EINDHOVEN
UNIVERSITY OF
TECHNOLOGY

Fault tolerance and timing of distributed systems : compositional specification and verification

Citation for published version (APA):
Schepers, H. J. J. H. (1994). *Fault tolerance and timing of distributed systems : compositional specification and verification*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science]. Technische Universiteit Eindhoven. https://doi.org/10.6100/IR417296

DOI:
10.6100/IR417296

Document status and date:
Published: 01/01/1994

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

Please check the document version of this publication:

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

Link to publication

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

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

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 04. Oct. 2023

Fault Tolerance and Timing of Distributed Systems

Compositional specification and verification

Henk Schepers

Fault Tolerance and Timing of Distributed Systems

Compositional specification and verification

Fault Tolerance and Timing of Distributed Systems

Compositional specification and verification

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven,
op gezag van de Rector Magnificus, prof.dr. J.H. van Lint,
voor een commissie aangewezen door het College van Dekanen
in het openbaar te verdedigen op
donderdag 26 mei 1994 te 16.00 uur

door

HENDRIK JAN JOZEF HUBERTUS SCHEPERS

geboren te Heerlen

druk: wibro dissertatiedrukkerij, helmond

Dit proefschrift is goedgekeurd door de promotoren

prof.dr.dipl.-ing. D.K. Hammer

en

prof.dr. M. Joseph

en de copromotor

dr. R.T. Gerth

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Schepers, Hendrik Jan Jozef Hubertus

Fault tolerance and timing of distributed systems :
compositional specification and verification / Hendrik
Jan Jozef Hubertus Schepers. - Eindhoven : Eindhoven
University of Technology
Thesis Eindhoven. - With index, ref. - With summary in
Dutch.
ISBN 90-386-0423-8
Subject headings: fault tolerance / distributed systems.

This work was sponsored by STW under grant number NWI88.1517:
'Fault Tolerance: Paradigms, Models, Logics, Construction'.

To Hetty,
Caerbannog,
and my other family.

Acknowledgements

I would like to thank the members of STW project 'Fault Tolerance: Paradigms, Models, Logics, Construction', especially Job Zwiers, for their comments on presentations of my research and stimulating discussions.

The presentation of my ideas has gained a lot from Mathai Joseph's many valuable comments on draft versions of my papers. I am grateful to Flaviu Cristian for his continuous interest in my research.

My colleagues at the Eindhoven University of Technology are thanked for the very pleasant atmosphere at work. The collaboration with Jozef Hooman has considerably contributed to the development of the basic formalism of this thesis. Collaborating with Rob Gerth resulted in the framework that is presented in Chapter 5. The formalism described in Chapter 4 is a result of joint work with Jos Coenen.

I am obliged to Rob Gerth, Dieter Hammer, Mathai Joseph, Willem-Paul de Roever, Peter van der Stok and Job Zwiers for their comments on a draft version of this manuscript. They provided many suggestions for improvements.

Contents

Chapter 1

Introduction

A *distributed system* consists of a number of physically separated computing components that work on a common goal using their private storage, and, when needed, communicate by explicit message passing. An example is a banking system consisting of a large number of terminals in the various different branches and point-of-sale terminals in the streets. Whenever a customer wants to withdraw money from his account, the system must check for a positive balance with the appropriate data base. If money is indeed withdrawn, the customer's account is updated accordingly.

A *real-time* system is a system whose correct functioning depends crucially on the timing of its actions. An example of a real-time distributed system is the anti-lock braking system in a car. Many sensors provide input to the system, and a prompt reaction to any of the wheel rotation sensors signaling a dangerously low rotation speed is imperative to avoid the wheel starting to slip. Moreover, if the system responds to the signals of one sensor significantly faster than to those of another the possibility exists that the car pulls to the left or the right.

A *failure* occurs when the behaviour of a system is abnormal, that is, deviates from that required by its specification [RLT78]. The failure of a component appears to the system as a *fault*. Notice that there is no conceptual difference between 'fault' and 'failure': they are merely used to distinguish the cause from the consequence. According to Laprie (cf. [Laprie85]) *fault tolerance* is the property of a system "to provide, by redundancy, service complying with the specification in spite of faults having occurred or occurring".

Faults are usually classified according to the specific aspects of the specification they violate, for instance timing faults. If it is possible to deduce from assertions about a component's behaviour that some fault has occurred, we call that fault detectable. Different fault classes arise from the assumptions about the correctness of the behaviour with respect to the various specification aspects, and, in case that behaviour is not assumed to be correct, the detectability of such faults.

In a fault tolerant system, three forms of behaviour are distinguished: normal, exceptional and catastrophic [LA90]. *Normal* behaviour is the behaviour that conforms to the specification. The discriminating factor between exceptional and catastrophic behaviour is the *failure hypothesis* which stipulates how faults affect the normal behaviour. Relative to the failure hypothesis an *exceptional* behaviour exhibits an abnormality which should be tolerated. A *catastrophic* behaviour has an abnormality that is not required to be tolerable (cf. [RLT78, AL86, LA90]). Consider, for instance, the failure hypothesis that a transmission medium might lose messages. For this medium the corruption of messages is catastrophic. The exceptional behaviour together with the normal behaviour constitutes the *acceptable* behaviour.

In general, the catastrophic behaviour of a component cannot be tolerated by a system. Under a particular failure hypothesis for each of its components, a system is designed to tolerate only the *anticipated* faults. Important for this design is the *fault hypothesis* which, in fact, determines the collection of components that must function correctly during any interval of operation (see, e.g., [Schepers93a] for some design examples).

A *distributed program* is an abstract description of the operations of a distributed system. The behaviour of a component is modeled by a so-called *process*, and a distributed program consists of a collection of processes that work concurrently. Interaction between the processes of a distributed program takes place by means of communication rather than by shared variables. We use a *failure prone process* to model the behaviour of an unreliable component.

It is difficult to prove the properties of a distributed program 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. This thesis is concerned with the specification and verification of fault tolerant real-time distributed systems.

The *reliability* of a system is usually defined as the probability that the system functions correctly over a certain period of time, and thus requires a probabilistic, and hence quantitative, specification and verification framework. However, the correctness of a program is a qualitative issue. In this thesis we reason about the reliability of a system in terms of qualitative statements concerning the system's behaviour. More precisely, we reason about properties that hold for all possible executions of a program. It is of little practical value to know that some of the executions tolerate a particular fault.

To specify a system we define an assertion language in which the properties of a system can be described. We use a *correctness formula* of the form P **sat** ϕ to express that the process P satisfies the property ϕ. Such a formula expresses that all executions of P satisfy ϕ. To verify that a process satisfies such a specification we present a *proof theory* (also called a proof system), that is, a collection of axioms and rules by which valid correctness formulae can be deduced.

A proof system is called *compositional* if the specification of a compound process can be deduced from specifications of its constituent parts without any further information about the internal structure of those components. In other words, a compositional proof theory allows reasoning with the specifications of processes without considering their implementation and the precise nature and occurrence of faults in such an implementation. In a compositional proof system every process can be developed in isolation. Moreover, it supports top-down program design where, to master the complexity, a program is decomposed into a number of smaller ones; in a compositional framework such design steps can be individually verified.

Apart from side conditions, in this thesis a *proof rule* has the form

$$\frac{\ldots,\ P_i \ \mathbf{sat}\ \phi_i,\ \ldots,\ \xi_i \ \rightarrow\ \eta_i,\ \ldots}{P\ \mathbf{sat}\ \phi}$$

where P and P_i are process terms and ϕ, ϕ_i, ξ_i and η_i are assertions. The interpretation of such an *inference rule* is that if the formulae above the line have been derived then the formula below the line may be concluded: if, for all i, P_i **sat** ϕ_i and $\xi_i \rightarrow \eta_i$ then P **sat** ϕ.

In particular, we investigate whether an existing compositional proof theory for reasoning about the normal behaviour of a system can be adapted to deal with its acceptable behaviour. To do so, we formalize a failure hypothesis as a relation between the normal and the acceptable behaviour of a system. Such a relation enables us to abstract from the precise nature and occurrence of a fault and to focus on any abnormal behaviour it causes. For a failure hypothesis χ we introduce the construct $P \wr \chi$ (read "P under χ") to indicate execution of process P under the assumption χ. Our approach allows a general treatment of paradigms for fault tolerance because it supports a modular treatment of acceptable behaviour: the acceptable behaviour of the process P under the failure hypothesis χ is the normal behaviour of the failure prone process $P \wr \chi$.

We consider networks of processes that communicate synchronously via directed point-to-point channels. Synchronous communication means that either the sender or the receiver has to wait until a partner is available. In the case of asynchronous communication a message can always be sent without delay but in effect it must be buffered until it can be delivered to the receiver. Thus, asynchronous communication can be modeled by synchronous communication by introducing the (infinite) buffer as an explicit process.

As mentioned above, processes do not share variables. In this thesis we focus on the formalization of fault tolerance in relation to concurrency. We abstract from the internal states of processes and concentrate on the input and output behaviour that is observable at their interface. So, in our proof theory we do not deal with the sequential aspects of processes and instead use a simple compositional formalism to reason about the properties of networks of processes. Termination and *divergence*, the situation where a process appears to be doing nothing because it has entered an infinite loop in which no communication command occurs, are not observable in our framework.

In this thesis we reason about both safety and liveness properties. In the absence of the factor time, a *safety* property expresses that "nothing bad will happen" whereas a *liveness* property expresses that "eventually something good will happen" [Lamport83]. Consider, for instance, a simple 1-place first-in first-out buffer B that has two observable channels *in* and *out*, with the obvious interpretation. Typical safety properties of B are "if there is a communication on *out* then the communicated value is equal to the most recently communicated value on *in*" and "the number of *out* communications is equal to or one less than the number of *in* communications". Observe that a safety property does not express that something must happen: it is trivially satisfied if nothing happens. A characteristic liveness property of B is "after an input eventually output is produced". To prove liveness properties, often fairness conditions such as "in an infinite execution communication on c occurs infinitely often" are enforced.

1.1 Tolerating faults

As mentioned before, fault tolerance is concerned with providing a specified service, even in the presence of faults. Practical fault tolerance, however, only provides protection against those faults that had been anticipated during the design of the system. Either way, fault tolerance depends upon the effective deployment and utilization of redundancy[1].

The most rigorous way to tolerate a fault is to use so much redundancy that it can be masked (see, e.g., [Krol91]), for instance the triple modular redundancy paradigm presented in Section A.3. But this kind of redundancy is generally too expensive.

If faults cannot be masked, then our first concern is how to identify an anticipated fault (*fault detection*). Before the system can be allowed to continue to provide its service, *fault diagnosis* must be applied and the fault's — unwanted — consequences must be undone. Leaving incorrect implementations out of consideration, the fault diagnosis must identify the components that are responsible for the fault and also whether that fault is transient or permanent.

If the fault is only transient, its consequences can be undone by simply restarting the system[2], i.e. by putting it in some initial state, or, in case a valid system state is regularly recorded as a checkpoint, by bringing the system back to its last checkpoint and then continuing operation from that state. This technique is called backward recovery, and it allows actions to be atomic [Lomet77]: they are either executed completely or not at all. Manipulating the

[1] In the literature there is a classification by what kind of element (for instance component and information) is replicated. This classification, however, is not orthogonal (for instance component redundancy also means information redundancy).

[2] This only helps, of course, if the application allows the involved delay; for time-critical applications this is often not the case.

current erroneous state to produce a valid new state is called forward recovery. Once taken to a consistent state the system can continue to provide its service.

If the fault is not transient but permanent the system needs repair first. If the faulty component can be replaced, the system can deliver its service without modification; otherwise, other components must take over the faulty component's tasks in addition to their own, and this may lead to a degradation of the service in case not all the tasks can be fulfilled. *Graceful* degradation allows as many tasks as possible to be still accomplished. Replacing a faulty component can be done either physically or logically by means of *reconfiguration*, where a faulty component is taken out of action and a spare, already present in the system, is put into service.

In Appendix A we present and analyze a few paradigms that are typical for fault tolerance. In particular, we qualitatively investigate under what conditions a particular paradigm successfully tolerates a given class of faults.

1.2 Overview of this thesis

This thesis is organized as follows. In Chapter 2 we present, after discussing existing formal methods for fault tolerance, our approach of formalizing the failure hypothesis of a process as a relation between the normal and the acceptable process behaviour. This method was first introduced in [Schepers93b].

The basic formalism, which is the result of joint work with Jozef Hooman [SH93a, SH93b], is presented in Chapter 3. To emphasize the essence of our approach, we do not consider deadlock and the timing of actions, and restrict ourselves to the specification and verification of safety properties of fault tolerant distributed systems. A process is in a state of *deadlock* if it is blocked in a communication that will never occur. Such a situation is possible because, in the case of synchronous communication, communication partners are each dependent on the other for their respective completion. Because safety properties are properties that can be falsified by finite observations (see for instance [Zwiers89]), our basic theory is based on a finite trace model.

Based on the formalism of Chapter 3, a compositional refinement theory is presented in Chapter 4. This theory is a result of research carried out together with Jos Coenen [SC94].

In Chapter 5 we introduce time into the formalism, to allow reasoning about properties of fault tolerant real-time systems. Then, the above mentioned characterization of safety and liveness properties is no longer appropriate (as indeed mentioned in [Lamport83]). Consider, for instance, a transmission medium that accepts messages via a channel *in* and relays them to a channel *out*. The real-time property "after a message is input to the medium via *in* it is output via *out* within 5 seconds" is a safety property, because it can be falsified 5 seconds after an *in* communication. Note, however, that it expresses that something must happen. Hence, by adding time, the class of safety properties also includes real-time properties, and, consequently, the importance of liveness and fairness decreases. 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. This work is based on joint research with Rob Gerth [SG93]. Besides real-time properties, the extended model allows liveness issues and deadlock to be considered.

The formalism of Chapter 5 assumes that each process has its own processor. But complex programs are typically executed on systems whose limited resources are shared according to some scheduling discipline. In such a case the order of execution is determined on the basis of the priorities of the various actions. In the final stage of the development of our proof theory the model is generalized to facilitate multiprogramming. To do so, the blocking and deblocking related to a synchronous communication are made explicit, and the infinite traces mentioned above are further decorated with timed histories of both the processor occupation and the outstanding requests. The resulting theory, which will appear in [Schepers94], does not require priorities to be fixed. In particular, it is possible to specify how priorities depend on the time already spent waiting for the processor.

Conclusions and suggestions for future research appear in Chapter 7.

Chapter 2

How to Characterize the Effects of Faults and Schedulers

A number of formal methods for fault tolerance have been proposed in the literature. Much of the, by now classical, work on the formalization of fault tolerance is state based. In the state machine approach the output of several instantiations of a program, each running on a distinct processor, is compared. Lamport's original description [Lamport78] dealt with fault-free environments only; for a survey of the efforts to generalize the state machine approach to deal with faults see [Schneider90]. A well-known application of the state machine approach is the implementation of fail-stop processors [SS83].

In layered architectures the exception handling concept (see e.g. [LA90]) is popular: a layer that provides service to some upper level layer raises an exception to signal that upper level layer that a problem occurred as a result of which the requested service could not be provided. The upper level layer contains handlers which deal with such exceptions. In a proof system based on Hoare triples (see [Hoare69]) one reasons about correctness formulae of the form $\{p\}S\{q\}$ where S is a program, and p and q are assertions expressed in a first order language. Informally, the triple $\{p\}S\{q\}$ means that if execution of S is started in a state satisfying p, and if S terminates, then the final state satisfies q. Cristian [Cristian85] uses Hoare logic to make the normal and exceptional domain of execution explicit by partitioning the initial state space into disjoint subspaces for normal and exceptional behaviour, and providing a separate specification for each part. Started in the normal subspace the program terminates normally, but started in the exceptional subspace the program terminates exceptionally, that is, by raising an exception.

Example 2.1 (Cristian's approach) Consider the following procedure to read the contents b of the block starting at address a on disk d

$$\mathbf{proc}\ \mathrm{READ}\ (d : \mathrm{disk}; a : \mathrm{address}; b : \mathrm{block}; u : \mathrm{bool});$$

where return code u (meaning: undefined block) is true if, and only if, the block starting at address a has a parity error. Under the convention that only the addresses of the blocks that have no parity error are contained in domain $dom(d)$ of disk d a successful read operation can be specified as follows:

$$\{\ a \in dom(d)\ \}\ \mathrm{READ}\ (d, a, b, u);\ \{\ b = d(a)\ \wedge\ \neg u\ \}$$

and a non-successful read operation as follows:

$$\{\ a \notin dom(d)\ \}\ \mathrm{READ}\ (d, a, b, u);\ \{\ u\ \}$$

△

Save processor crashes, only the effects of the faults that occurred before the invocation of the program are accounted for. Note that, in terms of Example 2.1, the specification $\{$ true $\}$ READ (d, a, b, u); $\{\ b = d(a)\ \vee\ u\ \}$ is trivially satisfied by any process that just raises the exception. In [Coenen93] deontic logic is proposed to overcome this lazy programmer paradox. All the same an exception-based approach is inadequate to reason about the behaviour of the bottom layer: a corrupted message, for instance, just appears at the physical interface.

In the formalisms of [JH87, JMS87] the execution of a process restarts as soon as a fault occurs. Hence, a failure prone execution of a process P consists of a number of partial executions of P that end in failure followed by a final and complete execution. The incorporation of checkpointing and backward recovery (see Section 1.1) into a program has been investigated in [LJ93, PJ93].

Processes that crash are investigated in [Peleska91]. More precisely, a dual computer system is proved correct. Such a system contains two replicas of the crash prone process, called master and slave. The slave shadows the master and takes over if and when the master crashes.

The formalism proposed in [CH93] allows a program to exhibit arbitrary behaviour after a fault occurs. This approach results in conditional specifications: a process behaves according to its specification as long as no faults have occurred. Fault tolerance is proved by virtue of the system's fault hypothesis and the available redundancy. A similar approach can be found in [CdeR93]. These approaches are not satisfactory in case the effects of faults cannot be masked. For instance, when verifying a system or protocol which employs an error detecting code (see Section A.4) it is crucial to be able to express that even in case of corruption one valid codeword is not changed into another.

In [Weber87] Weber sketches a formalism which takes the effects of faults on the process behaviour into account.

Example 2.2 (Weber's approach) The events from which the histories of a file system are constructed are 'read–file *data*' and 'write–file *data*'. The history ⟨write–file *contents*, read–file *contents*⟩ is obviously a valid trace of the file system. The history ⟨write–file *contents*, read–file *garbage*⟩, where *contents* and *garbage* are different, should not be an admissible trace of the system. Using the designated symbol '†' to denote the occurrence of a fault, the history ⟨write–file *contents*, †, read–file *garbage*⟩, on the other hand, is again a valid trace of the system. △

In [Nordahl93] the normal behaviour of a system S (characterized by the specification $S_{original}$) is distinguished from it's exceptional behaviour (characterized by the 'failure mode' S_f). However, unlike what one would expect in a compositional framework, S_f cannot be derived from $S_{original}$.

In this thesis we investigate how, based on a particular failure hypothesis, the set of behaviours that characterize a process must be expanded. To this end a failure hypothesis is formalized as a relation between the normal behaviour and the acceptable behaviour. Hence, in case we allow faults the distinction between normal and exceptional behaviour disappears. This approach results in a proof rule by which a specification of the acceptable behaviour can be obtained from the specification of the normal behaviour and a predicate characterizing the failure hypothesis. The method allows a modular treatment of acceptable behaviour: the acceptable behaviour of the process P under the failure hypothesis χ is the normal behaviour of the failure prone process $P \wr \chi$ (read "P under χ").

In this thesis we formalize the behaviour of a process by using traces, or histories, which record the communications along the observable channels of the process. Abstracting from the timing of computations, we represent the synchronous communication of value v on channel c by a pair (c, v). An untimed history h is a finite sequence $\langle (c_1, v_1), \ldots, (c_n, v_n) \rangle$. Then, a possible history h of the process *Square*, which alternately inputs an integer via the observable channel *in* and outputs its square via the observable channel *out* (see Figure 2.1), may be $\langle (in, 1), (out, 1), (in, 3), (out, 9) \rangle$.

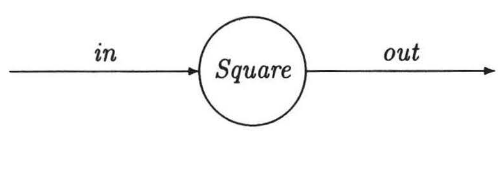

Figure 2.1: Process *Square*

Under the hypothesis that *Square*'s output channel may transiently be stuck at zero, we might observe, instead of the trace $\langle (in, 1), (out, 1), (in, 3), (out, 9) \rangle$, the trace $\langle (in, 1), (out, 1), (in, 3), (out, 0) \rangle$. However, we should not observe the sequence $\langle(in, 1), (out, 1), (in, 3), (out, 15217)\rangle$. Then, a natural way to abstract from the precise nature and occurrence of faults is to formalize a failure hypothesis as a relation between the set of normal behaviours and the set of acceptable behaviours. To do so, a failure hypothesis is characterized by a predicate, expressed in a first order assertion language, whose free variables are h and h_{old}. The interpretation is such that h_{old} denotes a normal behaviour, whereas h denotes a behaviour that is acceptable with respect to the failure hypothesis under discussion.

The relation *StuckAtZero* corresponding to the stuck at zero hypothesis mentioned above is characterized by the fact that

- with respect to the number of *in* and *out* communications h_{old} and h are equally long,
- the order of *in* and *out* communications as recorded by h_{old} is preserved by h,
- the ith input value as recorded by h equals the ith input value as recorded by h_{old}, and
- the ith output value as recorded by h equals the ith output value as recorded by h_{old}, or it is equal to zero.

Example 2.3 (Stuck at zero) The before mentioned relation *StuckAtZero* relates, for instance, trace $h_{old} = \langle(in, 1), (out, 1), (in, 3), (out, 9)\rangle$ to trace $h = \langle(in, 1), (out, 1), (in, 3), (out, 0)\rangle$. Notice that *StuckAtZero* does not relate, for instance, trace $h_{old} = \langle(in, 1), (out, 1), (in, 3), (out, 9)\rangle$ and trace $h = \langle(in, 1), (out, 1), (in, 3), (out, 15217)\rangle$ △

Multiprogramming leads to a new notion of acceptability, as executions may be interrupted in favour of higher priority tasks and continued later. If we consider the behaviour of a program in case each process has its own processor to be its normal behaviour, then a scheduling strategy can be regarded as just another transformation between this normal behaviour and the behaviour that is acceptable with respect to that strategy in case the processor has to be shared. In essence, scheduling introduces breaks in an execution that would not exist in case each process has its own processor. However, different scheduling strategies do not necessarily lead to different behaviours. The resulting behaviour depends namely primarily on the process behaviour and the processes the processor is shared with. The fact that a scheduling strategy may introduce breaks can sufficiently be accounted for by a straightforward extension of the definition of the normal behaviour. When composing behaviours we then have to make sure a particular scheduling strategy is followed.

Chapter 3

Fault Tolerant Distributed Systems

It is difficult to prove the properties of a distributed system. It requires an extra effort to prove the properties of a distributed system composed of failure prone processes, as such proofs must take into account the effects of faults occurring at any point in the execution of the individual processes. In this chapter we present our basic formalism for fault tolerance in which we model the effects of faults on the externally visible input and output behaviour of a process and let its syntactic interface remain unchanged.

To support top-down program design we wish to reason with the specifications of processes without considering their implementation and the precise nature and occurrence of faults in such an implementation. This implies that we aim at a *compositional* proof theory for fault tolerant distributed systems.

Typically, the correctness of a fault tolerant distributed system does not depend on its (initial) state: the system is initialized while communicating with its environment. We abstract from the internal states of processes and concentrate on the input and output behaviour that is observable at their interface. Especially, in this chapter we only describe the sequence of communications that are performed by the processes. In particular, we focus on the formalization of fault tolerance in relation to concurrency. Also, we do not yet consider the timing of those communications and the enabledness of a process to communicate (so we do not yet reason about deadlock). In our proof theory we do not deal with the sequential aspects of processes and instead use a simple compositional formalism to verify properties of networks of processes.

Our basic framework is restricted to the specification and verification of *safety* properties of fault tolerant distributed systems. Safety properties are important for reliability because, in the characterization by Lamport [Lamport83], they express that "nothing bad will happen". Termination and divergence are not observable in our framework. Because we do not consider liveness properties, no fairness assumptions are needed.

Given the classification of behaviour in Chapter 1, we investigate whether an existing compositional proof theory for reasoning about the normal behaviour of a system can be adapted to deal with its acceptable behaviour. To do so, we formalize a failure hypothesis as a relation between the normal and the acceptable behaviour of a system. Indeed, such a relation enables us to abstract from the precise nature and occurrence of a fault and to focus on any abnormal behaviour it causes. It is important to note that our goal is to examine whether it is possible to develop a compositional proof theory based on the idea of transforming behaviours; it is not our aim to find a logic to express failure hypotheses as elegantly as possible.

We consider networks of processes that communicate synchronously via directed channels, each of which connects exactly two processes. Processes do not share variables. We express a property of a process by means of a first order trace logic, using a special variable h to denote the trace, also called history, of the process. Such a history describes the observable behaviour of the process by recording the communications along its visible channels. For instance, a possible history of buffer B is $\langle(in, 1), (out, 1), (in, 3), (out, 3)\rangle$. To express that a process P satisfies a safety property ϕ we use a correctness formula of the form P **sat** ϕ.

Based on a particular failure hypothesis, the set of behaviours that characterize a process is expanded, as has been argued in Chapter 2. To keep such an expansion manageable, the failure hypothesis of a process P is formalized as a predicate, whose only free variables are h_{old} and h, representing a relation between the normal and acceptable histories of P. The interpretation is such that h_{old} represents a normal history of process P, whereas h is an *acceptable* history of P with respect to the failure hypothesis under discussion. For a predicate χ representing a failure hypothesis, we introduce the construct $P \wr \chi$ (read "P under χ") to indicate execution of process P under the assumption χ. This construct enables us to specify *failure prone processes.* Consider again buffer B. Under the hypothesis that, due to faults, values in the buffer are corrupted, which is formalized by some failure hypothesis predicate *Cor*, the history $\langle(in, 1), (out, 1), (in, 3), (out, 3)\rangle$ may be transformed into the history $\langle(in, 1), (out, 1), (in, 3), (out, 5)\rangle$. Then, we would like to prove that failure prone process $B \wr Cor$ still satisfies the property that "the number of *out* communications is equal to or less than the number of *in* communications".

We define the trace semantics of a failure prone process *FP*, and define when correctness formulae of the form *FP* **sat** ϕ are valid. We present a proof theory to verify that a system tolerates the abnormal behaviour of its components to the extent expressed by the failure hypothesis. The proof theory is compositional in the sense that it allows reasoning with the specifications satisfied by failure prone processes while ignoring their implementation details. Further, our approach supports a modular treatment of normal and acceptable behaviour. The usefulness of our method is illustrated by applying it to a triple modular redundant system and the alternating bit protocol, where, indeed, we only use the specifications of the components. Finally, we show that our proof theory is sound and obtain a completeness result by establishing preciseness preservation (see [WGS92]).

This chapter is organized as follows. Section 3.1 introduces the programming language. In Section 3.2 we present the computational model. Section 3.3 defines the denotational semantics. In Section 3.4 we present the assertion language and associated correctness formulae. Section 3.5 presents a proof system for the language of Section 3.1. In Section 3.6 we incorporate failure hypotheses in our formalism. Section 3.7 presents a compositional network proof theory for fault tolerant distributed systems. We illustrate our method by applying it, in Section 3.8, to a triple modular redundant system, and, in Section 3.9, to the alternating bit protocol. In Section 3.10 we prove that the proof theory of Section 3.7 is sound and relatively complete.

3.1 Programming language

In this section we present a programming language, inspired by CSP [Hoare78] and occam [INMOS88], which can be used to define networks of processes that communicate synchronously via directed channels. Channels always connect exactly two processes, that is, two different processes do not both use some channel as input channel or output channel. A channel via which a process communicates with its environment is called an *external* channel of that process. When two processes are composed in parallel their joint channels are said to be the *internal* channels of that composite process. After composing processes we usually no longer wish to observe the internal communications. To conceal communications along internal channels these channels can be hidden.

Let $\mathbb{N}$ denote the set of natural numbers (including 0). Let VAR be a nonempty set of program variables, $CHAN$ a nonempty set of channel names, and let VAL be a denumerable domain of values ($VAL \supseteq \mathbb{N}$). The syntax of our programming language is given in Table 3.1, where $n \in \mathbb{N}$, $n \geq 1$, $\mu \in VAL$, $x \in VAR$, $f \in VAL^n \rightarrow VAL$, $c \in CHAN$, and $cset \subseteq CHAN$.

Table 3.1: Syntax of the programming language

Expression	e	$::=$	$\mu \mid x \mid f(e_1, \ldots, e_n)$
Boolean Expression	b	$::=$	$e_1 = e_2 \mid e_1 < e_2 \mid \neg b \mid b_1 \vee b_2$
Guarded Command	G	$::=$	$[\ \|\!\|_{i=1}^{n}\ b_i \rightarrow P_i\ ]$
Process	P	$::=$	$\mathbf{skip} \mid x := e \mid c!e \mid c?x \mid P_1 \,;\, P_2 \mid$ $G \mid *G \mid P_1 \parallel P_2 \mid P \backslash cset$

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

Atomic statements

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

Compound statements

- P_1 ; P_2 indicates sequential composition: first execute P_1, and continue with the execution of P_2 if and when P_1 terminates.
- Guarded command $[\ \Box_{i=1}^{n}\ b_i \rightarrow P_i\ ]$. If none of the Boolean expressions b_i evaluates to true (i.e., is open) then this guarded command terminates. Otherwise, non-deterministically select one of the b_i that evaluates to true and execute the corresponding statement P_i.
- Iteration $* G$ indicates repeated execution of guarded command G as long as at least one of the guards is open. When none of the guards is open $* G$ terminates.
- $P_1 \parallel P_2$ indicates the parallel execution of the processes P_1 and P_2. This means that P_1 and P_2 execute independently except that the communications along their joint channels require the simultaneous participation of both P_1 and P_2.
- $P \setminus cset$ hides communications along the channels from a set *cset* of internal channels.

For a guarded command $G \equiv [\Box_{i=1}^{n} b_i \rightarrow P_i]$ we define $b_G \equiv b_1 \vee \ldots \vee b_n$. The set of *variables* occurring in process P, notation $var(P)$, is defined in Definition B.1. The set of visible, or observable, *input channels* of process P, denoted $in(P)$, is defined in Definition B.2; the set of observable *output channels* of process P, notation $out(P)$, is defined in Definition B.3.

Definition 3.1 (Observable channels of a process) The set $chan(P)$ of process P's *observable channels* is defined by $chan(P) = in(P) \cup out(P)$. $\diamond$

Example 3.2 (Observable channels of a process)

$$chan(*[\,\mathsf{true} \rightarrow c?x\,;\,[\,x<0 \rightarrow d!0\ [\!]\ x \geq 0 \rightarrow d!1\,]\,]) \;=\; \{c,d\}.$$

△

3.1.1 Syntactic restrictions

To guarantee that channels are unidirectional and point-to-point, that is, connect exactly two processes, we have the following syntactic constraints (for arbitrary expressions e, e_1, and e_2, for any $n \in \mathbb{N}$, $c, c_1, c_2 \in CHAN$, and $x, x_1, x_2 \in VAR$):

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

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

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

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

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

3.2 Model of computation

Since we abstract from the timing of computations, we represent a synchronous communication of value $\mu \in VAL$ along channel $c \in CHAN$ by a pair (c, μ), and define:

(*Channel*) $ch((c,\mu)) = c$;

(*Value*) $val((c,\mu)) = \mu$.

To denote the behaviour of a process P we use a history θ which is a finite sequence (also called a trace) of the form $\langle (c_1,\mu_1), \ldots, (c_n,\mu_n) \rangle$ of length $len(\theta) = n$, where $n \in \mathbb{N}$, $c_i \in chan(P)$, and $\mu_i \in VAL$, for $1 \leq i \leq n$. Such a history denotes the communications of P along its observable channels up to some point in an execution.

Example 3.3 (History) During some execution of process *Square*, which alternately inputs an integer via the observable channel *in* and outputs its square via the observable channel *out*, we may observe the traces $\langle\rangle$, $\langle(in,1)\rangle$, $\langle(in,1),(out,1)\rangle$, $\langle(in,1),(out,1),(in,3)\rangle$, $\langle(in,1),(out,1),(in,3),(out,9)\rangle$ and so on. $\triangle$

Let $\langle\rangle$ denote the empty history, i.e. the sequence of length 0. The concatenation of two histories $\theta_1 = \langle(c_1,\mu_1),\ldots,(c_k,\mu_k)\rangle$ and $\theta_2 = \langle(d_1,\nu_1),\ldots,(d_l,\nu_l)\rangle$, denoted $\theta_1{}^\wedge\theta_2$, is defined as $\langle(c_1,\mu_1),\ldots,(c_k,\mu_k),(d_1,\nu_1),\ldots,(d_l,\nu_l)\rangle$. We use $\theta^\wedge(c,\mu)$ as an abbreviation of $\theta^\wedge\langle(c,\mu)\rangle$.

