refuse } \bigcup_{j=1}^{3} \{\text{out}_j\} \text{ upto } K_C \\
\land \text{after } K_C: \\
\forall t_1 \cdot \bigcup_{j=1}^{3} \{\text{out}_j\} \text{ refused precisely upto } t_1 \\
\rightarrow \text{enable } \bigcup_{j=1}^{3} \{\text{out}_j\} \text{ for } t_1 \\
\land \bigcup_{j=1}^{3} \{\text{out}_j\} \text{ enabled at } t_1 \\
\rightarrow \text{after } t_1 + \varepsilon: \\
\forall t_2 \cdot \bigcup_{j=1}^{3} \{\text{in}_j\} \text{ refused precisely upto } t_2 \\
\rightarrow \text{enable } \bigcup_{j=1}^{3} \{\text{in}_j\} \text{ for } t_2

Under the assumption that faults do not change the rate at which a component accepts input, we formalize the hypothesis that per round at most one of the components C_1, C_2, and C_3 fails in the way described above as follows:

\[ \text{Loss}^{\leq 1} \equiv h \{\text{in}_{1,2,3}\} = h_{\text{old}} \{\text{in}_{1,2,3}\} \land \forall i \cdot 1 \leq i \leq \text{len}(h_{\text{old}} \{\text{out}_{1,2,3}\}/3) \rightarrow \exists k \neq i \cdot h_{\text{old}} \{\text{out}_k(i) \in h \land h_{\text{old}} \{\text{out}_i(i) \in h \}
\\
\land R \{\text{in}_{1,2,3}\} = R_{\text{old}} \{\text{in}_{1,2,3}\} \\
\land R \{\text{out}_{1,2,3}\} = R_{\text{old}} \{\text{out}_{1,2,3}\} \\
\bigcup_{j=1}^{3} \{\text{out}_j\} \times [t_1, t_2] \mid \exists t, v \cdot (t, \text{out}_j, v) \in h_{\text{old}} \land (t, \text{out}_i, v) \notin h \\
\land t_1 = ts(last(\text{until}(h \uparrow \text{in}_j, t))) \\
\land t_2 = ts(first(\text{from}(h \uparrow \text{in}_j, t))) \}

Observe that in this case the loss of a value boils down to refusing the communication involved until new input is accepted.

Failure hypothesis \text{Loss}^{\leq 1} expresses that per round only one output fails to occur, and, furthermore, that despite such a failure fresh input will be accepted as usual. Observe that it suffices to know that the environment did allow all output to conclude that a particular output does not occur due to a failure rather than the unavailability of a communication partner. Hence, by applying (Failure hypothesis introduction) and (Consequence) we conclude that after synchronous input via the channels in_1, in_2, and in_3 at least two of the components of failure prone process (C_1||C_2||C_3) \text{ Loss}^{\leq 1} \text{ will provide output within } K_C \text{ time units, and that if at the moment two such outputs occur the environment does not refuse any of the } \text{out}_j \text{ communications}, j = 1, 2, 3, \text{ then all three components will accept fresh input } \varepsilon \text{ time units thereafter.}
\((C_1 \| C_2 \| C_3) \uparrow \text{Loss}^{\leq 1}\)

sat

\(h = \emptyset \) \implies \text{enable } \bigcup_{j=1}^{3} \{i_{nj}\} \text{ up to } \infty \)
\(\land \forall t, v : (\bigwedge_{j=1}^{3} (t, i_{nj}, v) \in h) \implies \text{refuse } \bigcup_{j=1}^{3} \{o_{nj}\} \text{ up to } K_C \)
\(\land \text{ after } K_C : \forall t_1 : \bigcup_{j=1}^{3} \{o_{nj}\} \text{ refused precisely up to } t_1 \)
\(\implies \exists k \neq t : \text{enable } \{o_{nk}, o_{nk}\} \text{ for } t_1 \)
\(\land \forall v_1, v_2 : ((t_1, o_{nk}, v_1) \in h \land (t_1, o_{nk}, v_2) \in h) \implies v_1 = v_2 = f(v) \)
\(\land \bigcup_{j=1}^{3} \{o_{nj}\} \text{ enabled at } t_1 \)
\(\implies \text{ after } t_1 + \varepsilon : \forall t_2 : \bigcup_{j=1}^{3} \{i_{nj}\} \text{ refused precisely up to } t_2 \)
\(\implies \text{enable } \bigcup_{j=1}^{3} \{i_{nj}\} \text{ for } t_2 \)

Observe that, due to the assumptions concerning the environment’s enabledness to communicate, we only need the specifications of components \(C_1, C_2, \) and \(C_3\) and failure hypothesis \(\text{Loss}^{\leq 1}\) to establish this non-blocking property.

If the last communication of \(\text{Voter}\) relative to some instant \(t\) is a sync communication, or if \(\text{Voter}\) has not engaged in any communication up to and including time \(t\), then we know that \(\text{Voter}\) does not refuse any \(o_{nj}, j = 1, 2, 3,\) at time \(t\). Consequently, if an \(i_{nj}\) communication occurs at time \(t\) then the before mentioned readiness of \(\text{Voter}\) does not change until an \(o_{nj}\) communication, \(j = 1, 2, 3,\) actually takes place. Using \(h_{\text{Voter}} = h \upharpoonright \text{chan}(\text{Voter})\), we obtain, by (Parallel composition):

