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

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 $TIME$ be some ordered time domain ($\infty \in TIME$). For the scope of this report it is immaterial whether time domain $TIME$ is discrete, that is, $TIME = \{ u\tau \mid \tau \in \mathbb{N} \}$ for some positive smallest time unit u, dense, that is, $TIME = \{ \tau \in \mathbb{Q} \mid \tau \geq 0 \}$, or continuous, that is, $TIME = \{ \tau \in \mathbb{R} \mid \tau \geq 0 \}$. The syntax of our programming language is given in Table 1, with $n \in \mathbb{N}$, $n \geq 1$, $x \in VAR$, $\mu \in VAL$, $c \in CHAN$, $d \in TIME$, and $cset \subseteq CHAN$.

Table 1: Syntax of the Programming Language

Expression	e	$::=$	$\mu \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2$
Boolean Expression	b	$::=$	$e_1 = e_2 \mid e_1 < e_2 \mid \neg b \mid b_1 \vee b_2$
Guarded Command	G	$::=$	$[\ []_{i=1}^{n}\ b_i \rightarrow P_i\ ] \mid [\ []_{i=1}^{n}\ c_i?x_i \rightarrow P_i\ []\ \textbf{delay}\ d \rightarrow P\ ]$
Process	P	$::=$	$\textbf{skip} \mid x := e \mid c!e \mid c?x \mid P_1;P_2 \mid G \mid *G \mid P_1 \parallel P_2 \mid P \backslash cset$

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

<u>Atomic statements</u>

- **skip** terminates after K_{skip} units of time, where constant $K_{\mathsf{skip}} > 0$.
- 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 $[\ []_{i=1}^{n}\ c_i?x_i \rightarrow P_i\ []\ \textbf{delay}\ d \rightarrow P\ ]$. 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.
- $P \backslash cset$ hides the channels from $cset$.

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

- $var(\mu) = \emptyset$
- $var(x) = \{x\}$
- $var(e_1 + e_2) = var(e_1 - e_2) = var(e_1 \times e_2) = var(e_1 = e_2) = var(e_1 < e_2) = var(e_1) \cup var(e_2)$
- $var(\neg b) = var(b)$
- $var(b_1 \vee b_2) = var(b_1) \cup var(b_2)$
- $var(\textbf{skip}) = \emptyset$
- $var(x := e) = \{x\} \cup var(e)$
- $var(c!e) = var(e)$
- $var(c?x) = \{x\}$
- $var(P_1 \; ; \; P_2) = var(P_1) \cup var(P_2)$
- $var([\ []_{i=1}^{n}\ b_i \rightarrow P_i\ ]) = \cup_{i=1}^{n} var(b_i)\ \cup\ \cup_{i=1}^{n} var(P_i)$
- $var([\ []_{i=1}^{n}\ c_i?x_i \rightarrow P_i\ []\ \textbf{delay}\ d \rightarrow P_0\ ]) = \cup_{i=1}^{n}\{x_i\}\ \cup\ \cup_{i=0}^{n} var(P_i)$
- $var(*G) = var(G)$

- $var(P_1 \,\|\, P_2) = var(P_1) \cup var(P_2)$
- $var(P \setminus cset) = var(P)$ $\Diamond$

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

- $in(\mathbf{skip}) = in(x := e) = in(c!e) = \emptyset$
- $in(c?x) = \{c\}$
- $in(P_1 \,;\, P_2) = in(P_1) \cup in(P_2)$
- $in([\ []_{i=1}^{n}\ b_i \to P_i\ ]) = \cup_{i=1}^{n} in(P_i)$
- $in([\ []_{i=1}^{n}\ c_i?x_i \to P_i\ []\ \mathbf{delay}\ d \to P_0\ ]) = \cup_{i=1}^{n}\{c_i\}\ \cup\ \cup_{i=0}^{n} in(P_i)$
- $in(*G) = in(G)$
- $in(P_1 \,\|\, P_2) = in(P_1) \cup in(P_2)$
- $in(P \setminus cset) = in(P) - cset$ $\Diamond$

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

- $out(\mathbf{skip}) = out(x := e) = \emptyset$
- $out(c!e) = \{c\}$
- $out(c?x) = \emptyset$
- $out(P_1 \,;\, P_2) = out(P_1) \cup out(P_2)$
- $out([\ []_{i=1}^{n}\ b_i \to P_i\ ]) = \cup_{i=1}^{n} out(P_i)$
- $out([\ []_{i=1}^{n}\ c_i?x_i \to P_i\ []\ \mathbf{delay}\ d \to P_0\ ]) = \cup_{i=0}^{n} out(P_i)$
- $out(*G) = out(G)$
- $out(P_1 \,\|\, P_2) = out(P_1) \cup out(P_2)$
- $out(P \setminus cset) = out(P) - cset$ $\Diamond$

Definition 4 (Observable channels of a process) The set of *observable channels* of a process P, notation $chan(P)$, is defined by $chan(P) = in(P) \cup out(P)$. $\Diamond$

2.1 Syntactic Restrictions

To guarantee that channels are unidirectional and point-to-point, we have the following syntactic constraints (for any $n \in \mathbb{N}$, $d \in TIME$, $c_1, \ldots, c_n \in CHAN$, and $x_1, \ldots, x_n \in VAR$):

- For $P_1 \,;\, P_2$ we require that $in(P_1) \cap out(P_2) = \emptyset$ and $out(P_1) \cap in(P_2) = \emptyset$.
- For $[\ []_{i=1}^{n}\ b_i \to P_i\ ]$ we require that, for all $i, j \in \{1, \ldots, n\}$, $i \neq j$, $out(P_i) \cap in(P_j) = \emptyset$.
- For $[\ []_{i=1}^{n}\ c_i?x_i \to P_i\ []\ \mathbf{delay}\ d \to P_0\ ]$ we require that
  - $\cup_{i=1}^{n}\{c_i\}\ \cap\ \cup_{i=0}^{n} out(P_i) = \emptyset$, and,
  - for all $i, j \in \{0, \ldots, n\}$, $i \neq j$, $out(P_i) \cap in(P_j) = \emptyset$.

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

To avoid programs such as $(c?x)\backslash\{c\}$, which would be equivalent to a *random assignment* to x, we require that only internal channels are hidden:

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

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

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

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_{skip}, we assume given a constant K_{a} such that execution of each assignment statement takes K_{a} time units, a constant K_α denoting the overhead preceding a communication, a constant K_ω denoting the overhead following a communication, and a constant K_{g} 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 ($T \notin VAR$) to denote the global time.

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

Thus, besides assigning to each program variable x a value $\sigma(x)$, a state σ 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 $TIME$ and VAL.

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

- $\mathcal{E}[\![\mu]\!](\sigma) = \mu$,
- $\mathcal{E}[\![x]\!](\sigma) = \sigma(x)$,

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

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

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

We represent a synchronous communication of value $\mu \in VAL$ on channel $c \in CHAN$ at time $\tau \in TIME$ by a triple (τ, c, μ), and define

$(Timestamp)$ $ts((\tau, c, \mu)) = \tau$

$(Channel)$ $ch((\tau, c, \mu)) = c$

$(Value)$ $val((\tau, c, \mu)) = \mu$

To denote the observable input and output behaviour of a process P we use a *timed trace* θ which is a possibly infinite sequence of the form $\langle(\tau_1, c_1, \mu_1), (\tau_2, c_2, \mu_2), \ldots\rangle$, 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,

$$\begin{aligned} TRACE = \{\, \theta \in OBS^* \cup OBS^\omega \mid \forall i \cdot \quad & ts(\theta(i)) \leq ts(\theta(i+1)) \\ & \wedge \forall j \cdot ts(\theta(i)) = ts(\theta(j)) \rightarrow ch(\theta(i)) \neq ch(\theta(j)) \,\} \end{aligned}$$

$\diamond$

Let $\langle\rangle$ denote the empty trace, i.e. the sequence of length 0. The concatenation of two traces θ_1 and θ_2 is denoted $\theta_1{}^\wedge\theta_2$ (and equals θ_1 if θ_1 is infinite). We use $first(\theta)$ and, if θ is finite, $last(\theta)$ to refer to the first and last record of θ, 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. If the timed refusal of a process contains (c, τ) then this corresponds to the refusal of the process to participate in a communication on channel c at time τ.

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

$$REF \;=\; \{\, \mathfrak{R} \mid \mathfrak{R} \subseteq CHAN \times [0, \infty) \,\}$$

$\diamond$

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

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

- if $\sigma \neq \perp$ then it represents a terminating computation which starts in state σ_0, performs the communications as described in θ while refusing those in $\mathfrak{R}$, and terminates in state σ, and
- if $\sigma = \perp$ then it represents a computation which starts in state σ_0, performs the communications as described in θ while refusing those in $\mathfrak{R}$, 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 $\theta \in TRACE$ and a set of channels $cset \subseteq CHAN$, we define the *projection* of θ onto $cset$, denoted by $\theta\!\uparrow cset$, as the sequence obtained from θ by deleting all records with channels not in $cset$. Formally,

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

◇

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

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

◇

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

$$\theta \curvearrowleft \tau \;=\; \begin{cases} \langle\rangle & \text{iff } \theta = \langle\rangle \\ (t-\tau,c,v)^\wedge(\theta_0 \curvearrowleft \tau) & \text{iff } \theta = (t,c,v)^\wedge\theta_0 \end{cases}$$

◇

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

$$\mathfrak{R}\!\uparrow cset \;=\; \mathfrak{R} \cap (cset \times [0,\infty))$$

◇

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

$$\mathfrak{R}\backslash cset \;=\; \mathfrak{R} \cap ((CHAN - cset) \times [0,\infty))$$

◇

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

$$\mathfrak{R} \curvearrowleft \tau \;=\; \{\, (c, t-\tau) \mid (c,t) \in \mathfrak{R} \,\}$$

◇

Definition 14 (Variant of a state) The *variant* of a state σ with respect to a variable x and a value ϑ, denoted $(\sigma : x \mapsto \vartheta)$, is given by

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

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

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

- Exection of **skip** terminates after K_{skip} time units, all the while refusing no communication.

  $\mathcal{M}[\![\mathbf{skip}]\!] = \{ \ (\ \sigma_0 \ , \ (\langle\rangle, \emptyset) \ , \ (\sigma_0 : T \mapsto K_{\mathrm{skip}}) \ ) \mid \mathcal{E}[\![T]\!](\sigma_0) = 0 \ \}$

- 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.

  $\mathcal{M}[\![x := e]\!] = \{ \ (\ \sigma_0 \ , \ (\langle\rangle, \emptyset) \ , \ \left(\sigma_0 : \begin{cases} x & \mapsto & \mathcal{E}[\![e]\!](\sigma_0) \\ T & \mapsto & K_{\mathrm{a}} \end{cases}\right) \ ) \mid \mathcal{E}[\![T]\!](\sigma_0) = 0 \ \}$

- In the execution of a synchronous io-statement there comes, after an initial period of K_α 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_ω time units after the c communication has occured.

  $$\begin{aligned}
  \mathcal{M}[\![c!e]\!] = & \\
  & \{ \ (\ \sigma_0 \ , \ (\langle\rangle, \mathfrak{R}) \ , \ \bot \ ) \mid \mathcal{E}[\![T]\!](\sigma_0) = 0 \ \wedge \ \mathfrak{R} = \{c\} \times [0, K_\alpha) \ \} \\
  \cup & \{ \ (\ \sigma_0 \ , \ (\langle(\tau, c, \mathcal{E}[\![e]\!](\sigma_0))\rangle, \mathfrak{R}) \ , \ (\sigma_0 : T \mapsto \tau + K_\omega) \ ) \mid \ \mathcal{E}[\![T]\!](\sigma_0) = 0 \\
  & \qquad \wedge \ \tau \geq K_\alpha \\
  & \qquad \wedge \ \mathfrak{R} = \{c\} \times (\ [0, K_\alpha) \cup (\tau, \infty) \ ) \ \}
  \end{aligned}$$

  Recall that we allow at most one c communication at time $T = \tau$.