Definition 3.4 (Traces) Let *TRACE* be the set of traces, that is, the smallest set such that

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

Definition 3.5 (Projection) For a trace $\theta \in TRACE$ and a set of channels $cset \subseteq CHAN$, we define the *projection* of θ onto *cset*, denoted by $\theta\!\uparrow\! cset$, as the sequence obtained from θ by deleting all records with channels not in *cset*. Formally,

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

$\diamond$

Example 3.6 (Projection)

$$\langle(a,3),(c,4),(a,2),(a,3)\rangle\!\uparrow\!\{a\} = \langle(a,3),(a,2),(a,3)\rangle.$$

$\triangle$

We abbreviate $h\!\uparrow\!\{c\}$ as $h\!\uparrow\! c$.

Definition 3.7 (Hiding) Hiding is the complement of projection. Formally, the *hiding* of a set *cset* of channels from a trace $\theta \in TRACE$, notation $\theta\backslash cset$, is defined as

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

$\diamond$

Example 3.8 (Hiding)

$$\langle(a,3),(c,4),(a,2),(a,3)\rangle\backslash\{a\} = \langle(c,4)\rangle.$$

$\triangle$

Definition 3.9 (Channels occurring in a trace) The set of channels occurring in a trace θ, notation $chan(\theta)$, is defined by

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

$\diamond$

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

Definition 3.10 (Length of a trace) The length of trace θ, notation $len(\theta)$, is defined by

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

Definition 3.11 (Prefix) The trace θ_1 is a prefix of a trace θ_2, denoted by $\theta_1 \preceq \theta_2$, if, and only if, there exists a trace θ_3 such that $\theta_1{}^\wedge\theta_3 = \theta_2$. $\diamond$

If $\theta_1 \preceq \theta_2$ and $\theta_1 \neq \theta_2$ then θ_1 is a *strict* prefix of θ_2, notation $\theta_1 \prec \theta_2$.

3.3 Denotational semantics

In semantics, one is concerned with defining the meaning of programs in terms of a mathematical model. When developing a compositional proof system a convenient starting point is the formulation of a denotational, and hence compositional, semantics. In such a semantics the meaning of a statement is defined without any information about the environment in which it will be placed. Consequently, the semantics of a statement in isolation characterizes all potential executions of the statement, regardless of its environment. When composing statements, the semantic operators select the appropriate execution sequences.

In this section we define a denotational semantics for the programming language of Section 3.1 in terms of the finite trace model presented in the previous section.

Definition 3.12 (States) Define the set $STATE$ of states as the set of mappings σ which map a variable $x \in VAR$ to a value $\sigma(x) \in VAL$. $\diamond$

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

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

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

- $\mathcal{E}[\![f(e_1,\ldots,e_n)]\!](\sigma) = f(\mathcal{E}[\![e_1]\!](\sigma),\ldots,\mathcal{E}[\![e_n]\!](\sigma))$, where the function f on the right-hand side of the equality sign is the interpretation of the function f on the left-hand side.

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

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

Using the finite traces that have been defined in Section 3.2 we can denote the finite, i.e., terminating, computations of a program, and approximate its infinite executions. This is justified since [Scott70] in this chapter we only deal with safety properties (see for instance [Zwiers89]), and since (the semantics of) our programming language is such that an infinite trace represents a behaviour of the process if, and only if, all its prefixes do.

Example 3.13 Consider the process $P_1 \equiv *\,[x > 0 \;\rightarrow\; c!1; x := x-1]; d!0$. Because the variable x is not initialized, the traces that can be observed up to any point in the execution of P_1 are $\langle\rangle$, $\langle(d,0)\rangle$, $\langle(c,1)\rangle$, $\langle(c,1),(d,0)\rangle$, $\langle(c,1),(c,1)\rangle$, $\langle(c,1),(c,1),(d,0)\rangle$, etc. In this respect the process P_1 is equivalent to the process $P_2 \equiv *\,[x > 0 \;\rightarrow\; c!1; x := x-1]; [\mathsf{true} \rightarrow d!0 \;[\!]\; \mathsf{true} \rightarrow \mathbf{skip}]$. On the basis of these observations, the liveness property "eventually there is a communication on d" cannot be verified. △

Example 3.14 Consider the process

$$P \equiv [\mathsf{true} \rightarrow *[x > 0 \rightarrow c!1; x := x-1]; *[\mathsf{true} \rightarrow d!0]\,[\!]\,\mathsf{true} \rightarrow *[\mathsf{true} \rightarrow c!1]].$$

Under the fairness condition "in an infinite execution communication on c occurs infinitely often" the infinite executions of P satisfy the safety property "no communication on d occurs". This cannot be conclude from the finite approximations. △

Consequently, we can deal with so-called *reactive* processes (see [HP85]) that are typically non-terminating and that have an intense interaction with their environment. To reason about the input and output behaviour of both terminating and non-terminating processes, we want to observe, for any execution of a process P,

- the initial state of P,
- the sequence of communications performed by P, and,
- for a terminating computation of P, the final state of P.

In general, the semantics of a program is a set of denotations representing all finite observations of the program during its possible executions. At any point before termination we only observe the history of communications that have been performed by the process, using a special state $\perp$ (read "bottom") to indicate that the program has not yet terminated.

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

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

Since we abstract from the timing of actions we cannot distinguish between an unfinished and a deadlocked computation (there is no way of telling whether one has been observing the process infinitely long). Thus, the semantics of a program is a *prefix closed* set of denotations in the sense that if $(\sigma_0, \theta, \sigma)$ is an element of the semantics of a program then so is $(\sigma_0, \theta, \perp)$, and, furthermore, if $(\sigma_0, \theta, \perp)$ is an element of that semantics then so is $(\sigma_0, \widehat{\theta}, \perp)$, for all $\widehat{\theta} \preceq \theta$. Below we define the meaning of an atomic process as the smallest prefix closed set containing its terminating executions.

Definition 3.15 (Prefix closure) For a given set O consisting of triples from $STATE \times TRACE \times STATE_{\perp}$, the operator PC expresses its *prefix closure.*

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

$\diamond$

Example 3.16 (Prefix closure)

$$PC(\{(\sigma_0, \langle(c,1)\rangle, \sigma)\}) = \{(\sigma_0, \langle\rangle, \perp), (\sigma_0, \langle(c,1)\rangle, \perp), (\sigma_0, \langle(c,1)\rangle, \sigma)\}.$$

$\triangle$

Definition 3.17 (Variant of a function) The *variant* of a function f with respect to a variable x and a value ϑ, notation $(f : x \mapsto \vartheta)$, is defined below. The variant of an undefined function is again undefined:

- if $f = \perp$ then $(f : x \mapsto \vartheta) = \perp$, and
- if $f \neq \perp$ then $(f : x \mapsto \vartheta)(y) = \begin{cases} \vartheta & \text{if } y \equiv x, \\ f(y) & \text{if } y \not\equiv x. \end{cases}$

$\diamond$

Example 3.18 (Variant of a state) Consider a state σ such that $\sigma(x) = 3$ and $\sigma(y) = 4$. Then $(\sigma : x \mapsto 7)(x) = 7$, but still $(\sigma : x \mapsto 7)(y) = 4$. $\triangle$

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

- A **skip** statement does not communicate and terminates with a final state that is equal to its initial state.

$$\mathcal{M}[\![\mathbf{skip}]\!] \;=\; PC\ (\ \{\ (\sigma_0, \langle\rangle, \sigma_0) \mid \sigma_0 \in STATE\ \}\ ).$$

- The assignment $x := e$ does not communicate and terminates with a final state that is equal to its initial state, except that the value of variable x is replaced by the value that e had in its initial state.

$$\mathcal{M}[\![x := e]\!] \;=\; PC\ (\ \{\ (\sigma_0, \langle\rangle, (\sigma_0 : x \mapsto \mathcal{E}[\![e]\!](\sigma_0))) \mid \sigma_0 \in STATE\ \}\ ).$$

- A computation of the output statement $c!e$ communicates the value that e had in its initial state and terminates with a final state that is equal to that initial state. Note that, because the semantics is prefix closed, the observations of a deadlocked output statement are included automatically.

$$\mathcal{M}[\![c!e]\!] \;=\; PC\ (\ \{\ (\sigma_0, \langle(c, \mathcal{E}[\![e]\!](\sigma_0))\rangle, \sigma_0) \mid \sigma_0 \in STATE\ \}\ ).$$

- A computation of the input statement $c?x$ communicates a value and terminates with a final state that is equal to its initial state, except that the value of variable x is replaced by the communicated value. The semantics contains a triple for each value that can be received. Again, the observations of a deadlocked input statement are generated by the application of PC.

$$\begin{aligned}\mathcal{M}[\![c?x]\!] \;=\; PC\ (\ \{\ (\sigma_0, \theta, \sigma) \mid\ & \sigma_0 \in STATE \text{ and there exists a value} \\ & \mu \in VAL \text{ such that } \theta = \langle(c, \mu)\rangle \text{ and} \\ & \sigma = (\sigma_0 : x \mapsto \mu)\ \}\ ).\end{aligned}$$

- An execution of $P_1\ ;\ P_2$ is a non-terminating execution of P_1 or a terminating execution of P_1 followed by some execution of P_2.

$$\begin{aligned}\mathcal{M}[\![P_1\ ;\ P_2]\!] = \;& \{\ (\sigma_0, \theta, \bot) \mid (\sigma_0, \theta, \bot) \in \mathcal{M}[\![P_1]\!]\ \} \\ \cup\ & \{\ (\sigma_0, \theta_1{}^\wedge\theta_2, \sigma) \mid \text{there exists a } \sigma_1 \neq \bot \text{ such that} \\ & \qquad (\sigma_0, \theta_1, \sigma_1) \in \mathcal{M}[\![P_1]\!] \text{ and} \\ & \qquad (\sigma_1, \theta_2, \sigma) \in \mathcal{M}[\![P_2]\!]\ \}.\end{aligned}$$

- If in the initial state no guard is open, i.e., true, a Boolean guarded command terminates with final state equal to its initial state. Otherwise, the process corresponding to one of its open guards (non-deterministically chosen) is executed.

$$
\begin{aligned}
\mathcal{M}[\![[\,\square_{i=1}^{n} b_i \rightarrow P_i]]\!] = \quad & PC\ (\ \{\ (\sigma_0, \langle\rangle, \sigma_0) \mid \neg\mathcal{B}[\![b_1 \vee \ldots \vee b_n]\!](\sigma_0)\ \}\ ) \\
\cup\ & PC\ (\ \{\ (\sigma_0, \theta, \sigma) \mid \text{there exists a } k \in \{1, \ldots, n\} \\
& \qquad \text{such that } \mathcal{B}[\![b_k]\!](\sigma_0) \text{ and} \\
& \qquad (\sigma_0, \theta, \sigma) \in \mathcal{M}[\![P_k]\!]\ \}\ ).
\end{aligned}
$$

- If no guard is open in the initial state, $*\,G$ terminates with a final state equal to its initial state. Otherwise, $*\,G$ consists of one or more executions of the *body* G, each starting in a state in which at least one of the guards is open.

$$
\begin{aligned}
\mathcal{M}[\![*\,G]\!] = PC\ (\ \{\ (\sigma_0, \theta, \sigma) \mid\ & \text{there exists a } k \in \mathbb{N} \text{ and a list } (\sigma_0, \theta_1, \sigma_1), \\
& \ldots, (\sigma_{k-1}, \theta_k, \sigma_k) \text{ such that } \theta = \theta_1 \hat{}\ \ldots \hat{}\ \theta_k, \\
& \sigma = \sigma_k, \text{ and for all } i \in \{0, \ldots, k-1\}: \\
& \quad \sigma_i \neq \bot,\ \mathcal{B}[\![b_G]\!](\sigma_i), \\
& \quad (\sigma_i, \theta_{i+1}, \sigma_{i+1}) \in \mathcal{M}[\![G]\!], \\
& \text{and if } \sigma_k \neq \bot \text{ then } \mathcal{B}[\![\neg b_G]\!](\sigma_k)\ \}\ ).
\end{aligned}
$$

- Recall that the semantics of an input statement in isolation includes a triple for all possible values that could have been received. When two processes are composed in parallel, the triples that correspond to the actual values transmitted on the joint channels are selected. Since communication is synchronous a trace θ of the process $P_1 \| P_2$ has the property that $\theta \uparrow chan(P_1)$ and $\theta \uparrow chan(P_2)$ match traces of P_1 and P_2, respectively. Although necessary, this restriction is not sufficient as it allows θ to have records (c, μ) for arbitrary $c \in CHAN - chan(P_1 \parallel P_2)$ and arbitrary $\mu \in VAL$.

Example 3.19 Consider processes $P_1 \equiv a!3; b!4$ and $P_2 \equiv b?x; c!(x+1)$. Consequently, $chan(P_1) = \{a, b\}$ and $chan(P_2) = \{b, c\}$. Consider trace $\theta = \langle (a,3), (b,4), (d,3), (c,5) \rangle$. Although $\theta \uparrow chan(P_1) = \langle (a,3), (b,4) \rangle$ is an admissible trace of P_1 and $\theta \uparrow chan(P_2) = \langle (b,4), (c,5) \rangle$ is a possible sequence of communications of P_2, θ cannot be a trace of $P_1 \parallel P_2$. $\triangle$

We add the condition that $\theta = \theta \uparrow chan(P_1 \| P_2)$. Note that a computation of $P_1 \parallel P_2$ does not terminate until both P_1 and P_2 have terminated. Since P_1 and P_2 do not share variables, the final state of an execution of $P_1 \| P_2$ is a straightforward combination of the final states of the respective executions of P_1 and P_2.

$$\begin{aligned}\mathcal{M}[\![P_1 \parallel P_2]\!] = \{\, (\sigma_0, \theta, \sigma) \mid\ & \text{for } i = 1,2 \text{ there exist } \theta_i \text{ and } \sigma_i \text{ such that} \\ & (\sigma_0, \theta_i, \sigma_i) \in \mathcal{M}[\![P_i]\!], \\ & \text{if } \sigma_1 = \bot \text{ or } \sigma_2 = \bot \text{ then } \sigma = \bot, \text{ and,} \\ & \quad \text{otherwise, for all } x \in \mathit{VAR}, \\ & \quad \sigma(x) = \begin{cases} \sigma_i(x) & \text{if } x \in \mathit{var}(P_i), \\ \sigma_0(x) & \text{if } x \notin \mathit{var}(P_1 \parallel P_2), \end{cases} \\ & \theta \!\uparrow\! \mathit{chan}(P_i) = \theta_i, \text{ and } \theta \!\uparrow\! \mathit{chan}(P_1 \parallel P_2) = \theta \,\}.\end{aligned}$$

- Hiding of (internal) channels causes communications along those channels to be no longer observable but does not affect the internal state of a process.

$$\mathcal{M}[\![P \setminus \mathit{cset}]\!] = \{\, (\sigma_0, \theta \setminus \mathit{cset}, \sigma) \mid (\sigma_0, \theta, \sigma) \in \mathcal{M}[\![P]\!] \,\}.$$

Example 3.20 (Meaning of a process) Consider $(a!3; b!4) \| (b?x; c!(x+1))$. We have

$$\mathcal{M}[\![a!3]\!] = PC(\{(\sigma_0, \langle(a,3)\rangle, \sigma_0) \mid \sigma_0 \in \mathit{STATE}\})$$

and

$$\mathcal{M}[\![b!4]\!] = PC(\{(\sigma_0, \langle(b,4)\rangle, \sigma_0) \mid \sigma_0 \in \mathit{STATE}\}),$$

from which we easily obtain

$$\mathcal{M}[\![a!3\,;\, b!4]\!] = PC(\{(\sigma_0, \langle(a,3),(b,4)\rangle, \sigma_0) \mid \sigma_0 \in \mathit{STATE}\}).$$

Furthermore,

$$\begin{aligned}\mathcal{M}[\![b?x]\!] = PC(\{(\sigma_0, \theta, \sigma) \mid\ & \sigma_0 \in \mathit{STATE} \text{ and there exists a value} \\ & \mu \in \mathit{VAL} \text{ such that } \theta = \langle(b,\mu)\rangle \text{ and} \\ & \sigma = (\sigma_0 : x \mapsto \mu)\})\end{aligned}$$

and

$$\mathcal{M}[\![c!(x+1)]\!] = PC(\{(\sigma_0, \langle(c, \sigma_0(x)+1)\rangle, \sigma_0) \mid \sigma_0 \in \mathit{STATE}\}),$$

or, equivalently,

$$\mathcal{M}[\![c!(x+1)]\!] = PC\left(\left\{ \begin{pmatrix} (\sigma_0 : x \mapsto \mu) \\ \langle(c, (\sigma_0 : x \mapsto \mu)(x) + 1)\rangle \\ (\sigma_0 : x \mapsto \mu) \end{pmatrix} \middle| \begin{matrix} (\sigma_0 : x \mapsto \mu) \in \mathit{STATE} \\ \wedge \\ \mu \in \mathit{VAL} \end{matrix} \right\}\right),$$

lead to

$$\mathcal{M}[\![b?x\,;\, c!(x+1)]\!] = PC\left(\left\{ \begin{pmatrix} \sigma_0 \\ \langle(b,\mu), (c, (\sigma_0 : x \mapsto \mu)(x) + 1)\rangle \\ (\sigma_0 : x \mapsto \mu) \end{pmatrix} \middle| \begin{matrix} \sigma_0 \in \mathit{STATE} \\ \wedge \\ \mu \in \mathit{VAL} \end{matrix} \right\}\right),$$

that is,

$$\mathcal{M}[\![b?x\,;\, c!(x+1)]\!] = PC(\{(\sigma_0, \langle(b,\mu),(c,\mu+1)\rangle, (\sigma_0 : x \mapsto \mu)) \mid \sigma_0 \in \mathit{STATE} \wedge \mu \in \mathit{VAL}\}).$$

Consider a trace θ with $\theta\uparrow\{a,b\} = \langle(a,3),(b,4)\rangle$, $\theta\uparrow\{b,c\} = \langle(b,\mu),(c,\mu+1)\rangle$, and $\theta\uparrow\{a,b,c\} = \theta$. In this case $\mu = 4$, and hence $\theta = \langle(a,3),(b,4),(c,5)\rangle$. Since, obviously, $x \notin var(a!3\ ;\ b!4)$, we obtain

$$\begin{aligned}&\mathcal{M}[\![(a!3\ ;\ b!4) \parallel (b?x\ ;\ c!(x+1))]\!] = \\ &\qquad PC(\{(\sigma_0, \langle(a,3),(b,4),(c,5)\rangle, (\sigma_0 : x \mapsto 4)) \mid \sigma_0 \in STATE\}).\end{aligned}$$

△

Now we can abstract from the internal state of a process.

Definition 3.21 (Traces of a process) The *traces* of a process P, notation $\mathcal{H}[\![P]\!]$, are defined as follows:

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

◇

Example 3.22 (Traces of a process)

$$\begin{aligned}\mathcal{H}[\![(a!3\ ;\ b!4) \parallel (b?x\ ;\ c!(x+1))]\!] = \{\ &\langle\rangle, \\ &\langle(a,3)\rangle, \\ &\langle(a,3),(b,4)\rangle, \\ &\langle(a,3),(b,4),(c,5)\rangle\ \}.\end{aligned}$$

△

The set $\mathcal{H}[\![P]\!]$ represents the normal behaviour of the process P. In Section 3.6 we determine the set $\mathcal{H}[\![P \wr \chi]\!]$ representing the acceptable behaviour of P under the failure hypothesis χ, that is, the normal behaviour of the failure prone process $P \wr \chi$.

3.4 Assertion language and correctness formulae

Assertions are used to express the properties of a program in terms of its observable quantities. Since we abstract from the internal state of a process and focus on the pattern of communications, the only observable quantity is the trace of the process. More precisely, an assertion is a logical function of the communication history of the process. In this thesis 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. Informally, such a correctness formula expresses that any sequence of communications which P may exhibit satisfies ϕ.

Similar to the semantic denotation of traces in Section 3.2, we use in assertions communication record expressions such as (c,μ), with $c \in CHAN$ and $\mu \in VAL$. We have channel expressions, e.g. using the operator *ch* which yields the channel of a communication record, and value expressions, using the operator *val* which yields the value of a communication record, and a number of n-ary functions which remain uninterpreted. To reason about natural numbers, integer expressions include the length operator *len*. We use the empty trace, $\langle\rangle$, traces of one record, e.g. $\langle(c,\mu)\rangle$, as well as the concatenation operator $^\wedge$ and the projection operator $\uparrow$ to create trace expressions. Further, for a trace expression *texp* and an integer expression *iexp* we use $texp(iexp)$ to refer to record number *iexp* of *texp*, provided *iexp* is a positive natural number less than or equal to $len(texp)$. We write $texp[iexp]$ to denote the prefix of *texp* that has length *iexp*. A special variable h is used to refer to the communication history of a process. Then, we can write specifications like $c!2$ **sat** $h\uparrow\{c\} = \langle\rangle \vee h\uparrow\{c\} = \langle(c,2)\rangle$.

In assertions we furthermore use *logical variables* which serve as placeholders for arbitrary values. Let *IVAR*, with typical representatives i, j, k, l, and n, denote the set of logical value variables ranging over $\mathbb{N}$, let *VVAR*, with typical representative v, denote the set of logical value variables ranging over *VAL*, and let *TVAR*, with characteristic element s, be the set of logical trace variables ranging over *TRACE*. Assume that $IVAR \cap VVAR = \emptyset$, $IVAR \cap TVAR = \emptyset$, and $VVAR \cap TVAR = \emptyset$.

The syntax of the assertion language is given in Table 3.2, with $i \in IVAR$, $c \in CHAN$, $\mu \in VAL$, $v \in VVAR$, $f \in VAL^n \to VAL$, $s \in TVAR$, and $cset \subseteq CHAN$. An expression in the assertion language of Table 3.2 does not refer to program variables since we abstract from the internal state of a process.

Table 3.2: Syntax of the assertion language

Integer expression	$iexp$	$::=$	$0 \mid 1 \mid i \mid iexp_1 + iexp_2 \mid iexp_1 \times iexp_2 \mid len(texp)$
Channel expression	$cexp$	$::=$	$c \mid ch(rexp)$
Value expression	$vexp$	$::=$	$\mu \mid v \mid iexp \mid val(rexp) \mid f(vexp_1,\ldots,vexp_n)$
Record expression	$rexp$	$::=$	$(cexp, vexp) \mid texp(iexp)$
Trace expression	$texp$	$::=$	$s \mid h \mid \langle\rangle \mid \langle rexp\rangle \mid texp_1{}^\wedge texp_2 \mid texp\uparrow cset \mid texp[iexp]$
Assertion	ϕ	$::=$	$iexp_1 = iexp_2 \mid iexp_1 < iexp_2 \mid cexp_1 = cexp_2 \mid vexp_1 = vexp_2 \mid vexp_1 < vexp_2 \mid texp_1 = texp_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \exists i \cdot \phi \mid \exists v \cdot \phi \mid \exists s \cdot \phi$

Syntactic Restriction For any occurrence of $\langle(cexp, vexp)\rangle$ in an assertion we require that the term h does not appear in value expression *vexp*. The reason for this restriction will become clear after Lemma 3.51.

Definition 3.23 (Abbreviations)

- $ch(cexp, vexp) \;\equiv\; ch((cexp, vexp))$.
- $val(cexp, vexp) \;\equiv\; val((cexp, vexp))$.
- $texp\uparrow cexp \;\equiv\; texp\uparrow\{cexp\}$.
- $rexp_1 = rexp_2 \;\equiv\; ch(rexp_1) = ch(rexp_2) \;\wedge\; val(rexp_1) = val(rexp_2)$.
- $texp \setminus cset \;\equiv\; texp\uparrow(CHAN - cset)$.
- $last(texp) \;\equiv\; texp(len(texp))$.
- $texp_1 \preceq texp_2 \;\equiv\; \exists s \cdot texp_1{}^\wedge s = texp_2$.

  This expresses that $texp_1$ is a prefix of $texp_2$.
- $texp_1 \preceq^n texp_2 \;\equiv\; \exists s \cdot len(s) \leq n \wedge texp_1{}^\wedge s = texp_2$.

  This denotes that $texp_1$ is a prefix of $texp_2$ which is at most n records shorter.
- $texp_1 \prec texp_2 \;\equiv\; texp_1 \preceq texp_2 \wedge texp_1 \neq texp_2$.

  This denotes that $texp_1$ is a *strict* prefix of $texp_2$.
- $texp_1 \prec^n texp_2 \;\equiv\; \exists s \cdot 1 < len(s) \leq n \wedge texp_1{}^\wedge s = texp_2$.

  This expresses that $texp_1$ is a strict prefix of $texp_2$ which is at most n records shorter.
- $$\begin{aligned} texp_1 \trianglelefteq texp_2 \;\equiv\; \exists s \cdot\;\; & len(s) = len(texp_1) \\ & \wedge \forall i \cdot 1 \leq i < len(s) \;\rightarrow\; val(s(i)) < val(s(i+1)) \\ & \wedge \forall i \cdot 1 \leq i \leq len(s) \;\rightarrow\; texp_2(val(s(i))) = texp_1(i). \end{aligned}$$

  This expression denotes that $texp_1$ is a (not necessarily contiguous) subsequence of $texp_2$ (observe that it implies that $len(texp_1) \leq len(texp_2)$).
- $$\begin{aligned} Val(texp_1) \preceq Val(texp_2) \;\equiv\; & \forall i \cdot 1 \leq i \leq len(texp_1) \\ & \rightarrow\; val(texp_1(i)) = val(texp_2(i)). \end{aligned}$$

  This expresses that the sequence of values in the trace $texp_1$ is a prefix of the sequence of values in the trace $texp_2$.
- $$\begin{aligned} Val(texp_1) \preceq^n Val(texp_2) \;\equiv\; & \;0 \leq len(texp_2) - len(texp_1) \leq n \\ & \wedge\; Val(texp_1) \preceq Val(texp_2). \end{aligned}$$

  This expression denotes that the sequence of values in the trace $texp_1$ is a prefix of the sequence of values in the trace $texp_2$, that the trace $texp_2$ is at least as long as the trace $texp_1$, and that the trace $texp_1$ is at most n records shorter than the trace $texp_2$. ◇

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

Example 3.24 (Medium) Consider the medium M that accepts messages via *in* and delivers them via *out* in first-in first-out order. To specify that M has a capacity of one message, we write

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

$\triangle$

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

- $\mathcal{I}[\![0]\!](\theta,\gamma) = 0$;
- $\mathcal{I}[\![1]\!](\theta,\gamma) = 1$;
- $\mathcal{I}[\![i]\!](\theta,\gamma) = \gamma(i)$;
- $\mathcal{I}[\![iexp_1 + iexp_2]\!](\theta,\gamma) =$
$$\begin{cases} \dagger & \text{if } \mathcal{I}[\![iexp_1]\!](\theta,\gamma) = \dagger \text{ or} \\ & \mathcal{I}[\![iexp_2]\!](\theta,\gamma) = \dagger, \\ \mathcal{I}[\![iexp_1]\!](\theta,\gamma) + \mathcal{I}[\![iexp_2]\!](\theta,\gamma) & \text{otherwise;} \end{cases}$$
- $\mathcal{I}[\![iexp_1 \times iexp_2]\!](\theta,\gamma) =$
$$\begin{cases} \dagger & \text{if } \mathcal{I}[\![iexp_1]\!](\theta,\gamma) = \dagger \text{ or} \\ & \mathcal{I}[\![iexp_2]\!](\theta,\gamma) = \dagger, \\ \mathcal{I}[\![iexp_1]\!](\theta,\gamma) \times \mathcal{I}[\![iexp_2]\!](\theta,\gamma) & \text{otherwise;} \end{cases}$$

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

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

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

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

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

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

- $\mathcal{V}[\![f(vexp_1,\ldots,vexp_n)]\!](\theta,\gamma) =$
$$\begin{cases} \dagger & \text{if } \mathcal{V}[\![vexp_1]\!](\theta,\gamma) = \dagger, \\ & \text{or } \ldots, \text{ or} \\ & \mathcal{V}[\![vexp_n]\!](\theta,\gamma) = \dagger, \\ f(\mathcal{V}[\![vexp_1]\!](\theta,\gamma),\ldots,\mathcal{V}[\![vexp_n]\!](\theta,\gamma)) & \text{otherwise;} \end{cases}$$

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

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

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

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

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

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

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

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

- $\mathcal{T}[\![texp[iexp]]\!](\theta,\gamma) =$
$$\begin{cases} \dagger & \text{if } \mathcal{T}[\![texp]\!](\theta,\gamma) = \dagger \text{ or } len(\mathcal{T}[\![texp]\!](\theta,\gamma)) < \mathcal{I}[\![iexp]\!](\theta,\gamma), \\ \theta & \text{if } \theta \preceq \mathcal{T}[\![texp]\!](\theta,\gamma) \text{ and } len(\theta) = \mathcal{I}[\![iexp]\!](\theta,\gamma). \end{cases}$$

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

- $(\theta,\gamma) \models iexp_1 = iexp_2$ iff $\mathcal{I}[\![cexp_1]\!](\theta,\gamma) = \mathcal{I}[\![cexp_2]\!](\theta,\gamma)$ and $\mathcal{I}[\![cexp_1]\!](\theta,\gamma) \neq \dagger$;
- $(\theta,\gamma) \models iexp_1 < iexp_2$ iff $\mathcal{I}[\![vexp_1]\!](\theta,\gamma) < \mathcal{I}[\![vexp_2]\!](\theta,\gamma)$ and $\mathcal{I}[\![vexp_1]\!](\theta,\gamma) \neq \dagger$ and $\mathcal{I}[\![vexp_2]\!](\theta,\gamma) \neq \dagger$;
- $(\theta,\gamma) \models cexp_1 = cexp_2$ iff $\mathcal{C}[\![cexp_1]\!](\theta,\gamma) = \mathcal{C}[\![cexp_2]\!](\theta,\gamma)$ and $\mathcal{C}[\![cexp_1]\!](\theta,\gamma) \neq \dagger$;
- $(\theta,\gamma) \models vexp_1 = vexp_2$ iff $\mathcal{V}[\![vexp_1]\!](\theta,\gamma) = \mathcal{V}[\![vexp_2]\!](\theta,\gamma)$ and $\mathcal{V}[\![vexp_1]\!](\theta,\gamma) \neq \dagger$;
- $(\theta,\gamma) \models vexp_1 < vexp_2$ iff $\mathcal{V}[\![vexp_1]\!](\theta,\gamma) < \mathcal{V}[\![vexp_2]\!](\theta,\gamma)$ and $\mathcal{V}[\![vexp_1]\!](\theta,\gamma) \neq \dagger$ and $\mathcal{V}[\![vexp_2]\!](\theta,\gamma) \neq \dagger$;
- $(\theta,\gamma) \models texp_1 = texp_2$ iff $\mathcal{T}[\![texp_1]\!](\theta,\gamma) = \mathcal{T}[\![texp_2]\!](\theta,\gamma)$ and $\mathcal{T}[\![texp_1]\!](\theta,\gamma) \neq \dagger$;
- $(\theta,\gamma) \models \phi_1 \wedge \phi_2$ iff $(\theta,\gamma) \models \phi_1$ and $(\theta,\gamma) \models \phi_2$;
- $(\theta,\gamma) \models \neg\phi$ iff not $(\theta,\gamma) \models \phi$;
- $(\theta,\gamma) \models \exists i \cdot \phi$ iff there exists an integer n such that $(\theta,(\gamma : i \mapsto n)) \models \phi$;
- $(\theta,\gamma) \models \exists v \cdot \phi$ iff there exists a value μ such that $(\theta,(\gamma : v \mapsto \mu)) \models \phi$;
- $(\theta,\gamma) \models \exists s \cdot \psi$ iff there exists a trace $\widehat{\theta}$ such that $(\theta,(\gamma : s \mapsto \widehat{\theta})) \models \phi$.

Example 3.25 (Satisfaction) Consider the assertion

$$s \preceq h \;\rightarrow\; s\uparrow\{c\} = \langle\rangle.$$

Since h obtains its value from θ, we have $(\theta,\gamma) \models s \preceq h \rightarrow s\uparrow\{c\} = \langle\rangle$ for any environment γ and trace θ such that if $\gamma(s) \preceq \theta$ then $\gamma(s)\uparrow\{c\} = \langle\rangle$. $\triangle$

Definition 3.26 (Validity of an assertion) An assertion ϕ is *valid*, which we denote by $\models \phi$, if, and only if, for all θ and γ, $(\theta,\gamma) \models \phi$. $\diamond$

For an assertion ϕ Definition B.4 defines the set $chan(\phi)$ of channels such that $c \in chan(\phi)$ if a communication along c might affect the validity of ϕ. For instance, the validity of the assertion $h = \langle\rangle$ is affected by any communication and thus we have $chan(h = \langle\rangle) = CHAN$. On the other hand, the validity of the assertion $(h\uparrow\{c\})^\wedge(d,7) = \langle(d,7)\rangle$ can only be changed by a communication along channel c, although d also occurs in the assertion. Hence, $chan(\phi)$ consists of the channels to which references to h in ϕ are restricted, the so-called *history* channels of ϕ (cf. [Zwiers89, Hooman92]). Note that the value of a logical variable is not affected by any communication.