\(( (C_1 \| C_2 \| C_3) \uparrow \text{Loss}^{\leq 1} ) \mid \text{Voter} \)

sat

\(\text{ASAP}(R, \bigcup_{j=1}^{3} \{o_{nj}\}) \not\rightarrow \)
\(\forall t, v : (\bigwedge_{j=1}^{3} (t, i_{nj}, v) \in h) \land \text{until}(h_{\text{Voter}}, t) = \emptyset \land \exists t_1, v_1 : \text{last}(\text{until}(h_{\text{Voter}}, t)) = (t_1, \text{sync}, v_1) \quad)
\(\implies \exists t_1 : 0 \leq t_1 \leq \Delta \)
\(\land \text{refuse } \text{out } \text{upto } K_C + t_1 + K_{\text{Voter}} \)
\(\land \text{after } K_C + t_1 + K_{\text{Voter}} : \forall t_2 : \text{out refused precisely up to } t_2 \)
\(\implies \text{enable } \text{out for } t_2 \)
\(\land \forall v_1 : (t_2, o_{nk}, v_1) \in h \implies v_1 = v \)
\(\land \text{after } K_C + \varepsilon : \forall t_1 : \bigcup_{j=1}^{3} \{i_{nj}\} \text{ refused precisely up to } t_1 \)
\(\implies \text{enable } \bigcup_{j=1}^{3} \{i_{nj}\} \text{ for } t_1 \)
\(\land \forall t, v : (t, o_{nk}, v) \in h \implies \text{refuse } \text{sync } \text{upto } \varepsilon \)
\(\land \text{after } \varepsilon : \forall t_1 : \text{sync refused precisely up to } t_1 \implies \text{enable } \text{sync for } t_1 \)

Note that if \((r, c, \mu) \in h \land c \notin \text{cset}\) then also \((r, c, \mu) \in h \upharpoonright \text{cset}\). Further note that if \(h = \emptyset\) then \(h \upharpoonright \text{cset} = \emptyset\).

Because \(\text{In}\) will not accept new input until a sync communication occurs, we may conclude that if at time \(t\) a sync communication occurs and, for \(j = 1, 2, 3,\) there either has been no \(i_{nj}\) communication, or the preceding \(i_{nj}\) communications all happened at the same time, then \(C_j\) does not refuse \(i_{nj}, j = 1, 2, 3,\) at time \(t\). Again, this readiness does not change until an \(i_{nj}\) communication, \(j = 1, 2, 3,\) actually occurs. By (Hiding), the specification of \(\text{In}\), and (Parallel composition),

\(23\)
\[ \text{In} \quad \begin{array}{c} ((C_1||C_2||C_3) \downarrow \text{Loss}^{\leq 1}) \| \text{Voter} \end{array} \setminus \bigcup_{j=1}^{3} \{ \text{out}_j \} \]

\[
\text{sat}
\]

\[
\forall t, v \cdot \left( (t, \text{in}, v) \in h \right)
\]

\[
\land \begin{array}{c}
\text{until}(h, \bigcup_{j=1}^{3} \{ \text{in}_j \} , t) = () \\
\lor \exists t_1, v_1 \cdot \bigwedge_{j=1}^{3} \text{last} \left( \text{until}(h, c_j, t) \right) = (t_1, \text{in}_j, v_1)
\end{array}
\]

\[
\rightarrow \exists t_1 \cdot 0 \leq t_1 \leq \Delta
\]

\[
\land \begin{array}{c}
\text{refuse out upto } K_{in} + K_C + t_1 + K_{\text{Voter}}
\end{array}
\]

\[
\land \begin{array}{c}
\text{after } K_{in} + K_C + t_1 + K_{\text{Voter}} : \forall t_2 \cdot \text{out refused precisely upto } t_2
\end{array}
\]

\[
\rightarrow \begin{array}{c}
\text{enable out for } t_2
\end{array}
\]

\[
\land \begin{array}{c}
\forall v_1 \cdot (t_2, \text{out}, v_1) \in h \rightarrow v_1 = f(v)
\end{array}
\]

\[
\forall t, v \cdot (t, \text{out}, v) \in h \rightarrow \text{refuse in upto } 2 \varepsilon
\]

\[
\land \begin{array}{c}
\text{after } 2 \varepsilon : \forall t_1 \cdot \text{in refused precisely upto } t_1 \rightarrow \text{enable in for } t_1
\end{array}
\]

8 Soundness and Relative Network Completeness

In this section we show that the proof system of Section 6 is sound, that is, we prove that, if a correctness formula \( FP \) sat \( \phi \) is derivable, then it is valid. Furthermore, we prove the proof system to be complete, that is, we prove that, if a correctness formula \( FP \) sat \( \phi \) is valid, then it is derivable.

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

**Proof.** See Appendix A.

As usual when designing a proof theory, we only prove relative completeness, assuming that we can prove any valid formula of the underlying logic (cf. [5]). 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 \( \phi \),

\[ \vdash \phi \quad \text{if} \quad \models \phi \]

24
As in [24] 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 observations of the process.

**Definition 28 (Preciseness)** An assertion $\phi$ is **precise** for failure prone process $FP$ iff

1. $\models FP sat \phi$.
2. if $\theta \upharpoonright \text{chan}(FP) = \theta, \mathcal{R} \upharpoonright \text{chan}(FP) = \mathcal{R}$, and, for some $\gamma, (\theta, \mathcal{R}, \gamma) \models \phi$, then $(\theta, \mathcal{R}) \in O[FP]$.
3. $\phi \rightarrow \phi(h \upharpoonright \text{chan}(FP), R \upharpoonright \text{chan}(FP))$.

Let $\vdash P sat \phi$ denote that correctness formula $P sat \phi$ 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 29 (Network completeness)** Assume that for every process $P$ there exists a precise assertion $\phi$ with $\vdash P sat \phi$. Then, for any failure prone process $FP$ and assertion $\eta$, $\models FP sat \eta$ implies $\vdash FP sat \eta$.

The following lemma asserts that preciseness is preserved.