- 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.

  $$\begin{aligned}
  \mathcal{M}[\![c?x]\!] = & \\
  & \{ \ (\ \sigma_0 \ , \ (\langle\rangle, \mathfrak{R}) \ , \ \bot \ ) \mid \mathcal{E}[\![T]\!](\sigma_0) = 0 \ \wedge \ \mathfrak{R} = \{c\} \times [0, K_\alpha) \ \} \\
  \cup & \{ \ (\ \sigma_0 \ , \ (\langle(\tau, c, \mu)\rangle, \mathfrak{R}) \ , \ \left(\sigma_0 : \begin{cases} x \mapsto \mu \\ T \mapsto \tau + K_\omega + K_a \end{cases}\right) \ ) \mid \ \mathcal{E}[\![T]\!](\sigma_0) = 0 \\
  & \qquad \wedge \ \tau \geq K_\alpha \\
  & \qquad \wedge \ \mu \in \mathit{Val} \\
  & \qquad \wedge \ \mathfrak{R} = \{c\} \times (\ [0, K_\alpha) \cup (\tau, \infty) \ ) \ \}
  \end{aligned}$$

- 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.

$$\mathcal{M}[\![P_1\, ;\, P_2]\!] =$$

$$\begin{aligned}
&\{\ (\ \sigma_0, (\theta, \mathfrak{R} \cup (chan(P_2) - chan(P_1)) \times [0,\infty)), \perp\ )\ \mid\ (\ \sigma_0, (\theta, \mathfrak{R}), \perp\ ) \in \mathcal{M}[\![P_1]\!]\ \}\\
\cup\ &\{\ (\ \sigma_0, (\theta_1{}^\wedge\theta_2, \mathfrak{R}), \sigma\ )\\
&\quad \mid\\
&\quad \text{there exist a } \mathfrak{R}_1, \text{ a } \mathfrak{R}_2, \text{ a } \sigma_1 \neq \perp \text{ and a } \tau > 0 \text{ such that } \mathcal{E}[\![T]\!](\sigma_1) = \tau,\\
&\quad (\ \sigma_0\ ,\ (\ \theta_1\ ,\ \mathfrak{R}_1\ )\ ,\ \sigma_1\ ) \in \mathcal{M}[\![P_1]\!],\\
&\quad (\ (\sigma_1 : T \mapsto 0)\ ,\ (\ \theta_2\ ,\ \mathfrak{R}_2\ ) \curvearrowleft \tau\ ,\ (\sigma : T \mapsto T - \tau)\ ) \in \mathcal{M}[\![P_2]\!],\\
&\quad \text{and } \mathfrak{R} = \mathfrak{R}_1 \cup (chan(P_2) - chan(P_1)) \times [0,\tau) \cup \mathfrak{R}_2 \cup (chan(P_1) - chan(P_2)) \times [\tau, \infty)\ \}
\end{aligned}$$

where $(\theta, \mathfrak{R}) \curvearrowleft t$ equals $(\theta \curvearrowleft t, \mathfrak{R} \curvearrowleft t)$.

- 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 $\cup_i chan(P_i)$ are refused.

$$\mathcal{M}[\![\ [\ \|_{i=1}^{n}\ b_i \rightarrow P_i\ ]\ ]\!] =$$

$$\begin{aligned}
&\{\ (\sigma_0\ ,\ (\langle\rangle, \cup_i chan(P_i) \times [0,\infty))\ ,\ (\sigma_0 : T \mapsto K_g))\ \mid \mathcal{E}[\![T]\!](\sigma_0) = 0\ \wedge\ \neg\mathcal{B}[\![b_1 \vee \ldots \vee b_n]\!](\sigma_0)\ \}\\
\cup\ &\{\ (\ \sigma_0, (\theta, \mathfrak{R}), \sigma\ )\ \mid \mathcal{E}[\![T]\!](\sigma_0) = 0, \text{ and there exist a } k \in \{1, \ldots, n\} \text{ and a } \widehat{\mathfrak{R}} \text{ such that}\\
&\qquad \mathcal{B}[\![b_k]\!](\sigma_0),\ (\ \sigma_0\ ,\ (\theta, \widehat{\mathfrak{R}}) \curvearrowleft K_g\ ,\ (\sigma : T \mapsto T - K_g)\ ) \in \mathcal{M}[\![P_k]\!], \text{ and}\\
&\qquad \mathfrak{R} = \cup_i chan(P_i) \times [0, K_g]\ \cup\ \widehat{\mathfrak{R}}\ \cup\ (\cup_i chan(P_i) - chan(P_k)) \times [K_g, \infty)\ \}
\end{aligned}$$

- 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.

$$\mathcal{M}[\![\ [\ \|_{i=1}^{n}\ c_i?x \rightarrow P_i\ \|\ \mathbf{delay}\ d \rightarrow P\ ]\ ]\!] =$$

$$\begin{aligned}
\cup_i\ &\{\ (\ \sigma_0, (\langle(\tau, c_i, v)\rangle^\wedge\theta, \mathfrak{R}), \sigma\ )\\
&\quad \mid\\
&\quad \mathcal{E}[\![T]\!](\sigma_0) = 0,\ K_\alpha \leq \tau < d,\ v \in VAL, \text{ and there exists a } \widehat{\mathfrak{R}} \text{ such that}\\
&\quad \mathfrak{R}\ =\ ((\cup_j chan(P_j) \cup chan(P)) - \cup_j\{c_j\}) \times [0,\tau] - \{(c_i, \tau)\}\\
&\qquad\quad \cup\widehat{\mathfrak{R}}\\
&\qquad\quad \cup((\cup_j chan(P_j) \cup chan(P)) - chan(P_i)) \times [\tau, \infty)\ ,\\
&\quad \text{and } (\ \sigma_0\ ,\ (\theta, \widehat{\mathfrak{R}}) \curvearrowleft (\tau + K_\omega + K_a)\ ,\ (\sigma : T \mapsto T - \tau - K_\omega + K_a)\ ) \in \mathcal{M}[\![P_i]\!]\ \}\\
\cup\ &\{\ (\ \sigma_0, (\theta, \mathfrak{R}), \sigma\ )\\
&\quad \mid\\
&\quad \mathcal{E}[\![T]\!](\sigma_0) = 0, \text{ and there is a } \widehat{\mathfrak{R}} \text{ with } (\sigma_0\ ,\ (\theta, \widehat{\mathfrak{R}}) \curvearrowleft d\ ,\ (\sigma : T \mapsto T - d)) \in \mathcal{M}[\![P]\!], \text{ and}\\
&\quad \mathfrak{R} = ((\cup_j chan(P_j) \cup chan(P)) - \cup_j\{c_j\}) \times [0,d] \cup \widehat{\mathfrak{R}} \cup (\cup_j chan(P_j) - chan(P)) \times [d, \infty)\ \}
\end{aligned}$$

- An execution of $*G$ consists of either an infite 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.

$\mathcal{M}[\![* G]\!] =$

$\{ \ (\ \sigma_0, (\theta, \mathfrak{R}), \sigma \ ) \mid \mathcal{E}[\![T]\!](\sigma_0) = 0$ and there exists a $k \in \mathbb{N} \cup \{\infty\}$, and for every i, $0 \leq i < k$,
there exists a triple $(\sigma_i \ , \ (\theta_{i+1}, \mathfrak{R}_{i+1}) \ , \ \sigma_{i+1})$ such that
$\sigma_i \neq \bot$,
$\mathcal{B}[\![b_G]\!](\sigma_i)$,
$(\ (\sigma_i : T \mapsto 0) \ ,$
$(\theta_{i+1}, \mathfrak{R}_{i+1}) \curvearrowleft \mathcal{E}[\![T]\!](\sigma_i) \ ,$
$(\sigma_{i+1} : T \mapsto T - \mathcal{E}[\![T]\!](\sigma_i)) \ ) \in \mathcal{M}[\![G]\!]$, and
if $k = \infty$ then
for all j, $1 \leq j < k$, $\theta_1 {}^\wedge \ldots {}^\wedge \theta_j \ \preceq \ \theta$ and $\bigcap_{l=1}^{j} \mathfrak{R}_l \ \supseteq \ \mathfrak{R}$, and $\sigma = \bot$,
else if $k < \infty$ then
$\theta \ = \ \theta_1 {}^\wedge \ldots {}^\wedge \theta_k \ , \mathfrak{R} \ = \ \bigcap_{l=1}^{k} \mathfrak{R}_l \ , \sigma = \sigma_k$, and if $\sigma_k \neq \bot$ then $\mathcal{B}[\![\neg b_G]\!](\sigma_k) \ \}$

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

$\mathcal{M}[\![P_1 \parallel P_2]\!] = \{ \ (\ \sigma_0, (\theta, \mathfrak{R}), \sigma \ ) \mid$ for $i = 1, 2$ there exist $(\theta_i, \mathfrak{R}_i), \sigma_i$ such that
$(\sigma_0, (\theta_i, \mathfrak{R}_i), \sigma_i) \in \mathcal{M}[\![P_i]\!]$,
and if $\sigma_1 = \bot$ or $\sigma_2 = \bot$ then $\sigma = \bot$ else, for all $x \in VAR$,
$\sigma(x) = \begin{cases} \sigma_i(x) \text{if } x \in var(P_i) \\ \sigma_0(x) \text{if } x \notin var(P_1 \parallel P_2) \end{cases}$, $\sigma(T) = \mathbf{max}_i(\sigma_i(T))$,
$\theta \uparrow chan(P_i) = \theta_i$, $\theta \uparrow chan(P_1 \parallel P_2) = \theta$, and $\mathfrak{R} = \mathfrak{R}_1 \cup \mathfrak{R}_2 \quad \}$

- The observations of $P \backslash cset$, where $cset \subseteq in(P) \cap out(P)$, are characterized by the fact that the internal $cset$ 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 $P \backslash cset$ is characterized by the fact that $cset$ communications are continuously refused, except on single instants.

Definition 15 (As soon as possible) For a timed refusal set $\mathfrak{R}$ and a set $cset$ of channels:

$$ASAP(\mathfrak{R}, cset) \equiv \forall c \in cset \cdot \forall t_1, t_2 \cdot \{c\} \times [t_1, t_2] \ \cap \ \mathfrak{R} \ = \ \emptyset \ \rightarrow \ t_1 = t_2$$

$\diamond$

Then,

$\mathcal{M}[\![P \backslash cset]\!] = \{ \ (\ \sigma_0 \ , \ (\theta \backslash cset \ , \ \mathfrak{R} \backslash cset) \ , \ \sigma \ ) \mid (\ \sigma_0 \ , \ (\theta \ , \ \mathfrak{R}) \ , \ \sigma \ ) \in \mathcal{M}[\![P]\!] \ \wedge \ ASAP(\mathfrak{R}, cset) \ \}$

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:

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

$\diamond$

The set $\mathcal{O}[\![P]\!]$ represents the normal behaviour of process P. In Section 5 we determine the set $\mathcal{O}[\![P \wr \chi]\!]$ representing the acceptable behaviour of P under the assumption of failure hypothesis χ. Besides the already mentioned finite variability, other important properties of the semantic function $\mathcal{O}$ are that if $(\theta, \mathfrak{R}) \in \mathcal{O}[\![P]\!]$ then $\theta \uparrow chan(P) = \theta$ and $\mathfrak{R} \uparrow chan(P) = \mathfrak{R}$.

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 $\mathcal{O}$. In this report we specify the relation between a program P and an assertion ϕ by means of a so-called correctness formula of the form P **sat** ϕ. Intuitively, such a formula expresses that all executions of P satisfy ϕ.

Similar to the semantic denotation of traces in the previous section, we use record expressions such as (τ, c, μ), with $\tau \in \mathit{TIME}$, $c \in \mathit{CHAN}$, and $\mu \in \mathit{VAL}$, in assertions. We use instant expressions, e.g. using the function ts to obtain the timestamp of a record. We have channel expressions, e.g. using the operator ch which yields the channel of a record, and value expressions, including the operator val which yields the value of a communication record. We use the empty trace, $\langle\rangle$, traces of one record, e.g. $\langle(\tau, c, \mu)\rangle$, as well as the concatenation operator $^\wedge$ and the projection operator $\uparrow$ to create trace expressions. Further, for a trace expression $texp$ and a value expression $vexp$ we have $texp(vexp)$ to refer a particular record of $texp$, provided $vexp$ is a positive natural number less than or equal to the length of trace $texp$. We use expressions such as $cset \times [\tau_1, \tau_2)$ 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 $c!2$ **sat** $h\uparrow\{c\} = \langle\rangle \vee \exists t \geq 0 \cdot h\uparrow\{c\} = \langle(t, c, 2)\rangle$. To reason about natural numbers, the assertion language includes, for value expression $vexp$, the predicate $vexp \in \mathbb{N}$ which is true if, and only if, the value of value expression $vexp$ 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 $\forall i \cdot i \in \mathbb{N} \rightarrow \phi$. For an assertion ϕ we also write $\phi(h, R)$ to indicate that ϕ has two free variables h and R. We use $\phi(texp, rfxp)$ to denote the assertion which is obtained from ϕ by replacing h by trace expression $texp$, and R by refusal expression $rfxp$. Let IVAR, with typical representative t, denote the set of logical time variables ranging over TIME, let VVAR, with typical representative v, denote the set of logical value variables ranging over VAL, let TVAR, with characteristic element s, be the set of logical trace variables ranging over TRACE, and let RVAR, with typical element N, be the set of logical refusal variables ranging over REF.