Observe that the fact that $c \in chan(\phi)$ does not imply that the validity of ϕ indeed depends on communications along c. For instance, according to Definition B.4 $chan(h = \langle\rangle \vee h \neq \langle\rangle) = CHAN$ although the assertion will always be true. However, if some channel is not contained in $chan(\phi)$ then ϕ does not impose any restrictions on communications along that channel. It is this aspect that we are interested in.

We conclude this section by defining when a correctness formula P **sat** ϕ is valid. Since we focus on reliability, we are interested only in properties that hold for all observations.

Definition 3.27 (Validity of a correctness formula) For a process P and an assertion ϕ a correctness formula P **sat** ϕ is *valid*, denoted by $\models P$ **sat** ϕ, if, and only if, for all γ and all $\theta \in \mathcal{H}[\![P]\!]$, $(\theta, \gamma) \models \phi$. $\diamond$

3.5 A compositional proof theory for plain processes

In this section we present a compositional proof theory for the programming language of Section 3.1. The theory consists of an *axiom* for each atomic process and an *inference rule* for each composition operator of the programming language. To reason about the sequential details of processes we use an extended assertion language which includes program variables and a denotation to indicate termination (e.g. [Zwiers89]).

To reason about program variables, we extend the assertion language of Section 3.4 with the value expressions x and x_0. For a variable $x \in VAR$, the term x denotes the value of x in the (non-bottom) *final* state, and the assertion term x_0 denotes the *initial* state value of x. Further, to denote a terminated computation, we add the primitive predicate *fin*.

- $\mathcal{V}[\![x]\!](\sigma_0, \theta, \sigma, \gamma) = \begin{cases} \dagger & \text{if } \sigma = \bot, \\ \mathcal{E}[\![x]\!](\sigma) & \text{otherwise;} \end{cases}$
- $\mathcal{V}[\![x_0]\!](\sigma_0, \theta, \sigma, \gamma) = \mathcal{E}[\![x]\!](\sigma_0)$;

and

- $(\sigma_0, \theta, \sigma, \gamma) \models fin$ iff $\sigma \neq \bot$.

Sentences of this extended assertion language are called *proper* assertions and are typically denoted ζ. Let $var(\zeta)$ denote the program variables occurring in ζ and let $var_0(\zeta)$ denote the variables $x \in VAR$ such that x_0 appears in ζ. For a proper assertion ζ, the proper assertion $\zeta[fin]$ is obtained from ζ by replacing occurrences of *fin* by true. So, for instance,

- $(\zeta_1 \wedge \zeta_2)[fin] = \zeta_1[fin] \wedge \zeta_2[fin]$, and
- $fin[fin] = \mathsf{true}$.

Definition 3.28 (Valid proper assertion) A proper assertion ζ is *valid*, notation $\models \zeta$, if, and only if, for all σ_0, θ, σ, and γ, $(\sigma_0, \theta, \sigma, \gamma) \models \zeta$. ◇

Definition 3.29 (Valid proper correctness formula) For process P and proper assertion ζ the proper correctness formula P **sat** ζ is *valid*, notation $\models P \textbf{ sat } \zeta$, if, and only if, for all γ and all $(\sigma_0, \theta, \sigma) \in \mathcal{M}[\![P]\!]$, it is the case that $(\sigma_0, \theta, \sigma, \gamma) \models \zeta$. ◇

Axiom 3.30 (Skip)

$$\overline{\textbf{skip} \ \textbf{ sat } \ \mathsf{true}}$$

The axiom for the assignment statement is given next. Note that the expression e is evaluated in the initial state. Hence, when referring to the ultimate value of x in a proper assertion, any term x in expression e as it appears in $x := e$ must be replaced by x_0. Also note that execution of this assignment may only change the value of x, not the other variables referred to in e.

Axiom 3.31 (Assignment)

$$\overline{x := e \ \textbf{ sat } \ fin \ \rightarrow \ (x = e[x_0/x] \wedge Y = Y_0)}$$

where $Y \in VAR^*$ is a list containing all $y \in var(e) - \{x\}$, Y_0 the corresponding list of terms y_0, and $Y = Y_0$ abbreviates $\forall y \in Y \cdot y = y_0$.

Axiom 3.32 (Output)

$$\overline{c!e \ \textbf{ sat } \ h{\uparrow}c \preceq \langle(c, e[X_0/X])\rangle \ \wedge \ fin \ \rightarrow \ (h{\uparrow}c = \langle(c,e)\rangle \wedge X = X_0)}$$

where $X \in VAR^*$ is a list containing all $x \in var(e)$.

Axiom 3.33 (Input)

$$\overline{c?x \ \textbf{ sat } \ \exists v \cdot (h{\uparrow}c \preceq \langle(c,v)\rangle \ \wedge \ fin \ \rightarrow \ (h{\uparrow}c = \langle(c,v)\rangle \wedge x = v))}$$

The proof system contains the following two invariants.

Rule 3.34 (Channel invariance)

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

which captures the fact that the history of a process only records the communications along the channels of the process.

Rule 3.35 (Variable invariance)

$$\frac{vset \cap var(P) = \emptyset}{P \textbf{ sat } \forall x \in vset \cdot x = x_0}$$

which captures the fact that processes do not share variables.

The proof system contains the following two general rules.

Rule 3.36 (Conjunction)

$$\frac{P \textbf{ sat } \zeta_1 \ , \quad P \textbf{ sat } \zeta_2}{P \textbf{ sat } \zeta_1 \wedge \zeta_2}$$

Rule 3.37 (Consequence)

$$\frac{P \textbf{ sat } \zeta_1 \ , \quad \zeta_1 \rightarrow \zeta_2}{P \textbf{ sat } \zeta_2}$$

In case an execution of P_1 ; P_2 is a terminating execution of P_1 followed by an execution of P_2 then the initial state of P_2's execution equals the final state of P_1's execution. More specifically, given that P_1 **sat** ζ_1 and P_2 **sat** ζ_2 a term x appearing in ζ_1 is evaluated the same as a term x_0 appearing in ζ_2.

Definition 3.38 (Sequential composition operator) Let $X \in VAR^*$ be a list containing all $x \in VAR$ such that $x \in var(\zeta_1) \cap var_0(\zeta_2)$; X_0 is the corresponding list of terms x_0. Let $V \in VVAR^*$ be a list of fresh logical value variables of the same length as X. Then $\zeta_1;\zeta_2$ is expressed by:

$$\begin{array}{ll} & \zeta_1 \wedge \neg fin \\ \vee \ \exists s_1, s_2, V \cdot & \zeta_1[fin][s_1/h, V/X] \\ & \wedge\ \zeta_2[s_2/h, V/X_0] \\ & \wedge\ h\uparrow(chan(\zeta_1) \cup chan(\zeta_2)) = (s_1{}^\wedge s_2)\uparrow(chan(\zeta_1) \cup chan(\zeta_2)). \end{array}$$

◇

Then we have the following inference rule for sequential composition.

Rule 3.39 (Sequential composition)

$$\frac{P_1 \textbf{ sat } \zeta_1 \ , \quad P_2 \textbf{ sat } \zeta_2}{P_1 ; P_2 \textbf{ sat } \zeta_1 ; \zeta_2}$$

The inference rule for guarded commands is given below. Note that the guards are evaluated in the initial state. Consequently, when referring to the guard b_i in a proper assertion, any term x in the Boolean expression b_i as it appears in the program $[\ \square_{i=1}^{n}\ b_i \rightarrow P_i\ ]$ must be replaced by x_0.

Rule 3.40 (Guarded command)

$$\frac{\begin{array}{l}(\neg(\vee_{i=1}^{n}\ b_i[X_0/X])\ \wedge\ X = X_0\ \wedge\ h\uparrow \cup_{i=1}^{n}\ chan(P_i) = \langle\rangle) \rightarrow \zeta\ ,\\ \wedge_{i=1}^{n}\ P_i\ \mathbf{sat}\ \zeta_i\ ,\\ \wedge_{i=1}^{n}\ (\zeta_i \wedge b_i[X_0/X]) \rightarrow \zeta\end{array}}{[\ \square_{i=1}^{n}\ b_i \rightarrow P_i\ ]\ \mathbf{sat}\ \zeta}$$

where $X \in VAR^*$ is a list containing all $x \in var([\ \square_{i=1}^{n}\ b_i \rightarrow P_i\ ])$.

For iteration we have the following well-known inference rule.

Rule 3.41 (Iteration)

$$\frac{\begin{array}{l}\mathbf{skip}\ \mathbf{sat}\ \zeta_1\ ,\\ G\ \mathbf{sat}\ \zeta_2\ ,\\ \zeta_1;\zeta_2 \rightarrow \zeta_1\end{array}}{*G\ \mathbf{sat}\ \zeta_1}$$

The inference rule for parallel composition is:

Rule 3.42 (Parallel composition)

$$\frac{\begin{array}{l}P_1\ \mathbf{sat}\ \zeta_1\ ,\\ P_2\ \mathbf{sat}\ \zeta_2\ ,\\ var(\zeta_1) \subseteq var(P_1)\ ,\\ var(\zeta_2) \subseteq var(P_2)\ ,\\ chan(\zeta_1) \subseteq chan(P_1)\ ,\\ chan(\zeta_2) \subseteq chan(P_2)\end{array}}{P_1 \parallel P_2\ \mathbf{sat}\ \zeta_1 \wedge \zeta_2}$$

The conditions $var(\zeta_1) \subseteq var(P_1)$ and $var(\zeta_2) \subseteq var(P_2)$ capture the requirement that processes are not allowed to share variables. The conditions $chan(\zeta_1) \subseteq chan(P_1)$ and $chan(\zeta_2) \subseteq chan(P_2)$ express that the assertion that holds for one process refers only to channels of that process (cf. [Zwiers89, Hooman92]).

Example 3.43 Clearly, $c!2$ **sat** $h\uparrow c = \langle\rangle \vee h\uparrow c = \langle(c,2)\rangle$. Also, obviously, $d!0$ **sat** $h\uparrow c = \langle\rangle$. It is, however, not the case that $c!2 \parallel d!0$ **sat** $h\uparrow c = \langle\rangle$. △

Note that, as a consequence of this restriction, any occurrence of h in proper specification ζ_i of the process P_i should be projected onto a subset of $chan(P_i)$.

The effect of hiding a set *cset* of channels is simply that records of communications via channels in that set disappear from the history of the process. Then, the process $P\backslash cset$ satisfies ζ if P does so, unless a reference to h in ζ includes one or more channels from *cset*. The following inference rule captures this idea.

Rule 3.44 (Hiding)

$$\frac{P \ \mathbf{sat}\ \zeta[(h \setminus cset)/h]}{P \setminus cset \ \mathbf{sat}\ \zeta}$$

Example 3.45 (Calculator) Consider the following calculator program

$$C \ \equiv\ *[\ \mathsf{true} \ \rightarrow\ in?x\,;\ out!f(x)\ ].$$

By input axiom 3.33, channel invariance rule 3.34, conjunction rule 3.36, and consequence rule 3.37,

$$\begin{aligned} in?x\ \mathbf{sat}\ \exists v \cdot\ \ & h{\uparrow}\{in, out\} \preceq \langle(in, v)\rangle \\ & \wedge\ fin\ \rightarrow\ (h{\uparrow}\{in, out\} = \langle(in, v)\rangle\ \wedge\ x = v). \end{aligned} \tag{3.1}$$

By output axiom 3.32, channel invariance rule 3.34, conjunction rule 3.36, and consequence rule 3.37,

$$\begin{aligned} out?f(x)\ \mathbf{sat}\ \ & h{\uparrow}\{in, out\} \preceq \langle(out, f(x_0))\rangle \\ & \wedge\ fin\ \rightarrow\ (h{\uparrow}\{in, out\} = \langle(out_i, f(x_0))\rangle). \end{aligned} \tag{3.2}$$

By (3.1), (3.2), sequential composition rule 3.39, and consequence rule 3.37,

$$\begin{aligned} in?x\ ;\ out!f(x)\ \ \mathbf{sat}\ \ & \exists v \cdot h{\uparrow}\{in, out\} \preceq \langle(in, v)\rangle \\ & \vee\ \exists s_1, s_2, v \cdot\ \ s_1{\uparrow}\{in, out\} = \langle(in, v)\rangle \\ & \qquad\qquad \wedge\ s_2{\uparrow}\{in, out\} \preceq \langle(out, f(v))\rangle \\ & \qquad\qquad \wedge\ h{\uparrow}\{in, out\} = (s_1{}^\wedge s_2){\uparrow}\{in, out\}, \end{aligned}$$

from which we may conclude

$$in?x\ ;\ out!f(x)\ \ \mathbf{sat}\ \ \exists v \cdot h{\uparrow}\{in, out\} \preceq \langle(in, v), (out, f(v))\rangle.$$

Define $\zeta_1 \equiv \exists v \cdot h{\uparrow}\{in, out\} \preceq \langle(in, v), (out, f(v))\rangle$. Then, by guarded command rule 3.40,

$$[\ \mathsf{true} \ \rightarrow\ in?x\,;\ out!f(x)\ ]\ \ \mathbf{sat}\ \ \zeta_1. \tag{3.3}$$

Define $\zeta_2 \equiv \forall i \cdot 1 \leq i \leq len(h{\uparrow}out)\ \rightarrow\ val(h{\uparrow}out(i)) = f(val(h{\uparrow}in(i)))$. Obviously, since **skip sat** $len(h{\uparrow}out) = 0$,

$$\mathbf{skip}\ \ \mathbf{sat}\ \ \zeta_2. \tag{3.4}$$

By Definition 3.38 and consequence rule 3.37, $\zeta_2\ ;\ \zeta_1$ implies

$$\begin{aligned} & \forall i \cdot 1 \leq i \leq len(h{\uparrow}out)\ \rightarrow\ val(h{\uparrow}out(i)) = f(val(h{\uparrow}in(i))) \\ \vee\ & \exists s_1, s_2 \cdot\ \ \forall i \cdot 1 \leq i \leq len(s_1{\uparrow}out)\ \rightarrow\ val(s_1{\uparrow}out(i)) = f(val(s_1{\uparrow}in(i))) \\ & \qquad \wedge\ \exists v \cdot s_2{\uparrow}\{in, out\} \preceq \langle(in, v), (out, f(v))\rangle \\ & \qquad \wedge\ h{\uparrow}\{in, out\} = (s_1{}^\wedge s_2){\uparrow}\{in, out\}, \end{aligned}$$

which implies

$$
\begin{aligned}
&\forall i \cdot 1 \le i \le len(h \uparrow out) \;\rightarrow\; val(h \uparrow out(i)) = f(val(h \uparrow in(i))) \\
\vee\ \exists s_1, s_2 \cdot \quad &\forall i \cdot 1 \le i \le len(s_1 \uparrow out) \;\rightarrow\; val(s_1 \uparrow out(i)) = f(val(s_1 \uparrow in(i))) \\
&\wedge\ \forall i \cdot 1 \le i \le len(s_2 \uparrow out) \;\rightarrow\; val(s_2 \uparrow out(i)) = f(val(s_2 \uparrow in(i))) \\
&\wedge\ h \uparrow \{in, out\} = (s_1 {}^\wedge s_2) \uparrow \{in, out\}.
\end{aligned}
$$

Hence,

$$\zeta_2 \,;\, \zeta_1 \;\rightarrow\; \zeta_2. \tag{3.5}$$

By (3.4), (3.3), (3.5), and iteration rule 3.41,

$$C \textbf{ sat } \forall i \cdot 1 \le i \le len(h \uparrow out) \;\rightarrow\; val(h \uparrow out(i)) = f(val(h \uparrow in(i))).$$

△

3.6 Incorporating failure hypotheses

As mentioned in the introduction, a failure hypothesis χ of a process P is formalized as a predicate which represents a relation between the normal and acceptable histories of P. Such a predicate is expressed in a slightly extended version of the assertion language given in Table 3.2. This version contains, besides h, the special variable h_{old}. As in the previous section, variable h describes the observable behaviour of a program, but now this behaviour might be affected by faults. So, h represents an acceptable history of process P, whereas h_{old} represents a normal history of P. For instance, a possible history of process *Square*, which alternately inputs an integer via the observable channel *in* and outputs its square via the observable channel *out*, is $\langle(in, 1), (out, 1), (in, 3), (out, 9)\rangle$. Consider the exceptional behaviour caused by *Square*'s output channel transiently being stuck at zero. The relation between the normal and the acceptable behaviour can be defined using a predicate *StuckAtZero* asserting that

- with respect to the number of recorded *in* and *out* communications h_{old} and h are equally long,
- the order of *in* and *out* communications as recorded by h_{old} is preserved by h,
- the ith input value as recorded by h equals the ith input value as recorded by h_{old}, and
- the ith output value as recorded by h equals the ith output value as recorded by h_{old}, or it is equal to zero.

The construct *Square* $\wr$ *StuckAtZero* indicates execution of *Square* under the failure hypothesis *StuckAtZero*. Since predicate *StuckAtZero* holds for $h = h_{old} = \langle (in, 1), (out, 1), (in, 3), (out, 9) \rangle$, we have that *Square*'s normal behaviour $\langle (in, 1), (out, 1), (in, 3), (out, 9) \rangle$ is contained in $\mathcal{H}[\![Square \wr StuckAtZero]\!]$, the set representing the acceptable behaviour of *Square* under *StuckAtZero*. Also, because *StuckAtZero* holds for $h = \langle (in, 1), (out, 1), (in, 3), (out, 0) \rangle$ and $h_{old} = \langle (in, 1), (out, 1), (in, 3), (out, 9) \rangle$, we obtain that the abnormal behaviour $\langle (in, 1), (out, 1), (in, 3), (out, 0) \rangle$ is an element of $\mathcal{H}[\![Square \wr StuckAtZero]\!]$.

Example 3.46 (Stuck at zero) The predicate *StuckAtZero* mentioned before can formally be defined as follows:

$$
\begin{array}{rl}
StuckAtZero \equiv & len(h_{old} \uparrow \{in, out\}) = len(h \uparrow \{in, out\}) \\
\wedge & \forall i \cdot 1 \leq i \leq len(h \uparrow \{in, out\}) \\
 & \quad \rightarrow \; ch(h \uparrow \{in, out\}(i)) = ch(h_{old} \uparrow \{in, out\}(i)) \\
\wedge & \forall i \cdot 1 \leq i \leq len(h \uparrow in) \\
 & \quad \rightarrow \; val(h \uparrow in(i)) = val(h_{old} \uparrow in(i)) \\
\wedge & \forall i \cdot 1 \leq i \leq len(h \uparrow out) \\
 & \quad \rightarrow \quad val(h \uparrow out(i)) = val(h_{old} \uparrow out(i)) \\
 & \qquad \vee \; val(h \uparrow out(i)) = 0.
\end{array}
$$

$\triangle$

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

Example 3.47 (Corruption) We formalize corruption as follows:

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

$\triangle$

Example 3.48 (Loss) Consider the medium M of Example 3.24. To formalize the hypothesis that M may lose messages we define:

$$
\begin{array}{rl}
Loss \equiv & h \uparrow \{in, out\} \trianglelefteq h_{old} \uparrow \{in, out\} \\
\wedge & h \uparrow in = h_{old} \uparrow in.
\end{array}
$$

$\triangle$

We extend the assertion language with trace expression term h_{old}. Sentences of the extended language are called *transformation expressions*, with typical representative ψ. For a transformation expression ψ we also write $\psi(h_{old}, h)$ to indicate that ψ has two free variables h_{old} and h. We use $\psi(texp_1, texp_2)$ to denote the expression which is obtained from ψ by replacing h_{old} by $texp_1$, and h by $texp_2$. A transformation expression is interpreted with respect to a triple $(\theta_0, \theta, \gamma)$. Trace θ_0 gives h_{old} its value, and, in conformity with the foregoing, trace θ gives h its value, and environment γ interprets the logical variables of $IVAR \cup VVAR \cup TVAR$. The meaning of transformation expressions can easily be obtained from the meaning of assertions defined in Section 3.4. For instance,

- $\mathcal{T}[\![texp \uparrow cset]\!](\theta_0, \theta, \gamma) = \begin{cases} \dagger & \text{if } \mathcal{T}[\![texp]\!](\theta_0, \theta, \gamma) = \dagger, \\ \mathcal{T}[\![texp]\!](\theta_0, \theta, \gamma) \uparrow cset & \text{otherwise.} \end{cases}$

The only new clause is:

- $\mathcal{T}[\![h_{old}]\!](\theta_0, \theta, \gamma) = \theta_0.$

We write $(\theta_0, \theta, \gamma) \models \psi$ to denote that the traces θ_0 and θ, and the environment γ satisfy the transformation expression ψ. Since the term h_{old} does not occur in assertions, the following lemma is trivial.

Lemma 3.49 (Correspondence) For an assertion ϕ and for all θ_0 it is the case that $(\theta_0, \theta, \gamma) \models \phi$ if, and only if, $(\theta, \gamma) \models \phi$. ○

The following lemmas are easy to prove by structural induction.

Lemma 3.50 (Substitution) For the transformation expression $\psi(h_{old}, h)$,

(a) $(\theta_0, \theta, \gamma) \models \psi(h_{old}, texp)$ if, and only if, $(\theta_0, \mathcal{T}[\![texp]\!](\theta_0, \theta, \gamma), \gamma) \models \psi$;

(b) $(\theta_0, \theta, \gamma) \models \psi(texp, h)$ if, and only if, $(\mathcal{T}[\![texp]\!](\theta_0, \theta, \gamma), \theta, \gamma) \models \psi$. ○

Lemma 3.51 (Projection) Consider $cset \subseteq CHAN$ and transformation expression ψ. If $chan(\psi) \subseteq cset$ then, for all θ_0, θ, and γ,

(a) $(\theta_0, \theta, \gamma) \models \psi$ if, and only if, $(\theta_0, \theta \uparrow cset, \gamma) \models \psi$;

(b) $(\theta_0, \theta, \gamma) \models \psi$ if, and only if, $(\theta_0 \uparrow cset, \theta, \gamma) \models \psi$. ○

Notice that the projection lemma would not hold without the restriction, mentioned on page 24, that for any occurrence of $\langle\langle cexp, vexp \rangle\rangle$ in an assertion the term h does not appear in value expression $vexp$. The following example illustrates this. Consider assertion $\phi \equiv h \uparrow c = \langle (c, len(h \uparrow d)) \rangle \uparrow c$ and trace $\theta = \langle (c, 1), (d, 7) \rangle$. Clearly, $\theta \uparrow c = \langle (c, 1) \rangle = \langle (c, len(\theta \uparrow d)) \rangle$. Hence, for any θ_0 and γ, $(\theta_0, \theta, \gamma) \models \phi$. Let $cset = \{c\}$. Observe that $chan(\phi) = \{c\}$, but $\mathcal{T}[\![h \uparrow c]\!](\theta_0, \theta \uparrow c, \gamma) = \langle (c, 1) \rangle$ and $\mathcal{T}[\![\langle (c, len(h \uparrow d)) \rangle \uparrow c]\!](\theta_0, \theta \uparrow c, \gamma) = \langle (c, 0) \rangle$. Hence, $((\theta_0, \theta \uparrow c, \gamma) \not\models \phi$. Instead of the restriction on $\langle\langle cexp, vexp \rangle\rangle$ a valid projection lemma could have been obtained by defining $\mathcal{T}[\![texp \uparrow cset]\!](\theta, \gamma)$ to be

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

and $\mathcal{T}[\![texp \uparrow cset]\!](\theta_0, \theta, \gamma)$ to be

$$\begin{cases} \nmid & \text{if } \mathcal{T}[\![texp]\!](\theta_0, \theta, \gamma) = \nmid, \\ \mathcal{T}[\![texp]\!](\theta_0 \uparrow cset, \theta \uparrow cset, \gamma) \uparrow cset & \text{otherwise.} \end{cases}$$

We have decided against this because then the interpretation of transformation expressions would no longer be a straightforward adaptation of the interpretation of assertions.

Definition 3.52 (Validity of a transformation expression) A transformation expression ψ is *valid*, which we denote by $\models \psi$, if, and only if, for all θ_0, θ and γ, $(\theta_0, \theta, \gamma) \models \psi$. ◇

Since h and h_{old} may both occur free in a transformation expression, its validity might be affected by communications along the channels to which references to h *or* h_{old} are restricted. For instance, the validity of transformation expression $h \uparrow \{c\} = h_{old} \uparrow \{c, d\}$ might be affected by communications along c as well as d. The set of history channels of a transformation expression ψ, notation $chan(\psi)$, is as defined in Definition B.4 with the extra clause:

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

Definition 3.53 (Failure hypothesis) A *failure hypothesis* χ is a transformation expression which represents a reflexive relation on the normal behaviour, to guarantee that the normal behaviour is part of the acceptable behaviour:

- $\models \chi(h_{old}, h_{old})$.

As mentioned before, the semantics of a process contains the finite traces that can be observed up to any point in a normal execution. To maintain this property for acceptable behaviour, we require a failure hypothesis χ to preserve the prefix closedness:

- $\models (\chi(h_{old}, h) \wedge s \prec h) \rightarrow \exists s_{old} \preceq h_{old} \cdot \chi(s_{old}, s)$.

Furthermore, a failure hypothesis for a process FP does not impose restrictions on communications along those channels that are not in $chan(FP)$:

- $chan(\chi) \subseteq chan(FP)$. ◇

Example 3.54 Consider the process *Square* introduced at the beginning of this section. Examine

$$\begin{aligned} \psi \equiv \quad & h \uparrow \{in, out\} = h_{old} \uparrow \{in, out\} \\ \wedge \; & h \uparrow \{alarm\} = \langle\rangle. \end{aligned}$$

This transformation expression prohibits communications along channel *alarm*, which is not even a channel of *Square*. As we have seen before, this may cause problems when composing *Square* in parallel with another process. △

Using P to denote a process expressed in the programming language of Section 3.1, we define the syntax of our extended programming language in Table 3.3. Since we formalize fault tolerance in relation to concurrency, the language has no sequential constructs for failure prone processes.

Table 3.3: Extended syntax of the programming language

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

In $FP \wr \chi$, we have, by Definition 3.53, that $chan(\chi) \subseteq chan(FP)$. Consequently, $chan(FP \wr \chi) = chan(FP)$. As before, we define $chan(FP_1 \parallel FP_2) = chan(FP_1) \cup chan(FP_2)$, and $chan(FP \backslash cset) = chan(FP) - cset$.

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

- If θ is a trace of $FP_1 \parallel FP_2$ then $\theta \uparrow chan(FP_1)$ and $\theta \uparrow chan(FP_2)$ correspond to the sequence of communications performed by FP_1 and FP_2, respectively. Also, $\theta \backslash chan(FP_1 \parallel FP_2) = \langle\rangle$.

$$\begin{aligned}\mathcal{H}[\![FP_1 \parallel FP_2]\!] \;=\; \{\, \theta \mid\ & \text{for } i = 1,2,\ \theta \uparrow chan(FP_i) \in \mathcal{H}[\![FP_i]\!], \text{ and} \\ & \theta \uparrow chan(FP_1 \parallel FP_2) = \theta \,\}.\end{aligned}$$

- The effect of hiding internal channels is that communications along those channels are no longer observable.

$$\mathcal{H}[\![FP \backslash cset]\!] \;=\; \{\, \theta \backslash cset \mid \theta \in \mathcal{H}[\![FP]\!] \,\}.$$

- A trace θ is admitted by $FP \wr \chi$ if there is a trace $\theta_0 \in \mathcal{H}[\![FP]\!]$ to which θ is related, according to χ.

Example 3.55 (Loss) The failure hypothesis *Loss* defined in Example 3.48 holds for trace $\theta_0 = \langle (m_{in}, 1), (m_{out}, 1), (m_{in}, 3), (m_{out}, 3) \rangle$ and trace $\theta = \langle (m_{in}, 1), (m_{in}, 3), (m_{out}, 3) \rangle$. Unfortunately, *Loss* also holds for trace θ_0 and trace $\theta = \langle (m_{in}, 1), (a, 19), (m_{in}, 3), (m_{out}, 3) \rangle$, since it does not impose restrictions on communications along channels other than m_{in} and m_{out}. $\triangle$

We add the requirement that $\theta \backslash chan(FP) = \langle\rangle$.

$$\begin{aligned}\mathcal{H}[\![FP \wr \chi]\!] \;=\; \{\, \theta \mid\ & \text{there exists a } \theta_0 \in \mathcal{H}[\![FP]\!] \text{ such that, for all } \gamma, \\ & (\theta_0, \theta, \gamma) \models \chi, \text{ and } \theta \uparrow chan(FP) = \theta \,\}.\end{aligned}$$

Observe that this trace semantics is defined such that if $\theta \in \mathcal{H}[\![FP]\!]$ then $chan(\theta) \subseteq chan(FP)$. Notice that, for a process FP, the failure hypothesis $\chi_{FP} \equiv h \uparrow chan(FP) = h_{old} \uparrow chan(FP)$ serves as an identity relation, that is, $\mathcal{H}[\![FP]\!] = \mathcal{H}[\![FP \wr \chi_{FP}]\!]$. Also, note that, because of the reflexivity of χ on the traces of $\mathcal{H}[\![FP]\!]$, $\mathcal{H}[\![FP]\!] \subseteq \mathcal{H}[\![FP \wr \chi]\!]$.

Lemma 3.56 (Prefix closedness) If $\theta \in \mathcal{H}[\![FP]\!]$ and $\widehat{\theta} \preceq \theta$ then $\widehat{\theta} \in \mathcal{H}[\![FP]\!]$.

Proof. See Appendix C.1.

Definition 3.57 (Composite transformation expression) For the transformation expressions $\psi_1(h_{old}, h)$ and $\psi_2(h_{old}, h)$, the composite transformation expression $\psi_1 \wr \psi_2$ is defined as follows:

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

where s must be fresh. ◇

We will also use this operator to compose assertions and transformation expressions, e.g. $\phi \wr \psi \equiv \exists s \cdot \phi(s) \wedge \psi(s, h)$. Observe that, since ϕ is an assertion, h_{old} does not occur in ϕ, and hence also $\phi \wr \chi$ is an assertion.

From Definition 3.57 we easily obtain the following lemma.

Lemma 3.58 (Composite failure hypothesis)

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

Proof. See Appendix C.2.

Since the interpretation of assertions has not changed, the validity of a correctness formula FP **sat** ϕ is as defined in Definition 3.27, with P replaced by FP.

Definition 3.59 (Validity of a correctness formula) For a failure prone process FP and an assertion ϕ a correctness formula FP **sat** ϕ is *valid*, denoted by $\models FP$ **sat** ϕ, iff, for all γ and all $\theta \in \mathcal{H}[\![FP]\!]$, $(\theta, \gamma) \models \phi$ ◇

3.7 A compositional network proof theory for failure prone processes

In this section we present a compositional proof theory to prove safety properties of networks of processes. Since we focus on the relation between fault tolerance and concurrency, we have abstracted from the internal states of the processes and do not give rules for atomic statements or sequential composition. Such a proof theory is called a *network* proof theory.

The proof system contains the following two general rules.

Rule 3.60 (Consequence)

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

Rule 3.61 (Conjunction)

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

Since the history of a process *FP* only records the communications along the channels in $chan(FP)$, we have:

Rule 3.62 (Invariance)

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

From this rule we obtain the following lemma.

Lemma 3.63 (Invariance) FP **sat** $h \backslash chan(FP) = \langle\rangle$ ○

The inference rule for parallel composition is:

Rule 3.64 (Parallel composition)

$$\frac{\begin{array}{l} FP_1 \textbf{ sat } \phi_1, \\ FP_2 \textbf{ sat } \phi_2, \\ chan(\phi_1) \subseteq chan(FP_1), \\ chan(\phi_2) \subseteq chan(FP_2) \end{array}}{FP_1 \| FP_2 \textbf{ sat } \phi_1 \wedge \phi_2}$$

Next is the rule for hiding.

Rule 3.65 (Hiding)

$$\frac{FP \textbf{ sat } \phi(h \backslash cset)}{FP \backslash cset \textbf{ sat } \phi}$$

Finally, we formulate the rule for the introduction of a failure hypothesis.

Rule 3.66 (Failure hypothesis introduction)

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

Example 3.67 (Loss) Consider the medium M introduced in Example 3.48. By failure hypothesis introduction rule 3.66,

$$\begin{array}{lll} M \wr Loss \textbf{ sat } \exists s \cdot & (& Val(h\uparrow out) \preceq^1 Val(h\uparrow in) \;)[s/h] \\ & \wedge \; (& h\uparrow\{in, out\} \trianglelefteq h_{old}\uparrow\{in, out\} \\ & & \wedge \; h\uparrow in = h_{old}\uparrow in \;)[s/h_{old}], \end{array}$$

which reduces to

$$\begin{array}{ll} M \wr Loss \textbf{ sat } \exists s \cdot & Val(s\uparrow out) \preceq^1 Val(s\uparrow in) \\ & \wedge \; h\uparrow\{in, out\} \trianglelefteq s\uparrow\{in, out\} \\ & \wedge \; h\uparrow in = s\uparrow in. \end{array} \tag{3.6}$$

Now, for instance, by $h\uparrow\{in, out\} \trianglelefteq s\uparrow\{in, out\}$, it is obviously the case that $h\uparrow out \trianglelefteq s\uparrow out$, which, since s satisfies $Val(s\uparrow out) \preceq^1 Val(s\uparrow in)$, implies that $Val(h\uparrow out) \trianglelefteq Val(s\uparrow in)$. Then, by $s\uparrow in = h\uparrow in$, we obtain

$$M \wr Loss \textbf{ sat } Val(h\uparrow out) \trianglelefteq Val(h\uparrow in). \tag{3.7}$$

Property (3.7) expresses that, under *Loss*, M does not generate messages.