**Lemma 4 (Preciseness preservation)** Assume that for any process $P$ there exists an assertion $\phi$ which is precise for $P$ and $\vdash P sat \phi$. Then, for any failure prone process $FP$ there exists an assertion $\eta$ which is precise for $FP$ and $\vdash FP sat \eta$.

**Proof.** See Appendix B.

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 neither communicates on those other channels nor refuses to do so.

**Lemma 5 (Preciseness consequence)** If $\phi_1$ is precise for $FP$ and $\models FP sat \phi_2$ then $\models (\phi_1 \land h \upharpoonright \text{chan}(FP) = h \land R \upharpoonright \text{chan}(FP) = R) \rightarrow \phi_2$.

**Proof.** Assume that $\phi_1$ is precise for $FP$, and that $\models FP sat \phi_2$. Consider any $\theta, \mathcal{R}$, and $\gamma$. Assume $(\theta, \mathcal{R}, \gamma) \models \phi_1 \land h \upharpoonright \text{chan}(FP) = h \land R \upharpoonright \text{chan}(FP) = R$. Then, by the preciseness of $\phi_1$ for $FP$, $(\theta, \mathcal{R}) \in O[FP]$. By (1), for all $\tilde{\gamma}$, $(\theta, \mathcal{R}, \tilde{\gamma}) \models \phi_2$. Hence, $(\theta, \mathcal{R}, \gamma) \models \phi_2$. 

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 $\vdash P sat \phi$. Then, by the preciseness preservation lemma, for every failure prone process $FP$ there exists an assertion $\eta$ which is precise for $FP$ and $\vdash FP sat \eta$.

Assume $\models FP sat \xi$. By the definition of the semantics,

$\vdash FP sat h \upharpoonright \text{chan}(FP) = h \land R \upharpoonright \text{chan}(FP) = R$.

Then, by (1), (2), the preciseness consequence lemma, the relative completeness assumption, and (Consequence), $\vdash FP sat \xi$. 

25
9 Conclusions

We have defined a compositional proof theory for fault tolerant real-time distributed systems. In this theory, the failure hypothesis of a process is formalized as a relation between the normal and acceptable 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 failure hypothesis introduction rule, is needed. We illustrated our method by proving correctness of a triple modular redundant system.

An obvious continuation of the research described in this report is to find a logic to express failure hypotheses more elegantly, e.g. using the classification of failures that appears in [7].

Acknowledgement We would like to thank the members of STW project ‘Fault Tolerance: Paradigms, Models, Logics, Construction’, especially Job Zwiers, for their comments.

References


A Proof of the Soundness Theorem

A.1 Soundness of the consequence and conjunction rule

Trivial.

A.2 Soundness of the parallel composition rule

Assume $\vdash FP_1 \text{ sat } \phi_1, \vdash FP_2 \text{ sat } \phi_2$.

Consider any $\gamma$. Let $(\theta, \mathcal{R}) \in \mathcal{O}[FP_1][FP_2]$. By the definition of the semantics there exist, for $i = 1, 2$, $\mathcal{R}_i$ such that $(\theta \upharpoonright \text{chan}(FP_i), \mathcal{R}_i) \in \mathcal{O}[FP_i]$ (1), and $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ (2).

Let, for fresh $N_1$ and $N_2$, $\tilde{\gamma} = (\gamma : (N_1, N_2) \mapsto (\mathcal{R}_1, \mathcal{R}_2))$.

By (3), $(\theta, \mathcal{R}, \tilde{\gamma}) \vdash R = N_1 \cup N_2$, or $(\theta, \mathcal{R}, \gamma) \vdash \exists N_1, N_2 \cdot R = N_1 \cup N_2$ (4).

By (1) and (2), for all $\gamma'$, $(\theta \upharpoonright \text{chan}(FP_i), \mathcal{R}_i, \gamma') \vdash \phi_i$, $i = 1, 2$. Then, $(\theta \upharpoonright \text{chan}(FP_i), \mathcal{R}_i, \tilde{\gamma}) \vdash \phi_i$ (5).

Observe that $\mathcal{R}[N_1][\theta, \mathcal{R}, \tilde{\gamma}] = \mathcal{R}_1$ and that $T[h \upharpoonright \text{chan}(FP_i)][\theta, \mathcal{R}, \tilde{\gamma}] = \theta \upharpoonright \text{chan}(FP_i)$. Consequently, we have $(T[h \upharpoonright \text{chan}(FP_i)][\theta, \mathcal{R}, \tilde{\gamma}], \mathcal{R}[N_1][\theta, \mathcal{R}, \tilde{\gamma}], \tilde{\gamma}) \vdash \phi_i$. Then, by applying substitution lemma

27
b) and d), we obtain that \((\theta, \mathfrak{R}, \gamma) \models \phi_1[(h \mid \text{chan}(FP_1))/h, N_1/R]\), from which we may conclude that
\[(\theta, \mathfrak{R}, \gamma) \models \exists N_1, N_2 \cdot \phi_1[(h \mid \text{chan}(FP_1))/h, N_1/R]\]  
(5).

By (4) and (5) we conclude that the parallel composition rule is sound. \(\square\)

### A.3 Soundness of the hiding rule

Assume that \(\models FP \text{ sat } \text{ASAP}(R, \text{cset}) \rightarrow \phi(h \setminus \text{cset}, R \setminus \text{cset})\)  
(1).

Consider any \(\gamma\). Let \((\theta, \mathfrak{R}) \in \mathcal{O}[FP \setminus \text{cset}]\). Then, by the definition of the semantics there exists a \((\hat{\theta}, \mathfrak{S}) \in \mathcal{O}[FP]\)  
(2) 
for which \(\text{ASAP}(\mathfrak{S}, \text{cset})\),  
(3) 
such that \(\theta = \hat{\theta} \setminus \text{cset}\)  
(4), 
and \(\mathfrak{R} = \mathfrak{S} \setminus \text{cset}\).  
(5).