Table 2 presents the language we use to define assertions, with $\tau \in \mathit{TIME}$, $t \in \mathit{IVAR}$, $c \in \mathit{CHAN}$, $\mu \in \mathit{VAL}$, $v \in \mathit{VVAR}$, $s \in \mathit{TVAR}$, $N \in \mathit{RVAR}$, and $cset \subseteq \mathit{CHAN}$. Observe that an expression in the assertion language of Table 2 does not refer to program variables since we abstract from the internal state of a process in this report.

Table 2: Syntax of the Assertion Language

Instant expression	$iexp$	::=	$\tau \mid t \mid ts(rexp) \mid iexp_1 + iexp_2$
Channel expression	$cexp$	::=	$c \mid ch(rexp)$
Value expression	$vexp$	::=	$\mu \mid v \mid val(rexp) \mid len(texp)$
Record expression	$rexp$	::=	$(iexp, cexp, vexp) \mid texp(vexp)$
Trace expression	$texp$	::=	$s \mid h \mid \langle\rangle \mid \langle rexp\rangle \mid texp_1{}^\wedge texp_2 \mid texp\uparrow cset$
Interval expression	$inxp$	::=	$[iexp_1, iexp_2) \mid \{iexp\}$
Refusal expression	$rfxp$	::=	$N \mid R \mid \emptyset \mid cset \times inxp \mid rfxp_1 \cup rfxp_2 \mid rfxp\uparrow cset$
Assertion	ϕ	::=	$iexp_1 = iexp_2 \mid cexp_1 = cexp_2 \mid vexp_1 = vexp_2 \mid vexp \in \mathbb{N} \mid$ $texp_1 = texp_2 \mid rfxp_1 = rfxp_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid$ $\exists t \cdot \phi \mid \exists v \cdot \phi \mid \exists s \cdot \phi \mid \exists N \cdot \phi$

Definition 17 (Abbreviation) For record expression $rexp$ and trace expression $texp$, $rexp \in texp$ iff there exists an i such that $texp(i) = rexp$. $\diamond$

Furthermore, we use the standard abbreviations $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$, $\phi_1 \rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$, 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* $\equiv$
  $\forall \hat{t} \cdot ($ **refuse** *cset* **upto** $\hat{t} \leftrightarrow \hat{t} \leq iexp\ )$
- **after** $iexp : \phi \equiv \phi[t + iexp/t]$
  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 $(\theta, \Re) \in \mathcal{O}[\![P]\!]$. If P did not refuse a c communication at time t, that is, $(c, t) \notin \Re$, then the fact that no c communication occurred at t, that is, $\neg(\exists v \cdot (t, c, v) \in \theta)$, 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 $Match(h, R, N)$, iff

$$\begin{array}{l} \forall c, t \cdot (\ (c, t) \notin R \ \wedge\ \neg(\exists v \cdot (t, c, v) \in h)\ ) \rightarrow (c, t) \in N \\ \wedge\ \forall c, t, v \cdot (t, c, v) \in h \rightarrow (c, t) \notin N \end{array}$$

◇

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* $\equiv$
  $\forall N \cdot Match(h, R, N) \rightarrow (cset \times iexp) \cap N = \emptyset$
- *cset* **refused upto** *iexp* $\equiv$
  $\forall N \cdot Match(h, R, N) \rightarrow cset \times [t, t + iexp) \subseteq N$
- *cset* **refused precisely upto** *iexp* $\equiv$
  $\forall \hat{t} \cdot ($ *cset* **refused upto** $\hat{t} \leftrightarrow \hat{t} \leq iexp\ )$

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 ε time units. We specify C as follows:

$$\begin{array}{ll}
C \textbf{ sat} & \forall i \cdot 1 \leq i \leq len(h \uparrow out) \rightarrow val(h \uparrow out(i)) = f(val(h \uparrow in(i))) \\
& \wedge\ h = \langle\rangle \rightarrow \textbf{enable } in \textbf{ and refuse } out \textbf{ upto } \infty \\
& \wedge\ \forall t, v \cdot (t, in, v) \in h \rightarrow \\
& \qquad\qquad \textbf{refuse } \{in, out\} \textbf{ upto } K_C \\
& \qquad\quad \wedge\ \textbf{after } K_C : \forall \hat{t} \cdot out \textbf{ refused precisely upto } \hat{t} \\
& \qquad\qquad\qquad\qquad\qquad\qquad \rightarrow \textbf{enable } out \textbf{ and refuse } in \textbf{ for } \hat{t} \\
& \wedge\ \forall t, v \cdot (t, out, v) \in h \rightarrow \\
& \qquad\qquad \textbf{refuse } \{in, out\} \textbf{ upto } \varepsilon \\
& \qquad\quad \wedge\ \textbf{after } \varepsilon : \forall \hat{t} \cdot in \textbf{ refused precisely upto } \hat{t} \\
& \qquad\qquad\qquad\qquad\qquad\qquad \rightarrow \textbf{enable } in \textbf{ and refuse } out \textbf{ for } \hat{t}
\end{array}$$

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 ϕ we define the set $chan(\phi)$ of channels such that $c \in chan(\phi)$ if, and only if, a communication along c might affect the validity of ϕ. For instance, the validity of assertion $h = \langle\rangle$ is affected by any communication and thus we should have $chan(h = \langle\rangle) = CHAN$. 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 $R \uparrow \{c\} = \emptyset$, like assertion $R \uparrow \{c\} = \{c\} \times [0, \infty)$, is invalidated by a communication along c, and by a communication along c only. On the other hand, also the validity of assertion $(h \uparrow \{c\})^\wedge(5, d, 7) = \langle(5, d, 7)\rangle$ can only be changed by a communication along channel c, although d occurs in the assertion as well. Hence, $chan(\phi)$ consists of the channels to which references to h and R in ϕ are restricted rather than the channels occurring syntactically in ϕ. Note that the value of a logical variable is not affected by any communication.

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

- $chan(\tau) = chan(t) = \emptyset$
- $chan(ts(rexp)) = chan(rexp)$
- $chan(iexp_1 + iexp_2) = chan(iexp_1) \cup chan(iexp_2)$
- $chan(c) = \emptyset$
- $chan(ch(rexp)) = chan(rexp)$
- $chan(\mu) = chan(v) = \emptyset$
- $chan(val(rexp)) = chan(rexp)$
- $chan(len(texp)) = chan(texp)$
- $chan((iexp, cexp, vexp)) = chan(iexp) \cup chan(cexp) \cup chan(vexp)$
- $chan(texp(vexp)) = chan(texp) \cup chan(vexp)$
- $chan(s) = \emptyset$
- $chan(h) = CHAN$

- $chan(\langle\rangle) = \emptyset$
- $chan(\langle rexp \rangle) = chan(rexp)$
- $chan(texp_1 \hat{} texp_2) = chan(texp_1) \cup chan(texp_2)$
- $chan(texp \uparrow cset) = chan(texp) \cap cset$
- $chan([iexp_1, iexp_1)) = chan(iexp_1) \cup chan(iexp_2)$
- $chan(\{iexp\}) = chan(iexp)$
- $chan(N) = \emptyset$
- $chan(R) = CHAN$
- $chan(\emptyset) = \emptyset$
- $chan(cset \times inxp) = chan(inxp)$
- $chan(rfxp_1 \cup rfxp_2) = chan(rfxp_1) \cup chan(rfxp_2)$
- $chan(rfxp \uparrow cset) = chan(rfxp) \cap cset$
- $chan(iexp_1 = iexp_2) = chan(iexp_1) \cup chan(iexp_2)$
- $chan(cexp_1 = cexp_2) = chan(cexp_1) \cup chan(cexp_2)$
- $chan(vexp_1 = vexp_2) = chan(vexp_1) \cup chan(vexp_2)$
- $chan(vexp \in \mathbb{N}) = chan(vexp)$
- $chan(texp_1 = texp_2) = chan(texp_1) \cup chan(texp_2)$
- $chan(rfxp_1 = rfxp_2) = chan(rfxp_1) \cup chan(rfxp_2)$
- $chan(\phi_1 \wedge \phi_2) = chan(\phi_1) \cup chan(\phi_2)$
- $chan(\neg\phi) = chan(\exists t \cdot \phi) = chan(\exists v \cdot \phi) = chan(\exists s \cdot \phi) = chan(\exists N \cdot \phi) = chan(\phi)$ $\diamond$

Next we define the meaning of assertions. We use an environment γ to interpret the logical variables of $IVAR \cup VVAR \cup TVAR \cup RVAR$. 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 $(\theta, \mathfrak{R}, \gamma)$. Trace θ gives h its value, refusal set $\mathfrak{R}$ gives R its value, and, as said before, environment γ interprets the logical variables of $IVAR \cup VVAR \cup TVAR \cup RVAR$. We use the special symbol $\dagger$ to deal with the interpretation of $texp(vexp)$ where index $vexp$ is not a positive natural number, or if it is greater than the length of $texp$. The value of an expression is undefined whenever a subexpression yields $\dagger$. We define the value of an instant expression $iexp$ in the trace θ, refusal $\mathfrak{R}$, and an environment γ, denoted by $\mathcal{I}[\![iexp]\!](\theta, \mathfrak{R}, \gamma)$, yielding a value in $TIME \cup \{\dagger\}$, the value of a channel expression $cexp$ in the trace θ, refusal $\mathfrak{R}$, and an environment γ, denoted by $\mathcal{C}[\![cexp]\!](\theta, \mathfrak{R}, \gamma)$, yielding a value in $CHAN \cup \{\dagger\}$, the value of a value expression $vexp$ in the trace θ, refusal $\mathfrak{R}$, and an environment γ, denoted by $\mathcal{V}[\![vexp]\!](\theta, \mathfrak{R}, \gamma)$, yielding a value in $VAL \cup \{\dagger\}$, the value of a record expression $rexp$ in the trace θ, refusal $\mathfrak{R}$, and an environment γ, denoted by $\mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma)$, yielding a value in $(CHAN \times VAL) \cup \{\dagger\}$, the value of a trace expression $texp$ for trace θ, refusal $\mathfrak{R}$, and an environment γ, denoted by $\mathcal{T}[\![texp]\!](\theta, \mathfrak{R}, \gamma)$, yielding a value in $TRACE \cup \{\dagger\}$, the value of an interval expression $inxp$ for trace θ, refusal $\mathfrak{R}$, and an environment γ, denoted by $\mathcal{IN}[\![inxp]\!](\theta, \mathfrak{R}, \gamma)$, yielding a value in $\mathcal{P}(TIME) \cup \{\dagger\}$, and the value of a refusal expression $rfxp$ for trace θ, refusal $\mathfrak{R}$, and an environment γ, denoted by $\mathcal{RF}[\![rfxp]\!](\theta, \mathfrak{R}, \gamma)$, yielding a value in $REF \cup \{\dagger\}$,