Since, by (3.6), $Val(s\uparrow out) \preceq^1 Val(s\uparrow in)$, we have

$$\begin{array}{l} \forall i \cdot ch(s\uparrow\{in, out\}(i)) = out \\ \quad \rightarrow val(s\uparrow\{in, out\}(i)) = val(last(s\uparrow\{in, out\}[i]\uparrow in)). \end{array}$$

As, again by (3.6), $h\uparrow\{in, out\} \trianglelefteq s\uparrow\{in, out\}$ while $h\uparrow in = s\uparrow in$, this leads to

$$\begin{array}{ll} M \wr Loss \textbf{ sat } \forall i \cdot & ch(h\uparrow\{in, out\}(i)) = out \\ & \rightarrow val(h\uparrow\{in, out\}(i)) \\ & \quad = val(last(h\uparrow\{in, out\}[i]\uparrow in)), \end{array} \tag{3.8}$$

which expresses that whenever there is an output the value equals the value of the most recent input. △

3.8 Example I : Triple modular redundancy

Consider the triple modular redundant system of Figure 3.1. It consists of three identical components C_j, $j = 1, 2, 3$, an input triplicating component *In*, and a component *Voter* that determines the ultimate output. The intuition of the triple modular redundancy paradigm is that three identical components operate on the same input and send their output to a voter which outputs the result of a majority vote. Clearly, the failure of one component can be masked, and the failure of two or all three components can be detected, as long as they do not fail identically.

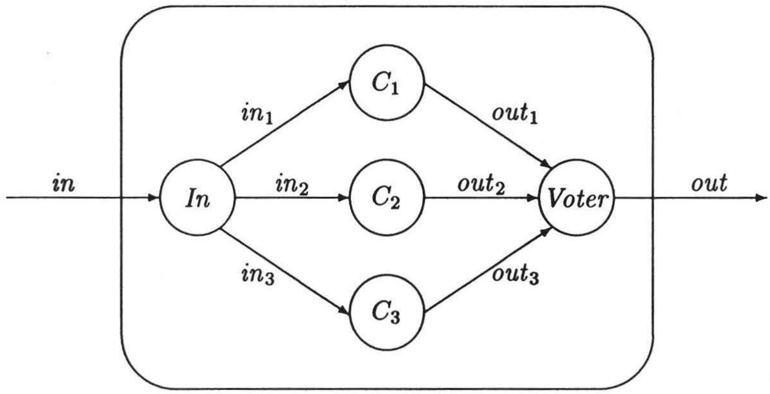

Figure 3.1: Triple modular redundant system

Definition 3.68 (Abbreviations) Throughout this section we use the following abbreviations:

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

Component C_j alternately awaits an input message from in_j, performs some computation f, and produces an output message on out_j. We abstract from the implementation details of a component; we only consider the following specification (see Example 3.45 for a possible implementation).

$$C_j \textbf{ sat } \forall i \cdot 1 \leq i \leq len(h \uparrow out_j) \rightarrow out_j(i) = f(in_j(i)).$$

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

$$\begin{array}{ll} Voter \textbf{ sat} & len(h \uparrow out) \leq \mathbf{min}(len(h \uparrow out_1), len(h \uparrow out_2), len(h \uparrow out_3)) \\ & \wedge\ \forall i, v \cdot 1 \leq i \leq len(h \uparrow out) \\ & \qquad \rightarrow ((\exists k \neq l \cdot out_k(i) = v \ \wedge\ out_l(i) = v) \rightarrow out(i) = v). \end{array}$$

Finally, component In conforms to

$$In \textbf{ sat } \forall i, j \cdot 1 \leq i \leq len(h \uparrow in_j) \rightarrow in_j(i) = in(i).$$

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

$$\begin{aligned} Cor^{\leq 1} \equiv \quad & \forall i \cdot 1 \leq i \leq \mathbf{min}(len(h\uparrow out_1), len(h\uparrow out_2), len(h\uparrow out_3)) \\ & \quad \rightarrow \exists k \neq l \cdot out_k(i) = out_k^{old}(i) \wedge out_l(i) = out_l^{old}(i) \\ & \wedge h\uparrow\{in_1, in_2, in_3\} = h_{old}\uparrow\{in_1, in_2, in_3\}. \end{aligned}$$

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

$$\begin{aligned} & (In \parallel ((C_1\|C_2\|C_3)\wr Cor^{\leq 1}) \parallel Voter)\setminus\{in_1, in_2, in_3, out_1, out_2, out_3\} \\ & \mathbf{sat} \\ & \forall i \cdot 1 \leq i \leq len(h\uparrow out) \rightarrow out(i) = f(in(i)). \end{aligned}$$

Proof. By parallel composition rule 3.64,

$$C_1\|C_2\|C_3 \ \mathbf{sat} \ \bigwedge_{j=1}^{3}(\forall i \cdot 1 \leq i \leq len(h\uparrow out_j) \rightarrow out_j(i) = f(in_j(i))).$$

By failure hypothesis introduction rule 3.66,

$$\begin{aligned} & (C_1\|C_2\|C_3)\wr Cor^{\leq 1} \\ & \mathbf{sat} \\ & \exists s \cdot \quad (\textstyle\bigwedge_{j=1}^{3}(\forall i \cdot 1 \leq i \leq len(h\uparrow out_j) \rightarrow out_j(i) = f(in_j(i))))[\, s/h \,] \\ & \qquad \wedge Cor^{\leq 1}[\, s/h_{old} \,], \end{aligned}$$

which, by definition, is equivalent to

$$\begin{aligned} & (C_1\|C_2\|C_3)\wr Cor^{\leq 1} \\ & \mathbf{sat} \\ & \exists s \cdot \quad \textstyle\bigwedge_{j=1}^{3}(\forall i \cdot 1 \leq i \leq len(s\uparrow out_j) \rightarrow out_j^s(i) = f(in_j^s(i))) \\ & \qquad \wedge \forall i \cdot 1 \leq i \leq \mathbf{min}(len(h\uparrow out_1), len(h\uparrow out_2), len(h\uparrow out_3)) \\ & \qquad\qquad \rightarrow \exists k \neq l \cdot out_k(i) = out_k^s(i) \wedge out_l(i) = out_l^s(i) \\ & \qquad \wedge h\uparrow\{in_1, in_2, in_3\} = s\uparrow\{in_1, in_2, in_3\}, \end{aligned}$$

and, thus, by consequence rule 3.60,

$$\begin{aligned} & (C_1\|C_2\|C_3)\wr Cor^{\leq 1} \\ & \mathbf{sat} \\ & \exists s \cdot \quad \forall i \cdot 1 \leq i \leq \mathbf{min}(len(h\uparrow out_1), len(h\uparrow out_2), len(h\uparrow out_3)) \\ & \qquad\qquad \rightarrow \exists k \neq l \cdot out_k(i) = f(in_k^s(i)) \wedge out_l(i) = f(in_l^s(i)) \\ & \qquad \wedge h\uparrow\{in_1, in_2, in_3\} = s\uparrow\{in_1, in_2, in_3\}. \end{aligned}$$

Note that $\bigwedge_{j=1}^{3}(\forall i \cdot 1 \leq i \leq len(h\uparrow in_j) \rightarrow s\uparrow in_j(i) = h\uparrow in_j(i))$ is implied by $h\uparrow\{in_1, in_2, in_3\} = s\uparrow\{in_1, in_2, in_3\}$. Hence

$$\begin{array}{l}(C_1 \| C_2 \| C_3) \wr Cor^{\leq 1} \\ \mathbf{sat} \\ \forall i \cdot 1 \leq i \leq \mathbf{min}(len(h \uparrow out_1), len(h \uparrow out_2), len(h \uparrow out_3)) \\ \quad \rightarrow \exists k \neq l \cdot out_k(i) = f(in_k(i)) \wedge out_l(i) = f(in_l(i)).\end{array}$$

By parallel composition rule 3.64, we get

$$\begin{array}{l}In \| ((C_1 \| C_2 \| C_3) \wr Cor^{\leq 1}) \\ \mathbf{sat} \\ \quad \forall i \cdot 1 \leq i \leq \mathbf{min}(len(h \uparrow out_1), len(h \uparrow out_2), len(h \uparrow out_3)) \\ \qquad \rightarrow \exists k \neq l \cdot out_k(i) = f(in_k(i)) \wedge out_l(i) = f(in_l(i)) \\ \wedge \forall i, j \cdot 1 \leq i \leq len(h \uparrow in_j) \rightarrow in_j(i) = in(i).\end{array}$$

Hence, by consequence rule 3.60,

$$\begin{array}{l}In \| ((C_1 \| C_2 \| C_3) \wr Cor^{\leq 1}) \\ \mathbf{sat} \\ \forall i \cdot 1 \leq i \leq \mathbf{min}(len(h \uparrow out_1), len(h \uparrow out_2), len(h \uparrow out_3)) \\ \quad \rightarrow \exists k \neq l \cdot out_k(i) = f(in(i)) \wedge out_l(i) = f(in(i)).\end{array}$$

By parallel composition rule 3.64 and consequence rule 3.60, we add the voter and obtain the relation between *in* and *out*.

$$\begin{array}{l}In \| ((C_1 \| C_2 \| C_3) \wr Cor^{\leq 1}) \| Voter \\ \mathbf{sat} \\ \forall i \cdot 1 \leq i \leq len(h \uparrow out) \rightarrow out(i) = f(in(i)).\end{array}$$

Finally, by hiding rule 3.65, we obtain

$$\begin{array}{l}(In \| ((C_1 \| C_2 \| C_3) \wr Cor^{\leq 1}) \| Voter) \backslash \{in_1, in_2, in_3, out_1, out_2, out_3\} \\ \mathbf{sat} \\ \forall i \cdot 1 \leq i \leq len(h \uparrow out) \rightarrow out(i) = f(in(i)).\end{array}$$

□

3.9 Example II : The alternating bit protocol

The alternating bit protocol [BSW69], extended with timers, is a simple way of achieving communication over a medium that may lose messages. Consider the duplex transmission medium of Figure 3.2, where A and M are media with failure hypothesis *Loss* as already discussed in Example 3.67.

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

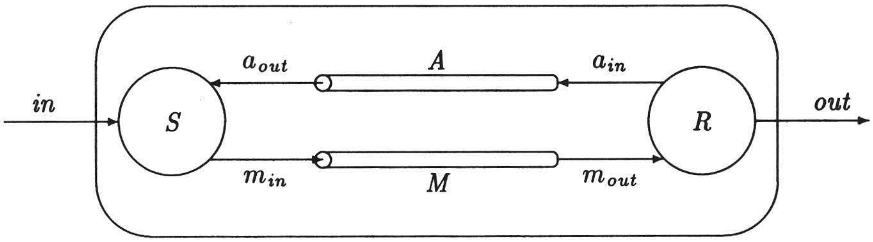

Figure 3.2: Duplex transmission medium

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

Definition 3.69 (Removal of duplicate messages) For a trace $texp$ that records only communications along $chan(M)$,

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

◇

Definition 3.70 (Removal of duplicate acknowledgements) For a trace $texp$ recording only a_{in} and a_{out} communications,

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

◇

Definition 3.71 (Abbreviations)

- For trace $texp_1$ and trace $texp_2$ such that $chan(texp_1) \subseteq chan(M)$ and $chan(texp_2) \subseteq chan(M)$,

  $$\begin{aligned}&Dat(texp_1) \preceq Dat(texp_2)\\ &\equiv \forall i \cdot 1 \leq i \leq len(texp_1)\\ &\qquad \rightarrow dat(val(texp_1(i))) = dat(val(texp_2(i))).\end{aligned}$$

  This expression denotes that the stream of data in the trace $texp_1$ is a prefix of the stream of data in the trace $texp_2$.

- For trace $texp_1$ and trace $texp_2$ such that $chan(texp_1) \subseteq chan(M)$ and $chan(texp_2) \subseteq chan(S)$,

  $$\begin{aligned}Dat(texp_1) \preceq Val(texp_2) \equiv\ &\forall i \cdot 1 \leq i \leq len(texp_1)\\ &\rightarrow dat(val(texp_1(i))) = val(texp_2(i)).\end{aligned}$$

  This expresses that the stream of data in the trace $texp_1$ is a prefix of the sequence of values in the trace $texp_2$.

- For trace $texp_1$ and trace $texp_2$ such that $chan(texp_1) \subseteq chan(A)$ and $chan(texp_2) \subseteq chan(M)$,

  $$\begin{aligned}Val(texp_1) \preceq Bit(texp_2) \equiv\ &\forall i \cdot 1 \leq i \leq len(texp_1)\\ &\rightarrow val(texp_1(i)) = bit(val(texp_2(i))).\end{aligned}$$

  This expresses that the sequence of values in the trace $texp_1$ is a prefix of the stream of bits in the trace $texp_2$.

- For trace $texp_1$ and trace $texp_2$ such that $chan(texp_1) \subseteq chan(R)$ and $chan(texp_2) \subseteq chan(M)$,

  $$\begin{aligned}Val(texp_1) \preceq Dat(texp_2) \equiv\ &\forall i \cdot 1 \leq i \leq len(texp_1)\\ &\rightarrow val(texp_1(i)) = dat(val(texp_2(i))).\end{aligned}$$

  This expression denotes that the sequence of values in the trace $texp_1$ is a prefix of the stream of data in the trace $texp_2$. ◇

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

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

$$\begin{aligned}S\ \mathbf{sat}\quad &Dat(RDMsg(h \uparrow m_{in})) \preceq^1 Val(h \uparrow in)\\ \wedge\ &Val(RDAck(h \uparrow a_{out})) \preceq^1 Bit(RDMsg(h \uparrow m_{in})).\end{aligned}$$

Similarly, we obtain the following specification for receiver R:

$$\begin{aligned}R\ \mathbf{sat}\quad &Val(h \uparrow out) \preceq^1 Dat(RDMsg(h \uparrow m_{out}))\\ \wedge\ &Val(RDAck(h \uparrow a_{in})) \preceq^1 Bit(RDMsg(h \uparrow m_{out})).\end{aligned}$$

Then, by consequence rule 3.60 and parallel composition rule 3.64, we obtain

$$ABP \text{ sat } Dat(RDMsg(h \uparrow m_{in})) \preceq^1 Val(h \uparrow in), \tag{3.9}$$

$$ABP \text{ sat } Val(RDAck(h \uparrow a_{out})) \preceq^1 Bit(RDMsg(h \uparrow m_{in})), \tag{3.10}$$

$$ABP \text{ sat } Val(h \uparrow out) \preceq^1 Dat(RDMsg(h \uparrow m_{out})) \tag{3.11}$$

and

$$ABP \text{ sat } Val(RDAck(h \uparrow a_{in})) \preceq^1 Bit(RDMsg(h \uparrow m_{out})). \tag{3.12}$$

Property (3.7) implies that

$$ABP \text{ sat } len(RDMsg(h \uparrow m_{out})) \leq len(RDMsg(h \uparrow m_{in})). \tag{3.13}$$

Since property (3.8) can only be invalidated by communications on m_{in} and m_{out}, we conclude

$$ABP \text{ sat } \forall i \cdot ch(h(i)) = m_{out} \rightarrow val(h(i)) = val(last(h[i] \uparrow m_{in})). \tag{3.14}$$

For medium A we obtain similarly

$$ABP \text{ sat } len(RDAck(h \uparrow a_{out})) \leq len(RDAck(h \uparrow a_{in})), \tag{3.15}$$

and

$$ABP \text{ sat } \forall i \cdot ch(h(i)) = a_{out} \rightarrow val(h(i)) = val(last(h[i] \uparrow a_{in})). \tag{3.16}$$

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

Lemma 3.72 (Persistency)

$$\begin{aligned} ABP \text{ sat } \quad & Val(RDAck(h \uparrow a_{out})) \preceq^1 Val(RDAck(h \uparrow a_{in})) \\ & \wedge\ Dat(RDMsg(h \uparrow m_{out})) \preceq^1 Dat(RDMsg(h \uparrow m_{in})). \end{aligned}$$

Proof. See Appendix C.3.

Then, by consequence rule 3.60, we have

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

which, by (3.9) and (3.11), yields

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

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

3.10 Soundness and relative network completeness

In this section we prove that the proof theory of Section 3.7 is sound: if a correctness formula FP **sat** ϕ is derivable, then it is valid. Furthermore, we prove that the proof system is complete: if a correctness formula FP **sat** ϕ is valid, then it is derivable.

Theorem 3.73 (Soundness) The proof system of Section 3.7 is sound.

Proof. See Appendix C.4.

As usual when proving completeness, we assume that we can prove any valid formula of the underlying (trace) logic (cf. [Cook78]). Thus, using $\vdash \phi$ to denote that assertion ϕ is derivable, we add the following axiom to our proof theory.

Axiom 3.74 (Relative completeness assumption) For an assertion ϕ,

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

○

A specification that exactly characterizes the behaviour of a process is called *precise*. As a consequence, any valid specification is implied by the precise specification. However, as we have seen before, a specification should not impose restrictions on communications along channels other than those of the process. A specification that conforms to this restriction and that exactly characterizes the behaviour of the process with respect to the communications along its channels is called *relatively* precise.

Definition 3.75 (Relative preciseness) An assertion ϕ is *relatively precise* for failure prone process FP if, and only if,

(*i*) $\models FP$ **sat** ϕ;

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

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

An (absolutely) precise specification can be obtained from a specification which is only relatively precise by means of the invariance and conjunction rules, that is, if assertion ϕ is relatively precise for some process FP then assertion $\phi \wedge h \backslash chan(FP) = \langle\rangle$ is absolutely precise for FP. In the sequel, preciseness refers to relative preciseness.

As in [WGS92], 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 also are the deduced specifications of systems composed of such processes.

Let $\vdash P$ **sat** ϕ denote that correctness formula P **sat** ϕ 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 3.76 (Network completeness) Assume that for every process P there exists a precise assertion ϕ with $\vdash P$ **sat** ϕ. Then, for any failure prone process FP and assertion ξ, $\models FP$ **sat** ξ implies $\vdash FP$ **sat** ξ. ◇

The following lemma asserts that preciseness is preserved by the proof rules of Section 3.7.

Lemma 3.77 (Preciseness preservation) Assume that for any process P there exists an assertion ϕ which is precise for P and $\vdash P$ **sat** ϕ. Then, for any failure prone process FP there exists an assertion η which is precise for FP and $\vdash FP$ **sat** η.

Proof. See Appendix C.5.

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 refers only to channels of the process, and a valid specification might refer to other channels, we have to add a clause expressing that the process does not communicate on those other channels.

Lemma 3.78 (Preciseness consequence) If ϕ is a precise specification for FP and $\models FP$ **sat** ξ then

$$\models (\phi \ \wedge \ h\!\uparrow\!(chan(\xi) - chan(FP)) = \langle\rangle) \ \ \rightarrow \ \ \xi.$$

Proof. Assume that ϕ is precise for FP, and that

$$\models FP \ \mathbf{sat} \ \xi. \tag{3.17}$$

Consider θ and γ. Assume

$$(\theta, \gamma) \models \phi \wedge h\!\uparrow\!(chan(\xi) - chan(FP)) = \langle\rangle. \tag{3.18}$$

By (3.18), we have $(\theta, \gamma) \models \phi$. Since, by the preciseness of ϕ_1 for FP, $chan(\phi) \subseteq chan(FP)$, projection lemma 3.51(a) yields $(\theta\!\uparrow\! chan(FP), \gamma) \models \phi$, thus, once more by the preciseness of ϕ for FP, $\theta\!\uparrow\! chan(FP) \in \mathcal{H}[\![FP]\!]$. By (3.17),

$$(\theta\!\uparrow\! chan(FP), \gamma) \models \xi. \tag{3.19}$$

By (3.18), we have $(\theta, \gamma) \models h\!\uparrow\!(chan(\xi) - chan(FP)) = \langle\rangle$. Consequently, $\theta\!\uparrow\!(chan(\xi) - chan(FP)) = \langle\rangle$ and we may conclude that $\theta\!\uparrow\! chan(FP) =$

$\theta\uparrow(chan(FP) \cup (chan(\xi) - chan(FP))) = \theta\uparrow(chan(FP) \cup chan(\xi))$. Hence, we obtain from (3.19) that $(\theta\uparrow(chan(FP) \cup chan(\xi)), \gamma) \models \xi$, and consequently, by projection lemma 3.51(a), $(\theta, \gamma) \models \xi$. □

Now we can establish relative network completeness.

Theorem 3.79 (Relative network completeness) The proof system given in Section 3.7 is relatively network complete.

Proof. Assume that for every process P there exists a precise specification ϕ with $\vdash P$ **sat** ϕ. Then, by preciseness preservation lemma 3.77, for any failure prone process FP there exists some assertion η which is precise for FP and

$$\vdash FP \textbf{ sat } \eta. \tag{3.20}$$

Assume $\models FP$ **sat** ξ. Since $(chan(\xi) - chan(FP)) \cap chan(FP) = \emptyset$, we obtain, by invariance rule 3.62,

$$\vdash FP \textbf{ sat } h\uparrow(chan(\xi) - chan(FP)) = \langle\rangle. \tag{3.21}$$

By (3.20) and (3.21), $\vdash FP$ **sat** $\eta \wedge h\uparrow(chan(\xi) - chan(FP)) = \langle\rangle$, and thus, by preciseness consequence lemma 3.78, relative completeness axiom 3.74, and consequence rule 3.60, $\vdash FP$ **sat** ξ. □

3.11 Discussion

In this chapter a trace-based compositional proof theory for fault tolerant distributed systems has been defined. In this theory, the failure hypothesis of a process is formalized as a relation between the normal and acceptable observable input and output behaviour of that process. Such a relation enables us to abstract from the precise nature of a fault and to focus on the abnormal behaviour it causes. This idea was first introduced in [Schepers93b]. Comparing our proof system with trace-based formalisms for normal behaviour (see e.g. [Zwiers89]), only one new rule, viz. the failure hypothesis introduction rule, has been added to capture acceptable executions.

We illustrated our method by proving safety of a triple modular redundant system and the alternating bit protocol, using only the specifications of the components. The triple modular redundant system example illustrated how the possibility of expressing the failure hypothesis of a subsystem allows us to formalize a fault hypothesis. The proof of correctness of the alternating bit protocol that appears in [PS91] is also based on traces. There, a less natural specification of the receiver, which contains the requirement that non-duplicate input messages have alternating bits, evades the necessity to prove the property of persistency.

Motivated by the ease with which it can be assured that one process cannot access the channels of another, the failure hypothesis of a process refers only to the channels of that process. This is, however, not imperative: the third failure hypothesis requirement can be dropped provided of course that we define $chan(FP \wr \chi) = chan(FP) \cup chan(\chi)$ and replace, in the definition of $\mathcal{H}[\![FP \wr \chi]\!]$, the clause $\theta \uparrow chan(FP) = \theta$ by the clause $\theta \uparrow chan(FP \wr \chi) = \theta$. In addition to this, we can specify for a process P a set $base(P)$ of channels such that the failure hypothesis χ of process P should satisfy $chan(\chi) \subseteq base(P)$ and, hence, $base(P \wr \chi) = base(P)$.

The proof system for failure prone processes abstracts from the internal states of the processes. Instead of defining a full programming language and presenting a corresponding semantics and proof theory, we could have introduced plain processes by just specifying their traces (as in [Schepers93b]). However, this would have been unfortunate, since a framework in which the basic building blocks have to be postulated is not complete.

Chapter 4

Compositional Refinement of Fault Tolerant Distributed Systems

In the previous chapter we developed a trace-based compositional proof theory to verify safety properties of fault tolerant distributed systems. In this theory, a failure hypothesis χ of a failure prone process *FP* is formalized as a relation between *FP*'s normal behaviour (i.e., the behaviour that conforms to the specification) and its acceptable behaviour, that is, the normal behaviour together with the exceptional behaviour (i.e., the behaviour whose abnormality should be tolerated). To characterize the acceptable behaviour of a failure prone process *FP* with respect to a failure hypothesis χ, the following inference rule was given:

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

In practice, a designer is faced with the problem of constructing a system which, given a failure hypothesis that characterizes the circumstances assumed for the system, satisfies a given specification. Although the failure hypothesis introduction rule repeated above can be used to obtain a specification of the acceptable behaviour from the specification of the normal behaviour, it cannot be used to identify the normal behaviour specification that results in the desired acceptable behaviour specification. Another problem one may encounter in practice concerns reusability: does a given system continue to satisfy its acceptable behaviour specification when the circumstances get worse? Then, failure hypothesis introduction rule 3.66 is again not of much use.

Essentially, a failure hypothesis relates the abstract level at which a process behaves normally to a concrete level at which that process behaves acceptably. More precisely, the specification of the normal process behaviour can be seen as a *refinement* of the acceptable process behaviour because it is more restrictive. In this chapter we study the relationship between the compositional proof theory of the previous chapter and the compositional refinement theory of [ZCdR91]. One particular aim is to classify the processes that, given a particular failure hypothesis, satisfy a given specification. Also, we try to determine the least restrictive failure hypothesis under which a process still satisfies a given specification.

In Section 4.1 we give an alternative interpretation of the assertions, failure hypotheses, and correctness formulae that were introduced in the previous chapter. Section 4.2 contains the compositional refinement theory. In Section 4.3 we illustrate our refinement method by investigating a transmission medium that might corrupt messages and one that might be transiently stuck at zero.

4.1 Assertions, failure hypotheses, and correctness formulae

The syntactic construct $FP \wr \chi$ that was introduced in the previous chapter mixes, in effect, process terms with failure hypotheses. In such a *mixed terms* formalism it is convenient to interpret an assertion, just as a process term, as a set of computations, rather than by means of truth values [Zwiers89, Olderog91]. In our case we interpret an assertion as a set of traces:

- $[\![\phi]\!]_{cset} \;=\; \{\ \theta \mid \theta \backslash cset = \langle\rangle \ \wedge\ \exists \gamma \cdot (\theta, \gamma) \models \phi\ \}.$

The reason for parameterizing the interpretation of an assertion with a set of channels will become clear after Example 4.4.

Example 4.1 (Interpretation of an assertion)

$$\begin{aligned}
[\![h \uparrow c = \langle\rangle]\!]_{\{c,d\}} &= \{\ \theta \mid \theta \backslash \{c,d\} = \langle\rangle \ \wedge\ \exists \gamma \cdot (\theta, \gamma) \models h \uparrow c = \langle\rangle\ \} \\
&= \{\ \theta \mid \theta \backslash \{c,d\} = \langle\rangle \ \wedge\ \theta \uparrow c = \langle\rangle\ \} \\
&= \{\ \theta \mid \theta \uparrow d = \theta\ \}.
\end{aligned}$$

△

A transformation expression is interpreted as a set of pairs of traces:

- $[\![\psi]\!]_{cset} \;=\; \{\ (\theta_0, \theta) \mid \theta_0 \backslash cset = \langle\rangle \ \wedge\ \theta \backslash cset = \langle\rangle \ \wedge\ \exists \gamma \cdot (\theta_0, \theta, \gamma) \models \psi\ \}.$

Example 4.2 (Interpretation of a transformation expression)

$$\begin{aligned}
[\![h \uparrow c = h_{old} \uparrow c]\!]_{\{c,d\}} &= \{\ (\theta_0, \theta) \mid\ \theta_0 \backslash \{c,d\} = \langle\rangle \ \wedge\ \theta \backslash \{c,d\} = \langle\rangle \\
&\qquad \wedge\ \exists \gamma \cdot (\theta_0, \theta, \gamma) \models h \uparrow c = h_{old} \uparrow c\ \} \\
&= \{\ (\theta_0, \theta) \mid\ \theta_0 \backslash \{c,d\} = \langle\rangle \ \wedge\ \theta \backslash \{c,d\} = \langle\rangle \\
&\qquad \wedge\ \theta \uparrow c = \theta_0 \uparrow c\ \}.
\end{aligned}$$

△

We conclude this section by (re)defining the validity of a correctness formula FP **sat** ϕ.

Definition 4.3 (Validity of a correctness formula)

$$FP \ \mathbf{sat} \ \phi \text{ if, and only if, } \mathcal{H}[\![FP]\!] \subseteq [\![\phi]\!]_{chan(FP)}.$$

◇

Example 4.4 (Validity of a correctness formula) For some process FP which outputs value 2 along channel c we know that $\mathcal{H}[\![FP]\!] = \{\langle\rangle, \langle(c,2)\rangle\}$. Since $[\![h\!\uparrow\! c = \langle\rangle \vee h\!\uparrow\! c = \langle(c,2)\rangle]\!]_c = \{\langle\rangle, \langle(c,2)\rangle\}$, the correctness formula FP **sat** $h\!\uparrow\! c = \langle\rangle \vee h\!\uparrow\! c = \langle(c,2)\rangle$ is valid. △

Recall that if trace θ is an element of $\mathcal{H}[\![FP]\!]$ then $\theta\!\uparrow\! chan(FP) = \theta$. However, as mentioned in the previous chapter, in a compositional approach the specification ϕ of failure prone process FP should not impose restrictions on communications along channels other than those of FP: it should be the case that $chan(\phi) \subseteq chan(FP)$. Consequently, the set $[\![\phi]\!]_{CHAN}$ contains, among others, every trace θ such that $\theta\!\uparrow\! chan(FP) = \langle\rangle$. Then, specification ϕ is precise for process FP in the sense of Definition 3.75 if, and only if, $\mathcal{H}[\![FP]\!] = [\![\phi]\!]_{chan(FP)}$.

4.2 Compositional refinement

Compositional refinement can be defined in terms of the relational composition $X \fatsemi R$, the weakest precondition $[R]X$, and the leads-to relation $X \leadsto Y$ [ZCdR91]. But this definition is complex due to its generality and its formulation in terms of parameterized, that is, higher order, processes. In this chapter, we restrict the definition of these operators on the mixed terms formalism of CSP-like processes and first order assertional trace specifications. We only consider those cases which are needed to develop our refinement theory.

Definition 4.5 (Composition operator) For a set $X \subseteq TRACE$ and a set $R \subseteq TRACE \times TRACE$ the composition $X \fatsemi R$ is defined as follows:

$$X \fatsemi R \ = \ \{ \ \theta \mid \exists \theta_0 \cdot \theta_0 \in X \ \wedge \ (\theta_0, \theta) \in R \ \}.$$

◇

Example 4.6 (Composition operator) Consider the sets X and R where

- $X = \{ \ \theta \in TRACE \mid \theta\!\uparrow\!\{c,d\} = \langle\rangle \ \}$, and
- $R = \{ \ (\theta_1, \theta_2) \in TRACE \times TRACE \mid \theta_1\!\uparrow\! c = \theta_2\!\uparrow\! c \ \}$.

The composition $X \mathbin{;} R$ conforms to

$$\begin{aligned} X \mathbin{;} R &= \{ \theta \mid \exists \theta_0 \cdot \theta_0 \in X \ \wedge \ (\theta_0, \theta) \in Y \} \\ &= \{ \theta \mid \exists \theta_0 \cdot \theta_0 \uparrow \{c, d\} = \langle\rangle \ \wedge \ \theta_0 \uparrow c = \theta \uparrow c \} \\ &= \{ \theta \mid \theta \uparrow c = \langle\rangle \}. \end{aligned}$$

$\triangle$

Definition 4.7 (Weakest precondition operator) For a set X such that $X \subseteq \mathit{TRACE}$ and a set $R \subseteq \mathit{TRACE} \times \mathit{TRACE}$ the weakest precondition for X with respect to R, notation $[R]X$, is defined as follows:

$$[R]X \ = \ \{ \theta \mid \forall \theta_0 \cdot (\theta, \theta_0) \in R \ \rightarrow \ \theta_0 \in X \}.$$

$\diamond$

Example 4.8 (Weakest precondition operator) Consider the sets X and R where

- $X = \{ \theta \in \mathit{TRACE} \mid \theta \neq \langle\rangle \}$, and
- $R = \{ (\theta_1, \theta_2) \in \mathit{TRACE} \times \mathit{TRACE} \mid \theta_1 \uparrow c = \theta_2 \uparrow c \}$.

The weakest precondition for X with respect to R follows from

$$\begin{aligned} [R]X &= \{ \theta \mid \forall \theta_0 \cdot (\theta, \theta_0) \in Y \ \rightarrow \ \theta_0 \in X \} \\ &= \{ \theta \mid \forall \theta_0 \cdot \theta \uparrow c = \theta_0 \uparrow c \ \rightarrow \ \theta_0 \neq \langle\rangle \} \\ &= \{ \theta \mid \theta \uparrow c \neq \langle\rangle \}. \end{aligned}$$

$\triangle$

Definition 4.9 (Leads-to operator) For sets $X, Y \subseteq \mathit{TRACE}$ the leads-to relation $X \leadsto Y$ is defined as follows:

$$X \leadsto Y \ = \ \{ (\theta_0, \theta) \mid \theta_0 \in X \ \rightarrow \ \theta \in Y \}.$$

$\diamond$

Example 4.10 (Leads-to operator) Consider the sets X and Y where

- $X = \{ \theta \in \mathit{TRACE} \mid \theta \uparrow c = \langle\rangle \}$, and
- $Y = \{ \theta \in \mathit{TRACE} \mid \theta \uparrow d = \langle\rangle \}$.