By (2) and (1), we have that, for all \(\gamma, (\hat{\theta}, \mathfrak{S}, \gamma) \models \text{ASAP}(R, \text{cset}) \rightarrow \phi(h \setminus \text{cset}, R \setminus \text{cset})\). Then, by (3), \((\hat{\theta}, \mathfrak{S}, \gamma) \models \phi(h \setminus \text{cset}, R \setminus \text{cset})\). By substitution lemma b) and d), we obtain \((\hat{\theta} \setminus \text{cset}, \hat{\mathfrak{S}} \setminus \text{cset}, \gamma) \models \phi\). Hence, by (4) and (5), \((\theta, \mathfrak{R}, \gamma) \models \phi\), from which we conclude that the hiding rule is sound. \(\square\)

### A.4 Soundness of the failure hypothesis introduction rule

Assume that \(\models FP \text{ sat } \phi\)  
(1).

Consider any \(\gamma\). Let \((\theta, \mathfrak{R}) \in \mathcal{O}[FP] \chi\). Then, by the definition of the semantics, there exists a \((\theta_0, \mathfrak{R}_0) \in \mathcal{O}[FP]\), such that, for all \(\gamma, (\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \chi\)  
(2).

Let, for fresh \(s\) and \(N, \tilde{\gamma} = (\gamma : (s, N) \mapsto (\theta_0, \mathfrak{R}_0))\).

Since \((\theta_0, \mathfrak{R}_0) \in \mathcal{O}[FP]\), we know, by (1), that, for all \(\gamma, (\theta_0, \mathfrak{R}_0, \gamma) \models \phi\). Consequently, we have \((\theta_0, \mathfrak{R}_0, \tilde{\gamma}) \models \phi\). As, for all \(\hat{\theta}\) and \(\mathfrak{S}\), \(T[s](\theta, \mathfrak{R}, \tilde{\gamma}) = \theta_0\) and \(\mathcal{R}[N][\theta, \mathfrak{R}, \tilde{\gamma}] = \mathfrak{R}_0\), we may conclude \((T[s](\theta, \mathfrak{R}, \tilde{\gamma}), \mathcal{R}[N][\theta, \mathfrak{R}, \tilde{\gamma}], \tilde{\gamma}) \models \phi\). Hence, by applying substitution lemma b) and d), we obtain \((\theta, \mathfrak{R}, \gamma) \models \phi[h \setminus \text{old}, N/R]\).  
(3).

By (2), \((\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \chi\), that is, \((T[s](\theta, \mathfrak{R}, \tilde{\gamma}), \mathcal{R}[N][\theta, \mathfrak{R}, \tilde{\gamma}], \tilde{\gamma}) \models \chi\). Then, by substitution lemma a) and c), \((\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \chi[s/h \setminus \text{old}, N/R\setminus\text{old}]\). Since \(h\text{old}\) and \(R\text{old}\) obviously do not appear in \(\chi[s/h \setminus \text{old}, N/R\setminus\text{old}]\) we may conclude that \((\theta, \mathfrak{R}, \gamma) \models \chi[s/h \setminus \text{old}, N/R\setminus\text{old}]\).  
(4).

By (3) and (4) we obtain that \((\theta, \mathfrak{R}, \gamma) \models \phi[h \setminus \text{old}, N/R] \wedge \chi[s/h \setminus \text{old}, N/R\setminus\text{old}]\), from which we conclude \((\theta, \mathfrak{R}, \gamma) \models \exists s, N \cdot \phi[h \setminus \text{old}, N/R] \wedge \chi[s/h \setminus \text{old}, N/R\setminus\text{old}]\). Hence, the failure hypothesis introduction rule is sound. \(\square\)

### B Proof of the Preciseness Preservation Lemma

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

a) Assume \(\vdash FP_1 \text{ sat } \phi_1 \) and \(\vdash FP_2 \text{ sat } \phi_2\), with \(\phi_1 \) and \(\phi_2\) precise for \(FP_1\) and \(FP_2\), respectively.

By applying (Parallel composition), we obtain
\[\vdash FP_1 \parallel FP_2 \text{ sat } \exists N_1, N_2 \cdot R = N_1 \cup N_2 \wedge \phi_1(h \mid \text{chan}(FP_1), N_1) \wedge \phi_2(h \mid \text{chan}(FP_2), N_2)\]  
(1).

We show that the above specification is precise for \(FP_1 \parallel FP_2\).

i) By (1) and soundness, we obtain
\[\models FP_1 \parallel FP_2 \text{ sat } \exists N_1, N_2 \cdot R = N_1 \cup N_2 \wedge \phi_1(h \mid \text{chan}(FP_1), N_1) \wedge \phi_2(h \mid \text{chan}(FP_2), N_2)\]
ii) Let $\text{chan}(\theta) \subseteq \text{chan}(FP_1\|FP_2)$ and $\mathcal{R} \vdash \text{chan}(FP_1\|FP_2) = \mathcal{R}$

Assume further that, for some $\gamma$, $(\theta, \mathcal{R}, \gamma) \models \exists N_1, N_2 \cdot R = N_1 \cup N_2$

\[ \wedge \phi_1(h \uparrow \text{chan}(FP_1), N_1) \]

\[ \wedge \phi_2(h \uparrow \text{chan}(FP_2), N_2) \]

Hence, there exist $\mathcal{R}_1$ and $\mathcal{R}_2$ such that

\[ (\theta, \mathcal{R}, (\gamma : (N_1, N_2) \mapsto (\mathcal{R}_1, \mathcal{R}_2))) \models \exists N_1, N_2 \cdot R = N_1 \cup N_2 
\]

\[ \wedge \phi_1(h \uparrow \text{chan}(FP_1), N_1) \]

\[ \wedge \phi_2(h \uparrow \text{chan}(FP_2), N_2) \]