- $\mathcal{I}[\![\tau]\!](\theta, \mathfrak{R}, \gamma) = \tau$
- $\mathcal{I}[\![t]\!](\theta, \mathfrak{R}, \gamma) = \gamma(t)$
- $\mathcal{I}[\![ts(rexp)]\!](\theta, \mathfrak{R}, \gamma) = \begin{cases} \dagger & \text{iff } \mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma) = \dagger \\ \tau & \text{iff there exist } c \text{ and } \mu \text{ such that } \mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma) = (\tau, c, \mu) \end{cases}$
- $\mathcal{I}[\![iexp_1 + iexp_2]\!](\theta, \mathfrak{R}, \gamma) = \mathcal{I}[\![iexp_1]\!](\theta, \mathfrak{R}, \gamma) + \mathcal{I}[\![iexp_2]\!](\theta, \mathfrak{R}, \gamma)$
- $\mathcal{C}[\![c]\!](\theta, \mathfrak{R}, \gamma) = c$
- $\mathcal{C}[\![ch(rexp)]\!](\theta, \mathfrak{R}, \gamma) = \begin{cases} \dagger & \text{iff } \mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma) = \dagger \\ c & \text{iff there exist } \tau \text{ and } \mu \text{ such that } \mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma) = (\tau, c, \mu) \end{cases}$
- $\mathcal{V}[\![\mu]\!](\theta, \mathfrak{R}, \gamma) = \mu$
- $\mathcal{V}[\![v]\!](\theta, \mathfrak{R}, \gamma) = \gamma(v)$
- $\mathcal{V}[\![val(rexp)]\!](\theta, \mathfrak{R}, \gamma) = \begin{cases} \dagger & \text{iff } \mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma) = \dagger \\ \mu & \text{iff there exist } \tau \text{ and } c \text{ such that } \mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma) = (\tau, c, \mu) \end{cases}$
- $\mathcal{V}[\![len(texp)]\!](\theta, \mathfrak{R}, \gamma) = \begin{cases} \dagger & \text{iff } \mathcal{T}[\![texp]\!](\theta, \mathfrak{R}, \gamma) = \dagger \\ len(\mathcal{T}[\![texp]\!](\theta, \mathfrak{R}, \gamma)) & \text{otherwise} \end{cases}$
- $\mathcal{R}[\![(cexp, vexp)]\!](\theta, \mathfrak{R}, \gamma) =$
  $\begin{cases} \dagger & \text{iff } \mathcal{C}[\![cexp]\!](\theta, \mathfrak{R}, \gamma) = \dagger \text{ or } \mathcal{V}[\![vexp]\!](\theta, \mathfrak{R}, \gamma) = \dagger \\ (\mathcal{C}[\![cexp]\!](\theta, \mathfrak{R}, \gamma), \mathcal{V}[\![vexp]\!](\theta, \mathfrak{R}, \gamma)) & \text{otherwise} \end{cases}$
- $\mathcal{R}[\![texp(vexp)]\!](\theta, \mathfrak{R}, \gamma) =$
  $\begin{cases} (\tau, c, \mu) & \text{iff there exist } \theta_1 \text{ and } \theta_2 \text{ such that } len(\theta_1) = \mathcal{V}[\![vexp]\!](\theta, \mathfrak{R}, \gamma) - 1 \\ & \text{and } \mathcal{T}[\![texp]\!](\theta, \mathfrak{R}, \gamma) = \theta_1 {}^\wedge (\tau, c, \mu) {}^\wedge \theta_2 \\ \dagger & \text{otherwise} \end{cases}$
- $\mathcal{T}[\![s]\!](\theta, \mathfrak{R}, \gamma) = \gamma(s)$
- $\mathcal{T}[\![h]\!](\theta, \mathfrak{R}, \gamma) = \theta$
- $\mathcal{T}[\![\langle\rangle]\!](\theta, \mathfrak{R}, \gamma) = \langle\rangle$
- $\mathcal{T}[\![\langle rexp\rangle]\!](\theta, \mathfrak{R}, \gamma) = \begin{cases} \dagger & \text{iff } \mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma) = \dagger \\ \langle(c, \mu)\rangle & \text{iff } \mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma) = (c, \mu) \end{cases}$
- $\mathcal{T}[\![texp_1 {}^\wedge texp_2]\!](\theta, \mathfrak{R}, \gamma) =$
  $\begin{cases} \dagger & \text{iff } \mathcal{T}[\![texp_1]\!](\theta, \mathfrak{R}, \gamma) = \dagger \text{ or } \mathcal{T}[\![texp_2]\!](\theta, \mathfrak{R}, \gamma) = \dagger \\ \mathcal{T}[\![texp_1]\!](\theta, \mathfrak{R}, \gamma) {}^\wedge \mathcal{T}[\![texp_2]\!](\theta, \mathfrak{R}, \gamma) & \text{otherwise} \end{cases}$
- $\mathcal{T}[\![texp \upharpoonright cset]\!](\theta, \mathfrak{R}, \gamma) = \begin{cases} \dagger & \text{iff } \mathcal{T}[\![texp]\!](\theta, \mathfrak{R}, \gamma) = \dagger \\ \mathcal{T}[\![texp]\!](\theta \upharpoonright cset, \mathfrak{R}, \gamma) \upharpoonright cset & \text{otherwise} \end{cases}$
- $\mathcal{IN}[\![[iexp_1, iexp_2)]\!](\theta, \mathfrak{R}, \gamma) = [\mathcal{I}[\![iexp_1]\!](\theta, \mathfrak{R}, \gamma), \mathcal{I}[\![iexp_2]\!](\theta, \mathfrak{R}, \gamma))$
- $\mathcal{IN}[\![\{iexp\}]\!](\theta, \mathfrak{R}, \gamma) = \{\mathcal{I}[\![iexp]\!](\theta, \mathfrak{R}, \gamma)\}$
- $\mathcal{RF}[\![N]\!](\theta, \mathfrak{R}, \gamma) = \gamma(N)$
- $\mathcal{RF}[\![R]\!](\theta, \mathfrak{R}, \gamma) = \mathfrak{R}$
- $\mathcal{RF}[\![\emptyset]\!](\theta, \mathfrak{R}, \gamma) = \emptyset$

- $\mathcal{RF}[\![cset \times inxp]\!](\theta, \mathfrak{R}, \gamma) = cset \;\times\; \mathcal{IN}[\![inxp]\!](\theta, \mathfrak{R}, \gamma)$
- $\mathcal{RF}[\![rfxp_1 \cup rfxp_2]\!](\theta, \mathfrak{R}, \gamma) = \mathcal{RF}[\![rfxp_1]\!](\theta, \mathfrak{R}, \gamma) \;\cup\; \mathcal{RF}[\![rfxp_2]\!](\theta, \mathfrak{R}, \gamma)$
- $\mathcal{RF}[\![rfxp \uparrow cset]\!](\theta, \mathfrak{R}, \gamma) = \mathcal{RF}[\![rfxp]\!](\theta, \mathfrak{R} \uparrow cset, \gamma) \uparrow cset$

Definition 22 (Variant of an environment) The *variant* of an environment γ 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 ϕ holds for trace θ, refusal $\mathfrak{R}$, and an environment γ, denoted by $(\theta, \mathfrak{R}, \gamma) \models \phi$. To avoid the complexity of a three-valued logic, an equality predicate is interpreted strictly with respect to $\dagger$, that is, it is false if it contains some expression that has an undefined value.

- $(\theta, \mathfrak{R}, \gamma) \models vexp_1 = vexp_2$ iff
  $\mathcal{V}[\![vexp_1]\!](\theta, \mathfrak{R}, \gamma) = \mathcal{V}[\![vexp_2]\!](\theta, \mathfrak{R}, \gamma)$ and $\mathcal{V}[\![vexp_1]\!](\theta, \mathfrak{R}, \gamma) \neq \dagger$
- $(\theta, \mathfrak{R}, \gamma) \models cexp_1 = cexp_2$ iff
  $\mathcal{C}[\![cexp_1]\!](\theta, \mathfrak{R}, \gamma) = \mathcal{C}[\![cexp_2]\!](\theta, \mathfrak{R}, \gamma)$ and $\mathcal{C}[\![cexp_1]\!](\theta, \mathfrak{R}, \gamma) \neq \dagger$
- $(\theta, \mathfrak{R}, \gamma) \models texp_1 = texp_2$ iff
  $\mathcal{T}[\![texp_1]\!](\theta, \mathfrak{R}, \gamma) = \mathcal{T}[\![texp_2]\!](\theta, \mathfrak{R}, \gamma)$ and $\mathcal{T}[\![texp_1]\!](\theta, \mathfrak{R}, \gamma) \neq \dagger$
- $(\theta, \mathfrak{R}, \gamma) \models \phi_1 \wedge \phi_2$ iff $(\theta, \mathfrak{R}, \gamma) \models \phi_1$ and $(\theta, \mathfrak{R}, \gamma) \models \phi_2$
- $(\theta, \mathfrak{R}, \gamma) \models \neg\phi$ iff not $(\theta, \mathfrak{R}, \gamma) \models \phi$
- $(\theta, \mathfrak{R}, \gamma) \models \exists v \cdot \phi$ iff there exists a value μ such that $(\theta, \mathfrak{R}, (\gamma : v \mapsto \mu)) \models \phi$
- $(\theta, \mathfrak{R}, \gamma) \models \exists s \cdot \phi$ iff there exists a trace $\widehat{\theta}$ such that $(\theta, \mathfrak{R}, (\gamma : s \mapsto \widehat{\theta})) \models \phi$
- $(\theta, \mathfrak{R}, \gamma) \models \exists N \cdot \phi$ iff there exists a refusal $\widehat{\mathfrak{R}}$ such that $(\theta, \mathfrak{R}, (\gamma : N \mapsto \widehat{\mathfrak{R}})) \models \phi$

Example 2 (Satisfaction) In Example 1 we came across assertion

$$\forall t, v \cdot (t, in, v) \in h \;\rightarrow\; \textbf{refuse}\ \{in, out\}\ \textbf{upto}\ K_C$$

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 $(\theta, \mathfrak{R}, \gamma)$ if, and only if, for any instant τ and value μ we have, for environment $\widehat{\gamma} = (\gamma : t \mapsto \tau, v \mapsto \mu)$ which gives logical variables t and v the value of τ and μ respectively,

$$(\theta, \mathfrak{R}, \widehat{\gamma}) \models (t, in, v) \in h \;\rightarrow\; \{in, out\} \times [t, t + K_C) \subseteq R$$

Since h and R obtain their value from θ and $\mathfrak{R}$, respectively, this implication holds for those traces θ and refusals $\mathfrak{R}$ such that if θ contains a record (τ, in, μ) then $\mathfrak{R}$ contains $\{in, out\} \times [\tau, \tau + K_C)$. △

Definition 23 (Validity of an assertion) An assertion is *valid*, notation $\models \phi$, iff for all θ, $\mathfrak{R}$, and γ, $(\theta, \mathfrak{R}, \gamma) \models \phi$. ◇

As mentioned before, we use a correctness formula P **sat** ϕ to express that process P satisfies property ϕ. 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 ϕ.

Definition 24 (Validity of a correctness formula) For process P and assertion ϕ correctness formula P **sat** ϕ is *valid*, notation $\models P$ **sat** ϕ, iff for all γ, and all $(\theta, \mathfrak{R}) \in \mathcal{O}[\![P]\!]$, $(\theta, \mathfrak{R}, \gamma) \models \phi$. ◇

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 χ of process P is formalized as a predicate whose only free variables are h, h_{old}, R, and 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 χ. 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 representative ψ. To indicate that transformation expression ψ 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 ψ 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 $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma)$. Trace θ_0 gives h_{old} its value, refusal $\mathfrak{R}_0$ does so for R_{old}, and, in conformity with the foregoing, trace θ and refusal $\mathfrak{R}$ give h and R their value, and environment γ 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

- $\mathcal{T}[\![h_{old}]\!](\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) = \theta_0$
- $\mathcal{RF}[\![R_{old}]\!](\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) = \mathfrak{R}_0$

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

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

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

Lemma 1 (Correspondence) For assertion ϕ it is for all θ_0 and $\mathfrak{R}_0$ the case that $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \phi$ iff $(\theta, \mathfrak{R}, \gamma) \models \phi$. ○

Definition 25 (Failure hypothesis) A *failure hypothesis* χ 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 $chan(FP)$, that is, $\models \chi \rightarrow \chi(h_{old}\!\uparrow\! chan(FP), h\!\uparrow\! chan(FP), R_{old}\!\uparrow\! chan(FP), R\!\uparrow\! chan(FP))$ ◇

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 one another arbitrarily fast or even coincide.

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 $h\!\uparrow\!\{in, out\}$ and $h_{old}\!\uparrow\!\{in, out\}$ are equally long, if the ith element of $h_{old}\!\uparrow\!\{in, out\}$ records an in communication then it is equal to the ith element of $h\!\uparrow\!\{in, out\}$, if the ith element of $h_{old}\!\uparrow\!\{in, out\}$ records an out communication then so does the ith element of $h\!\uparrow\!\{in, out\}$ 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.

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

$\triangle$

For a failure hypothesis χ we introduce, similar to [20], the construct $P \wr \chi$ to indicate execution of process P under the assumption of χ. 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.

Table 3: Extended Syntax of the Programming Language

Failure Prone Process	FP	$::=$	$P \mid FP_1 \| FP_2 \mid FP \backslash cset \mid FP \wr \chi$

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

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 $chan(FP_1) \cap chan(FP_2)$. Hence, if θ is a trace of $FP_1 \| FP_2$ then $\theta \uparrow chan(FP_1)$ and $\theta \uparrow chan(FP_2)$ are the corresponding traces of FP_1 and FP_2, respectively. As we already saw in Section 3, a communication is refused by $FP_1 \| FP_2$ if, and only if, it is refused by FP_1 or FP_2.

  $\mathcal{O}[\![FP_1 \| FP_2]\!] =$
  $\{ (\theta, \mathfrak{R}) \mid$ there exist $(\theta_1, \mathfrak{R}_1) \in \mathcal{O}[\![FP_1]\!]$ and $(\theta_2, \mathfrak{R}_2) \in \mathcal{O}[\![FP_2]\!]$ such that
  $\theta \uparrow chan(FP_i) = \theta_i$, for $i = 1, 2$, $\theta \uparrow chan(FP_1 \| FP_2) = \theta$, and $\mathfrak{R} = \mathfrak{R}_1 \cup \mathfrak{R}_2 \}$

- The observations of $FP \backslash cset$ are, as before, characterized by the fact that $cset$ communications are continuously refused, except on single instants.