For the leads-to relation $X \leadsto Y$ we find

$$\begin{aligned} X \leadsto Y &= \{ (\theta_0, \theta) \mid \theta_0 \in X \ \rightarrow \ \theta \in Y \} \\ &= \{ (\theta_0, \theta) \mid \theta_0 \uparrow c = \langle\rangle \ \rightarrow \ \theta \uparrow d = \langle\rangle \}. \end{aligned}$$

$\triangle$

The following two lemmas relate the above defined operators.

Lemma 4.11 Let $X, Y \subseteq TRACE$ and $R \subseteq TRACE \times TRACE$, then

$$X \mathbin{\S} R \subseteq Y \text{ if, and only if, } X \subseteq [R]Y\ .$$

Proof.

$$\begin{array}{l}
X \mathbin{\S} R \subseteq Y \\
\Leftrightarrow\ \forall\theta \cdot ((\exists\theta_0 \cdot (\theta_0 \in X\ \wedge\ (\theta_0, \theta) \in R)) \rightarrow \theta \in Y) \\
\Leftrightarrow\ \forall\theta \cdot (\forall\theta_0 \cdot ((\theta_0 \in X\ \wedge\ (\theta_0, \theta) \in R) \rightarrow \theta \in Y)) \\
\Leftrightarrow\ \forall\theta \cdot (\forall\theta_0 \cdot (\theta_0 \in X \rightarrow ((\theta_0, \theta) \in R \rightarrow \theta \in Y))) \\
\Leftrightarrow\ \forall\theta_0 \cdot (\theta_0 \in X \rightarrow \forall\theta \cdot ((\theta_0, \theta) \in R \rightarrow \theta \in Y)) \\
\Leftrightarrow\ X \subseteq [R]Y
\end{array}$$

□

Lemma 4.12 Let $X, Y \subseteq TRACE$ and $R \subseteq TRACE \times TRACE$, then

$$X \mathbin{\S} R \subseteq Y \text{ if, and only if, } R \subseteq X \rightsquigarrow Y.$$

Proof.

$$\begin{array}{l}
X \mathbin{\S} R \subseteq Y \\
\Leftrightarrow\ \forall\theta \cdot (\exists\theta_0 \cdot (\theta_0 \in X\ \wedge\ (\theta_0, \theta) \in R) \rightarrow \theta \in Y) \\
\Leftrightarrow\ \forall\theta \cdot (\forall\theta_0 \cdot ((\theta_0 \in X\ \wedge\ (\theta_0, \theta) \in R) \rightarrow \theta \in Y)) \\
\Leftrightarrow\ \forall\theta \cdot (\forall\theta_0 \cdot ((\theta_0, \theta) \in R \rightarrow (\theta_0 \in X \rightarrow \theta \in Y))) \\
\Leftrightarrow\ R \subseteq X \rightsquigarrow Y
\end{array}$$

□

For assertions ϕ and ξ, and transformation expression ψ, the above operators can be expressed in the first order assertion language given in Section 3.4 as follows:

- $[\![\phi]\!] \mathbin{\S} [\![\psi]\!]$ is expressed by $\exists s \cdot (\phi(s) \wedge \psi(s, h))$;
- $[[\![\psi]\!]][\![\phi]\!]$ is expressed by $\forall s \cdot (\psi(h, s) \rightarrow \phi(s))$;
- $[\![\phi]\!] \rightsquigarrow [\![\xi]\!]$ is expressed by $\phi(h_{old}) \rightarrow \xi(h)$.

Previously, the assertion $\exists s \cdot (\phi(s) \wedge \psi(s, h))$ was abbreviated as $\phi \wr \psi$.

The inference rule for introducing failure hypotheses that was given in Section 3.7 is reformulated below as Theorem 4.13. Recall from Definition 3.53 that a failure hypothesis only refers to a subset of the channels of the process.

Theorem 4.13 (Failure hypothesis introduction)

$$\text{If } FP \textbf{ sat } \phi \text{ then } FP \wr \chi \textbf{ sat } \phi \wr \chi.$$

Proof.

$$\begin{aligned}
& FP \ \mathbf{sat}\ \phi \\
\Leftrightarrow\ & \mathcal{H}[\![FP]\!] \subseteq [\![\phi]\!]_{chan(FP)} \\
\Rightarrow\ & \mathcal{H}[\![FP]\!] \mathbin{;} [\![\chi]\!]_{chan(FP)} \subseteq [\![\phi]\!]_{chan(FP)} \mathbin{;} [\![\chi]\!]_{chan(FP)} \\
\Leftrightarrow\ & \mathcal{H}[\![FP \wr \chi]\!] \subseteq [\![\exists s \cdot (\phi(s) \wedge \chi(s,h))]\!]_{chan(FP)} \\
\Leftrightarrow\ & FP \wr \chi \ \mathbf{sat}\ \exists s \cdot (\phi(s) \wedge \chi(s,h))
\end{aligned}$$

□

Next we investigate how, given a failure hypothesis χ and an assertion ϕ, we can find a specification for failure prone process *FP* such that $FP \wr \chi$ **sat** ϕ. In this context, observe that a trace h of *FP* is characterized by the fact that any trace s with $\chi(h, s)$ conforms to ϕ.

Theorem 4.14 (Failure hypothesis elimination)

$$FP \wr \chi \ \mathbf{sat}\ \phi \text{ if, and only if, } FP \ \mathbf{sat}\ \forall s \cdot (\chi(h,s) \rightarrow \phi(s)).$$

Proof.

$$\begin{aligned}
& FP \wr \chi \ \mathbf{sat}\ \phi \\
\Leftrightarrow\ & \mathcal{H}[\![FP \wr \chi]\!] \subseteq [\![\phi]\!]_{chan(FP)} \\
\Leftrightarrow\ & \mathcal{H}[\![FP]\!] \mathbin{;} [\![\chi]\!]_{chan(FP)} \subseteq [\![\phi]\!]_{chan(FP)} \\
\Leftrightarrow\ & \mathcal{H}[\![FP]\!] \subseteq [[\![\chi]\!]_{chan(FP)}][\![\phi]\!]_{chan(FP)} \\
\Leftrightarrow\ & \mathcal{H}[\![FP]\!] \subseteq [\![\forall s \cdot (\chi(h,s) \rightarrow \phi(s))]\!]_{chan(FP)} \\
\Leftrightarrow\ & FP \ \mathbf{sat}\ \forall s \cdot (\chi(h,s) \rightarrow \phi(s))
\end{aligned}$$

□

Suppose ξ_{FP} is a (relatively) precise specification of process *FP*: it is the case that $\theta \in \mathcal{H}[\![FP]\!]$ if, and only if, $\theta \in [\![\xi_{FP}]\!]_{chan(FP)}$. The following theorem identifies the weakest, that is, the least restrictive, class χ of failure hypotheses is, such that $FP \wr \chi$ **sat** ϕ for a suitable specification ϕ. Remember that, like failure hypothesis χ, the specification ϕ should only refer to a subset of $chan(FP)$.

Theorem 4.15 (Failure hypothesis isolation)

$$FP \wr \chi \ \mathbf{sat}\ \phi \text{ if, and only if, } \chi \rightarrow (\xi_{FP}(h_{old}) \rightarrow \phi(h)).$$

Proof.

$$\begin{aligned}
& FP \wr \chi \ \mathbf{sat}\ \phi \\
\Leftrightarrow\ & \mathcal{H}[\![FP \wr \chi]\!] \subseteq [\![\phi]\!]_{chan(FP)} \\
\Leftrightarrow\ & \mathcal{H}[\![FP]\!] \mathbin{;} [\![\chi]\!]_{chan(FP)} \subseteq [\![\phi]\!]_{chan(FP)} \\
\Leftrightarrow\ & [\![\xi_{FP}]\!] \mathbin{;} [\![\chi]\!]_{chan(FP)} \subseteq [\![\phi]\!]_{chan(FP)} \\
\Leftrightarrow\ & [\![\chi]\!]_{chan(FP)} \subseteq [\![\xi_{FP}]\!]_{chan(FP)} \leadsto [\![\phi]\!]_{chan(FP)} \\
\Leftrightarrow\ & \chi \rightarrow (\xi_{FP}(h_{old}) \rightarrow \phi(h))
\end{aligned}$$

□

4.3 Examples

In this section we illustrate the use of the failure hypothesis elimination and isolation theorems by investigating a transmission medium that might corrupt messages and one that might be transiently stuck at zero.

4.3.1 A transmission medium that might corrupt messages

Consider the transmission medium M introduced in Example 3.24 and the failure hypothesis Cor discussed in Example 3.47. Using failure hypothesis introduction theorem 4.13 we obtain:

$$\begin{array}{ll} M \wr Cor \ \mathbf{sat}\ \exists s \cdot & Val(s\uparrow out) \preceq^1 Val(s\uparrow in) \\ & \wedge\ len(h\uparrow\{in, out\}) = len(s\uparrow\{in, out\}) \\ & \wedge\ \forall i \cdot 1 \leq i \leq len(h\uparrow\{in, out\}) \\ & \qquad \rightarrow\ ch(h\uparrow\{in, out\}(i)) = ch(s\uparrow\{in, out\}(i)) \\ & \wedge\ \forall i \cdot 1 \leq i \leq len(h\uparrow in)\ \rightarrow\ val(h\uparrow in(i)) = val(s\uparrow in(i)). \end{array}$$

Because there is no relationship any more between the values input and those output, the strongest property of $M \wr Cor$ is:

$$M \wr Cor \ \mathbf{sat}\ len(h\uparrow out) \leq len(h\uparrow in) \leq len(h\uparrow out) + 1,$$

which no longer specifies a transmission medium. By failure hypothesis elimination theorem 4.14 we know that

$$CM \wr Cor \ \mathbf{sat}\ Val(h\uparrow out) \preceq^1 Val(h\uparrow in)$$

if, and only if,

$$\begin{array}{ll} CM \ \mathbf{sat}\ \forall s \cdot (& len(s\uparrow\{in, out\}) = len(h\uparrow\{in, out\}) \\ & \wedge\ \forall i \cdot 1 \leq i \leq len(s\uparrow\{in, out\}) \\ & \qquad \rightarrow\ ch(s\uparrow\{in, out\}(i)) = ch(h\uparrow\{in, out\}(i)) \\ & \wedge\ \forall i \cdot 1 \leq i \leq len(s\uparrow in)\ \rightarrow\ val(s\uparrow in(i)) = val(h\uparrow in(i))\ ) \\ & \rightarrow\ \ Val(s\uparrow out) \preceq^1 Val(s\uparrow in). \end{array}$$

However, this implication does not hold for *arbitrary* traces s as the premise may hold even if not $\forall i \cdot 1 \leq i \leq len(s\uparrow out) \rightarrow val(s\uparrow out(i)) = val(s\uparrow in(i))$. Hence, the assertion is equivalent to false, and therefore such a CM cannot be implemented. A possible way to deal with corruption is to use coding (see for instance Section A.4). An encoding function transforms a dataword into a codeword which contains some redundant bits. Thus the set of datawords is mapped into only a small fraction of a much larger set of codewords. The codewords some dataword is mapped into are called valid, and the encoding ensures that it is very unlikely that due to corruption one valid codeword is changed into another.

Using the function *Valid* with the obvious interpretation we formalize the detectable corruption hypothesis as follows:

$$\begin{array}{lll} DetCor \equiv & & len(h\uparrow\{in, out\}) = len(h_{old}\uparrow\{in, out\}) \\ & \wedge & \forall i \cdot 1 \leq i \leq len(h\uparrow\{in, out\}) \\ & & \quad \rightarrow ch(h\uparrow\{in, out\}(i)) = ch(h_{old}\uparrow\{in, out\}(i)) \\ & \wedge & \forall i \cdot 1 \leq i \leq len(h\uparrow in) \rightarrow val(h\uparrow in(i)) = val(h_{old}\uparrow in(i)) \\ & \wedge & \forall i \cdot 1 \leq i \leq len(h\uparrow out) \\ & & \quad \rightarrow \quad val(h\uparrow out(i)) = val(h_{old}\uparrow out(i)) \\ & & \qquad \vee \neg Valid(val(h\uparrow out(i))). \end{array}$$

Now, we seek CM such that

$$\begin{array}{l} CM \wr DetCor \ \mathbf{sat} \ \forall i \cdot 1 \leq i \leq len(h\uparrow out) \\ \qquad\qquad \rightarrow \ Valid(val(h\uparrow out(i))) \\ \qquad\qquad\qquad \rightarrow \ val(h\uparrow out(i)) = val(h\uparrow in(i)). \end{array}$$

Using failure hypothesis elimination theorem 4.14 once more we obtain:

$$\begin{array}{l} CM \ \mathbf{sat} \ \forall s \cdot (\quad len(s\uparrow\{in, out\}) = len(h\uparrow\{in, out\}) \\ \qquad \wedge \ \forall i \cdot 1 \leq i \leq len(s\uparrow\{in, out\}) \\ \qquad\qquad \rightarrow \ ch(s\uparrow\{in, out\}(i)) = ch(h\uparrow\{in, out\}(i)) \\ \qquad \wedge \ \forall i \cdot 1 \leq i \leq len(s\uparrow in) \ \rightarrow \ val(s\uparrow in(i)) = val(h\uparrow in(i)) \\ \qquad \wedge \ \forall i \cdot 1 \leq i \leq len(s\uparrow out) \\ \qquad\qquad \rightarrow \quad (s\uparrow out(i)) = val(h\uparrow out(i)) \\ \qquad\qquad\quad \vee \ \neg Valid(val(s\uparrow out(i))) \ ) \\ \quad \rightarrow \ (\ \forall i \cdot 1 \leq i \leq len(s\uparrow out) \\ \qquad\qquad \rightarrow \ Valid(val(s\uparrow out(i))) \\ \qquad\qquad\quad \rightarrow \ val(s\uparrow out(i)) = val(s\uparrow in(i)) \ ). \end{array}$$

4.3.2 A transmission medium that might be transiently stuck at zero

Suppose the medium M presented in Example 3.24 might be transiently stuck at zero. What is a suitable failure hypothesis *StuckAtZero* such that

$$\begin{array}{l} M \wr StuckAtZero \ \mathbf{sat} \ \forall i \cdot 1 \leq i \leq len(h\uparrow out) \\ \qquad\qquad \rightarrow \quad val(h\uparrow out(i)) = val(h\uparrow in(i)) \\ \qquad\qquad\quad \vee \ val(h\uparrow out(i)) = 0 \ ? \end{array}$$

Using failure hypothesis isolation rule 4.15 we can classify *StuckAtZero* as follows:

$$
\begin{array}{l}
\mathit{StuckAtZero} \\
\rightarrow \ (\quad \forall i \cdot 1 \leq i \leq len(h_{old} \uparrow out) \ \rightarrow \ val(h_{old} \uparrow out(i)) = val(h_{old} \uparrow in(i)) \\
\qquad \wedge \ len(h_{old} \uparrow out) \leq len(h_{old} \uparrow in) \leq len(h_{old} \uparrow out) + 1 \\
\qquad \rightarrow \ \forall i \cdot 1 \leq i \leq len(h \uparrow out) \ \rightarrow \quad val(h \uparrow out(i)) = val(h \uparrow in(i)) \\
\qquad \qquad \qquad \qquad \qquad \qquad \qquad \vee \ val(h \uparrow out(i)) = 0 \ ).
\end{array}
$$

A natural candidate is the predicate *StuckAtZero* defined in Example 3.46.

4.4 Discussion

Failure hypothesis introduction rule 3.66 is a so-called *forward* rule. A forward rule is characterized by the fact that the assertion appearing in the conclusion is constructed from the assertions appearing in the premises. Apart from side conditions, a forward rule has the following general form:

$$
\frac{\begin{array}{c} \ldots, \ P_i \ \mathbf{sat} \ \phi_i \ , \\ \ldots, \ \xi_i \ \rightarrow \ \eta_i, \ \ldots \end{array}}{op_{proc}(P_1, \ldots, P_n) \ \mathbf{sat} \ op_{spec}(\phi_1, \ldots, \phi_n, \xi_1, \ldots, \xi_m, \eta_1, \ldots, \eta_m)}
$$

A forward rule is useful for verification. However, when designing systems a so-called *backward* rule is required. A backward rule has the property that a simple correctness formula can be concluded whenever the complex premises hold. The difference between forward and backward rules is rather subtle: P **sat** ϕ if P **sat** $op_{spec}(\phi_1, \ldots, \phi_n, \xi_1, \ldots, \xi_m, \eta_1, \ldots, \eta_m)$ and $op_{spec}(\phi_1, \ldots, \phi_n, \xi_1, \ldots, \xi_m, \eta_1, \ldots, \eta_m) \rightarrow \phi$. The typical form of a backward rule is the following:

$$
\frac{\begin{array}{l} \ldots, \ P_i \ \mathbf{sat} \ \phi_i \ , \\ \ldots, \ \xi_i \ \rightarrow \ \eta_i, \ \ldots \ , \\ op_{spec}(\phi_1, \ldots, \phi_n, \xi_1, \ldots, \xi_m, \eta_1, \ldots, \eta_m) \ \rightarrow \ \phi \end{array}}{op_{proc}(P_1, ..., P_n) \ \mathbf{sat} \ \phi}
$$

Using failure hypothesis isolation theorem 4.15, we obtain the following backward failure hypothesis introduction rule:

$$
\frac{FP \ \mathbf{sat} \ \xi \ , \quad (\chi(h_{old}, h) \wedge \xi(h_{old})) \ \rightarrow \ \phi(h)}{FP \wr \chi \ \mathbf{sat} \ \phi}
$$

Chapter 5

Fault Tolerant Real-Time Distributed Systems

In this chapter we extend the proof theory of Chapter 3 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 timing failures and deadlock to be taken into account. To exclude unrealistic behaviour, it incorporates finite variability [BKP86], also called non-Zeno-ness (cf. [AL92] and Appendix D), 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 [Lamport83].

This chapter is organized as follows. Section 5.1 introduces the programming language. In Section 5.2 we present the computational model. Section 5.3 introduces the denotational semantics. In Section 5.4 we present the assertion language and associated correctness formulae. In Section 5.5 we once again incorporate failure hypotheses in our formalism. Section 5.6 presents a compositional network proof theory for fault tolerant real-time distributed systems. We illustrate our method by applying it, in Section 5.7, to a triple modular redundant system. In Section 5.8 we show that the proof system of Section 5.6 is sound and relative network complete.

5.1 Programming language

To enable the programming of time-outs we extend the language of Section 3.1 with a communication guarded command that contains, as one of the guards, a delay statement. Let $\mathbb{Q}$ denote the rationals, and $\mathbb{R}$ the reals. Let *TIME* be some ordered time domain ($\infty \in TIME$). For the scope of this thesis it is immaterial whether the time domain *TIME* is discrete, i.e., $TIME = \{u\tau \mid \tau \in \mathbb{N}\}$

for some positive smallest time unit u, dense, i.e., $TIME = \{\tau \in \mathbb{Q} \mid \tau \geq 0\}$, or continuous, i.e., $TIME = \{\tau \in \mathbb{R} \mid \tau \geq 0\}$. The syntax of our programming language is given in Table 5.1, with $n \in \mathbb{N}$, $n \geq 1$, $x, x_1, \ldots, x_n \in VAR$, $\mu \in VAL$, $f \in VAL^n \to VAL$, $d \in TIME$, $c, c_1, \ldots, c_n \in CHAN$, and $cset \subseteq CHAN$.

Table 5.1: Syntax of the programming language

Expression	e	$::=$	$\mu \mid x \mid f(e_1, \ldots, e_n)$
Boolean Expression	b	$::=$	$e_1 = e_2 \mid e_1 < e_2 \mid \neg b \mid b_1 \vee b_2$
Guarded Command	G	$::=$	$[\ \|\!\|_{i=1}^{n}\ b_i \to P_i\ ] \mid$ $[\ \|\!\|_{i=1}^{n}\ c_i?x_i \to P_i\ \|\!\|\ \mathbf{delay}\ d \to P\ ]$
Process	P	$::=$	$\mathbf{skip} \mid x := e \mid c!e \mid c?x \mid P_1; P_2 \mid$ $G \mid *G \mid P_1 \,\|\, P_2 \mid P \backslash cset$

We give the informal meaning of the new or modified statements.

Atomic statements

- **skip** terminates after K_{skip} units of time, where constant $K_{\mathsf{skip}} > 0$.

Compound statements

- The evaluation of the guards appearing in the Boolean guarded command $[\ \|\!\|_{i=1}^{n}\ b_i \to P_i\ ]$ and the non-deterministic selection of one of the open guards requires K_g time units.
- Communication guarded command $[\ \|\!\|_{i=1}^{n}\ c_i?x_i \to P_i\ \|\!\|\ \mathbf{delay}\ d \to P\ ]$. Wait for 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. If d equals 0 then P is executed immediately.

The set of variables that occur in process P, notation $var(P)$, is inductively defined as in Definition B.1 with the extra clause:

- $var([\ \|\!\|_{i=1}^{n}\ c_i?x_i \to P_i\ \|\!\|\ \mathbf{delay}\ d \to P_0\ ]) = \cup_{i=1}^{n}\{x_i\} \cup \cup_{i=0}^{n} var(P_i)$.

The set of *observable channels* of a process P, notation $chan(P)$, is as defined in Chapter 3. The only new clauses are:

- $in([\ \|\!\|_{i=1}^{n}\ c_i?x_i \to P_i\ \|\!\|\ \mathbf{delay}\ d \to P_0\ ]) = \cup_{i=1}^{n}\{c_i\} \cup \cup_{i=0}^{n} in(P_i)$, and
- $out([\ \|\!\|_{i=1}^{n}\ c_i?x_i \to P_i\ \|\!\|\ \mathbf{delay}\ d \to P_0\ ]) = \cup_{i=0}^{n} out(P_i)$.

5.1.1 Syntactic restrictions

In addition to the syntactic constraints mentioned in Section 3.1.1 we have the following restriction (for any $n \in \mathbb{N}$, $x_1, \ldots, x_n \in VAR$, $c_1, \ldots, c_n \in CHAN$ and $d \in TIME$):

- For communication guarded command $[\, [\!]_{i=1}^{n}\, c_i?x_i \rightarrow P_i \,[\!]\, \mathbf{delay}\, d \rightarrow P_0 \,]$ we require that if P_i contains $c!e$ then P_j does not contain $c?x$, that is, $out(P_i) \cap in(P_j) = \emptyset$, for all $i, j \in \{0, \ldots, n\}$, $i \neq j$. We require furthermore that P_j does not contain $c_i!e$, that is, $\{c_i\} \cap out(P_j) = \emptyset$, for all $i \in \{1, \ldots, n\}$ and $j \in \{0, \ldots, n\}$.

5.1.2 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.

We assume that the execution time of atomic statements, except for communication statements, is given by fixed constants. By assumption, communication takes no time: conceptually a message is received at the same time it is sent. Apart from an assumed fixed constant overhead before and after the actual communication, the execution time of a (synchronous) communication statement consists of the time spent waiting for a partner.

In this chapter we assume maximal parallelism, that is, we assume that each process has its own processor. Hence, a process executes a local, that is, non-communication, command immediately. Since communication is synchronous, a process is forced to wait until a communication partner is available. In the case of maximal parallelism the communication occurs as soon as such a partner is ready: 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.

Observe that in the semantic model of Section 3.3 actions can be arbitrarily delayed, due to the abstraction from the timing of computations. Then, the input and output behaviour of process $P_1 \parallel P_2$ is simply an interleaving of the communication sequences of processes P_1 and P_2 which respects the order of communications along the channels in $chan(P_1) \cap chan(P_2)$. The untimed trace semantics of process $[\, c?x \rightarrow d?y \,[\!]\, d?y \rightarrow c?x \,] \parallel (c!0 \parallel (z := 1 \,;\, d!z))$, for instance, includes $\langle (c, 0), (d, 1) \rangle$ as well as $\langle (d, 1), (c, 0) \rangle$. Taking the timing of computations into account, we notice that the d communication cannot be performed at the start of the program, since execution of the assignment $z := 1$ takes a positive amount of time. In the maximal parallelism case, the c communication can take place immediately, and, consequently, the c communication precedes the d communication.

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 the constant K_{skip}, we assume that execution of each assignment statement takes a constant K_{a} time units. A constant K_α denotes the overhead preceding a communication, and a constant K_ω denotes the overhead following a communication. Furthermore, we assume that to evaluate the guards of a Boolean guarded command and non-deterministically select one of the open guards a constant K_{g} time units are required.

5.2 Model of computation

The events in the various processes of a distributed system are related to each other by means of a conceptual global clock (as in [RR86, KSdRGA88]). This global notion of time is introduced at a metalevel of reasoning and is not incorporated in the distributed system itself. In essence, to reason about the real-time behaviour of a process we observe for each communication the time at which it occurs. We represent a synchronous communication of value $\mu \in \mathit{VAL}$ on channel $c \in \mathit{CHAN}$ at time $\tau \in \mathit{TIME}$ by a triple (τ, c, μ), and define:

$$\begin{array}{ll}(\mathit{Timestamp}) & \mathit{ts}((\tau, c, \mu)) = \tau; \\ (\mathit{Channel}) & \mathit{ch}((\tau, c, \mu)) = c; \\ (\mathit{Value}) & \mathit{val}((\tau, c, \mu)) = \mu.\end{array}$$

To denote the observable input and output communication behaviour of a process P we use a *timed* trace θ which is a possibly infinite sequence of the form $\langle(\tau_1, c_1, \mu_1), (\tau_2, c_n, \mu_n), \ldots\rangle$, where $\tau_i \geq \tau_{i-1}$, $c_i \in \mathit{chan}(P)$, and $\mu_i \in \mathit{Val}$, 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 5.1 (Timed traces) Let, for $\mathit{Obs} = \mathit{TIME} \times \mathit{CHAN} \times \mathit{VAL}$, TRACE be the set of timed traces, that is,

$$\begin{array}{rll}\mathit{TRACE} = \{\, \theta \in \mathit{Obs}^* \cup \mathit{Obs}^\omega \mid \forall i \cdot & \mathit{ts}(\theta(i)) \leq \mathit{ts}(\theta(i+1)) & \\ & \wedge\ \forall j \neq i \cdot \mathit{ts}(\theta(i)) = \mathit{ts}(\theta(j)) & \\ & \qquad \rightarrow \mathit{ch}(\theta(i)) \neq \mathit{ch}(\theta(j))\ \}. & \end{array}$$

◇

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 $\theta_1{}^\frown\theta_2$ (and equals θ_1 if θ_1 is infinite). We use $\mathit{first}(\theta)$ and, if θ is finite, $\mathit{last}(\theta)$ to refer to the first and last record of θ, respectively.

Example 5.2 (Timed traces) Let $K_\alpha = K_\mathrm{a} = K_\mathrm{g} = 1$. Consider the processes $P_1 \equiv c?x$ and $P_2 \equiv [\ z = 0 \rightarrow c!z \ [] \ z \neq 0 \rightarrow z := 0 ; c!z\ ]$. Extending the function $\mathcal{H}$ of the previous chapter to generate a set of timed traces leads to $\mathcal{H}[\![P_1]\!] = \{ (\tau, c, \mu) \mid \tau \geq 1 \wedge \mu \in \mathit{VAL} \}$ and $\mathcal{H}[\![P_2]\!] = \{ (\tau, c, 0) \mid \tau \geq 2 \vee \tau \geq 3 \}$, that is, $\mathcal{H}[\![P_2]\!] = \{ (\tau, c, 0) \mid \tau \geq 2 \}$. Doing so we also obtain that $\mathcal{H}[\![P_1 \parallel P_2]\!] = \{ (\tau, c, 0) \mid \tau \geq 2 \}$. However, this is in conflict with our maximal parallelism assumption on the basis of which $\mathcal{H}[\![P_1 \parallel P_2]\!] = \{ (2, c, 0), (3, c, 0) \}$. △

The above example illustrates that a model based on merely timed traces is too abstract to define a compositional semantics, as has been argued in [RR86] and [GB87]. 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 not only the communications that take place are recorded 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 5.3 (Timed refusals) Let *REF* be the set of timed refusal sets, that is,

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

◇

We usually define a timed refusal by a Cartesian product $\mathit{cset} \times \mathit{INT}$, where $\mathit{cset} \subseteq \mathit{CHAN}$ is a set of channels and *INT* an interval from *TIME*, that is, an element of $\mathcal{P}(\mathit{TIME})$.

Definition 5.4 (Projection on traces) For a trace $\theta \in \mathit{TRACE}$ and a set of channels $\mathit{cset} \subseteq \mathit{CHAN}$, we define the *projection* of θ onto *cset*, denoted by $\theta \uparrow \mathit{cset}$, as the sequence obtained from θ by deleting all records with channels not in *cset*. Formally,

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

◇

Example 5.5 (Projection on traces)

$$\langle (1, a, 3), (2, c, 4), (5, a, 2), (7, a, 3) \rangle \uparrow \{a\} = \langle (1, a, 3), (5, a, 2), (7, a, 3) \rangle.$$

△

Definition 5.6 (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

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

◇

Example 5.7 (Hiding on traces)

$$\langle(2,a,3),(3,c,4),(4,a,2),(6,a,3)\rangle \backslash \{a\} = \langle(3,c,4)\rangle.$$

△

Definition 5.8 (Time shift on traces) For a timed trace θ for which it is the case that $ts(first(\theta)) \geq \tau$ we define the *time shift* operation $\curvearrowleft$ as follows:

$$\theta \curvearrowleft \tau \;=\; \begin{cases} \langle\rangle & \text{if } \theta = \langle\rangle, \\ (\hat{\tau} - \tau, c, \mu)^\wedge(\theta_0 \curvearrowleft \tau) & \text{if } \theta = (\hat{\tau}, c, \mu)^\wedge\theta_0. \end{cases}$$

◇

Example 5.9 (Time shift on traces)

$$\langle(2,a,3),(3,c,4),(4,a,2),(6,a,3)\rangle \curvearrowleft 1 = \langle(1,a,3),(2,c,4),(3,a,2),(5,a,3)\rangle.$$

△

Definition 5.10 (Channel projection on refusals) For refusal $\mathfrak{R} \in REF$ and a set of channels $cset \subseteq CHAN$, we define the *channel projection* of $\mathfrak{R}$ onto *cset*, denoted by $\mathfrak{R}\uparrow cset$ as follows:

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

◇

Example 5.11 (Channel projection on refusals)

$$\{(a,1),(b,1),(a,2),(c,2)\}\uparrow\{a\} = \{(a,1),(a,2)\}.$$

△

Definition 5.12 (Interval projection on refusals) For a refusal $\mathfrak{R} \in REF$ and an interval $INT \in \mathcal{P}(TIME)$, we define the *interval projection* of $\mathfrak{R}$ onto INT, denoted by $\mathfrak{R}{\uparrow}INT$ as follows:

$$\mathfrak{R}{\uparrow}INT \;=\; \mathfrak{R} \cap (CHAN \times INT).$$

$\diamond$

Example 5.13 (Interval projection on refusals)

$$\{(a,1),(b,1),(a,2),(c,2)\}{\uparrow}[1,2) = \{(a,1),(b,1)\}.$$

$\triangle$

Definition 5.14 (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:

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

$\diamond$

Example 5.15 (Hiding on refusals)

$$\{(a,1),(b,1),(a,2),(c,2)\}\backslash\{a\} = \{(b,1),(c,2)\}.$$

$\triangle$

Definition 5.16 (Time shift on refusals) For $\mathfrak{R} \in REF$ the time shift operation $\mathfrak{R} \curvearrowleft \tau$ is defined as follows:

$$\mathfrak{R} \curvearrowleft \tau \;=\; \{\ (c, \hat{\tau} - \tau) \mid (c,\hat{\tau}) \in \mathfrak{R} \ \wedge\ \hat{\tau} \geq \tau\ \}.$$

$\diamond$

Example 5.17 (Time shift on refusals)

$$\{(a,2),(b,2),(a,3),(c,3)\} \curvearrowleft 1 = \{(a,1),(b,1),(a,2),(c,2)\}.$$

$\triangle$

5.3 Denotational semantics

In this section we define an — again — denotational semantics for the programming language of Section 5.1. We use a special symbol T ($T \notin VAR$) to denote the global time.

Definition 5.18 (States) The set *STATE* of states is 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*.

As pointed out in Section 5.2 the concept of a timed refusal set helps to achieve compositionality. Consequently, in addition to the observable quantities mentioned in Chapter 3, we want to observe, for any execution of a process P,

- the initial state of P including the starting time of the execution,
- the sequence of communications performed by P and the times at which communications occur,
- the times at which P refused to communicate and the names of those channels, and,
- for a terminating computation of P, the final state of P including the termination time of the execution.

Using the model of Section 5.2 we can describe both the terminating and the non-terminating computations of a program P, where a special state $\perp$ indicates an infinite execution. In general, whereas the semantics given in Chapter 3 contained the behaviour that may be observed up to a particular point in an execution, the semantics of a program is now a set of denotations representing the *maximal* observations of the possible executions of the program.

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.