Then, by substitution lemma b) and d),

\[ (\theta \uparrow \text{chan}(FP_1), \mathcal{R}_1, \gamma : (N_1, N_2) \mapsto (\mathcal{R}_1, \mathcal{R}_2)) \models \phi_1, \]

and since $N_1$ and $N_2$ do not occur free in $\phi_1$, $(\theta \uparrow \text{chan}(FP_1), \mathcal{R}_1, \gamma) \models \phi_1$. By the preciseness of $\phi$, for $FP_1$,

\[ (\theta \uparrow \text{chan}(FP_1), \mathcal{R}_1, \gamma) \models \phi_1(h \uparrow \text{chan}(FP_1) / h, R \uparrow \text{chan}(FP_1) / R). \]

By substitution lemma b) and d), using $(\theta \uparrow \text{chan}(FP_1)) \uparrow \text{chan}(FP_1) = \theta \uparrow \text{chan}(FP_1)$,

\[ (\theta \uparrow \text{chan}(FP_1), \mathcal{R}_1 \uparrow \text{chan}(FP_1), \gamma) \models \phi_1 \]

Trivially, $\text{chan}(\theta \uparrow \text{chan}(FP_1)) \subseteq \text{chan}(FP_1)$

It is also obvious that $(\mathcal{R}_1 \uparrow \text{chan}(FP_1)) \uparrow \text{chan}(FP_1) = \mathcal{R}_1 \uparrow \text{chan}(FP_1)$

By (5b), (5c), and (5a), the preciseness of $\phi_1$ for $FP_1$ leads to

\[ ((\theta \uparrow \text{chan}(FP_1), \mathcal{R}_1 \uparrow \text{chan}(FP_1))) \in \mathcal{O}[FP_1] \]

Similarly, $(\theta \uparrow \text{chan}(FP_2), \mathcal{R}_2 \uparrow \text{chan}(FP_2))) \in \mathcal{O}[FP_2]$

By (2), trivially, $\theta \uparrow \text{chan}(FP_1\|FP_2) = \emptyset$

By (6), by substitution lemma b) and d), considering $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, that is, by (3), $\mathcal{R} = (\mathcal{R}_1 \uparrow \text{chan}(FP_1)) \cup (\mathcal{R}_2 \uparrow \text{chan}(FP_2))$

By (5), (6), (7), and (8), $(\theta, \mathcal{R}) \in \mathcal{O}[FP_1\|FP_2]$

iii) Consider any $\theta, \mathcal{R}$, and $\gamma$ such that

\[ (\theta, \mathcal{R}, \gamma) \models \exists N_1, N_2 \cdot R = N_1 \cup N_2 
\]

\[ \wedge \phi_1(h \uparrow \text{chan}(FP_1), N_1) \]

\[ \wedge \phi_2(h \uparrow \text{chan}(FP_2), N_2) \]

which is, obviously, equivalent to

\[ (\theta, \mathcal{R}, \gamma) \models \exists N_1, N_2 \cdot R = N_1 \cup N_2 
\]

\[ \wedge \phi_1((h \uparrow \text{chan}(FP_1\|FP_2)) \uparrow \text{chan}(FP_1), N_1) \]

\[ \wedge \phi_2((h \uparrow \text{chan}(FP_1\|FP_2)) \uparrow \text{chan}(FP_2), N_2) \]

By the preciseness of $\phi_1$ and $\phi_2$ for $FP_1$ and $FP_2$, we have, using $\mathcal{N}_i = N_i \uparrow \text{chan}(FP_i)$, $i = 1, 2$,

\[ (\theta, \mathcal{R}, \gamma) \models \exists N_1, N_2 \cdot R = N_1 \cup N_2 
\]

\[ \wedge \phi_1((h \uparrow \text{chan}(FP_1\|FP_2)) \uparrow \text{chan}(FP_1), \mathcal{N}_1) \]

\[ \wedge \phi_2((h \uparrow \text{chan}(FP_1\|FP_2)) \uparrow \text{chan}(FP_2), \mathcal{N}_2) \]

Following the steps that were taken sub ii), we obtain, using $\mathcal{R} = R \uparrow \text{chan}(FP_1\|FP_2)$,

\[ (\theta, \mathcal{R}, \gamma) \models \exists N_1, N_2 \cdot \mathcal{R} = \mathcal{N}_1 \cup \mathcal{N}_2 
\]

\[ \wedge \phi_1((h \uparrow \text{chan}(FP_1\|FP_2)) \uparrow \text{chan}(FP_1), \mathcal{N}_1) \]

\[ \wedge \phi_2((h \uparrow \text{chan}(FP_1\|FP_2)) \uparrow \text{chan}(FP_2), \mathcal{N}_2) \]

or, equivalently,

\[ (\theta, \mathcal{R}, \gamma) \models \exists \mathcal{N}_1, \mathcal{N}_2 \cdot \mathcal{R} = \mathcal{N}_1 \cup \mathcal{N}_2 
\]

\[ \wedge \phi_1((h \uparrow \text{chan}(FP_1\|FP_2)) \uparrow \text{chan}(FP_1), \mathcal{N}_1) \]

\[ \wedge \phi_2((h \uparrow \text{chan}(FP_1\|FP_2)) \uparrow \text{chan}(FP_2), \mathcal{N}_2) \]
b) Assume $\vdash FP$ sat $\phi$

with $\phi$ precise for $FP$. Define $\widehat{\phi} \equiv \exists s, N \cdot \phi(s, N)$
\[ \land \text{ASAP}(N, cset) \]
\[ \land h = s \setminus cset \]
\[ \land R = N \setminus cset \]

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

**Lemma 6** $\models \phi \rightarrow \widehat{\phi}(h \setminus cset, R \setminus cset, E \setminus cset)$

By Lemma 6 and the relative completeness assumption,

$\vdash \phi \rightarrow \widehat{\phi}(h \setminus cset, R \setminus cset)$

Hence, by (1) and (Consequence),

$\vdash FP$ sat $\widehat{\phi}(h \setminus cset, R \setminus cset)$

Then, by (Hiding), $\vdash FP \setminus cset$ sat $\widehat{\phi}$ (2).