  $\mathcal{O}[\![FP \backslash cset]\!] = \{ (\theta \backslash cset, \mathfrak{R} \backslash cset) \mid (\theta, \mathfrak{R}) \in \mathcal{O}[\![FP]\!] \wedge ASAP(\mathfrak{R}, cset) \}$

- The observations of failure prone process $FP \wr \chi$ are those observations that are related, according to χ, to the observations of FP.

  $\mathcal{O}[\![FP \wr \chi]\!] = \{ (\theta, \mathfrak{R}) \mid$ there exists a $(\theta_0, \mathfrak{R}_0) \in \mathcal{O}[\![FP]\!]$ such that, for all γ,
  $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \chi$, $\theta \uparrow chan(FP) = \theta$, and $\mathfrak{R} \uparrow chan(FP) = \mathfrak{R} \}$

From this definition of the semantics:

Rule 5.1 (Invariance 1)

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

Rule 5.2 (Invariance 2)

$$\frac{cset \cap chan(FP) = \emptyset}{FP \ \mathbf{sat} \ R \uparrow cset = \emptyset}$$

Definition 26 (Composite transformation expression) For transformation expressions ψ_1 and ψ_2, the *composite transformation expression* $\psi_1 \wr \psi_2$ is defined as follows

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

where s and N must be fresh. $\diamond$

Since the interpretation of assertions has not changed, the validity of correctness formula FP **sat** ϕ 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) $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \psi(texp, h, R_{old}, R)$ iff $(\mathcal{T}[\![texp]\!](\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma), \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \psi$

b) $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \psi(h_{old}, texp, R_{old}, R)$ iff $(\theta_0, \mathcal{T}[\![texp]\!](\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma), \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \psi$

c) $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \psi(h_{old}, h, rfxp, R)$ iff $(\theta_0, \theta, \mathcal{R}[\![rfxp]\!](\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma), \mathfrak{R}, \gamma) \models \psi$

d) $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \psi(h_{old}, h, R_{old}, rfxp)$ iff $(\theta_0, \theta, \mathfrak{R}_0, \mathcal{R}[\![rfxp]\!](\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma), \gamma) \models \psi$ ○

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)

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

Rule 6.2 (Conjunction)

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

If h is a timed history of process $FP_1 \| FP_2$ 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 $FP_1 \| FP_2$ 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)

$$\frac{FP_1 \ \textbf{sat} \ \phi_1(h, R) \ , \ \ FP_2 \ \textbf{sat} \ \phi_2(h, R)}{FP_1 \| FP_2 \ \textbf{sat} \ \exists N_1, N_2 \cdot \begin{array}{l} R = N_1 \cup N_2 \\ \wedge \ \phi_1(h \uparrow chan(FP_1), N_1) \\ \wedge \ \phi_2(h \uparrow chan(FP_2), N_2) \end{array}}$$

Observations of $FP \backslash cset$ are characterized by the fact that $cset$ communications occur as soon as possible. Then, the effect of hiding a set $cset$ 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, $FP \backslash cset$ satisfies an assertion ϕ if FP satisfies $ASAP(R, cset) \rightarrow \phi$, unless a reference to h or R in ϕ includes one or more channels from $cset$.

Rule 6.4 (Hiding)

$$\frac{FP \ \textbf{sat} \ ASAP(R, cset) \rightarrow \phi(h \backslash cset, R \backslash cset)}{FP \backslash cset \ \textbf{sat} \ \phi(h, R)}$$

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

a) $(FP_1 \backslash cset) \ \| \ FP_2 = (FP_1 \| FP_2) \backslash cset$ iff $chan(FP_2) \cap cset = \emptyset$

b) $(FP \backslash cset_1) \backslash cset_2 = FP \backslash (cset_1 \cup cset_2)$ ○

Finally, for the introduction of a failure hypothesis we have

Rule 6.5 (Failure hypothesis introduction)

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

Observe that, since ϕ is an assertion, h_{old} and R_{old} do not appear in ϕ, and hence also the composite expression $\phi \wr \chi$ 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.

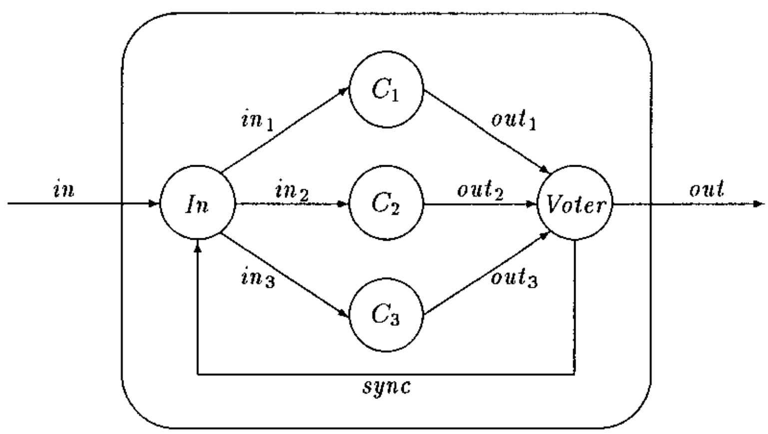

Figure 1: Triple modular redundant system

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

- $\textbf{until}(texp, t) = \begin{cases} \langle\rangle & \text{if } texp = \langle\rangle \text{ or } ts(first(texp)) > t \\ texp_1 & \text{if } texp = texp_1 \hat{}\, texp_2 \text{ such that } ts(last(texp_1)) \leq t \text{ and} \\ & ts(first(texp_2)) > t \end{cases}$

  to denote trace *texp*'s prefix up to and including t.

- $\textbf{from}(texp, t) = \begin{cases} \langle\rangle & \text{if } texp = \langle\rangle \text{ or } ts(last(texp)) < t \\ texp_2 & \text{if } texp = texp_1 \hat{}\, texp_2 \text{ such that } ts(last(texp_1)) \leq t \text{ and} \\ & ts(first(texp_2)) > t \end{cases}$

  to denote trace *texp*'s suffix starting at t. ◇

In accepts a value from the environment via channel in and distributes that value via channels in_1, in_2 and in_3 after K_{In} time units. When all three of them have occurred In tries to communicate via $sync$. ε time units after this communication has been taken, it enables in again.

$$
\begin{array}{lll}
In \ \mathbf{sat} & \forall i,j \cdot 1 \le i \le len(h \uparrow in_j) \ \rightarrow \ val(h \uparrow in_j(i)) = val(h \uparrow in(i)) \\
& \wedge\ h = \langle\rangle \ \rightarrow \ \textbf{enable } in \textbf{ and refuse } \cup_{j=1}^{3} \{in_j\} \textbf{ upto } \infty \\
& \wedge\ \forall t, v \cdot (t, in, v) \in h \ \rightarrow \\
& \qquad\qquad \textbf{refuse } chan(In) \textbf{ upto } K_{In} \\
& \qquad \bigwedge_{j=1}^{3} \ \textbf{after } K_{In} : \forall t_1 \cdot in_j \textbf{ refused precisely upto } t_1 \\
& \qquad\qquad\qquad\qquad \rightarrow \textbf{ enable } in_j \textbf{ for } t_1 \\
& \wedge\ \forall t, v \cdot (\bigwedge_{j=1}^{3} (t, in_j, v) \in h) \ \rightarrow \\
& \qquad \textbf{refuse } chan(In) \textbf{ upto } \varepsilon \\
& \qquad \wedge\ \textbf{after } \varepsilon : \forall t_1 \cdot sync \textbf{ refused precisely upto } t_1 \\
& \qquad\qquad\qquad \rightarrow \textbf{ enable } sync \textbf{ for } t_1 \\
& \wedge\ \forall t, v \cdot (t, sync, v) \in h \ \rightarrow \\
& \qquad \textbf{refuse } chan(In) \textbf{ upto } \varepsilon \\
& \qquad \wedge\ \textbf{after } \varepsilon : \forall t_1 \cdot in \textbf{ refused precisely upto } t_1 \\
& \qquad\qquad\qquad \rightarrow \textbf{ enable } in \textbf{ for } t_1
\end{array}
$$

$Voter$ awaits a communication via any of the channels out_1, out_2 and 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 Δ time units the timer expires, and K_{Voter} time units thereafter the tentative vote is communicated to the environment via out. Thus, timing is essential as it ends the waiting for a value that got lost. ε time units after an output occurs, $Voter$ tries to synchronize via $sync$. When this communication is taken, it enables channels out_1, out_2 and out_3 again.

$$
\begin{array}{lll}
Voter \ \mathbf{sat} & h = \langle\rangle \ \rightarrow \ \textbf{enable } \{in_1, in_2, in_3\} \textbf{ upto } \infty \\
& \wedge\ \forall k, l, m, t, v \cdot k \ne l \wedge k \ne m \wedge l \ne m \ \rightarrow \\
& \qquad\qquad ((t, out_k, v) \in h \ \wedge \ (t, out_l, v) \in h) \ \rightarrow \\
& \qquad\qquad \forall t_1 \cdot out_m \textbf{ refused precisely upto } t_1 \\
& \qquad\qquad\qquad \rightarrow \quad \textbf{refuse } out \textbf{ upto } \min(t_1, \Delta) + K_{Voter} \\
& \qquad\qquad\qquad\quad \wedge\ \textbf{after } \min(t_1, \Delta) + K_{Voter} : \\
& \qquad\qquad\qquad\qquad \forall t_2 \cdot out \textbf{ refused precisely upto } t_2 \\
& \qquad\qquad\qquad\qquad\qquad \rightarrow \quad \textbf{enable } out \textbf{ for } t_2 \\
& \qquad\qquad\qquad\qquad\qquad\quad \wedge\ \forall v_1 \cdot (t_2, out, v_1) \in h \ \rightarrow \ v_1 = v \\
& \wedge\ \forall t, v \cdot (t, out, v) \in h \ \rightarrow \\
& \qquad \textbf{refuse } chan(Voter) \textbf{ upto } \varepsilon \\
& \qquad \wedge\ \textbf{after } \varepsilon : \forall t_1 \cdot sync \textbf{ refused precisely upto } t_1 \\
& \qquad\qquad\qquad \rightarrow \textbf{ enable } sync \textbf{ and refuse } chan(Voter) - \{sync\} \textbf{ for } t_1 \\
& \wedge\ \forall t, v \cdot (t, sync, v) \in h \ \rightarrow \\
& \qquad\qquad \textbf{refuse } chan(Voter) \textbf{ upto } \varepsilon \\
& \qquad \bigwedge_{j=1}^{3} \ \textbf{after } \varepsilon : \forall t_1 \cdot in_j \textbf{ refused precisely upto } t_1 \\
& \qquad\qquad\qquad\qquad \rightarrow \textbf{ enable } in_j \textbf{ for } t_1
\end{array}
$$

Since C_1, C_2, and C_3 do not share a single channel, we easily obtain, by (Parallel composition) and (Consequence), that

$$
\begin{array}{lll}
C_1 \| C_2 \| C_3 \ \textbf{sat} & \forall i, j \cdot 1 \leq i \leq len(h \uparrow out_j) \ \rightarrow \ val(h \uparrow out_j(i)) = f(val(h \uparrow in_j(i))) \\
& \wedge\ h = \langle\rangle \ \rightarrow \ \textbf{enable} \ \bigcup_{j=1}^{3} \{in_j\} \ \textbf{upto} \ \infty \\
& \wedge\ \forall t, v \cdot (\bigwedge_{j=1}^{3} (t, in_j, v) \in h) \\
& \qquad \rightarrow \quad \textbf{refuse} \ \bigcup_{j=1}^{3} \{out_j\} \ \textbf{upto} \ K_C \\
& \qquad\quad \wedge\ \textbf{after} \ K_C : \\
& \qquad\qquad \forall t_1 \cdot \bigcup_{j=1}^{3} \{out_j\} \ \textbf{refused precisely upto} \ t_1 \\
& \qquad\qquad\quad \rightarrow \quad \textbf{enable} \ \bigcup_{j=1}^{3} \{out_j\} \ \textbf{for} \ t_1 \\
& \qquad\qquad\qquad \wedge\ \bigcup_{j=1}^{3} \{out_j\} \ \textbf{enabled at} \ t_1 \\
& \qquad\qquad\qquad\quad \rightarrow \ \textbf{after} \ t_1 + \varepsilon : \\
& \qquad\qquad\qquad\qquad \forall t_2 \cdot \bigcup_{j=1}^{3} \{in_j\} \ \textbf{refused precisely upto} \ t_2 \\
& \qquad\qquad\qquad\qquad\quad \rightarrow \ \textbf{enable} \ \bigcup_{j=1}^{3} \{in_j\} \ \textbf{for} \ t_2
\end{array}
$$