The semantic function $\mathcal{M}$ is inductively defined as follows. Notice that a terminated process will indefinitely refuse to communicate on its channels. The variant of a state σ with respect to a variable x and a value ϑ, notation $(\sigma : x \mapsto \vartheta)$, has been defined in Section 3.3. There we have also defined when a Boolean expression b holds in a state σ, notation $\mathcal{B}[\![b]\!](\sigma)$.

- Execution of **skip** terminates after $K_{\mathbf{skip}}$ time units. Because the process **skip** has no channels, it does not refuse any communication.

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

- Execution of the assignment $x := e$ terminates after $K_{\mathbf{a}}$ time units, and in its final state x has the value that e had in its initial state. Since the process $x := e$ has no channels, it does not refuse any communication.

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

- In the execution of the synchronous output statement $c!e$ there comes, after an initial period of K_α time units during which a communication on channel c is refused, a waiting period for a communication partner to become available. From the start of this latter period a communication via channel c is not refused until after such a communication occurs. As channel c is the only channel of the process $c!e$, this process does not refuse any other communication. Execution of the 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 occurred.

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

Recall that we allow at most one communication via channel c at time τ.

- Execution of the input statement $c?x$ either never terminates (in case no communication partner is ever available) or terminates when the communication on channel c has occurred and the received value is assigned to x, that is, $K_\omega + K_{\mathbf{a}}$ time units after that communication has occurred.

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

- 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 the case of sequential and similar compositions, expand the refusal sets of the respective components to be the union of the channels of those components. Assuming that the execution of P_1 terminates at τ and that the execution of P_2 starts at τ, process $P_1 \; ; \; P_2$ refuses to communicate along the channels in $chan(P_2) - chan(P_1)$ during the interval $[0, \tau)$, and along the channels in $chan(P_1) - chan(P_2)$ from time τ onwards.

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

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 execution of the Boolean guarded command $[\; []_{i=1}^{n} \; b_i \rightarrow P_i \;]$ terminates after evaluating the guards, which takes K_g time units. Otherwise, the process corresponding to one of its open guards (non-deterministically chosen) is executed. Since $chan([\; []_{i=1}^{n} \; b_i \rightarrow P_i \;]) = \cup_{i=1}^{n} chan(P_i)$, communications on $\cup_{i=1}^{n} chan(P_i)$ are refused while evaluating the guards; in case execution continues with the process P_k, communications on $\cup_{i=1}^{n} chan(P_i) - chan(P_k)$ are refused.

$$
\begin{array}{l}
\mathcal{M}[\![\; [\; []_{i=1}^{n} \; b_i \rightarrow P_i \;] \;]\!] = \\
\quad \{ \; (\; \sigma_0 \; , \; (\langle\rangle, \cup_{i=1}^{n} chan(P_i) \times [0, \infty)) \; , \; (\sigma_0 : T \mapsto K_g) \;) \\
\qquad | \; \mathcal{E}[\![T]\!](\sigma_0) = 0 \; \wedge \; \neg \mathcal{B}[\![b_1 \vee \ldots \vee b_n]\!](\sigma_0) \; \} \\
\cup \; \{ \; (\; \sigma_0, (\theta, \mathfrak{R}), \sigma \;) \\
\qquad | \;\; \mathcal{E}[\![T]\!](\sigma_0) = 0 \text{ and there exist a } k \in \{1, \ldots, n\} \text{ and a } \widehat{\mathfrak{R}} \text{ such that} \\
\qquad\;\; \mathcal{B}[\![b_k]\!](\sigma_0), \\
\qquad\;\; \widehat{\mathfrak{R}} \uparrow [0, K_g) = \emptyset, \\
\qquad\;\; (\; \sigma_0 \; , \; (\theta, \widehat{\mathfrak{R}}) \curvearrowleft K_g \; , \; (\sigma : T \mapsto T - K_g) \;) \in \mathcal{M}[\![P_k]\!], \\
\qquad\;\; \text{and } \mathfrak{R} \; = \; \cup_{i=1}^{n} chan(P_i) \times [0, K_g) \\
\qquad\qquad\qquad\quad \cup \; \widehat{\mathfrak{R}} \\
\qquad\qquad\qquad\quad \cup \; (\cup_{i=1}^{n} chan(P_i) - chan(P_k)) \times [K_g, \infty) \; \}.
\end{array}
$$

- In the communication guarded command $[[]_{i=1}^{n} c_i?x_i \rightarrow P_i [] \mathbf{delay}\, d \rightarrow P_0]$, 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, the process P_0 is executed.

$$
\begin{array}{l}
\mathcal{M}[\![\, [\ []_{i=1}^{n}\ c_i?x_i \rightarrow P_i\ []\ \mathbf{delay}\ d \rightarrow P_0\ ]\,]\!] = \\
\cup_{i=1}^{n}\ \{\ (\ \sigma_0, (\langle(\tau, c_i, \mu)\rangle\hat{}\theta, \mathfrak{R}), \sigma\ ) \\
\qquad |\ \mathcal{E}[\![T]\!](\sigma_0) = 0,\ K_\alpha \leq \tau < d,\ \mu \in VAL, \text{ and there exists an } \widehat{\mathfrak{R}} \\
\qquad\ \text{with } \widehat{\mathfrak{R}}\uparrow[0, \tau + K_\omega + K_{\mathbf{a}}) = \emptyset, \\
\qquad\ \mathfrak{R}\ =\ \ (\cup_{j=0}^{n} chan(P_j)\ \cup\ \cup_{j=1}^{n}\{c_j\}) \times [0, \tau + K_\omega + K_{\mathbf{a}}) \\
\qquad\qquad\quad - \{(c_i, \tau)\} - \cup_{j=1}^{n}\{c_j\} \times [K_\alpha, \tau) \\
\qquad\qquad\ \cup\ \widehat{\mathfrak{R}} \\
\qquad\qquad\ \cup\ ((\cup_{j=0}^{n} chan(P_j)\ \cup\ \cup_{j=1}^{n}\{c_j\}) - chan(P_i)) \\
\qquad\qquad\quad \times [\tau + K_\omega + K_{\mathbf{a}}, \infty)\ , \\
\qquad\ \text{and } (\ (\sigma_0 : x_i \mapsto \mu)\ , \\
\qquad\qquad (\theta, \widehat{\mathfrak{R}}) \curvearrowleft (\tau + K_\omega + K_{\mathbf{a}})\ , \\
\qquad\qquad (\sigma : T \mapsto T - \tau - K_\omega - K_{\mathbf{a}})\ ) \in \mathcal{M}[\![P_i]\!]\ \} \\
\cup \quad \{\ (\ \sigma_0, (\theta, \mathfrak{R}), \sigma\ ) \\
\qquad |\ \mathcal{E}[\![T]\!](\sigma_0) = 0, \text{ and there is a } \widehat{\mathfrak{R}} \text{ such that } \widehat{\mathfrak{R}}\uparrow[0, d) = \emptyset, \\
\qquad\ (\sigma_0\ ,\ (\theta, \widehat{\mathfrak{R}}) \curvearrowleft d\ ,\ (\sigma : T \mapsto T - d)) \in \mathcal{M}[\![P]\!], \text{ and} \\
\qquad\ \mathfrak{R}\ =\ \ \cup_{j=0}^{n} chan(P_j) \times [0, d)\ \cup\ \cup_{j=1}^{n}\{c_j\} \times [0, K_\alpha) \\
\qquad\qquad\ \cup\ \widehat{\mathfrak{R}}\ \cup\ (\cup_{j=1}^{n}(chan(P_j)\ \cup\ \{c_j\}) - chan(P_0)) \times [d, \infty)\ \}.
\end{array}
$$

- An execution of $*\,G$ consists of either an infinite number of executions of G that terminate in a state in which at least one of its 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.

$$
\begin{array}{l}
\mathcal{M}[\![*\,G]\!] = \\
\{\ (\ \sigma_0, (\theta, \mathfrak{R}), \sigma\ ) \\
\ \ |\ \mathcal{E}[\![T]\!](\sigma_0) = 0 \text{ and there exists a } k \in \mathbb{N} \cup \{\infty\}, \text{ and for every } i, \\
\quad 0 \leq i < k, \text{ there exists a triple } (\sigma_i\ ,\ (\theta_{i+1}, \mathfrak{R}_{i+1})\ ,\ \sigma_{i+1}) \text{ such that} \\
\quad \sigma_i \neq \perp,\ \mathcal{B}[\![b_G]\!](\sigma_i), \\
\quad \mathfrak{R}_{i+1}\uparrow[0, \mathcal{E}[\![T]\!](\sigma_i)) = chan(G) \times [0, \mathcal{E}[\![T]\!](\sigma_i)), \\
\quad (\ (\sigma_i : T \mapsto 0)\ , \\
\quad\ \ (\theta_{i+1}, \mathfrak{R}_{i+1}) \curvearrowleft \mathcal{E}[\![T]\!](\sigma_i)\ , \\
\quad\ \ (\sigma_{i+1} : T \mapsto T - \mathcal{E}[\![T]\!](\sigma_i))\ ) \in \mathcal{M}[\![G]\!], \text{ and} \\
\quad \text{if } k = \infty \text{ then} \\
\qquad \text{for all } j,\ 1 \leq j < k,\ \theta_1\hat{}\ldots\hat{}\theta_j\ \preceq\ \theta,\ \cap_{l=1}^{j}\ \mathfrak{R}_l\ \supseteq\ \mathfrak{R}, \text{ and } \sigma = \perp, \\
\quad \text{else} \\
\qquad \theta\ =\ \theta_1\hat{}\ldots\hat{}\theta_k,\ \mathfrak{R}\ =\ \cap_{l=1}^{k}\ \mathfrak{R}_l,\ \sigma = \sigma_k, \\
\qquad \text{and if } \sigma_k \neq \perp \text{ then } \mathcal{B}[\![\neg b_G]\!](\sigma_k)\ \}.
\end{array}
$$

- 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. As in Section 3.3, we further require that $\theta \uparrow chan(P_1 \| P_2) = \theta$. Communications along the channels in $chan(P_1) \cap chan(P_2)$ are refused by $P_1 \| P_2$ if, and only 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, and only if, they are refused by P_1 or P_2. Observe that process $P_1 \parallel P_2$ terminates if, and only if, both P_1 and P_2 terminate.

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

- The observations of $P \setminus cset$, 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 that they are no longer refused. Recall that we allow only one communication per channel to occur at a particular instant. Furthermore, by our definition of the semantics it takes a non-zero period before such a taken communication can become enabled again. Hence, an observation of $P \setminus cset$ is characterized by the fact that $cset$ communications are continuously refused, except at particular instants.

Definition 5.19 (As soon as possible) For a timed refusal set $\mathfrak{R}$ and a set $cset$ of channels:

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

◇

Then,

$$
\begin{array}{rcl}
\mathcal{M}[\![P \setminus cset]\!] & = & \{ \ (\ \sigma_0 \ , \ (\theta \setminus cset \ , \ \mathfrak{R} \setminus cset) \ , \ \sigma \ ) \\
& & \quad | \ (\ \sigma_0 \ , \ (\theta \ , \ \mathfrak{R}) \ , \ \sigma \ ) \in \mathcal{M}[\![P]\!] \ \wedge \ ASAP(\mathfrak{R}, cset) \ \}.
\end{array}
$$

Notice that this definition incorporates finite variability (also called non-Zenoness). Having defined the meaning of processes we again abstract from the internal states.

Definition 5.20 (Timed observations) The *timed observations* of a process P, notation $\mathcal{O}[\![P]\!]$, follow from:

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

$\diamond$

The set $\mathcal{O}[\![P]\!]$ represents the normal behaviour of process P. In Section 5.5 we determine the set $\mathcal{O}[\![P \wr \chi]\!]$ representing the acceptable behaviour of P under the 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 $\mathfrak{R} \uparrow chan(P) = \mathfrak{R}$.

5.4 Assertion language and correctness formulae

In this chapter the observable quantities are the communication history and the refusal set of the process. Similar to the semantic denotation of traces in Section 5.2, we use in assertions record expressions such as (τ, c, μ), with $\tau \in TIME$, $c \in CHAN$ and $\mu \in VAL$. We use time 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, and a number of n-ary functions which remain uninterpreted. To reason about natural numbers, the assertion language includes the operator *len*. 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 an integer expression *iexp* we use *texp*(*iexp*) to refer a particular record of *texp*, provided *iexp* is a positive natural number less than or equal to the length of trace *texp*. Similar to the semantic denotation of refusal sets in Section 5.2, we use expressions such as $cset \times [\tau_1, \tau_2)$ and the projection operator $\uparrow$ to form refusal expressions.

Let *IVAR*, with typical representatives i, j, k, l, and m, denote the set of logical value variables ranging over $\mathbb{N}$, let *TIVAR*, 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*.

To refer to the timed observation of a process we use the special variables h and R to denote the trace and the refusal set of the process, respectively. Then, we can write specifications like $c!2$ **sat** $h\uparrow\{c\} = \langle\rangle \vee \exists t \geq 0 \cdot h\uparrow\{c\} = \langle(t, c, 2)\rangle$ and $x := e$ **sat** $R = \emptyset$. 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*.

Table 5.2 presents the language we use to define assertions, with $i \in \mathbb{N}$, $\tau \in \mathit{TIME}$, $t \in \mathit{IVAR}$, $c \in \mathit{CHAN}$, $\mu \in \mathit{VAL}$, $v \in \mathit{VVAR}$, $s \in \mathit{TVAR}$, $N \in \mathit{RVAR}$, and $\mathit{cset} \subseteq \mathit{CHAN}$. Observe that an expression in the assertion language of Table 5.2 does not refer to program variables.

Table 5.2: Syntax of the assertion language

Integer expression	iexp	$::=$	$0 \mid 1 \mid i \mid \mathit{iexp}_1 + \mathit{iexp}_2 \mid \mathit{iexp}_1 \times \mathit{iexp}_2 \mid \mathit{len}(\mathit{texp})$
Time expression	tixp	$::=$	$\tau \mid t \mid \mathit{ts}(\mathit{rexp}) \mid \mathit{tixp}_1 + \mathit{tixp}_2$
Channel expression	cexp	$::=$	$c \mid \mathit{ch}(\mathit{rexp})$
Value expression	vexp	$::=$	$\mu \mid v \mid \mathit{iexp} \mid \mathit{val}(\mathit{rexp}) \mid f(\mathit{vexp}_1, \ldots, \mathit{vexp}_n)$
Record expression	rexp	$::=$	$(\mathit{tixp}, \mathit{cexp}, \mathit{vexp}) \mid \mathit{texp}(\mathit{iexp})$
Trace expression	texp	$::=$	$s \mid h \mid \langle\rangle \mid \langle \mathit{rexp} \rangle \mid \mathit{texp}_1{}^\wedge\mathit{texp}_2 \mid \mathit{texp}\!\uparrow\!\mathit{cset}$
Interval expression	inxp	$::=$	$[\mathit{tixp}_1, \mathit{tixp}_2) \mid \{\mathit{tixp}\}$
Refusal expression	rfxp	$::=$	$N \mid R \mid \emptyset \mid \mathit{cset} \times \mathit{inxp} \mid \mathit{rfxp}_1 \cup \mathit{rfxp}_2 \mid \mathit{rfxp}\!\uparrow\!\mathit{cset}$
Assertion	ϕ	$::=$	$\mathit{iexp}_1 = \mathit{iexp}_2 \mid \mathit{iexp}_1 < \mathit{iexp}_2 \mid \mathit{tixp}_1 = \mathit{tixp}_2 \mid \mathit{tixp}_1 < \mathit{tixp}_2 \mid \mathit{cexp}_1 = \mathit{cexp}_2 \mid \mathit{vexp}_1 = \mathit{vexp}_2 \mid \mathit{vexp}_1 < \mathit{vexp}_2 \mid \mathit{texp}_1 = \mathit{texp}_2 \mid \mathit{rfxp}_1 = \mathit{rfxp}_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \exists i \cdot \phi \mid \exists t \cdot \phi \mid \exists v \cdot \phi \mid \exists s \cdot \phi \mid \exists N \cdot \phi$

Definition 5.21 (Primitive predicates I) *Primitive predicates* have a free variable t, the current reference point of time (called the 'base time'). For a set *cset* of channels and a time expression *tixp*, a few typical examples are:

- **enable** *cset* **at** *tixp* $\equiv$ $(\mathit{cset} \times \mathit{tixp}) \cap R = \emptyset$;
- **enable** *cset* **for** *tixp* $\equiv$ $(\mathit{cset} \times [t, t + \mathit{tixp}]) \cap R = \emptyset$;
- **refuse** *cset* **upto** *tixp* $\equiv$ $\mathit{cset} \times [t, t + \mathit{tixp}) \subseteq R$;
- **refuse** *cset* **precisely upto** *tixp* $\equiv$ $\forall \widehat{t} \cdot ($ **refuse** *cset* **upto** $\widehat{t} \leftrightarrow \widehat{t} \leq \mathit{tixp}$);
- **after** *tixp* : $\phi \equiv \phi[t + \mathit{tixp}/t]$,

  where $[t + \mathit{iexp}/t]$ denotes syntactic substitution of $t + \mathit{iexp}$ for t. This construct allows the base time to be updated.

plus obvious combinations, e.g. **refuse** *cset* **precisely for** *iexp* and 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 where messages are lost due to faults. 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 yet come forward. Using assumptions about the readiness of the environment to receive a message elegantly resolves such issues.

If P did not refuse a c communication at time τ, then the fact that no c communication occurred at τ, implies that the environment was not prepared to engage in a c communication at that time. On the other hand, a c communication that did occur at time τ could not have been refused by the environment. Thus, we can define possible refusal sets of the environment.

Definition 5.22 (Match) A timed refusal set N *matches* timed trace h and timed refusal set R, notation $Match(h, R, N)$, if, and only if,

$$\begin{aligned} &\forall c, t \cdot (\ (c,t) \notin R \ \land \ \neg(\exists v \cdot (t,c,v) \in h)\ ) \ \rightarrow \ (c,t) \in N \\ \land\ &\forall c, t, v \cdot (t,c,v) \in h \ \rightarrow \ (c,t) \notin N. \end{aligned}$$

◇

Definition 5.23 (Primitive predicates II) For a set *cset* of channels and a time expression *tixp*, a few typical examples of primitive predicates are:

- *cset* **enabled at** *tixp* $\equiv$
  $\forall N \cdot Match(h,R,N) \ \rightarrow \ (cset \times tixp) \cap N \ = \ \emptyset$;
- *cset* **enabled for** *tixp* $\equiv$
  $\forall N \cdot Match(h,R,N) \ \rightarrow \ (cset \times [t, t + tixp]) \cap N \ = \ \emptyset$;
- *cset* **refused for** *tixp* $\equiv$
  $\forall N \cdot Match(h,R,N) \ \rightarrow \ cset \times [t, t + tixp] \ \subseteq \ N$;
- *cset* **refused upto** *tixp* $\equiv$
  $\forall N \cdot Match(h,R,N) \ \rightarrow \ cset \times [t, t + tixp) \ \subseteq \ N$;
- *cset* **refused precisely upto** *tixp* $\equiv$
  $\forall \widehat{t} \cdot (\ cset\ \textbf{refused upto}\ \widehat{t} \ \leftrightarrow \ \widehat{t} \leq tixp\ )$.

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 5.24 (Calculator) Consider the 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, the next input becomes enabled after ε time units. We specify C as follows:

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

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 Definition E.1 determines the set $\mathit{chan}(\phi)$ of channels such that $c \in \mathit{chan}(\phi)$ if a communication along c might affect the validity of ϕ. As before, $\mathit{chan}(\phi)$ contains the history channels of ϕ. Since, by the definition of the semantics, communications on a channel are refused for some time after a communication on that channel has occurred, the assertion $R{\uparrow}\{c\} = \emptyset$ is invalidated by a communication along c. This is also the case for the assertion $R{\uparrow}\{c\} = \{c\} \times [0, \infty)$, since a communication cannot occur when refused. Hence, $\mathit{chan}(\phi)$ consists of the channels to which references to h and R in ϕ are restricted, the so-called ***observation*** channels of ϕ.

We use an environment γ to interpret the logical variables of $\mathit{IVAR} \cup \mathit{TIVAR} \cup \mathit{VVAR} \cup \mathit{TVAR} \cup \mathit{RVAR}$. This environment maps a logical value variable i to a value $\gamma(i) \in \mathbb{N}$, logical time variable t to a value $\gamma(t) \in \mathit{TIME}$, a logical value variable v to a value $\gamma(v) \in \mathit{VAL}$, a logical trace variable s to a trace $\gamma(s) \in \mathit{TRACE}$, and a logical refusal variable N to a refusal set $\gamma(N) \in \mathit{REF}$. An assertion is interpreted with respect to a triple $(\theta, \mathfrak{R}, \gamma)$. Trace θ gives h its value, and refusal set $\mathfrak{R}$ gives R its value, and the environment γ interprets the logical variables. We use the special symbol $\dagger$ to deal with the interpretation of $\mathit{texp}(\mathit{iexp})$ where index iexp is not positive, or greater than the length of texp. The value of an expression is undefined whenever a subexpression yields $\dagger$. The meaning of assertions is given in Definition E.2.

Definition E.3 expresses when an assertion ϕ holds for trace θ, refusal $\mathfrak{R}$, and an environment γ, notation $(\theta, \mathfrak{R}, \gamma) \models \phi$. Again, to avoid the complexity of a three-valued logic, an (in)equality predicate is interpreted strictly with respect to $\nmid$, that is, it is false if it contains some expression that has an undefined value.

Example 5.25 (Satisfaction) In Example 5.24 we came across the 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 maps the logical variables t and v to the respective values τ and μ,

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

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

Definition 5.26 (Validity of an assertion) An assertion ϕ is *valid*, notation $\models \phi$, if, and only if, for all θ, $\mathfrak{R}$, and γ, $(\theta, \mathfrak{R}, \gamma) \models \phi$. $\diamond$

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

Definition 5.27 (Validity of a correctness formula) For process P and assertion ϕ correctness formula P **sat** ϕ is *valid*, notation $\models P$ **sat** ϕ, if, and only if, for all γ and all $(\theta, \mathfrak{R}) \in \mathcal{O}[\![P]\!]$, $(\theta, \mathfrak{R}, \gamma) \models \phi$. $\diamond$

5.5 Incorporating failure hypotheses

As we have observed earlier, the set of observations that characterize a process must be expanded to take account of a particular failure hypothesis. To be able to formulate a nice proof rule, we follow the approach taken in Chapter 3 and formalize failure hypothesis χ of process P 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. Now, 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 some process is willing sooner than usual to receive new input, then this input will still not occur before the environment is able to provide it.

We extend the assertion language to include the trace expression term h_{old} and the refusal expression term R_{old}. Sentences of the extended language are called *transformation expressions*, with typical representative ψ. To indicate that the 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 5-tuple $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma)$. 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 values, and the environment γ interprets the logical variables of $IVAR \cup TIVAR \cup VVAR \cup TVAR \cup RVAR$. The meaning of transformation expressions can be obtained from Definition E.2 by adding the clauses:

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

Let $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \psi$ denote that ψ holds for 5-tuple $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma)$. Since the terms h_{old} and R_{old} do not occur in assertions, the following lemma is trivial.

Lemma 5.28 (Correspondence) For assertion ϕ, trace θ_0, and refusal $\mathfrak{R}_0$ it is the case that $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \phi$ if, and only if, $(\theta, \mathfrak{R}, \gamma) \models \phi$. ○

The following lemma is easy to prove by structural induction.

Lemma 5.29 (Substitution) For the transformation expression ψ,

(a) $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \psi(texp, h, R_{old}, R)$
if, and only if, $(\mathcal{T}[\![texp]\!](\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma), \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \psi$;

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

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

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

Definition 5.30 (Validity of a transformation expression) A transformation expression ψ is *valid*, notation $\models \psi$, if, and only if, for all θ_0, θ, $\mathfrak{R}_0$, $\mathfrak{R}$ and γ, $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \psi$. ◇

The observation channels that appear in a transformation expression are as defined in Definition E.1 with the extra clauses

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

Definition 5.31 (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 the failure prone process *FP* does not impose restrictions on communications along those channels that are not in *chan*(*FP*), that is, $\models \chi \rightarrow \chi(h_{old}\uparrow chan(FP), h\uparrow chan(FP), R_{old}\uparrow chan(FP), R\uparrow chan(FP))$. $\diamond$

Care has to be taken that a failure hypothesis upholds the principle that communications cannot occur while being refused, or must occur as soon as no longer refused. Also, a failure hypothesis must not allow communications via the same channel to follow each other arbitrarily fast, or to coincide.

Example 5.32 Consider the process C introduced in Example 5.24. Examine:

$$\begin{aligned} \psi \equiv \quad & h\uparrow\{in, out\} = \langle\rangle \\ & \wedge\ R\uparrow\{in, out\} = R_{old}\uparrow\{in, out\}. \end{aligned}$$

Observe that this transformation expression expresses that no communications along the channels *in* and *out* occur, but does not require $R\uparrow\{in, out\}$ to be $\{in, out\} \times [0, \infty)$. $\triangle$

Example 5.33 (Corruption) Consider once more the process C. Assuming that corruption does not influence the real-time behaviour of C, we formalize corruption by asserting that

- with respect to the number of recorded *in* and *out* communications h_{old} and h are equally long,
- the order of *in* and *out* communications as recorded by h_{old} is preserved by h,
- the ith input value as recorded by h equals the ith input value as recorded by h_{old},
- as far as the timestamps of the *out* communications are concerned h conforms to h_{old}, and
- with respect to the refused attempts to communicate along *in* and *out* R equals R_{old}.

Formally,

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

$\triangle$

To specify *failure prone processes* we again use the construct $P \wr \chi$ to indicate execution of process P under the assumption χ. Using P to denote a process expressed in the programming language of Section 5.1, we define the syntax of our extended programming language in Table 5.3.

Table 5.3: 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 5.31 we obtain $chan(\chi) \subseteq chan(FP)$. Hence, $chan(FP \wr \chi) = chan(FP) \cup chan(\chi) = chan(FP)$. As before, we define $chan(FP_1 \parallel FP_2) = chan(FP_1) \cup chan(FP_2)$, and $chan(FP \backslash cset) = chan(FP) - cset$.

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

- Notice that failure prone processes FP_1 and FP_2 synchronize only 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 5.3, 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. As in Section 3.6 we require that $\theta \uparrow chan(FP) = \theta$ and, for similar reasons, $\mathfrak{R} \uparrow chan(FP) = \mathfrak{R}$.

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

Definition 5.34 (Composite transformation expression) For the transformation expressions $\psi_1(h_{old}, h, R_{old}, R)$ and $\psi_2(h_{old}, h, R_{old}, R)$, we define the *composite transformation expression* $\psi_1 \wr \psi_2$ 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. ◇

We will also use this operator to compose assertions and transformation expressions, e.g. $\phi \wr \psi \equiv \exists s, N \cdot \phi(s, N) \wedge \psi(s, h, N, R)$. 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.

Since the interpretation of assertions has not changed, the validity of a correctness formula FP **sat** ϕ is defined as in Definition 5.27, with P replaced by FP.

Definition 5.35 (Validity of a correctness formula) For process FP and assertion ϕ correctness formula FP **sat** ϕ is *valid*, notation $\models FP$ **sat** ϕ, if, and only if, for all γ and all $(\theta, \mathfrak{R}) \in \mathcal{O}[\![FP]\!]$, $(\theta, \mathfrak{R}, \gamma) \models \phi$. ◇

5.6 A compositional network proof theory for failure prone processes

In this section we give a compositional network proof system for the correctness formulae of Section 5.4. As in Section 3.7 we do not give rules for atomic statements or sequential composition. The proof system contains the following two general rules.

Rule 5.36 (Consequence)

$$\frac{FP \ \mathbf{sat}\ \phi_1\ ,\quad \phi_1 \ \rightarrow\ \phi_2}{FP\ \mathbf{sat}\ \phi_2}$$

Rule 5.37 (Conjunction)

$$\frac{FP\ \mathbf{sat}\ \phi_1\ ,\quad FP\ \mathbf{sat}\ \phi_2}{FP\ \mathbf{sat}\ \phi_1\ \wedge\ \phi_2}$$

From the definition of the semantics we get:

Rule 5.38 (Invariance)

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

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.

Rule 5.39 (Parallel composition)

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

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 history of the process, as do records of refused attempts from the refusal set of the process. 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 5.40 (Hiding)

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

Lemma 5.41 (Hiding) With respect to hiding the following equalities are useful:

(a) $(FP_1 \backslash cset) \| FP_2 = (FP_1 \| FP_2) \backslash cset$ if, and only if, $chan(FP_2) \cap cset = \emptyset$;

(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 5.42 (Failure hypothesis introduction)

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

5.7 Example : Triple modular redundancy

Consider the triple modular redundant system of Figure 5.1. It consists of three identical components C_j, $j = 1, 2, 3$, already discussed in Example 5.24, an input triplicating component *In*, and a component *Voter* that determines the ultimate output. We assume that each component needs exactly 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 accept fresh input arbitrarily fast, and hence confuses *Voter*, usually a synchronization channel *sync* is added. In this section we give the main steps of the proof that failure of at most one component per round can be tolerated.

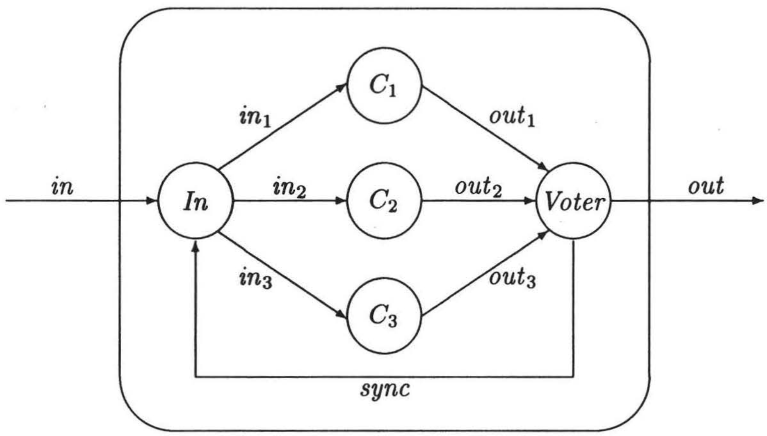

Figure 5.1: Triple modular redundant system

Definition 5.43 (Abbreviations)

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

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

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

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

In accepts a value from the environment via channel *in* and distributes that value via channels in_1, in_2 and in_3 after K_{In} time units. To keep the proof concise we assume that *In* simultaneously enables the in_1, in_2, and in_3 communications. When these three communications have occurred *In* tries to communicate via *sync*. ε time units after the communication on *sync* has occurred, it enables *in* again. The specification of *In* only deals with the occurrence of in_1, in_2, and in_3 communications as far as they coincide.

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

Voter awaits a communication on any of the channels out_1, out_2 and out_3. Upon the occurrence of such a communication it starts a timer and awaits the remaining communications: if they do not occur within Δ time units the timer expires, and *Voter* determines the output that is to be communicated to the environment on the basis of the values that are available. Thus, timing is essential as it makes it possible to avoid waiting for a value that will never be produced. ε time units after an output occurs, *Voter* tries to synchronize with *In* on *sync*. After this communication takes place, it enables channels out_1, out_2 and out_3 again. The specification of *Voter* only deals with out_1, out_2 and out_3 communications as far as at least two of them coincide.

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

Since C_1, C_2, and C_3 do not share a single channel, we easily obtain, by parallel composition rule 5.39 and consequence rule 5.36, that

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

Under the assumption that the environment offers input sufficiently far apart, faults do not change the rate at which a component accepts input. We formalize the hypothesis that on each round at most one of the components C_1, C_2, and C_3 fails to provide output by asserting that

- with respect to the channels in_1, in_2, and in_3 h equals h_{old},
- per round i at least two of the out_1, out_2, and out_3 communications as recorded by h_{old} also occur according to h,
- with respect to the channels in_1, in_2, and in_3 R equals R_{old}, and
- whenever an out_j communication (j = 1, 2, or 3) is omitted that communication is refused from the preceeding to the succeeding in_j.