It remains to be shown that $\widehat{\phi}$ is precise for $FP \setminus cset$

i) By (2) and soundness $\models FP \setminus cset$ sat $\widehat{\phi}$.

ii) Let $\text{chan}(\theta) \subseteq \text{chan}(FP \setminus cset)$

$\models \text{chan}(FP \setminus cset) = \mathcal{A}$

and assume that, for some $\gamma$, $(\theta, \mathcal{A}, \gamma) \models \widehat{\phi}$. Then, there exist some $\widehat{\theta}$ and $\widehat{\mathcal{A}}$ such that

$(\theta, \widehat{\mathcal{A}}, (\gamma : (s, N) \mapsto (\widehat{\theta}, \widehat{\mathcal{A}}))) \models \phi(s, N)$
\[ \land \text{ASAP}(N, cset) \]
\[ \land h = s \setminus cset \]
\[ \land R = N \setminus cset \]

Then, $\text{ASAP}(\widehat{\mathcal{A}}, cset)$
\[ \theta = \widehat{\theta} \setminus cset \]

and $\widehat{\mathcal{A}} = \widehat{\mathcal{A}} \setminus cset$

By (5), $(\widehat{\theta}, \widehat{\mathcal{A}}, \gamma) \models \phi$ (9a).

By (3) and (7), $\text{chan}(\widehat{\theta}) \subseteq \text{chan}(FP \setminus cset)$, and, hence, $\text{chan}(\widehat{\theta}) \subseteq \text{chan}(FP)$ (9b).

By (4) and (8), and the fact that $cset \subseteq \text{chan}(FP)$, we obtain $\models \widehat{\mathcal{A}} = \text{chan}(FP)$ (9c).

By (9b), (9c), and (9a), and the preciseness of $\phi$ for $FP$, $(\widehat{\theta}, \widehat{\mathcal{A}}) \in \mathcal{O}[FP]$ (9).

By (9), (6), (7), and (8), $(\theta, \mathcal{A}) \in \mathcal{O}[FP \setminus cset]$.

iii) Assume $(\theta, \mathcal{A}, \gamma) \models \phi$. Then, there exist $\widehat{\theta}$ and $\widehat{\mathcal{A}}$ such that

$(\theta, \mathcal{A}, (\gamma : (s, N) \mapsto (\widehat{\theta}, \widehat{\mathcal{A}}))) \models \phi(s, N)$
\[ \land \text{ASAP}(N, cset) \]
\[ \land h = s \setminus cset \]
\[ \land R = N \setminus cset \]

By the preciseness of $\phi$ for $FP$,

$\phi(s, N) \rightarrow \phi(s \upharpoonright \text{chan}(FP), N \setminus \text{chan}(FP))$ (2).

It is obvious that $\text{ASAP}(N, cset) \rightarrow \text{ASAP}(N \setminus \text{chan}(FP), cset)$ (3).

Note that $h = s \setminus cset \rightarrow h \upharpoonright \text{chan}(FP \setminus cset) = (s \upharpoonright \text{chan}(FP)) \setminus cset$ (4).

By (1), $R = N \setminus cset$, that is, $R \upharpoonright \text{chan}(FP \setminus cset) = (N \upharpoonright \text{chan}(FP)) \setminus cset$ (5).
By (1) - (5), \((\theta, \mathfrak{M}, (\gamma : (s, N) \mapsto (\bar{\theta}, \bar{\mathfrak{M}}))) \models \phi(s \upharpoonright \text{chan}(FP), N \upharpoonright \text{chan}(FP))
\land \text{ASAP}(N \upharpoonright \text{chan}(FP), \text{cset})
\land h[\text{chan}(FP) \setminus \text{cset}] = (s \upharpoonright \text{chan}(FP)) \setminus \text{cset}
\land R[\text{chan}(FP) \setminus \text{cset}] = (N \upharpoonright \text{chan}(FP)) \setminus \text{cset}

From which we may conclude \((\theta, \mathfrak{M}, \gamma) \models \bar{\phi}[h \upharpoonright \text{chan}(FP) \setminus \text{cset}]/h, R[\text{chan}(FP) \setminus \text{cset}]/R].

c) Assume \(\models FP \text{ sat } \phi\)
with \(\phi\) precise for \(FP\). Define \(\bar{\phi} \equiv \phi \backslash \chi\), that is

\[
\bar{\phi} \equiv \exists s, N : \phi(s/h, N/R] \land \chi[s/h_{\text{old}}, N/R_{\text{old}}]
\]

Then, by (1) and (Failure hypothesis introduction), \(\models FP \backslash \chi \text{ sat } \bar{\phi}\)
We show that \(\bar{\phi}\) is precise for \(FP\).

i) By (2) and soundness, we have \(\models FP \backslash \chi \text{ sat } \bar{\phi}\).

ii) Let \(\text{chan}(\theta) \subseteq \text{chan}(FP \backslash \chi)\)
\(\mathfrak{M} \upharpoonright \text{chan}(FP \backslash \chi) = \mathfrak{M}\)
and assume that, for some \(\gamma, (\theta, \mathfrak{M}, \gamma) \models \bar{\phi}\). Consequently, there exist some \(\hat{\theta}\) and \(\hat{\mathfrak{M}}\) such that
\((\hat{\theta}, \hat{\mathfrak{M}}, (\gamma : (s, N) \mapsto (\hat{\theta}, \hat{\mathfrak{M}}))) \models \phi(s/h, N/R] \land \chi[s/h_{\text{old}}, N/R_{\text{old}}]\)