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:

$$
\begin{array}{lll}
Loss^{\leq 1} & \equiv & h \uparrow \{in_1, in_2, in_3\} \ = \ h_{old} \uparrow \{in_1, in_2, in_3\} \\
& & \wedge\ \forall i \cdot 1 \leq i \leq \lfloor len(h_{old} \uparrow \{out_1, out_2, out_3\})/3 \rfloor \ \rightarrow \ \exists k \neq l \cdot \quad h_{old} \uparrow out_k(i) \in h \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge\ h_{old} \uparrow out_l(i) \in h \\
& & \wedge\ R \uparrow \{in_1, in_2, in_3\} \ = \ R_{old} \uparrow \{in_1, in_2, in_3\} \\
& & \wedge\ R \uparrow \{out_1, out_2, out_3\} \\
& & \quad = \ R_{old} \uparrow \{out_1, out_2, out_3\} \\
& & \qquad \bigcup_{j=1}^{3} \{ \ \{out_j\} \times [t_1, t_2) \mid \exists t, v \cdot \quad (t, out_j, v) \in h_{old} \ \wedge \ (t, out_j, v) \notin h \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge\ t_1 = ts(last(\textbf{until}(h \uparrow in_j, t))) \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge\ t_2 = ts(first(\textbf{from}(h \uparrow in_j, t))) \qquad \}
\end{array}
$$

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

Failure hypothesis $Loss^{\leq 1}$ 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 $(C_1 \| C_2 \| C_3) \wr Loss^{\leq 1}$ will provide output within K_C time units, and that if at the moment two such outputs occur the environment does not refuse any of the out_j communications, $j = 1, 2, 3$, then all three components will accept fresh input ε time units thereafter.

$$(C_1 \| C_2 \| C_3) \wr Loss^{\leq 1}$$

sat

$$\begin{array}{l}
h = \langle\rangle \;\rightarrow\; \textbf{enable } \cup_{j=1}^{3} \{in_j\} \textbf{ upto } \infty \\
\wedge\ \forall t, v \cdot (\bigwedge_{j=1}^{3} (t, in_j, v) \in h) \\
\qquad \rightarrow \quad \textbf{refuse } \cup_{j=1}^{3} \{out_j\} \textbf{ upto } K_C \\
\qquad\quad \wedge\ \textbf{after } K_C : \\
\qquad\qquad \forall t_1 \cdot \cup_{j=1}^{3} \{out_j\} \textbf{ refused precisely upto } t_1 \\
\qquad\qquad\quad \rightarrow \quad \exists k \neq l \cdot \quad \textbf{enable } \{out_k, out_l\} \textbf{ for } t_1 \\
\qquad\qquad\qquad\qquad\qquad \wedge\ \forall v_1, v_2 \cdot ((t_1, out_k, v_1) \in h \ \wedge\ (t_1, out_l, v_2) \in h) \ \rightarrow\ v_1 = v_2 = f(v) \\
\qquad\qquad\qquad \wedge\ \cup_{j=1}^{3} \{out_j\} \textbf{ enabled at } t_1 \\
\qquad\qquad\qquad\quad \rightarrow \textbf{ after } t_1 + \varepsilon : \forall t_2 \cdot \cup_{j=1}^{3} \{in_j\} \textbf{ refused precisely upto } t_2 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \rightarrow \textbf{ enable } \cup_{j=1}^{3} \{in_j\} \textbf{ for } t_2
\end{array}$$

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 $Loss^{\leq 1}$ to establish this non-blocking property.

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

$$(\ (C_1 \| C_2 \| C_3) \wr Loss^{\leq 1}\ ) \quad \| \quad Voter$$

sat

$$\begin{array}{l}
ASAP(R, \cup_{j=1}^{3} \{out_j\}) \;\rightarrow \\
\quad \forall t, v \cdot (\quad \bigwedge_{j=1}^{3} (t, in_j, v) \in h \\
\qquad\qquad \wedge\ \textbf{until}(h_{Voter}, t) = \langle\rangle \ \vee\ \exists t_1, v_1 \cdot \textbf{last}(\textbf{until}(h_{Voter}, t)) = (t_1, sync, v_1) \ ) \\
\qquad \rightarrow \quad \exists t_1 \cdot \quad 0 \leq t_1 \leq \Delta \\
\qquad\qquad\qquad\qquad \wedge\ \textbf{refuse } out \textbf{ upto } K_C + t_1 + K_{Voter} \\
\qquad\qquad\qquad\qquad \wedge\ \textbf{after } K_C + t_1 + K_{Voter} : \forall t_2 \cdot out \textbf{ refused precisely upto } t_2 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \rightarrow \quad \textbf{enable } out \textbf{ for } t_2 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge\ \forall v_1 \cdot (t_2, out, v_1) \in h \ \rightarrow\ v_1 = v \\
\qquad\qquad \wedge\ \textbf{after } K_C + \varepsilon : \forall t_1 \cdot \cup_{j=1}^{3} \{in_j\} \textbf{ refused precisely upto } t_1 \\
\qquad\qquad\qquad\qquad\qquad\qquad \rightarrow \textbf{ enable } \cup_{j=1}^{3} \{in_j\} \textbf{ for } t_1 \\
\wedge\ \forall t, v \cdot (t, out, v) \in h \ \rightarrow \quad \textbf{refuse } sync \textbf{ upto } \varepsilon \\
\qquad\qquad\qquad\qquad\qquad \wedge\ \textbf{after } \varepsilon : \forall t_1 \cdot sync \textbf{ refused precisely upto } t_1 \ \rightarrow\ \textbf{enable } sync \textbf{ for } t_1
\end{array}$$

Note that if $(\tau, c, \mu) \in h$ and $c \notin cset$ then also $(\tau, c, \mu) \in h \uparrow cset$. Further note that if $h = \langle\rangle$ then $h \uparrow cset = \langle\rangle$.

Because *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 in_j communication, or the preceding in_j communications all happened at the same time, then C_j does not refuse in_j, $j = 1, 2, 3$, at time t. Again, this readiness does not change until an in_j communication, $j = 1, 2, 3$, actually occurs. By (Hiding), the specification of *In*, and (Parallel composition),

$$In \;\|\; (((C_1\|C_2\|C_3) \wr Loss^{\leq 1})\| Voter) \setminus \bigcup_{j=1}^{3}\{out_j\}$$

$$\textbf{sat}$$

$$\begin{array}{l}
ASAP(R, \cup_{j=1}^{3}\{in_j\} \cup \{sync\}) \;\rightarrow \\
\quad \forall t, v \cdot (\quad (t, in, v) \in h \\
\qquad\qquad \wedge\ \textbf{until}(h\!\uparrow \cup_{j=1}^{3}\{in_j\}, t) = \langle\rangle \ \vee\ \exists t_1, v_1 \cdot \bigwedge_{j=1}^{3} \textbf{last}(\textbf{until}(h_{C_j}, t)) = (t_1, in_j, v_1) \ ) \\
\qquad\quad \rightarrow \exists t_1 \cdot \quad 0 \leq t_1 \leq \Delta \\
\qquad\qquad\qquad \wedge\ \textbf{refuse}\ out\ \textbf{upto}\ K_{In} + K_C + t_1 + K_{Voter} \\
\qquad\qquad\qquad \wedge\ \textbf{after}\ K_{In} + K_C + t_1 + K_{Voter} : \forall t_2 \cdot out\ \textbf{refused precisely upto}\ t_2 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \rightarrow \quad \textbf{enable}\ out\ \textbf{for}\ t_2 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge\ \forall v_1 \cdot (t_2, out, v_1) \in h \ \rightarrow\ v_1 = f(v) \\
\wedge\ \forall t, v \cdot (t, out, v) \in h \ \rightarrow \quad \textbf{refuse}\ in\ \textbf{upto}\ 2\varepsilon \\
\qquad\qquad\qquad\qquad\qquad \wedge\ \textbf{after}\ 2\varepsilon : \forall t_1 \cdot in\ \textbf{refused precisely upto}\ t_1 \ \rightarrow\ \textbf{enable}\ in\ \textbf{for}\ t_1
\end{array}$$

If the first in communication occurs at time t then we know that $\textbf{until}(h_{C_1\|C_2\|C_3}, t) = \langle\rangle$. Consequently, C_j does not refuse in_j at t, $j = 1,2,3$. Since this willingness does not change until an in_j communication, $j = 1,2,3$, actually occurs, the inductive structure that appears above can easily be resolved under the assumption that communications on in_j, $j = 1,2,3$, occur as soon as possible. Formally, by (Hiding)

$$(\ In \ \|\ ((C_1\|C_2\|C_3) \wr Loss^{\leq 1}) \ \|\ Voter \ ) \setminus \bigcup_{j=1}^{3}\{in_j\} \cup \bigcup_{j=1}^{3}\{out_j\} \cup \{sync\}$$

$$\textbf{sat}$$

$$\begin{array}{l}
\quad \forall t, v \cdot (t, in, v) \in h \ \rightarrow \\
\qquad \exists t_1 \cdot \quad 0 \leq t_1 \leq \Delta \\
\qquad\qquad \wedge\ \textbf{refuse}\ out\ \textbf{upto}\ K_{In} + K_C + t_1 + K_{Voter} \\
\qquad\qquad \wedge\ \textbf{after}\ K_{In} + K_C + t_1 + K_{Voter} : \forall t_2 \cdot out\ \textbf{refused precisely upto}\ t_2 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \rightarrow \quad \textbf{enable}\ out\ \textbf{for}\ t_2 \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \wedge\ \forall v_1 \cdot (t_2, out, v_1) \in h \ \rightarrow\ v_1 = f(v) \\
\wedge\ \forall t, v \cdot (t, out, v) \in h \ \rightarrow \quad \textbf{refuse}\ in\ \textbf{upto}\ 2\varepsilon \\
\qquad\qquad\qquad\qquad\qquad \wedge\ \textbf{after}\ 2\varepsilon : \forall t_1 \cdot in\ \textbf{refused precisely upto}\ t_1 \ \rightarrow\ \textbf{enable}\ in\ \textbf{for}\ t_1
\end{array}$$

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** ϕ is derivable, then it is valid. Furthermore, we prove the proof system to be complete, that is, we prove that, if a correctness formula FP **sat** ϕ is valid, then it is derivable.

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 ϕ is derivable, we add the following axiom to our proof theory.

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

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

○

As in [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 ϕ is *precise* for failure prone process FP iff

i) $\models FP \ \mathbf{sat} \ \phi$.

ii) if $\theta \uparrow chan(FP) = \theta$, $\mathfrak{R} \uparrow chan(FP) = \mathfrak{R}$, and, for some γ, $(\theta, \mathfrak{R}, \gamma) \models \phi$, then $(\theta, \mathfrak{R}) \in \mathcal{O}[\![FP]\!]$.

iii) $\phi \rightarrow \phi(h \uparrow chan(FP), R \uparrow chan(FP))$ $\diamond$

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

Definition 29 (Network completeness) Assume that for every process P there exists a precise assertion ϕ with $\vdash P \ \mathbf{sat} \ \phi$. Then, for any failure prone process FP and assertion η, $\models FP \ \mathbf{sat} \ \eta$ implies $\vdash FP \ \mathbf{sat} \ \eta$. $\diamond$

The following lemma asserts that preciseness is preserved.

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

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 ϕ_1 is precise for FP and $\models FP \ \mathbf{sat} \ \phi_2$ then
$\models (\ \phi_1 \ \wedge \ h \uparrow chan(FP) = h \ \wedge \ R \uparrow chan(FP) = R\ ) \ \rightarrow \ \phi_2$

Proof. Assume that ϕ_1 is precise for FP, and that $\models FP \ \mathbf{sat} \ \phi_2$ (1).
Consider any θ, $\mathfrak{R}$, and γ. Assume $(\theta, \mathfrak{R}, \gamma) \models \phi_1 \wedge h \uparrow chan(FP) = h \wedge R \uparrow chan(FP) = R$. Then, by the preciseness of ϕ_1 for FP, $(\theta, \mathfrak{R}) \in \mathcal{O}[\![FP]\!]$. By (1), for all $\widehat{\gamma}$, $(\theta, \mathfrak{R}, \widehat{\gamma}) \models \phi_2$. Hence, $(\theta, \mathfrak{R}, \gamma) \models \phi_2$. $\square$

Now we can establish relative network completeness.