Formally,

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

The failure hypothesis $Loss^{\leq 1}$ means 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 output on out_1, out_2, and out_3 to conclude that if a particular output does not occur it is due to a failure rather than to the unavailability of a communication partner. Hence, by applying failure hypothesis introduction rule 5.42 and consequence rule 5.36 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 the third, then all three components will accept fresh input ε time units thereafter.

$$
\begin{array}{l}
(C_1\|C_2\|C_3) \wr Loss^{\leq 1} \\
\textbf{sat} \\
\quad h = \langle\rangle \rightarrow \textbf{enable } \bigcup_{j=1}^{3} \{in_j\} \textbf{ upto } \infty \\
\wedge\ \forall t, v \cdot (\bigwedge_{j=1}^{3}(t, in_j, v) \in h) \\
\qquad\quad \rightarrow \quad \textbf{refuse } \bigcup_{j=1}^{3} \{out_j\} \textbf{ upto } K_C \\
\qquad\qquad \wedge\ \textbf{after } K_C : \\
\qquad\qquad\quad \forall t_1 \cdot \bigcup_{j=1}^{3}\{out_j\} \textbf{ refused precisely upto } t_1 \\
\qquad\qquad\qquad \rightarrow \quad \exists k \neq l \cdot \\
\qquad\qquad\qquad\qquad \textbf{enable } \{out_k, out_l\} \textbf{ for } t_1 \\
\qquad\qquad\qquad\qquad \wedge\ \forall v_1, v_2 \cdot ((t_1, out_k, v_1) \in h\ \wedge\ (t_1, out_l, v_2) \in h) \\
\qquad\qquad\qquad\qquad\qquad\qquad \rightarrow v_1 = v_2 = f(v) \\
\qquad\qquad\qquad \wedge\ \bigcup_{j=1}^{3}\{out_j\} \textbf{ enabled at } t_1 \\
\qquad\qquad\qquad\quad \rightarrow \textbf{after } t_1 + \varepsilon : \\
\qquad\qquad\qquad\qquad \forall t_2 \cdot \bigcup_{j=1}^{3}\{in_j\} \textbf{ refused precisely upto } t_2 \\
\qquad\qquad\qquad\qquad\qquad \rightarrow \textbf{enable } \bigcup_{j=1}^{3} \{in_j\} \textbf{ for } t_2.
\end{array}
$$

Observe that, due to the assumptions concerning the environment's enabledness to communicate, we only need the specifications of the components C_1, C_2, and C_3 and the failure hypothesis $Loss^{\leq 1}$ to establish this non-blocking property. This property assures that if *In* simultaneously enables communication via in_1, in_2, and in_3 then these communications indeed occur simultaneously. This justifies the incompleteness in the specifications of *In*, $C_1\|C_2\|C_3$, and *Voter*.

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 readiness of *Voter* does not change until an out_j communication, $j = 1, 2, 3$, actually takes place. Using $h_{Voter} \equiv h{\uparrow}chan(Voter)$, we obtain, by parallel composition rule 5.39:

$$
\begin{array}{l}
(\ (C_1 \| C_2 \| C_3) \wr Loss^{\leq 1}\ )\ \|\ Voter \\
\mathbf{sat} \\
ASAP(R, \cup_{j=1}^{3}\{out_j\}) \rightarrow \\
\quad \forall t, v \cdot (\ \ \bigwedge_{j=1}^{3}(t, in_j, v) \in h \\
\qquad\qquad \wedge\ \ \mathbf{until}(h_{Voter}, t) = \langle\rangle \\
\qquad\qquad \vee\ \exists t_1, v_1 \cdot \mathbf{last}(\mathbf{until}(h_{Voter}, t)) = (t_1, sync, v_1)\ ) \\
\qquad\quad \rightarrow \quad \exists t_1 \cdot \quad t_1 = 0 \vee t_1 = \Delta \\
\qquad\qquad\qquad\qquad \wedge\ \mathbf{refuse}\ out\ \mathbf{upto}\ K_C + t_1 + K_{Voter} \\
\qquad\qquad\qquad\qquad \wedge\ \mathbf{after}\ K_C + t_1 + K_{Voter} : \\
\qquad\qquad\qquad\qquad\quad \forall t_2 \cdot out\ \mathbf{refused\ precisely\ upto}\ t_2 \\
\qquad\qquad\qquad\qquad\qquad \rightarrow \quad \mathbf{enable}\ out\ \mathbf{for}\ t_2 \\
\qquad\qquad\qquad\qquad\qquad\qquad \wedge\ \forall v_1 \cdot (t_2, out, v_1) \in h \rightarrow v_1 = v \\
\qquad\qquad\qquad \wedge\ \mathbf{after}\ K_C + \varepsilon : \forall t_1 \cdot \cup_{j=1}^{3}\{in_j\}\ \mathbf{refused\ precisely\ upto}\ t_1 \\
\qquad\qquad\qquad\qquad\qquad\qquad \rightarrow\ \mathbf{enable}\ \cup_{j=1}^{3}\ \{in_j\}\ \mathbf{for}\ t_1 \\
\wedge\ \forall t, v \cdot (t, out, v) \in h \rightarrow \quad \mathbf{refuse}\ sync\ \mathbf{upto}\ \varepsilon \\
\qquad\qquad\qquad\qquad\qquad \wedge\ \mathbf{after}\ \varepsilon : \forall t_1 \cdot sync\ \mathbf{refused\ precisely\ upto}\ t_1 \\
\qquad\qquad\qquad\qquad\qquad\qquad \rightarrow\ \mathbf{enable}\ sync\ \mathbf{for}\ t_1.
\end{array}
$$

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

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 there either has been no in_1, in_2, or in_3 communication, or the preceding in_1, in_2, and in_3 communications all happened at the same time, then, for $j = 1, 2, 3$, C_j does not refuse to communicate via in_j at time t. Again, this readiness does not change until an in_j communication, $j = 1, 2, 3$, actually occurs. By hiding rule 5.40, the specification of *In*, and parallel composition rule 5.39,

$$
\begin{array}{l}
In \;\|\; (\;((C_1\|C_2\|C_3) \wr Loss^{\leq 1})\|\, Voter\;) \setminus \bigcup_{j=1}^{3}\{out_j\} \\
\textbf{sat} \\
ASAP(R, \cup_{j=1}^{3}\{in_j\} \cup \{sync\}) \rightarrow \\
\quad \forall t, v \cdot (\quad (t, in, v) \in h \\
\qquad\qquad \wedge \quad \textbf{until}(h \uparrow \cup_{j=1}^{3} \{in_j\}, t) = \langle\rangle \\
\qquad\qquad \vee\; \exists t_1, v_1 \cdot \bigwedge_{j=1}^{3} \textbf{last}(\textbf{until}(h_{C_j}, t)) = (t_1, in_j, v_1)\;) \\
\qquad \rightarrow \exists t_1 \cdot \quad t_1 = 0 \vee t_1 = \Delta \\
\qquad\qquad\qquad \wedge\; \textbf{refuse}\; out\; \textbf{upto}\; K_{In} + K_C + t_1 + K_{Voter} \\
\qquad\qquad\qquad \wedge\; \textbf{after}\; K_{In} + K_C + t_1 + K_{Voter} : \\
\qquad\qquad\qquad\quad \forall t_2 \cdot out\; \textbf{refused precisely upto}\; t_2 \\
\qquad\qquad\qquad\qquad \rightarrow \quad \textbf{enable}\; out\; \textbf{for}\; t_2 \\
\qquad\qquad\qquad\qquad\quad \wedge\; \forall v_1 \cdot (t_2, out, v_1) \in h \rightarrow v_1 = f(v) \\
\wedge\; \forall t, v \cdot (t, out, v) \in h \rightarrow \quad \textbf{refuse}\; in\; \textbf{upto}\; 2\varepsilon \\
\qquad\qquad\qquad\qquad\qquad \wedge\; \textbf{after}\; 2\varepsilon : \forall t_1 \cdot in\; \textbf{refused precisely upto}\; t_1 \\
\qquad\qquad\qquad\qquad\qquad\qquad \rightarrow \textbf{enable}\; in\; \textbf{for}\; t_1.
\end{array}
$$

If the first in communication occurs at time t then $\textbf{until}(h_{C_1\|C_2\|C_3}, t) = \langle\rangle$. 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 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 rule 5.40

$$
\begin{array}{l}
(\; In \;\|\; ((C_1\|C_2\|C_3) \wr Loss^{\leq 1}) \;\|\; Voter\;) \setminus \bigcup_{j=1}^{3}\{in_j\} \cup \bigcup_{j=1}^{3}\{out_j\} \cup \{sync\} \\
\textbf{sat} \\
\quad \forall t, v \cdot (t, in, v) \in h \rightarrow \\
\qquad \exists t_1 \cdot \quad t_1 = 0 \vee t_1 = \Delta \\
\qquad\qquad \wedge\; \textbf{refuse}\; out\; \textbf{upto}\; K_{In} + K_C + t_1 + K_{Voter} \\
\qquad\qquad \wedge\; \textbf{after}\; K_{In} + K_C + t_1 + K_{Voter} : \\
\qquad\qquad\quad \forall t_2 \cdot out\; \textbf{refused precisely upto}\; t_2 \\
\qquad\qquad\qquad \rightarrow \quad \textbf{enable}\; out\; \textbf{for}\; t_2 \\
\qquad\qquad\qquad\quad \wedge\; \forall v_1 \cdot (t_2, out, v_1) \in h \rightarrow v_1 = f(v) \\
\wedge\; \forall t, v \cdot (t, out, v) \in h \rightarrow \quad \textbf{refuse}\; in\; \textbf{upto}\; 2\varepsilon \\
\qquad\qquad\qquad\qquad\qquad \wedge\; \textbf{after}\; 2\varepsilon : \forall t_1 \cdot in\; \textbf{refused precisely upto}\; t_1 \\
\qquad\qquad\qquad\qquad\qquad\qquad \rightarrow \textbf{enable}\; in\; \textbf{for}\; t_1.
\end{array}
$$

which shows that omission faults are tolerated. In practice, Δ is determined on the basis of the variations in the respective processing times; in the above expression the clause $t_1 = 0 \vee t_1 = \Delta$ then has to be replaced by $0 \leq t_1 \leq \Delta$.

5.8 Soundness and relative network completeness

In this section we show that the proof system of Section 5.6 is sound and relatively network complete. To do so, we follow the approach of Section 3.10.

Theorem 5.44 (Soundness) The proof system of Section 5.6 is sound.

Proof. See Appendix F.1.

Axiom 5.45 (Relative completeness assumption) For an assertion ϕ,

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

○

Definition 5.46 (Relative preciseness) An assertion ϕ is *relatively precise* for failure prone process *FP* if, and only if,

i) $\models FP$ **sat** ϕ;

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) $\phi \rightarrow \phi(h{\uparrow}chan(FP), R{\uparrow}chan(FP))$. ◇

As in Chapter 3, an (absolutely) precise specification can be obtained from a specification which is only relatively precise by means of the invariance and conjunction rules. In the sequel, preciseness refers to relative preciseness.

Let, as before, $\vdash P$ **sat** ϕ denote that correctness formula P **sat** ϕ is derivable.

Definition 5.47 (Network completeness) Assume that for every process P there exists a precise assertion ϕ with $\vdash P$ **sat** ϕ. Then, for any failure prone process *FP* and assertion ξ, $\models FP$ **sat** ξ implies $\vdash FP$ **sat** ξ. ◇

Lemma 5.48 (Preciseness preservation) Assume that for any process P there exists an assertion ϕ which is precise for P and $\vdash P$ **sat** ϕ. Then, for any failure prone process *FP* there exists an assertion η which is precise for *FP* and $\vdash FP$ **sat** η.

Proof. See Appendix F.2.

Lemma 5.49 (Preciseness consequence) If assertion ϕ is precise for *FP* and $\models FP$ **sat** ξ then $\models (\ \phi \ \wedge \ h{\uparrow}chan(FP) = h \ \wedge \ R{\uparrow}chan(FP) = R\ ) \rightarrow \xi$.

Proof. Assume that ϕ is precise for *FP*, and that

$$\models FP \textbf{ sat } \xi. \tag{5.1}$$

Consider any θ, $\mathfrak{R}$, and γ.
Assume $(\theta, \mathfrak{R}, \gamma) \models \phi \wedge h{\uparrow}chan(FP) = h \wedge R{\uparrow}chan(FP) = R$
Then, by the preciseness of ϕ for *FP*, $(\theta, \mathfrak{R}) \in \mathcal{O}[\![FP]\!]$. By (5.1), for all $\widehat{\gamma}$, $(\theta, \mathfrak{R}, \widehat{\gamma}) \models \xi$. Hence, $(\theta, \mathfrak{R}, \gamma) \models \xi$. □

Theorem 5.50 (Relative network completeness) The proof system of Section 5.6 is relatively network complete.

Proof. Assume that for every process P there exists a precise specification ϕ with $\vdash P$ **sat** ϕ. Then, by preciseness preservation lemma 5.49, for every failure prone process *FP* there exists an assertion η which is precise for *FP* and

$$\vdash FP \textbf{ sat } \eta. \tag{5.2}$$

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

$$\vdash FP \textbf{ sat } h{\uparrow}chan(FP) = h \ \wedge \ R{\uparrow}chan(FP) = R. \tag{5.3}$$

Then, by (5.2), (5.3), preciseness consequence lemma 5.49, relative completeness axiom 5.45, and consequence rule 5.36, $\vdash FP$ **sat** ξ. □

5.9 Discussion

To enable the programming of time-outs the programming language includes a communication guarded command that contains, as one of the guards, a delay statement. This use of a delay statement, which is similar to its use in the **select** construct of Ada [ANSI83], also appears in [Hooman92].

The convention that a process can only refuse communications on its own channels differs from common practice. Usually (e.g. [RR86]), a process constantly refuses to communicate on channels other than its own. Then, a communication on a channel in $chan(FP_1) \cap chan(FP_2)$ is still refused by $FP_1 \| FP_2$ if, and only if, it is refused by either FP_1 or FP_2. However, a communication on a channel in $CHAN - (chan(FP_1) \cap chan(FP_2))$ is refused by $FP_1 \| FP_2$ if, and only if, it is refused by both FP_1 and FP_2:

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

Thus, our convention results in a simpler rule for parallel composition. However, sequential and similar compositions become more complicated because the composite process typically has more channels than each of its components. Referring to the environment's enabledness to communicate proved useful.

It is very convenient to identify a communication record by referring to its timestamp. Unlike its index, which was used in Chapter 3 for identification, the timestamp is invariant under projection.

Unlike parallel composition rule 3.64 parallel composition rule 5.39 does not have a side condition: obviously $chan(\phi_1(h\uparrow chan(FP_1), N_1)) \subseteq chan(FP_1)$ and $chan(\phi_2(h\uparrow chan(FP_2), N_2)) \subseteq chan(FP_2)$. Observe that parallel composition rule 5.39 ensures, by $R = N_1\uparrow chan(FP_1) \cup N_2\uparrow chan(FP_2)$, that the refusal set R of the process $FP_1\|FP_2$ satisfies $R\setminus chan(FP_1\|FP_2) = \emptyset$. This is essential because $\phi_1(h\uparrow chan(FP_1), N_1)$ does not necessarily imply that $N_1 \setminus chan(FP_1) = \emptyset$; the clause $R = N_1 \cup N_2$ would then easily lead to inconsistencies, e.g. in case $N_1\uparrow chan(FP_1) = \emptyset$ and $N_2\uparrow chan(FP_2) = \emptyset$.

Chapter 6

Fault Tolerant Real-Time Distributed Systems with Shared Resources

The timing properties of a reactive real-time program must conform to the requirements of its environment. Given a specification of these timing requirements, one problem is then to construct a real-time program which can be shown to satisfy these timing requirements, despite the occurrence of faults. In the previous chapter we have studied this assuming maximal parallelism, that is, assuming that every process has its own resource. But real-time programs are typically executed on systems whose limited resources are shared according to some scheduling discipline. So another problem is to determine whether a real-time program which meets some timing requirements 'in isolation' will continue to satisfy them when executed under a particular scheduling discipline.

In this chapter we extend the proof theory of the previous chapter to reason about multiprogramming, where several processes share a processor. We assume on-line preemptive dynamic priority scheduling where the priority is a function of the initial priority and the time spent waiting for the resource. We do not explicitly consider the scheduler but concentrate on its effect on the observable process behaviour. We introduce the notion of a multiprocess to conceptually capture the set of processes that share the same processor. Any (nested) parallelism within a multiprocess leads to interleaving the actions of the respective individual processes. This interleaving respects the (dynamic) priorities.

In the previous chapters we discussed the acceptability of an abnormal behaviour with respect to a given failure hypothesis. Multiprogramming leads to a new notion of acceptability, as executions may be interrupted in favour of higher priority tasks and continued later. In essence, scheduling introduces breaks in an execution and the observable behaviour of a process in case of resource sharing is a straightforward extension of the observable behaviour in case of maximal parallelism. To extend the transformation-based compositional reasoning to include failure prone multiprocesses we decorate the timed, infinite traces used in Chapter 5 further with timed histories of both the processor occupation and the outstanding requests. Thus, as opposed to simulation, we can reason already at the specification level about implementability, e.g. in terms of numbers of processors.

In the case of synchronous communication, processes that are communication partners are each dependent on the other for their respective completion. The timing of the actions illustrates how the need for synchronization can cause one partner process to be delayed until its counterpart is ready, and this may occur whether or not the processes are competing for computing resources. If communicating partner processes are scheduled on the same processor, at most one of them can be executed at any time; it would then be incorrect for a communicating process to occupy the processor by busy-waiting while waiting for its partner to be ready. Thus, we can no longer consider communication to be atomic and must assume that a process releases the resource when blocked in a communication.

This chapter is organized as follows. Section 6.1 introduces the programming language for multiprogramming. In Section 6.2 we present the computational model. A denotational semantics is defined in Section 6.3. This semantics incorporates preemption. In Section 6.4 we present the assertion language and associated correctness formulae. In Section 6.5 we once more incorporate failure hypotheses in our formalism. Section 6.6 presents a by now straightforward compositional network proof theory for fault tolerant real-time distributed systems composed of failure prone multiprocesses.

6.1 Programming language for multiprogramming

To characterize scheduling we introduce the notion of a multiprocess in the programming language defined in Table 5.1. A *multiprocess* is a set of possibly parallel processes that share a processor. To distinguish parallel processes executing on a single processor and concurrent processes each having their own resource, we introduce the interleaving composition operator $/\!/$. This interleaving can be restricted by assigning priorities to processes. Priorities take values from $\mathbb{N}$, where a larger value implies a higher priority. The statement **prio** e (P) assigns the value of expression e as priority to the (multi)process P. By default, a process is assigned priority 0. We use the processor closure brackets $\ll$ and $\gg$ to express that the multiprocess that appears inside these brackets is executed on a single processor and that no other process executes on this processor; the communications via the internal channels of the multiprocess are hidden. The processor closure operator creates processes as defined in the previous chapter.

The syntax of the programming language for multiprogramming is given in Table 6.1, with $n \in \mathbb{N}$, $n \geq 1$, $x \in \mathit{VAR}$, $\mu \in \mathit{VAL}$, $c \in \mathit{CHAN}$, $d \in \mathit{TIME}$, and $\mathit{cset} \subseteq \mathit{CHAN}$.

Table 6.1: Syntax of the programming language

Expression	e	$::=$	$\mu \mid x \mid f(e_1,\ldots,e_n)$
Boolean Expression	b	$::=$	$e_1 = e_2 \mid e_1 < e_2 \mid \neg b \mid b_1 \vee b_2$
Guarded Command	G	$::=$	$[\,\|_{i=1}^{n}\, b_i \rightarrow P_i\,] \mid$ $[\,\|_{i=1}^{n}\, c_i?x_i \rightarrow P_i \;\|\; \mathbf{delay}\; d \rightarrow P\,]$
multiProcess	P	$::=$	$x := e \mid c!e \mid c?x \mid P_1;P_2 \mid$ $G \mid *G \mid P_1/\!/P_2 \mid \mathbf{prio}\; e\; (P)$
Network	NP	$::=$	$\ll P \gg \mid NP_1 \parallel NP_2 \mid NP\backslash \mathit{cset}$

Informally, the new statements have the following meaning:

- $P_1/\!/P_2$ indicates the interleaved execution of the processes P_1 and P_2 on the same processor.
- The statement **prio** e (P) assigns the value of expression e as priority to the (multi)process P. Nesting of priority assignments has a cumulative effect: the multiprocess **prio** 3 $(P_1/\!/(\textbf{prio}\ 1\ (P_2)))$ is equivalent with the multiprocess $(\textbf{prio}\ 3\ (P_1))/\!/(\textbf{prio}\ 4\ (P_2))$.
- The processor closure $\ll P \gg$ denotes that the process P has its own processor and no process outside P executes on this processor. This operator hides P's internal channels.

The sets *var* and *chan* are as defined in Section 5.1 with the extra clauses:

- $\mathit{var}(P_1/\!/P_2) = \mathit{var}(P_1) \cup \mathit{var}(P_2)$
- $\mathit{var}(\textbf{prio}\ e\ (P)) = \mathit{var}(e) \cup \mathit{var}(P)$
- $\mathit{var}(\ll P \gg) = \mathit{var}(P)$
- $\mathit{in}(P_1/\!/P_2) = \mathit{in}(P_1) \cup \mathit{in}(P_2)$
- $\mathit{out}(P_1/\!/P_2) = \mathit{out}(P_1) \cup \mathit{out}(P_2)$
- $\mathit{in}(\textbf{prio}\ e\ (P)) = \mathit{in}(P)$
- $\mathit{out}(\textbf{prio}\ e\ (P)) = \mathit{out}(P)$
- $\mathit{in}(\ll P \gg) = \mathit{in}(P) - \mathit{out}(P)$
- $\mathit{out}(\ll P \gg) = \mathit{out}(P) - \mathit{in}(P)$

Let $\mathit{io}(P)$ denote the internal channels of process P, i.e, $\mathit{io}(P) = \mathit{in}(P) \cap \mathit{out}(P)$.

6.1.1 Syntactic restrictions

To the syntactic restrictions of the previous chapter we add the following restrictions on interleaving composition to make sure that interleaved processes do not share variables and that channels are point-to-point:

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

To guarantee that priorities are integers, we add the following:

- For the priority assignment **prio** e (P) we require that e evaluates to a positive natural number.

6.1.2 Basic timing assumptions

We assume that atomic statements are executed as a single block, that is, without preemption, and that such a block has a fixed constant computation time. In the case of multiprogramming the execution of each statement starts with requesting the processor. Then, the execution time of atomic statements, except for communication statements, consists of a variable period of waiting for the resource and a fixed constant period of processor occupation.

We assume special purpose hardware which manages the communications autonomously. We further assume minimal waiting with respect to communication: no process is blocked in the execution of $c?x$ while another is blocked in the execution of $c!e$. By assumption, communication takes no time. The execution time of a (synchronous) communication statement consists of a variable period of waiting for the resource and a fixed constant period of processor occupation both before and after the actual communication, and the time spent waiting for a partner, that is, the period the process is blocked in the communication.

Besides when starting, a process is executable if it has been preempted in favour of a higher priority task, or if the blocking communication has occurred. Based on the priorities, the scheduler grants the processor to some executable process. The priority of a process is a function of the initial priority and the time spent waiting for the resource. More precisely, a function $\Pi : TIME \rightarrow \mathbb{N}$ indicates how the priority increases while being queued. In Section 5.1.2 we concluded that the c communication precedes the d communication in any execution of the process $[\ c?x \rightarrow d?y \ [\!]\ d?y \rightarrow c?x\ ] \parallel (c!0 \parallel (z := 1\ ;\ d!z))$. Now consider $\ll [\ c?x \rightarrow d?y \ [\!]\ d?y \rightarrow c?x\ ] \gg \ \parallel\ \ll c!0 /\!/ (z := 1\ ;\ d!z) \gg$. It is very well possible that the execution of this network starts by executing the assignment $z := 1$. During this execution the processor request of the process $c!0$ remains pending. Depending on Π, the priority of the process $c!0$ at the termination of the assignment may or may not exceed 0, the priority of the process $d!z$ at that time. In the latter case it is possible that execution continues with the process $d!z$. So, in the case of resource sharing, the d communication might precede the c communication.

For simplicity, we assume that there is no overhead for compound statements and for the scheduler. We assume that execution of each assignment statement takes a constant K_a time units, and assume given a constant K_α denoting the overhead preceding a communication, and a constant K_ω denoting the overhead following a communication.

6.2 Model of computation

In the case of multiprogramming, we have to establish when the resource is taken. We use a triple of the form (τ_1, π, τ_2) to express that a process occupies the processor from τ_1 to τ_2 and that it's priority at time τ_1 is π. In such an interval, which we refer to as a *block*, the process is never preempted. The processor occupation throughout an execution is given by a set $\mathfrak{O}$ of such triples.

Definition 6.1 (Timed occupation history) Let OCC be the set of *timed occupation histories*.

$$OCC = \{\ \mathfrak{O} \subset TIME \times \mathbb{N} \times TIME \mid \forall(\tau_1, \pi, \tau_2) \in \mathfrak{O} \cdot \tau_1 < \tau_2\ \}.$$

$\diamond$

If a process does not have the resource, it is requesting it, unless it is blocked in a communication. However, if processes are merged then so are their respective refusal sets. In order to keep track of which requests are pending at some point in time we represent the request history separately. To denote that at time τ_1 a request is issued with priority π and that the request is granted at τ_2 we use a triple of the form (τ_1, π, τ_2). The history of requests throughout an execution is given by a set $\mathfrak{Q}$ of such triples.

Definition 6.2 (Timed request history) Let REQ be the set of *timed request histories*.

$$REQ \ = \ \{\ \mathfrak{Q} \subset TIME \times \mathbb{N} \times TIME \mid \forall(\tau_1, \pi, \tau_2) \in \mathfrak{Q} \cdot \tau_1 \leq \tau_2\ \}.$$

$\diamond$

Henceforth, we will refer to a 4-tuple $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q})$, that is, a timed trace decorated with a timed refusal set, a timed occupation history, and a timed request history, as *timed observation*.

Definition 6.3 (Projection on occupation histories) Given the occupation history $\mathfrak{O}$ we define the *projection* of $\mathfrak{O}$ onto interval $[\tau_1, \tau_2]$, denoted by $\mathfrak{O}\uparrow[\tau_1, \tau_2]$ as follows:

$$\mathfrak{O}\uparrow[\tau_1, \tau_2] = \{ (\tau_3, \pi, \tau_4) \in \mathfrak{O} \mid \tau_1 \leq \tau_3 \wedge \tau_4 \leq \tau_2 \}.$$

$\diamond$

Definition 6.4 (Time shift on occupation histories) For an occupation history $\mathfrak{O}$ the *time shift* operation $\curvearrowleft$ is defined as follows:

$$\mathfrak{O}\curvearrowleft\tau = \{ (\tau_1 - \tau, \pi, \tau_2 - \tau) \mid (\tau_1, \pi, \tau_2) \in \mathfrak{O} \wedge \tau_1 \geq \tau \}.$$

$\diamond$

These operations are defined likewise for request histories.

6.3 Denotational semantics

In this section we define a — once more — denotational semantics, in terms of the model of Section 6.2, for the programming language of Section 6.1. For a multiprocess P we want to observe:

- the initial state of P including the starting time of the execution,
- the sequence of communications on P's channels performed by the special purpose hardware used by P and the times at which communications occur,
- the times at which the special purpose hardware used by P refused to communicate on P's channels and the names of those channels,
- P's occupation of the resource,
- when P is requesting the processor, and,
- for a terminating computation of P, the final state of P including the termination time of the execution.

Consider the set $STATE_\perp$ of states defined in Section 5.3. In this chapter, the semantic function $\mathcal{M}$ assigns to a multiprocess P a set of triples $(\sigma_0, (\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}), \sigma)$ with $\sigma_0 \in STATE$, $\theta \in TRACE$, $\mathfrak{R} \in REF$, $\mathfrak{O} \in OCC$, $\mathfrak{Q} \in REQ$ and $\sigma \in STATE_\perp$. Informally, a triple $(\sigma_0, (\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}), \sigma) \in \mathcal{M}[\![P]\!]$ has the following meaning:

- if $\sigma \neq \perp$ then it represents a terminating computation of P which performs the communications as expressed by θ, refuses those in $\mathfrak{R}$, occupies the processor as described in $\mathfrak{O}$, requests the resource as reflected by $\mathfrak{Q}$ and terminates in state σ, and
- if $\sigma = \perp$ then it represents a non-terminating computation of P which performs the communications given by θ, refuses those in $\mathfrak{R}$, occupies the processor as described in $\mathfrak{O}$, and requests the resource as reflected by $\mathfrak{Q}$.

For a multiprocess P, the semantic function $\mathcal{M}$ is inductively defined as follows. Observe that the execution of any statement starts by requesting the processor, and during this period communications along its channels are refused.

- Execution of the assignment $x := e$ terminates K_a time units after being granted the processor (time τ in Figure 6.1), all the while refusing no communication. In its final state x has the value e had in its initial state.

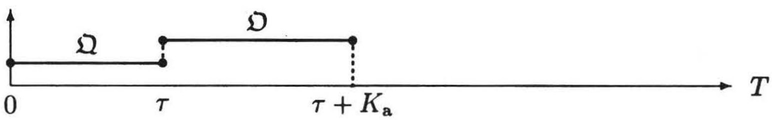

Figure 6.1: Request and occupation for an assignment

$$\mathcal{M}[\![x := e]\!] =$$
$$\{\ (\sigma_0, (\langle\rangle, \emptyset, \{(\tau, \Pi(\tau), \tau + K_a)\}, \{(0,0,\tau)\}), \left(\sigma_0 : \begin{cases} x \mapsto \mathcal{E}[\![e]\!](\sigma_0) \\ T \mapsto \tau + K_a \end{cases}\right))$$
$$\mid \mathcal{E}[\![T]\!](\sigma_0) = 0 \ \wedge\ \tau \geq 0\ \}.$$

- In the execution of the synchronous output statement $c!e$ there comes, K_α time units after the processor has been granted (time τ_1 in Figure 6.2), a waiting period for a communication partner to become available. At the start of this period the processor is released and as soon as the communication occurs (time τ_2 in Figure 6.2) the processor is requested a second time. Execution of the output statement $c!e$ terminates K_ω time units after the processor has been granted this second time (time τ_3 in Figure 6.2). In case there is no communication partner the execution never terminates (and the processor is not requested again).

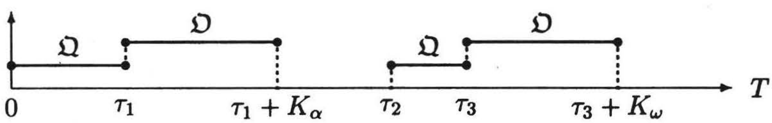

Figure 6.2: Request and occupation for a synchronous output

$$\mathcal{M}[\![c!e]\!] =$$
$$\{\ (\sigma_0, (\langle\rangle, \{c\} \times [0, \tau_1 + K_\alpha), \{(\tau_1, \Pi(\tau_1), \tau_1 + K_\alpha)\}, \{(0,0,\tau_1)\}), \perp)$$
$$\mid \mathcal{E}[\![T]\!](\sigma_0) = 0 \ \wedge\ \tau_1 \geq 0\ \}$$
$$\cup\ \{\ (\sigma_0, (\langle(\tau_2, c, \mathcal{E}[\![e]\!](\sigma_0))\rangle, \mathfrak{R}, \mathfrak{D}, \{(0,0,\tau_1), (\tau_2, 0, \tau_3)\}), \sigma)$$
$$\mid\ \ \mathcal{E}[\![T]\!](\sigma_0) = 0 \ \wedge\ \tau_1 \geq 0 \ \wedge\ \tau_3 \geq \tau_2 \geq \tau_1 + K_\alpha$$
$$\wedge\ \mathfrak{R} = \{c\} \times [0, \tau_1 + K_\alpha) \cup \{c\} \times (\tau_2, \infty)$$
$$\wedge\ \mathfrak{D} = \{\ (\tau_1, \Pi(\tau_1), \tau_1 + K_\alpha), (\tau_3, \Pi(\tau_3 - \tau_2), \tau_3 + K_\omega))\ \}$$
$$\wedge\ \sigma = (\sigma_0 : T \mapsto \tau_3 + K_\omega)\ \}.$$

- Execution of the input statement $c?x$ either never terminates (in case of no communication partner) or terminates $K_\omega + K_{\mathrm{a}}$ time units after the processor has been obtained th esecond time (time τ_3 in Figure 6.3). The received value is assigned to x.