Then, by substitution lemma b) and d), \((\hat{\theta}, \hat{\mathfrak{M}}, (\gamma : (s, N) \mapsto (\hat{\theta}, \hat{\mathfrak{M}}))) \models \phi, and thus, since \(s\) and \(N\) do not occur free in \(\phi, (\hat{\theta}, \hat{\mathfrak{M}}, \gamma) \models \phi\). Since \(\phi\) is precise for \(FP\), we may conclude that \((\hat{\theta}, \hat{\mathfrak{M}}, \gamma) \models \phi(h \upharpoonright \text{chan}(FP))/h, (R \upharpoonright \text{chan}(FP))/R]. Hence, by substitution lemma b) and d),
\((\hat{\theta} \upharpoonright \text{chan}(FP), \hat{\mathfrak{M}} \upharpoonright \text{chan}(FP), \gamma) \models \phi\)

(6).

Trivially, \(\text{chan}(\hat{\theta} \upharpoonright \text{chan}(FP)) \subseteq \text{chan}(FP)\)

(7).

It is also obvious that \((\hat{\mathfrak{M}} \upharpoonright \text{chan}(FP)) \upharpoonright \text{chan}(FP) = \hat{\mathfrak{M}} \upharpoonright \text{chan}(FP)\)

(8).

By results (7), (8), (3), (4), (6), and the fact that \(\phi\) is precise for \(FP\), we may conclude that
\((\hat{\theta} \upharpoonright \text{chan}(FP), \hat{\mathfrak{M}} \upharpoonright \text{chan}(FP), \gamma) \in \mathcal{O}[FP\backslash \chi]\)

(9).

By (5) and the correspondence lemma, for all \(\theta_0\) and \(\mathfrak{M}_0\)
\((\theta_0, \theta, \mathfrak{M}_0, \mathfrak{M}, (\gamma : (s, N) \mapsto (\hat{\theta}, \hat{\mathfrak{M}}))) \models \chi[s/h_{\text{old}}, N/R_{\text{old}}].

(10).

By (9), (10), (7), and (8), \((\theta, \mathfrak{M}, \gamma) \in \mathcal{O}[FP\backslash \chi].\)

iii) Follows from the fact that, since \(\phi\) is precise for \(FP\),
\(\phi \rightarrow \phi(h \upharpoonright \text{chan}(FP))/h, (R \upharpoonright \text{chan}(FP))/R], the fact that, since \(\chi\) is a failure hypothesis,
\(\chi \rightarrow \chi[(h \upharpoonright \text{chan}(FP))/h, (R \upharpoonright \text{chan}(FP))/R], and the fact that \(\text{chan}(FP \backslash \chi) = \text{chan}(FP).\)}
### In this series appeared:

<table>
<thead>
<tr>
<th>Volume</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>91/02</td>
<td>R.P. Nederpelt, H.C.M. de Swart</td>
<td>Implication. A survey of the different logical analyses &quot;if...then...&quot;, p. 26.</td>
</tr>
<tr>
<td>91/03</td>
<td>J.P. Katoen, L.A.M. Schoenmakers</td>
<td>Parallel Programs for the Recognition of P-invariant Segments, p. 16.</td>
</tr>
<tr>
<td>91/05</td>
<td>D. de Reus</td>
<td>An Implementation Model for GOOD, p. 18.</td>
</tr>
<tr>
<td>91/06</td>
<td>K.M. van Hee</td>
<td>SPECIFICATIEMETHODEN, een overzicht, p. 20.</td>
</tr>
<tr>
<td>91/07</td>
<td>E. Poll</td>
<td>CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.</td>
</tr>
<tr>
<td>91/12</td>
<td>E. van der Sluis</td>
<td>A parallel local search algorithm for the travelling salesman problem, p. 12.</td>
</tr>
<tr>
<td>91/14</td>
<td>P. Lemmens</td>
<td>The PDB Hypermedia Package. Why and how it was built, p. 63.</td>
</tr>
<tr>
<td>91/16</td>
<td>A.J.J.M. Marcelis</td>
<td>An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.</td>
</tr>
</tbody>
</table>
91/18 Rik van Geldrop
Transformational Query Solving, p. 35.

91/19 Erik Poll
Some categorical properties for a model for second order
lambda calculus with subtyping, p. 21.

91/20 A.E. Eiben
R.V. Schuwer

91/21 J. Coenen
Assertional Data Reification Proofs: Survey and
W.-P. de Roever
Perspective, p. 18.
J.Zwiers

91/22 G. Wolf

91/23 K.M. van Hee
Z and high level Petri nets, p. 16.
L.J. Somers
M. Voorhoeve

91/24 A.T.M. Aerts
Formal semantics for BRM with examples, p. 25.
D. de Reus

91/25 P. Zhou
A compositional proof system for real-time systems based
J. Hooman on explicit clock temporal logic: soundness and complete
R. Kuiper
teness, p. 52.

91/26 P. de Bra
The GOOD based hypertext reference model, p. 12.
G.J. Houben
J. Paredaens

91/27 F. de Boer
Embedding as a tool for language comparison: On the
C. Palamidessi CSP hierarchy, p. 17.

91/28 F. de Boer
A compositional proof system for dynamic proces
creation, p. 24.

91/29 H. Ten Eikelder
Correctness of Acceptor Schemes for Regular Languages,
R. van Geldrop p. 31.
J.C.M. Baeten
F.W. Vaandrager
An Algebra for Process Creation, p. 29.