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

Proof. Assume that for every process P there exists a precise specification ϕ with $\vdash P \ \mathbf{sat} \ \phi$. Then, by the preciseness preservation lemma, for every failure prone process FP there exists an assertion η which is precise for FP and $\vdash FP \ \mathbf{sat} \ \eta$ (1).
Assume $\models FP \ \mathbf{sat} \ \xi$. By the definition of the semantics,
$\vdash FP \ \mathbf{sat} \ h \uparrow chan(FP) = h \ \wedge \ R \uparrow chan(FP) = R$ (2).
Then, by (1), (2), the preciseness consequence lemma, the relative completeness assumption, and (Consequence), $\vdash FP \ \mathbf{sat} \ \xi$. $\square$

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

[1] M. Abadi and L. Lamport, An old-fashioned recipe for real time, in: *Proc. REX Workshop on Real-Time: Theory in Practice*, Lecture Notes in Computer Science **600** (Springer, 1992) 1–27.

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

[3] H. Barringer, R. Kuiper, A. Pnueli, A really abstract concurrent model and its temporal logic, in: *Proc. 13th ACM Symposium on Principles of Programming Languages* (ACM, 1986) 173–183.

[4] J. Coenen and J. Hooman, Parameterized semantics for fault tolerant real-time systems, in: J. Vytopil, ed., *Formal Techniques in Real-Time and Fault Tolerant Systems* (Kluwer Academic Publishers, 1993) 51–78.

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

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

[7] F. Cristian, Understanding fault tolerant distributed systems, *Communications of the ACM* **34**(2) (1991) 56 - 78.

[8] R. Gerth and A. Boucher, A timed failures model for Extended Communicating Processes, in: *Proc. 14th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science **267** (Springer, 1987) 95–114.

[9] INMOS Limited, occam 2 *Reference Manual* (Prentice Hall, 1988).

[10] H. Jifeng and C.A.R. Hoare, Algebraic specification and proof of a distributed recovery algorithm, *Distributed Computing* **2** (1987) 1–12.

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

[12] R. Koymans, R.K. Shyamasundar, W.-P. de Roever, R. Gerth, S. Arun-Kumar, Compositional semantics for real-time distributed computing, *Information and Computation* **79** (3) 210 – 256, 1988.

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

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

[15] J. Nordahl, Design for dependability, in: *Proc. 3rd IFIP Int. Working Conference on Dependable Computing for Critical Applications*, Dependable Computing and Fault Tolerant Systems **8** (Springer, 1993) 65–89.

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

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

[18] G.M. Reed and A.W. Roscoe, A timed model for Communicating Sequential Processes, in: *Proc. 13th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science **266** (Springer, 1986) 314–323; Theoretical Computer Science **58** (1988) 249–261.

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

[20] H. Schepers, Tracing fault tolerance, in: *Proc. 3rd IFIP Int. Working Conference on Dependable Computing for Critical Applications*, Dependable Computing and Fault Tolerant Systems **8** (Springer, 1993) 91–110.

[21] H. Schepers and R. Gerth, A compositional proof theory for fault tolerant real-time distributed systems, in: *Proc. 12th Symp. on Reliable Distributed Systems* (IEEE Computer Society Press, 1993) – .

[22] H. Schepers and J. Hooman, Trace-based compositional reasoning about fault tolerant systems, in: *Proc. Parallel Architectures and Languages Europe (PARLE) '93*, Lecture Notes in Computer Science **694** (Springer, 1993) 197–208.

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

[24] J. Widom, D. Gries and F. Schneider, Trace-based network proof systems: expressiveness and completeness, *ACM Trans. on Programming Languages and Systems* **14**(3) (July 1992) 396–416.

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 $\models FP_1$ **sat** ϕ_1, $\models FP_2$ **sat** ϕ_2 (1).
Consider any γ. Let $(\theta, \mathfrak{R}) \in \mathcal{O}[\![FP_1 \| FP_2]\!]$. By the definition of the semantics there exist, for $i = 1, 2$, $\mathfrak{R}_i$ such that $(\theta \uparrow chan(FP_i), \mathfrak{R}_i) \in \mathcal{O}[\![FP_i]\!]$ (2),
and $\mathfrak{R} = \mathfrak{R}_1 \cup \mathfrak{R}_2$ (3).

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

By (3), $(\theta, \mathfrak{R}, \hat{\gamma}) \models R = N_1 \cup N_2$, or $(\theta, \mathfrak{R}, \gamma) \models \exists N_1, N_2 \cdot R = N_1 \cup N_2$ (4).
By (1) and (2), for all γ', $(\theta \uparrow chan(FP_i), \mathfrak{R}_i, \gamma') \models \phi_i$, $i = 1, 2$. Then, $(\theta \uparrow chan(FP_i), \mathfrak{R}_i, \hat{\gamma}) \models \phi_i$. Observe that $\mathcal{RF}[\![N_i]\!](\theta, \mathfrak{R}, \hat{\gamma}) = \mathfrak{R}_i$ and that $\mathcal{T}[\![h \uparrow chan(FP_i)]\!](\theta, \mathfrak{R}, \hat{\gamma}) = \theta \uparrow chan(FP_i)$. Consequently, we have $(\mathcal{T}[\![h \uparrow chan(FP_i)]\!](\theta, \mathfrak{R}, \hat{\gamma})\ ,\ \mathcal{RF}[\![N_i]\!](\theta, \mathfrak{R}, \hat{\gamma})\ ,\ \hat{\gamma}) \models \phi_i$. Then, by applying substitution lemma

b) and d), we obtain that $(\theta, \mathfrak{R}, \widehat{\gamma}) \models \phi_i[\ (h \uparrow chan(FP_i))/h\ ,\ N_i/R\ ]$, from which we may conclude that $(\theta, \mathfrak{R}, \gamma) \models \exists N_1, N_2 \cdot \phi_i[\ (h \uparrow chan(FP_i))/h\ ,\ N_i/R\ ]$ (5).
By (4) and (5) we conclude that the parallel composition rule is sound. □

A.3 Soundness of the hiding rule

Assume that $\models FP$ **sat** $ASAP(R, cset) \rightarrow \phi(h \backslash cset, R \backslash cset)$ (1).
Consider any γ. Let $(\theta, \mathfrak{R}) \in \mathcal{O}[\![FP \backslash cset]\!]$. Then, by the definition of the semantics there exists a $(\widehat{\theta}, \widehat{\mathfrak{R}}) \in \mathcal{O}[\![FP]\!]$ (2)
for which $ASAP(\widehat{\mathfrak{R}}, cset)$ (3),
such that $\theta = \widehat{\theta} \backslash cset$ (4),
and $\mathfrak{R} = \widehat{\mathfrak{R}} \backslash cset$ (5).
By (2) and (1), we have that, for all γ, $(\widehat{\theta}, \widehat{\mathfrak{R}}, \gamma) \models ASAP(R, cset) \rightarrow \phi(h \backslash cset, R \backslash cset)$. Then, by (3), $(\widehat{\theta}, \widehat{\mathfrak{R}}, \gamma) \models \phi(h \backslash cset, R \backslash cset)$. By substitution lemma b) and d), we obtain $(\widehat{\theta} \backslash cset, \widehat{\mathfrak{R}} \backslash cset, \gamma) \models \phi$. Hence, by (4) and (5), $(\theta, \mathfrak{R}, \gamma) \models \phi$, from which we conclude that the hiding rule is sound. □

A.4 Soundness of the failure hypothesis introduction rule

Assume that $\models FP$ **sat** ϕ (1).
Consider any γ. Let $(\theta, \mathfrak{R}) \in \mathcal{O}[\![FP \wr \chi]\!]$. Then, by the definition of the semantics, there exists a $(\theta_0, \mathfrak{R}_0) \in \mathcal{O}[\![FP]\!]$, such that, for all γ, $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \chi$ (2).

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

Since $(\theta_0, \mathfrak{R}_0) \in \mathcal{O}[\![FP]\!]$, we know, by (1), that, for all γ, $(\theta_0, \mathfrak{R}_0, \gamma) \models \phi$. Consequently, we have $(\theta_0, \mathfrak{R}_0, \widehat{\gamma}) \models \phi$. As, for all $\widehat{\theta}$ and $\widehat{\mathfrak{R}}$, $\mathcal{T}[\![s]\!](\widehat{\theta}, \widehat{\mathfrak{R}}, \widehat{\gamma}) = \theta_0$ and $\mathcal{RF}[\![N]\!](\widehat{\theta}, \widehat{\mathfrak{R}}, \widehat{\gamma}) = \mathfrak{R}_0$, we may conclude $(\mathcal{T}[\![s]\!](\theta, \mathfrak{R}, \widehat{\gamma}), \mathcal{RF}[\![M]\!](\theta, \mathfrak{R}, \widehat{\gamma}), \widehat{\gamma}) \models \phi$. Hence, by applying substitution lemma b) and d), we obtain $(\theta, \mathfrak{R}, \widehat{\gamma}), \models \phi[\ s/h\ ,\ N/R\ ]$ (3).
By (2), $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \widehat{\gamma}) \models \chi$, that is, $(\mathcal{T}[\![s]\!](\theta, \mathfrak{R}, \widehat{\gamma}), \theta, \mathcal{RF}[\![N]\!](\theta, \mathfrak{R}, \widehat{\gamma}), \mathfrak{R}, \widehat{\gamma}) \models \chi$. Then, by substitution lemma a) and c), $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \widehat{\gamma}) \models \chi[s/h_{old}\ ,\ N/R_{old}]$. Since h_{old} and R_{old} obviously do not appear in $\chi[s/h_{old}\ ,\ N/R_{old}]$ we may conclude that $(\theta, \mathfrak{R}, \widehat{\gamma}) \models \chi[s/h_{old}\ ,\ N/R_{old}]$ (4).
By (3) and (4) we obtain that $(\theta, \mathfrak{R}, \widehat{\gamma}) \models \phi[\ s/h\ ,\ N/R\ ] \wedge \chi[\ s/h_{old}\ ,\ N/R_{old}\ ]$, from which we conclude $(\theta, \mathfrak{R}, \gamma) \models \exists s, N \cdot \phi[\ s/h\ ,\ N/R\ ]\ \wedge\ \chi[\ s/h_{old}\ ,\ N/R_{old}\ ]$. Hence, the failure hypothesis introduction rule is sound. □

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 $\vdash FP_1$ **sat** ϕ_1 and $\vdash FP_2$ **sat** ϕ_2, with ϕ_1 and ϕ_2 precise for FP_1 and FP_2, respectively. By applying (Parallel composition), we obtain

$$\vdash FP_1 \parallel FP_2 \ \textbf{sat}\ \exists N_1, N_2 \cdot \begin{array}{l} \quad R = N_1 \cup N_2 \\ \wedge\ \phi_1(h \uparrow chan(FP_1), N_1) \\ \wedge\ \phi_2(h \uparrow chan(FP_2), N_2) \end{array} \qquad (1).$$

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

i) By (1) and soundness, we obtain

$$\models FP_1 \parallel FP_2 \ \textbf{sat}\ \exists N_1, N_2 \cdot \begin{array}{l} \quad R = N_1 \cup N_2 \\ \wedge\ \phi_1(h \uparrow chan(FP_1), N_1) \\ \wedge\ \phi_2(h \uparrow chan(FP_2), N_2) \end{array}$$

ii) Let $chan(\theta) \subseteq chan(FP_1 \| FP_2)$ (2),
and $\mathfrak{R}\!\uparrow chan(FP_1 \| FP_2) = \mathfrak{R}$ (3).
Assume further that, for some γ, $(\theta, \mathfrak{R}, \gamma) \models \exists N_1, N_2 \cdot \quad R = N_1 \cup N_2 \wedge \phi_1(h\!\uparrow chan(FP_1), N_1) \wedge \phi_2(h\!\uparrow chan(FP_2), N_2)$

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

$$(\theta, \mathfrak{R}, (\gamma : (N_1, N_2) \mapsto (\widehat{\mathfrak{R}_1}, \widehat{\mathfrak{R}_2}))) \models \exists N_1, N_2 \cdot \quad \begin{array}{l} R = N_1 \cup N_2 \\ \wedge\ \phi_1(h\!\uparrow chan(FP_1), N_1) \\ \wedge\ \phi_2(h\!\uparrow chan(FP_2), N_2) \end{array} \tag{4}$$