91/31 H. ten Eikelder
Some algorithms to decide the equivalence of recursive

91/32 P. Struijk

91/33 W. v.d. Aalst
The modelling and analysis of queueing systems with
QNM-ExSpect, p. 23.

91/34 J. Coenen
Specifying fault tolerant programs in deontic logic,
p. 15.

91/35 F.S. de Boer
Asynchronous communication in process algebra, p. 20.
J.W. Klop
C. Palamidessi
<table>
<thead>
<tr>
<th>Volume</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>92/01</td>
<td>J. Coenen, J. Zwiers, W.-P. de Roever</td>
<td>A note on compositional refinement, p. 27.</td>
</tr>
<tr>
<td>92/02</td>
<td>J. Coenen, J. Hooman</td>
<td>A compositional semantics for fault tolerant real-time systems, p. 18.</td>
</tr>
<tr>
<td>92/03</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Real space process algebra, p. 42.</td>
</tr>
<tr>
<td>92/05</td>
<td>J.P.H.W.v.d.Eijnde</td>
<td>Conservative fixpoint functions on a graph, p. 25.</td>
</tr>
<tr>
<td>92/06</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Discrete time process algebra, p. 45.</td>
</tr>
<tr>
<td>92/07</td>
<td>R.P. Nederpelt</td>
<td>The fine-structure of lambda calculus, p. 110.</td>
</tr>
<tr>
<td>92/10</td>
<td>P.M.P. Rambags</td>
<td>Composition and decomposition in a CPN model, p. 55.</td>
</tr>
<tr>
<td>92/13</td>
<td>F. Kamareddine</td>
<td>Set theory and nominalisation, Part II, p. 22.</td>
</tr>
<tr>
<td>92/14</td>
<td>J.C.M. Baeten</td>
<td>The total order assumption, p. 10.</td>
</tr>
<tr>
<td>92/15</td>
<td>F. Kamareddine</td>
<td>A system at the cross-roads of functional and logic programming, p. 36.</td>
</tr>
<tr>
<td>92/16</td>
<td>R.R. Seljée</td>
<td>Integrity checking in deductive databases; an exposition, p. 32.</td>
</tr>
<tr>
<td>92/17</td>
<td>W.M.P. van der Aalst</td>
<td>Interval timed coloured Petri nets and their analysis, p. 20.</td>
</tr>
<tr>
<td>92/18</td>
<td>R.Nederpelt, F. Kamareddine</td>
<td>A unified approach to Type Theory through a refined lambda-calculus, p. 30.</td>
</tr>
<tr>
<td>92/20</td>
<td>F.Kamareddine</td>
<td>Are Types for Natural Language? P. 32.</td>
</tr>
<tr>
<td>92/21</td>
<td>F.Kamareddine</td>
<td>Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.</td>
</tr>
<tr>
<td>Code</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<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>
</tr>
<tr>
<td>92/23</td>
<td>F. Kamareddine, E. Klein</td>
<td>Nominalization, Predication and Type Containment, p. 40.</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>
</tr>
<tr>
<td>92/25</td>
<td>E. Poll</td>
<td>A Programming Logic for Fō, p. 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, p. 36.</td>
</tr>
<tr>
<td>93/02</td>
<td>T. Verhoeff</td>
<td>A continuous version of the Prisoner's Dilemma, p. 17</td>
</tr>
<tr>
<td>93/03</td>
<td>T. Verhoeff</td>
<td>Quicksort for linked lists, p. 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, p. 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, p. 18.</td>
</tr>
<tr>
<td>93/06</td>
<td>J. P. Velkamp</td>
<td>On the unavoidability of metastable behaviour, p. 29</td>
</tr>
<tr>
<td>93/07</td>
<td>P. D. Moerland</td>
<td>Exercises in Multiprogramming, p. 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, p. 32.</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>
</tr>
</tbody>
</table>
A useful lambda notation, p. 17.
Nominalization, Predication and Type Containment, p. 40.
Bottom-up Abstract Interpretation of Logic Programs, p. 33.
A Programming Logic for For, p. 15.
A modelling method using MOVIE and SimCon/ExSpect, p. 15.
A taxonomy of keyword pattern matching algorithms, p. 50.
Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
A continuous version of the Prisoner's Dilemma, p. 17
Quicksort for linked lists, p. 8.
Deterministic and randomized local search, p. 78.
A congruence theorem for structured operational semantics with predicates, p. 18.
On the unavoidability of metastable behaviour, p. 29
Exercises in Multiprogramming, p. 97
A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
Systems Engineering: a Formal Approach
Systems Engineering: a Formal Approach
Part II: Frameworks, p. 44.
Systems Engineering: a Formal Approach
Systems Engineering: a Formal Approach
Part IV: Analysis Methods, p. 63.
Systems Engineering: a Formal Approach
<table>
<thead>
<tr>
<th>Paper No.</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>93/16</td>
<td>H. Schepers, J. Hooman</td>
<td>A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27</td>
</tr>
<tr>
<td>93/17</td>
<td>D. Alstein, P. van der Stok</td>
<td>Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.</td>
</tr>
<tr>
<td>93/18</td>
<td>C. Verhoef</td>
<td>A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.</td>
</tr>
<tr>
<td>93/19</td>
<td>G.J. Houben</td>
<td>The Design of an Online Help Facility for ExSpect, p. 21.</td>
</tr>
<tr>
<td>93/22</td>
<td>E. Poll</td>
<td>A Typechecker for Bijective Pure Type Systems, p. 28.</td>
</tr>
<tr>
<td>93/23</td>
<td>E. de Kogel</td>
<td>Relational Algebra and Equational Proofs, p. 23.</td>
</tr>
<tr>
<td>93/24</td>
<td>E. Poll and Paula Severi</td>
<td>Pure Type Systems with Definitions.</td>
</tr>
</tbody>
</table>