Then, by substitution lemma b) and d),
$(\theta\!\uparrow chan(FP_1), \widehat{\mathfrak{R}_1}, (\gamma : (N_1, N_2) \mapsto (\widehat{\mathfrak{R}_1}, \widehat{\mathfrak{R}_2}))) \models \phi_1$,
and since N_1 and N_2 do not occur free in ϕ_1, $(\theta\!\uparrow chan(FP_1), \widehat{\mathfrak{R}_1}, \gamma) \models \phi_1$.
By the preciseness of ϕ_1 for FP_1,
$(\theta\!\uparrow chan(FP_1), \widehat{\mathfrak{R}_1}, \gamma) \models \phi_1[h\!\uparrow chan(FP_1)/h, R\!\uparrow chan(FP_1)/R]$.
By substitution lemma b) and d), using $(\theta\!\uparrow chan(FP_1))\!\uparrow chan(FP_1) = \theta\!\uparrow chan(FP_1)$,
$(\theta\!\uparrow chan(FP_1), \widehat{\mathfrak{R}_1}\!\uparrow chan(FP_1), \gamma) \models \phi_1$ (5a).
Trivially, $chan(\theta\!\uparrow chan(FP_1)) \subseteq chan(FP_1)$ (5b).
It is also obvious that $(\widehat{\mathfrak{R}_1}\!\uparrow chan(FP_1))\!\uparrow chan(FP_1) = \widehat{\mathfrak{R}_1}\!\uparrow chan(FP_1)$ (5c).
By (5b), (5c), and (5a), the preciseness of ϕ_1 for FP_1 leads to
$((\theta\!\uparrow chan(FP_1), \widehat{\mathfrak{R}_1}\!\uparrow chan(FP_1),) \in \mathcal{O}[\![FP_1]\!]$ (5).
Similarly, $((\theta\!\uparrow chan(FP_2), \widehat{\mathfrak{R}_2}\!\uparrow chan(FP_2),) \in \mathcal{O}[\![FP_2]\!]$ (6).
By (2), trivially, $\theta\!\uparrow chan(FP_1 \| FP_2) = \theta$ (7).
By (4), $\mathfrak{R} = \widehat{\mathfrak{R}_1} \cup \widehat{\mathfrak{R}_2}$, that is, by (3), $\mathfrak{R} = (\widehat{\mathfrak{R}_1}\!\uparrow chan(FP_1)) \cup (\widehat{\mathfrak{R}_2}\!\uparrow chan(FP_2))$ (8).
By (5), (6), (7), and (8), $(\theta, \mathfrak{R}) \in \mathcal{O}[\![FP_1 \| FP_2]\!]$.

iii) Consider any θ, $\mathfrak{R}$, and γ such that

$$(\theta, \mathfrak{R}, \gamma) \models \exists N_1, N_2 \cdot \quad \begin{array}{l} R = N_1 \cup N_2 \\ \wedge\ \phi_1(h\!\uparrow chan(FP_1), N_1) \\ \wedge\ \phi_2(h\!\uparrow chan(FP_2), N_2) \end{array}$$

which is, obviously, equivalent to

$$(\theta, \mathfrak{R}, \gamma) \models \exists N_1, N_2 \cdot \quad \begin{array}{l} R = N_1 \cup N_2 \\ \wedge\ \phi_1((h\!\uparrow chan(FP_1 \| FP_2))\!\uparrow chan(FP_1), N_1) \\ \wedge\ \phi_2((h\!\uparrow chan(FP_1 \| FP_2))\!\uparrow chan(FP_2), N_2) \end{array}$$

By the preciseness of ϕ_1 and ϕ_2 for FP_1 and FP_2, we have, using $\widehat{N_i} = N_i\!\uparrow chan(FP_i)$, $i = 1, 2$,

$$(\theta, \mathfrak{R}, \gamma) \models \exists N_1, N_2 \cdot \quad \begin{array}{l} R = N_1 \cup N_2 \\ \wedge\ \phi_1((h\!\uparrow chan(FP_1 \| FP_2))\!\uparrow chan(FP_1), \widehat{N_1}) \\ \wedge\ \phi_2((h\!\uparrow chan(FP_1 \| FP_2))\!\uparrow chan(FP_2), \widehat{N_2}) \end{array}$$

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

$$(\theta, \mathfrak{R}, \gamma) \models \exists N_1, N_2 \cdot \quad \begin{array}{l} \widehat{R} = \widehat{N_1} \cup \widehat{N_2} \\ \wedge\ \phi_1((h\!\uparrow chan(FP_1 \| FP_2))\!\uparrow chan(FP_1), \widehat{N_1}) \\ \wedge\ \phi_2((h\!\uparrow chan(FP_1 \| FP_2))\!\uparrow chan(FP_2), \widehat{N_2}) \end{array}$$

or, equivalently,

$$(\theta, \mathfrak{R}, \gamma) \models \exists \widehat{N_1}, \widehat{N_2} \cdot \quad \begin{array}{l} \widehat{R} = \widehat{N_1} \cup \widehat{N_2} \\ \wedge\ \phi_1((h\!\uparrow chan(FP_1 \| FP_2))\!\uparrow chan(FP_1), \widehat{N_1}) \\ \wedge\ \phi_2((h\!\uparrow chan(FP_1 \| FP_2))\!\uparrow chan(FP_2), \widehat{N_2}) \end{array}$$

b) Assume $\vdash FP$ **sat** ϕ (1),

with ϕ precise for FP. Define $\widehat{\phi} \equiv \exists s, N \cdot \quad \phi(s,N) \wedge ASAP(N, cset) \wedge h = s \setminus cset \wedge R = N \setminus cset$

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 $chan(\theta) \subseteq chan(FP \setminus cset)$ (3),

$\mathfrak{R} \uparrow chan(FP \setminus cset) = \mathfrak{R}$ (4),

and assume that, for some γ, $(\theta, \mathfrak{R}, \gamma) \models \widehat{\phi}$. Then, there exist some $\widehat{\theta}$ and $\widehat{\mathfrak{R}}$ such that $(\theta, \mathfrak{R}, (\gamma : (s, N) \mapsto (\widehat{\theta}, \widehat{\mathfrak{R}}))) \models \quad \phi(s,N) \wedge ASAP(N, cset) \wedge h = s \setminus cset \wedge R = N \setminus cset$ (5).

Then, $ASAP(\widehat{\mathfrak{R}}, cset)$ (6),

$\theta = \widehat{\theta} \setminus cset$ (7),

and $\mathfrak{R} = \widehat{\mathfrak{R}} \setminus cset$ (8).

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

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

By (4) and (8), and the fact that $cset \subseteq chan(FP)$, we obtain $\widehat{\mathfrak{R}} \uparrow chan(FP) = \widehat{\mathfrak{R}}$ (9c).

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

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

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

$(\theta, \mathfrak{R}, (\gamma : (s, N) \mapsto (\widehat{\theta}, \widehat{\mathfrak{R}}))) \models \quad \phi(s,N) \wedge ASAP(N, cset) \wedge h = s \setminus cset \wedge R = N \setminus cset$ (1).

By the preciseness of ϕ for FP,

$\phi(s, N) \rightarrow \phi(s \uparrow chan(FP), N \uparrow chan(FP))$ (2).

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

Note that $h = s \setminus cset \rightarrow h \uparrow chan(FP \setminus cset) = (s \uparrow chan(FP)) \setminus cset$ (4),

By (1), $R = N \setminus cset$, that is, $R \uparrow chan(FP \setminus cset) = (N \uparrow chan(FP)) \setminus cset$ (5).

By (1) – (5), $(\theta, \mathfrak{R}, (\gamma : (s, N) \mapsto (\widehat{\theta}, \widehat{\mathfrak{R}}))) \models$

$$\begin{array}{l} \phi(s \uparrow chan(FP), N \uparrow chan(FP)) \\ \wedge\ ASAP(N \uparrow chan(FP), cset) \\ \wedge\ h \uparrow chan(FP \setminus cset) = (s \uparrow chan(FP)) \setminus cset \\ \wedge\ R \uparrow chan(FP \setminus cset) = (N \uparrow chan(FP)) \setminus cset \end{array}$$

From which we may conclude $(\theta, \mathfrak{R}, \gamma) \models \widehat{\phi}[h \uparrow chan(FP \setminus cset)/h, R \uparrow chan(FP \setminus cset)/R]$.

c) Assume $\vdash FP$ **sat** ϕ (1),

with ϕ precise for FP. Define $\widehat{\phi} \equiv \phi \wr \chi$, that is

$$\widehat{\phi} \ \equiv\ \exists s, N \cdot \phi[\, s/h \ ,\ N/R \,] \ \wedge\ \chi[\, s/h_{old} \ ,\ N/R_{old} \,]$$

Then, by (1) and (Failure hypothesis introduction), $\vdash FP \wr \chi$ **sat** $\widehat{\phi}$ (2).

We show that $\widehat{\phi}$ is precise for $FP \wr \chi$.

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

ii) Let $chan(\theta) \subseteq chan(FP \wr \chi)$ (3),
$\mathfrak{R} \uparrow chan(FP \wr \chi) = \mathfrak{R}$ (4),
and assume that, for some γ, $(\theta, \mathfrak{R}, \gamma) \models \widehat{\phi}$. Consequently, there exist some $\widehat{\theta}$ and $\widehat{\mathfrak{R}}$ such that $(\theta, \mathfrak{R}, (\gamma : (s, N) \mapsto (\widehat{\theta}, \widehat{\mathfrak{R}}))) \models \phi[\, s/h, N/R \,] \wedge \chi[\, s/h_{old}, N/R_{old} \,]$ (5).
Then, by substitution lemma b) and d), $(\widehat{\theta}, \widehat{\mathfrak{R}}, (\gamma : (s, N) \mapsto (\widehat{\theta}, \widehat{\mathfrak{R}}))) \models \phi$, and thus, since s and N do not occur free in ϕ, $(\widehat{\theta}, \widehat{\mathfrak{R}}, \gamma) \models \phi$. Since ϕ is precise for FP, we may conclude that $(\widehat{\theta}, \widehat{\mathfrak{R}}, \gamma) \models \phi[(h \uparrow chan(FP))/h, (R \uparrow chan(FP))/R]$. Hence, by substitution lemma b) and d), $(\widehat{\theta} \uparrow chan(FP), \widehat{\mathfrak{R}} \uparrow chan(FP), \gamma) \models \phi$ (6).
Trivially, $chan(\widehat{\theta} \uparrow chan(FP)) \subseteq chan(FP)$ (7).
It is also obvious that $(\widehat{\mathfrak{R}} \uparrow chan(FP)) \uparrow chan(FP) \ = \ \widehat{\mathfrak{R}} \uparrow chan(FP)$ (8).
By results (7), (8), (3), (4), (6), and the fact that ϕ is precise for FP, we may conclude that $(\widehat{\theta} \uparrow chan(FP), \widehat{\mathfrak{R}} \uparrow chan(FP),) \in \mathcal{O}[\![FP]\!]$ (9).
By (5) and the correspondence lemma, for all θ_0 and $\mathfrak{R}_0$
$(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, (\gamma : (s, N) \mapsto (\widehat{\theta}, \widehat{\mathfrak{R}}))) \models \chi[\, s/h_{old}, N/R_{old} \,]$. By substitution lemma a) and c) we obtain $(\widehat{\theta}, \theta, \widehat{\mathfrak{R}}, \mathfrak{R}, (\gamma : (s, N) \mapsto (\widehat{\theta}, \widehat{\mathfrak{R}}))) \models \chi$, and thus, since s and N do not occur free in χ, $(\widehat{\theta}, \theta, \widehat{\mathfrak{R}}, \mathfrak{R}, \gamma) \models \chi$. Since χ is a failure hypothesis, we may conclude that $(\widehat{\theta}, \theta, \widehat{\mathfrak{R}}, \mathfrak{R}, \gamma) \models \chi[(h_{old} \uparrow chan(FP))/h_{old}, (R_{old} \uparrow chan(FP))/R_{old}]$. By substitution lemma a) and c) $(\widehat{\theta} \uparrow chan(FP), \theta, \widehat{\mathfrak{R}} \uparrow chan(FP), \mathfrak{R}, \gamma) \models \chi$ (10).
By (9), (10), (7), and (8), $(\theta, \mathfrak{R}) \in \mathcal{O}[\![FP \wr \chi]\!]$.

iii) Follows from the fact that, since ϕ is precise for FP,
$\phi \rightarrow \phi[(h \uparrow chan(FP))/h, (R \uparrow chan(FP))/R]$, the fact that, since χ is a failure hypothesis, $\chi \rightarrow \chi[(h \uparrow chan(FP))/h, (R \uparrow chan(FP))/R]$, and the fact that $chan(FP \wr \chi) = chan(FP)$.

□

In this series appeared:

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

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

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

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

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

93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.
93/16	H. Schepers J. Hooman	A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
93/17	D. Alstein P. van der Stok	Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
93/18	C. Verhoef	A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
93/19	G-J. Houben	The Design of an Online Help Facility for ExSpect, p.21.
93/20	F.S. de Boer	A Process Algebra of Concurrent Constraint Programming, p. 15.
93/21	M. Codish D. Dams G. Filé M. Bruynooghe	Freeness Analysis for Logic Programs - And Correctness?, p. 24.
93/22	E. Poll	A Typechecker for Bijective Pure Type Systems, p. 28.
93/23	E. de Kogel	Relational Algebra and Equational Proofs, p. 23.
93/24	E. Poll and Paula Severi	Pure Type Systems with Definitions.