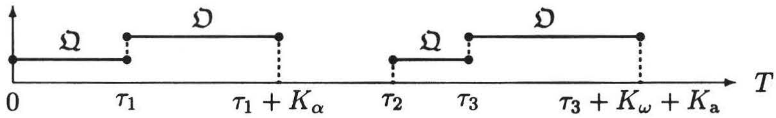

Figure 6.3: Request and occupation for a synchronous input

$$
\begin{array}{l}
\mathcal{M}[\![c?x]\!] = \\
\quad \{\ (\sigma_0, (\langle\rangle, \{c\} \times [0, \tau_1 + K_\alpha), \{(\tau_1, \Pi(\tau_1), \tau_1 + K_\alpha)\}, \{(0,0,\tau_1)\}), \bot) \\
\qquad \mid\ \ \mathcal{E}[\![T]\!](\sigma_0) = 0 \\
\qquad \wedge\ \tau_1 \geq 0\ \} \\
\cup\ \{\ (\sigma_0, (\langle(\tau_2, c, \mu)\rangle, \mathfrak{R}, \mathfrak{O}, \{(0,0,\tau_1), (\tau_2, 0, \tau_3)\}), \sigma) \\
\qquad \mid\ \ \mathcal{E}[\![T]\!](\sigma_0) = 0 \\
\qquad \wedge\ \tau_1 \geq 0 \\
\qquad \wedge\ \tau_3 \geq \tau_2 \geq \tau_1 + K_\alpha \\
\qquad \wedge\ \mu \in \mathit{VAL} \\
\qquad \wedge\ \mathfrak{R} = \{c\} \times [0, \tau_1 + K_\alpha) \cup \{c\} \times (\tau_2, \infty) \\
\qquad \wedge\ \mathfrak{O} = \{\ (\tau_1, \Pi(\tau_1), \tau_1 + K_\alpha), (\tau_3, \Pi(\tau_3 - \tau_2), \tau_3 + K_\omega))\ \} \\
\qquad \wedge\ \sigma = \left(\sigma_0 : \begin{cases} x \mapsto \mu \\ T \mapsto \tau_3 + K_\omega + K_{\mathrm{a}} \end{cases}\right)\ \}.
\end{array}
$$

- 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.

$$
\begin{array}{l}
\mathcal{M}[\![P_1\ ;\ P_2]\!] = \\
\quad \{\ (\ \sigma_0, (\theta, \widehat{\mathfrak{R}}, \mathfrak{O}, \mathfrak{Q}), \bot\ ) \\
\qquad \mid \text{there exists an } \mathfrak{R} \text{ such that } (\ \sigma_0, (\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}), \bot\ ) \in \mathcal{M}[\![P_1]\!] \\
\qquad\ \ \text{and } \widehat{\mathfrak{R}} = \mathfrak{R}\ \cup\ (\mathit{chan}(P_2) - \mathit{chan}(P_1)) \times [0, \infty)\ \} \\
\cup\ \{\ (\ \sigma_0, (\theta_1{}^\wedge\theta_2, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}), \sigma\ ) \\
\qquad \mid \text{there exist an } \mathfrak{R}_1, \text{ an } \mathfrak{R}_2, \text{ a } \sigma_1 \neq \bot \text{ and a } \tau > 0 \text{ such that} \\
\qquad\ \ \mathcal{E}[\![T]\!](\sigma_1) = \tau, \\
\qquad\ \ \mathfrak{R}_2 \uparrow [0, \tau) = \emptyset, \\
\qquad\ \ (\sigma_0\ ,\ (\theta_1\ ,\ \mathfrak{R}_1\ ,\ \mathfrak{O} \uparrow [0, \tau]\ ,\ \mathfrak{Q} \uparrow [0, \tau])\ ,\ \sigma_1) \in \mathcal{M}[\![P_1]\!], \\
\qquad\ \ (\ (\sigma_1 : T \mapsto 0)\ ,\ (\theta_2, \mathfrak{R}_2, \mathfrak{O}, \mathfrak{Q}) \curvearrowleft \tau\ ,\ (\sigma : T \mapsto T - \tau)\ ) \in \mathcal{M}[\![P_2]\!], \\
\qquad\ \ \text{and } \mathfrak{R} = \ \ \mathfrak{R}_1 \uparrow [0, \tau)\ \cup\ (\mathit{chan}(P_2) - \mathit{chan}(P_1)) \times [0, \tau) \\
\qquad\qquad\qquad \cup\ \mathfrak{R}_2\ \cup\ (\mathit{chan}(P_1) - \mathit{chan}(P_2)) \times [\tau, \infty)\ \},
\end{array}
$$

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

- If no guard is open, the Boolean guarded command $[\ \Box_{i=1}^{n}\ b_i \rightarrow P_i\ ]$ terminates after evaluating the guards (K_g time units after being granted the processor). Otherwise, the process corresponding to one of its open guards (non-deterministically chosen) is executed, possibly after preemption in favour of a higher priority task. While evaluating the guards, communications on $chan([\ \Box_{i=1}^{n}\ b_i \rightarrow P_i\ ])$ are refused.

$$
\begin{aligned}
&\mathcal{M}[\![\ [\ \Box_{i=1}^{n}\ b_i \rightarrow P_i\ ]\ ]\!] = \\
&\quad \{\ (\ \sigma_0\ ,\ (\langle\rangle, \mathfrak{R}, \{(\tau, \Pi(\tau), \tau + K_g)\}, \{(0,0,\tau)\})\ ,\ (\sigma_0 : T \mapsto \tau + K_g)\ ) \\
&\qquad \mid \mathcal{E}[\![T]\!](\sigma_0) = 0, \\
&\qquad\ \ \neg\mathcal{B}[\![b_1 \vee \ldots \vee b_n]\!](\sigma_0), \text{ and} \\
&\qquad\ \ \mathfrak{R} = \cup_{i=1}^{n} chan(P_i) \times [0, \infty)\ \} \\
&\cup \{\ (\ \sigma_0, (\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}), \sigma\ ) \\
&\qquad \mid \mathcal{E}[\![T]\!](\sigma_0) = 0, \\
&\qquad\ \ \mathfrak{O}\uparrow[0, \tau + K_g] = \{(\tau, \Pi(\tau), \tau + K_g)\}, \\
&\qquad\ \ \mathfrak{Q}\uparrow[0, \tau + K_g] = \{(0,0,\tau)\} \\
&\qquad\ \ \text{and there exist an } \widehat{\mathfrak{R}} \text{ and a } k \in \{1, \ldots, n\} \text{ such that} \\
&\qquad\ \ \widehat{\mathfrak{R}}\uparrow[0, \tau + K_g) = \emptyset, \\
&\qquad\ \ \mathcal{B}[\![b_k]\!](\sigma_0),\ (\ \sigma_0\ , \\
&\qquad\qquad\qquad\qquad (\theta, \widehat{\mathfrak{R}}, \mathfrak{O}, \mathfrak{Q}) \curvearrowleft \tau + K_g\ , \\
&\qquad\qquad\qquad\qquad (\sigma : T \mapsto T - \tau - K_g)\ ) \in \mathcal{M}[\![P_k]\!], \text{ and} \\
&\qquad\ \ \mathfrak{R} = \quad \cup_{i=1}^{n} chan(P_i) \times [0, \tau + K_g) \\
&\qquad\qquad\quad \cup\ \widehat{\mathfrak{R}} \\
&\qquad\qquad\quad \cup\ (\cup_{i=1}^{n} chan(P_i) - chan(P_k)) \times [\tau + K_g, \infty)\ \}.
\end{aligned}
$$

- K_α time units after the processor is granted, a waiting period commences for one of the c_i communications that appear in the communication guarded command $[\ \|_{i=1}^{n}\ c_i?x_i \rightarrow P_i\ \|$ **delay** $d \rightarrow P_0\ ]$ to occur. At the start of this period, which is at most d time units long, the processor is released. The first communication that occurs resolves the choice of which process to execute. Upon occurrence of this communication the resource is requested and $K_\omega + K_\mathbf{a}$ time units after it is granted execution continues with the appropriate P_i. If no communication occurs before d time units $(0 \leq d \leq \infty)$ have elapsed, the processor is requested and when granted the process P_0 is executed.

$$
\begin{aligned}
&\mathcal{M}[\![\ [\ \|_{i=1}^{n}\ c_i?x_i \rightarrow P_i\ \|\ \mathbf{delay}\ d \rightarrow P_0\ ]\ ]\!] = \\
&\cup_{i=1}^{n}\ \{\ (\ \sigma_0, (\langle(\tau_2, c_i, \mu)\rangle^\wedge\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}), \sigma\ ) \\
&\qquad \mid \mathcal{E}[\![T]\!](\sigma_0) = 0, \\
&\qquad\ \mu \in VAL, \\
&\qquad\ \text{and there exist a } \tau_1 \geq 0, \text{ a } \tau_2, \text{ a } \tau_3 \text{ and an } \widehat{\mathfrak{R}} \text{ with} \\
&\qquad\ \tau_1 + K_\alpha \leq \tau_2 < \tau_1 + K_\alpha + d,\ \tau_2 \leq \tau_3, \\
&\qquad\ \widehat{\mathfrak{R}}\uparrow[0, \tau_3 + K_\omega + K_\mathbf{a}) = \emptyset, \\
&\qquad\ \mathfrak{D}\uparrow[0, \tau_3 + K_\omega + K_\mathbf{a}] \\
&\qquad\ = \{(\tau_1, \Pi(\tau_1), \tau_1 + K_\alpha), (\tau_3, \Pi(\tau_3 - \tau_2), \tau_3 + K_\omega + K_\mathbf{a})\}, \\
&\qquad\ \mathfrak{Q}\uparrow[0, \tau_3 + K_\omega + K_\mathbf{a}] = \{(0, 0, \tau_1), (\tau_2, 0, \tau_3)\}, \\
&\qquad\ (\ (\sigma_0 : x_i \mapsto \mu)\ , \\
&\qquad\quad (\theta, \widehat{\mathfrak{R}}, \mathfrak{D}, \mathfrak{Q}) \curvearrowleft (\tau_3 + K_\omega + K_\mathbf{a})\ , \\
&\qquad\quad (\sigma : T \mapsto T - \tau_3 - K_\omega - K_\mathbf{a})\ ) \in \mathcal{M}[\![P_i]\!], \text{ and} \\
&\qquad\ \mathfrak{R} = \quad (\cup_{j=0}^{n} chan(P_j)\ \cup\ \cup_{j=1}^{n}\{c_j\}) \times [0, \tau_3 + K_\omega + K_\mathbf{a}) \\
&\qquad\qquad\qquad - \{(c_i, \tau_2)\} - \cup_{j=1}^{n}\{c_j\} \times [\tau_1 + K_\alpha, \tau_2) \\
&\qquad\qquad\quad \cup\ \widehat{\mathfrak{R}} \\
&\qquad\qquad\quad \cup\ ((\cup_{j=0}^{n} chan(P_j)\ \cup\ \cup_{j=1}^{n}\{c_j\}) - chan(P_i)) \\
&\qquad\qquad\qquad \times \\
&\qquad\qquad\qquad [\tau_3 + K_\omega + K_\mathbf{a}, \infty)\ \} \\
&\cup \quad \{\ (\ \sigma_0, (\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}), \sigma\ ) \\
&\qquad \mid \mathcal{E}[\![T]\!](\sigma_0) = 0, \text{ and there is a } \tau_1 \geq 0 \text{ and a } \widehat{\mathfrak{R}} \text{ such that} \\
&\qquad\ \widehat{\mathfrak{R}}\uparrow[0, \tau_1 + K_\alpha + d) = \emptyset, \\
&\qquad\ \mathfrak{D}\uparrow[0, \tau_1 + K_\alpha + d] = \{(\tau_1, \Pi(\tau_1), \tau_1 + K_\alpha)\}, \\
&\qquad\ \mathfrak{Q}\uparrow[0, \tau_1 + K_\alpha + d] = \{(0, 0, \tau_1)\}, \\
&\qquad\ (\ \sigma_0\ , \\
&\qquad\quad (\theta, \widehat{\mathfrak{R}}, \mathfrak{D}, \mathfrak{Q}) \curvearrowleft (\tau_1 + K_\alpha + d)\ , \\
&\qquad\quad (\sigma : T \mapsto T - \tau_1 - K_\alpha - d)\ ) \in \mathcal{M}[\![P_0]\!], \text{ and} \\
&\qquad\ \mathfrak{R} = \quad \cup_{j=0}^{n} chan(P_j) \times [0, \tau_1 + K_\alpha + d) \\
&\qquad\qquad\quad \cup\ \cup_{j=1}^{n}\{c_j\} \times [0, \tau_1 + K_\alpha) \\
&\qquad\qquad\quad \cup\ \widehat{\mathfrak{R}} \\
&\qquad\qquad\quad \cup\ (\cup_{j=1}^{n}(chan(P_j)\ \cup\ \{c_j\}) - chan(P_0)) \\
&\qquad\qquad\qquad \times \\
&\qquad\qquad\qquad [\tau_1 + K_\alpha + d, \infty)\ \}.
\end{aligned}
$$

- After each execution of the body G the execution of $*G$ may be preempted.

$$
\begin{array}{l}
\mathcal{M}[\![* G]\!] = \\
\{ \ (\ \sigma_0, (\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}), \sigma \ ) \\
\quad | \ \mathcal{E}[\![T]\!](\sigma_0) = 0 \text{ and there exists a } k \in \mathbb{N} \cup \{\infty\}, \text{ and for every } i, \\
\quad\quad 0 \leq i < k, \text{ there is a triple } (\sigma_i \ , \ (\theta_{i+1}, \mathfrak{R}_{i+1}, \mathfrak{O}_{i+1}, \mathfrak{Q}_{i+1}) \ , \ \sigma_{i+1}) \\
\quad\quad \text{such that } \sigma_i \neq \bot, \ \mathcal{B}[\![b_G]\!](\sigma_i), \\
\quad\quad \mathfrak{R}_{i+1} \uparrow [0, \mathcal{E}[\![T]\!](\sigma_i)) = chan(G) \times [0, \mathcal{E}[\![T]\!](\sigma_i)), \\
\quad\quad \mathfrak{O}_{i+1} \uparrow [0, \mathcal{E}[\![T]\!](\sigma_i)] = \emptyset, \\
\quad\quad \mathfrak{Q}_{i+1} \uparrow [0, \mathcal{E}[\![T]\!](\sigma_i)] = \emptyset, \\
\quad\quad (\ (\sigma_i : T \mapsto 0) \ , \\
\quad\quad\quad (\theta_{i+1}, \mathfrak{R}_{i+1}, \mathfrak{O}_{i+1}, \mathfrak{Q}_{i+1}) \curvearrowleft \mathcal{E}[\![T]\!](\sigma_i) \ , \\
\quad\quad\quad (\sigma_{i+1} : T \mapsto T - \mathcal{E}[\![T]\!](\sigma_i)) \ ) \in \mathcal{M}[\![G]\!], \text{ and} \\
\quad\quad \text{if } k = \infty \text{ then} \\
\quad\quad\quad \text{for all } j, \ 1 \leq j < k, \ \theta_1{}^\wedge \ldots {}^\wedge\theta_j \ \preceq \ \theta, \ \cap_{l=1}^{j} \ \mathfrak{R}_l \ \supseteq \ \mathfrak{R}, \\
\quad\quad\quad \cup_{l=1}^{j} \ \mathfrak{O}_l \ \subseteq \mathfrak{O}, \ \cup_{l=1}^{j} \ \mathfrak{Q}_l \ \subseteq \mathfrak{Q}, \text{ and } \sigma = \bot, \\
\quad\quad \text{else if } k < \infty \text{ then} \\
\quad\quad\quad \theta \ = \ \theta_1{}^\wedge \ldots {}^\wedge\theta_k, \ \mathfrak{R} \ = \ \cap_{l=1}^{k} \ \mathfrak{R}_l, \\
\quad\quad\quad \mathfrak{O} = \cup_{l=1}^{j} \ \mathfrak{O}_l, \ \mathfrak{Q} = \cup_{l=1}^{j} \ \mathfrak{Q}_l, \ \sigma = \sigma_k, \\
\quad\quad\quad \text{and if } \sigma_k \neq \bot \text{ then } \mathcal{B}[\![\neg b_G]\!](\sigma_k) \ \}.
\end{array}
$$

- The executions of $P_1 /\!/ P_2$ are obtained by interleaving executions of P_1 and P_2 that do not conflict in their occupation of the shared processor and that respect the priorities.

Definition 6.5 (Absence of conflict) The requirement that the processor occupation $\mathfrak{O}_1$ of process P_1 and the processor occupation $\mathfrak{O}_2$ of process P_2 do not conflict can be formalized as follows:

$$
NoConflict(\mathfrak{O}_1, \mathfrak{O}_2) \ \equiv \ \forall (\tau_1, \pi_1, \tau_2) \in \mathfrak{O}_1, (\tau_3, \pi_2, \tau_4) \in \mathfrak{O}_2 \cdot \quad \begin{array}{l} \tau_4 < \tau_1 \\ \vee \ \tau_2 < \tau_3. \end{array}
$$

$\diamond$

Definition 6.6 (Respect) A particular processor occupation $\mathfrak{O}$ is said to respect the priorities of a history of pending requests $\mathfrak{Q}$, notation $Respect(\mathfrak{O}, \mathfrak{Q})$, if, and only if, the priority with which each block starts equals the maximal priority at that point in time.

$$
Respect(\mathfrak{O}, \mathfrak{Q}) \ \equiv \ \forall (\tau_1, \pi, \tau_2) \in \mathfrak{O} \cdot \pi \ = \ \mathbf{max} \, \pi_1 \cdot \exists (\tau_3, \pi_2, \tau_4) \in \mathfrak{Q} \cdot \begin{array}{l} \tau_3 \leq \tau_1 < \tau_4 \\ \wedge \, \pi_1 = \pi_2 + \Pi(\tau_1 - \tau_3). \end{array}
$$

$\diamond$

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

- The effect of assigning a priority π to a process is that the priorities with which the resource is requested are increased by π. Since we define the semantics of P regardless of its environment, the priorities with which the resource is occupied must, consequently, also be increased by π.

Definition 6.7 (Increase of priority) For a $\pi \in \mathbb{N}$ and a processor occupation $\mathfrak{O} \in OCC$,

$$\mathit{IncPr}(\pi, \mathfrak{O}) \quad = \quad \{\ (\tau_1, \pi_1 + \pi, \tau_2) \mid (\tau_1, \pi_1, \tau_2) \in \mathfrak{O}\ \}.$$

The operation $\mathit{IncPr}(\pi, \mathfrak{Q})$ is defined likewise. ◇

$$
\begin{array}{rcl}
\mathcal{M}[\![\mathbf{prio}\ e\ (P)]\!] & = & \{\ (\sigma_0, (\theta, \mathfrak{R}, \mathit{IncPr}(\mathcal{E}[\![e]\!]\sigma_0, \mathfrak{O}), \mathit{IncPr}(\mathcal{E}[\![e]\!]\sigma_0, \mathfrak{Q})), \sigma) \\
& & \mid (\sigma_0, (\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}), \sigma) \in \mathcal{M}[\![P]\!]\ \}.
\end{array}
$$

Notice that this definition incorporates, besides finite variability, preemption, since the processor has to be requested for each atomic statement.

Having thus defined the meaning of a multiprocess we can abstract from its internal state.

Definition 6.8 (Timed observations) The *timed observations* of the multiprocess P, notation $\mathcal{O}[\![P]\!]$, follow from:

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

◇

The set $\mathcal{O}[\![P]\!]$ represents the normal behaviour of process P. In Section 6.5 we determine the set $\mathcal{O}[\![P \wr \chi]\!]$ representing the acceptable behaviour of P under the failure hypothesis χ. Note that if $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}) \in \mathcal{O}[\![P]\!]$ then $\theta{\uparrow}\mathit{chan}(P) = \theta$ and $\mathfrak{R}{\uparrow}\mathit{chan}(P) = \mathfrak{R}$.

The timed observations of the network $\ll P \gg$ are defined as follows (notice how we abstract from the information concerning processor occupation and pending requests):

- Since the processor closure $\ll P \gg$ denotes that no process outside P executes on the processor, the observations of $\ll P \gg$ are the observations of P corresponding to the case where the processor is idle if, and only if, there are no tasks to execute. Recall that the processor closure operator hides the internal communications. Hence, an observation of $\ll P \gg$ is characterized by the fact that $io(P)$ communications occur as soon as possible.

Definition 6.9 (No strike)

$$\begin{aligned} NoStrike(\mathfrak{O}, \mathfrak{Q}) \;\equiv\; & \forall (\tau_1, \pi, \tau_2) \in \mathfrak{Q} \cdot \\ & \forall \tau_1 \leq \tau \leq \tau_2 \cdot \exists (\tau_3, \widehat{\pi}, \tau_4) \in \mathfrak{O} \cdot \tau_3 \leq \tau \leq \tau_4. \end{aligned}$$

$\diamond$

$$\begin{aligned} & \mathcal{O}[\![\ll P \gg]\!] = \\ & \{\ (\theta \setminus io(P), \mathfrak{R} \setminus io(P)) \\ & \quad \mid (\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}) \in \mathcal{O}[\![P]\!] \ \wedge\ ASAP(\mathfrak{R}, io(P)) \ \wedge\ NoStrike(\mathfrak{O}, \mathfrak{Q})\ \}. \end{aligned}$$

The timed observations of $NP_1 \parallel NP_2$ and $NP \setminus cset$ follow from the definitions given in Chapter 5.

6.4 Assertion language and correctness formulae

At the network level the occupation and request histories are irrelevant. Therefore we distinguish in this chapter between specifications of multiprocesses, typically represented by φ, and specifications of networks, with typical representative ϕ, already discussed in Chapter 5.

Besides the expressions introduced in Section 5.4 we have expressions like (τ_1, π, τ_2), with $\tau_1, \tau_2 \in TIME$ and $\pi \in \mathbb{N}$, to create occupation and request history expressions. To refer to the timed observation of a multiprocess we use the special variables h, R, O, and Q to denote the trace, the refusal set, the occupation history and the request history of the process, respectively.

For an assertion φ we also write $\varphi(h, R, O, Q)$ to indicate that φ has free variables h, R, O, and Q. We use $\varphi(texp, rfxp, ohxp, rhxp)$ to denote the assertion which is obtained from φ by replacing h by trace expression $texp$, R by refusal expression $rfxp$, O by occupation history expression $ohxp$, and Q by request history expression $rhxp$.

Let $IVAR$, with typical representative i, denote the set of logical value variables ranging over $\mathbb{N}$, let $TIVAR$, 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 $TRACE$, let $RVAR$, with typical element N, be the set of logical refusal variables ranging over REF, let $PRVAR$, with characteristic element p, be the set of logical priority variables ranging over $\mathbb{N}$, let $OVAR$, with typical element K, be the set of logical occupation history variables ranging over OCC, and let $QVAR$, with typical element L, be the set of logical request history variables ranging over REQ. Then, we can write specifications such as $\mathbf{prio}\ 3\ (c!2)\ \mathbf{sat}\ \exists t \cdot (0, 3, t) \in Q\ \wedge\ (t, 3 + \Pi(t), t + K_\alpha) \in O$.

Table 6.2 summarizes the assertion language, with $\tau \in TIME$, $t \in IVAR$, $c \in CHAN$, $\mu \in VAL$, $v \in VVAR$, $s \in TVAR$, $N \in RVAR$, $\pi \in \mathbb{N}$, $p \in PRVAR$, $K \in OVAR$, $L \in QVAR$, and $cset \subseteq CHAN$. As a reminder, a specification ϕ of a network of multiprocesses is a sentence of the language defined in Table 5.2 and has the form $\phi(h, R)$.

We use the primitive predicates defined in Definitions 5.21 and 5.23. Observe that the ability to refer to the willingness of the environment to communicate allows us to specify a deadline not only relative to the point in time at which a communication occurs, but even in relation to the instant at which the environment started to offer it.

Example 6.10 (Calculator) Consider the process C that accepts a value via *in*, applies a function f to it and produces the result via *out*. To specify that an input is always taken within K_d time units after it was first offered we write:

$$C\ \mathbf{sat}\ \forall t, \hat{t} \cdot in\ \mathbf{enabled\ for}\ \hat{t}\ \rightarrow\ \hat{t} \leq K_d.$$

△

When verifying a time-critical system it is often crucial to be able to express a lower bound on the frequency at which the environment will be offering input.

Example 6.11 (Calculator) To specify that inputs are offered at least K_f time units apart we write:

$$C\ \mathbf{sat}\ \forall t, \hat{t} \cdot in\ \mathbf{enabled\ precisely\ for}\ \hat{t}\ \rightarrow\ \mathbf{after}\ \hat{t} : in\ \mathbf{refused\ for}\ K_f.$$

△

We define a third category of primitive predicates.

Definition 6.12 (Primitive predicates III) For time expression *texp*,

- **occupied at** $texp\ \equiv\ \exists(t_1, p, t_2) \in O \cdot t_1 \leq texp \leq t_2$;
- **pending at** $texp\ \equiv\ \exists(t_1, p, t_2) \in Q \cdot t_1 \leq texp \leq t_2$;
- **busy upto** $texp$
  $\equiv\ \forall \hat{t} \cdot t < \hat{t} < texp\ \rightarrow\ \mathbf{occupied\ at}\ \hat{t}\ \vee\ \mathbf{pending\ at}\ \hat{t}.$ ◇

Table 6.2: Syntax of the assertion language

Integer expression	$iexp$	$::=$	$0 \mid 1 \mid i \mid iexp_1 + iexp_2 \mid iexp_1 \times iexp_2 \mid len(texp)$
Time expression	$tixp$	$::=$	$\tau \mid t \mid ts(rexp) \mid tixp_1 + tixp_2$
Channel expression	$cexp$	$::=$	$c \mid ch(rexp)$
Value expression	$vexp$	$::=$	$\mu \mid v \mid val(rexp) \mid f(vexp_1, \ldots, vexp_n)$
Record expression	$rexp$	$::=$	$(tixp, cexp, vexp) \mid texp(iexp)$
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	$inxp$	$::=$	$[tixp_1, tixp_2) \mid \{tixp\}$
Refusal expression	$rfxp$	$::=$	$N \mid R \mid \emptyset \mid cset \times inxp \mid rfxp_1 \cup rfxp_2$
Priority expression	$prxp$	$::=$	$\pi \mid p \mid \Pi(tixp)$
Block expression	$blxp$	$::=$	$(tixp_1, prxp, tixp_2)$
Occupation expression	$ohxp$	$::=$	$K \mid O \mid \emptyset \mid \{blxp\} \mid ohxp_1 \cup ohxp_2$
Queue expression	$quxp$	$::=$	$(tixp_1, prxp, tixp_2)$
Request expression	$rhxp$	$::=$	$L \mid Q \mid \emptyset \mid \{quxp\} \mid rhxp_1 \cup rhxp_2$
Assertion	φ	$::=$	$iexp_1 = iexp_2 \mid iexp_1 < iexp_2 \mid tixp_1 = tixp_2 \mid tixp_1 < tixp_2 \mid cexp_1 = cexp_2 \mid vexp_1 = vexp_2 \mid vexp_1 < vexp_2 \mid texp_1 = texp_2 \mid rfxp_1 = rfxp_2 \mid prxp_1 = prxp_2 \mid ohxp_1 = ohxp_2 \mid rhxp_1 = rhxp_2 \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \exists i \cdot \varphi \mid \exists t \cdot \varphi \mid \exists v \cdot \varphi \mid \exists s \cdot \varphi \mid \exists N \cdot \varphi \mid \exists p \cdot \varphi \mid \exists K \cdot \varphi \mid \exists L \cdot \varphi$

We are primarily interested in the accumulative processor occupation of the process.

Definition 6.13 (Accumulative processor occupation) Counting from base time t, the *accumulative processor occupation* of the process at time $\widehat{t}$, notation $APO(\widehat{t})$, is defined as follows:

$$APO(\widehat{t}) = \mathbf{sum}\ t_2 - t_1 \cdot t_1 \geq t \wedge t_2 \leq \widehat{t} \wedge \exists p \cdot (t_1, p, t_2) \in O.$$

◇

Definition 6.14 (Abbreviation) The instant, relative to base time t, at which the processor has accumulatively been occupied for K time units, notation $RO(K)$, follows from:

$$RO(K) \;=\; \mathbf{min}\, \widehat{t} \cdot (APO(\widehat{t}) = K).$$

$\diamond$

Example 6.15 (Calculator) After an input it takes $RO(K_{C_1})$ time units of execution before the corresponding output becomes enabled. Once an output has occurred, a next input becomes enabled after $RO(K_{C_2})$ time units of resource occupation. We specify C as follows:

$$\begin{array}{ll}
C\ \mathbf{sat} & \forall i \cdot 1 \le i \le len(h{\uparrow}out) \rightarrow val(h{\uparrow}out(i)) = f(val(h{\uparrow}in(i))) \\
& \wedge\ h = \langle\rangle \rightarrow \mathbf{enable}\ in\ \mathbf{and\ refuse}\ out\ \mathbf{upto}\ \infty \\
& \wedge\ \forall t, v \cdot (t, in, v) \in h \rightarrow \\
& \qquad\quad \mathbf{busy\ upto}\ RO(K_{C_1}) \\
& \qquad \wedge\ \mathbf{refuse}\ \{in, out\}\ \mathbf{upto}\ RO(K_{C_1}) \\
& \qquad \wedge\ \mathbf{after}\ RO(K_{C_1}) : \\
& \qquad\quad \forall \widehat{t} \cdot out\ \mathbf{refused\ precisely\ upto}\ \widehat{t} \\
& \qquad\qquad\quad \rightarrow \mathbf{enable}\ out\ \mathbf{and\ refuse}\ in\ \mathbf{for}\ \widehat{t} \\
& \wedge\ \forall t, v \cdot (t, out, v) \in h \rightarrow \\
& \qquad\quad \mathbf{busy\ upto}\ RO(K_{C_2}) \\
& \qquad \wedge\ \mathbf{refuse}\ \{in, out\}\ \mathbf{upto}\ RO(K_{C_2}) \\
& \qquad \wedge\ \mathbf{after}\ RO(K_{C_2}) : \\
& \qquad\quad \forall \widehat{t} \cdot in\ \mathbf{refused\ precisely\ upto}\ \widehat{t} \\
& \qquad\qquad\quad \rightarrow \mathbf{enable}\ in\ \mathbf{and\ refuse}\ out\ \mathbf{for}\ \widehat{t}.
\end{array}$$

Notice how references to the readiness of the environment to communicate are used to determine, for instance, the time $RO(K_{C_1}) + \widehat{t}$ at which an out communication occurs after an input. $\triangle$

For an assertion φ the set $chan(\varphi)$ of observation channels is defined in Definition G.1. An assertion is interpreted with respect to a 5-tuple $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma)$. Trace θ gives h its value, refusal set $\mathfrak{R}$ gives R its value, occupation history O obtains its value from $\mathfrak{O}$, request history Q does so from $\mathfrak{Q}$ and the environment γ interprets the logical variables. The meaning of assertions is given in Definition G.2. When an assertion φ holds for trace θ, refusal $\mathfrak{R}$, occupation history $\mathfrak{O}$, request history $\mathfrak{Q}$ and an environment γ, notation $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \varphi$, is a straightforward extension of the definition given in Definition E.3.

Example 6.16 (Satisfaction) In Example 6.15 we came across assertion

$$\forall t, v \cdot (t, in, v) \in h \rightarrow \mathbf{refuse}\ \{in, out\}\ \mathbf{upto}\ RO(K_{C_1}),$$

which is an abbreviation of

$$\forall t, v \cdot (t, in, v) \in h \rightarrow \{in, out\} \times [t, t + RO(K_{C_1})) \subseteq R.$$

This assertion holds for 5-tuple $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma)$ 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,

$$(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \widehat{\gamma}) \models (t, in, v) \in h \rightarrow \{in, out\} \times [t, t + RO(K_{C_1})) \subseteq R.$$

Since h and R obtain their value from θ and $\mathfrak{R}$, respectively, this implication holds for those traces θ and refusals $\mathfrak{R}$ for which it is the case that if θ contains a record (τ, in, μ) then $\mathfrak{R}$ contains $\{in, out\} \times [\tau, \tau + \widehat{\tau})$, where $\widehat{\tau}$ is the smallest instant with $K_{C_1} = \textbf{sum } \tau_2 - \tau_1 \cdot \tau_1 \geq \tau \wedge \tau_2 \leq \widehat{\tau} \wedge \exists \pi \cdot (\tau_1, \pi, \tau_2) \in \mathfrak{O}$. $\triangle$

Definition 6.17 (Valid assertion) An assertion φ is *valid*, notation $\models \varphi$, if, and only if, for all θ, $\mathfrak{R}$, $\mathfrak{O}$, $\mathfrak{Q}$ and all γ, $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \varphi$. $\diamond$

Prompted by the observation that a fault is tolerated only if it does not cause abnormalities in any execution and that an on-line scheduler cannot backtrack, correctness formula P **sat** φ expresses that all executions of multiprocess P satisfy φ.

Definition 6.18 (Valid correctness formula) For multiprocess P and assertion φ the correctness formula P **sat** φ is *valid*, notation $\models P$ **sat** φ, if, and only if, for all γ and all $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}) \in \mathcal{O}[\![P]\!]$, $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \varphi$. $\diamond$

The validity of the correctness formula NP **sat** ϕ is as defined in Definition 5.27.

6.5 Incorporating failure hypotheses

To take account of a particular failure hypothesis, the set of observations that characterize a multiprocess must again be expanded. This time, a predicate that is used to formalize a failure hypothesis has free variables h, h_{old}, R, R_{old}, O, O_{old}, Q and Q_{old}.

We extend the assertion language to include the trace expression term h_{old}, refusal expression term R_{old}, occupation history expression term O_{old} and request history expression term Q_{old}. Sentences of this extended language are again called *transformation expressions*, with typical representative ψ. We also write $\psi(h_{old}, h, R_{old}, R, O_{old}, O, Q_{old}, Q)$ to indicate that transformation expression ψ has free variables h_{old}, h, R_{old}, R, O_{old}, O, Q_{old} and Q. Then, $\psi(texp_1, texp_2, rfxp_1, rfxp_2, ohxp_1, ohxp_2, rhxp_1, rhxp_2)$ denotes the expression which is obtained from ψ by substituting $texp_1$ for h_{old}, $texp_2$ for h, $rfxp_1$ for R_{old}, $rfxp_2$ for R, $ohxp_1$ for O_{old}, $ohxp_2$ for O, $rhxp_1$ for Q_{old} and $rhxp_2$ for Q. Notice that at the network level a transformation expression has the form $\psi(h_{old}, h, R_{old}, R)$. A transformation expression is interpreted with respect to a tuple $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \mathfrak{O}_0, \mathfrak{O}, \mathfrak{Q}_0, \mathfrak{Q}, \gamma)$. Trace θ_0 gives h_{old} its value, refusal $\mathfrak{R}_0$, occupation history $\mathfrak{O}$, and request history $\mathfrak{Q}$ does so for R_{old}, O_{old}, and Q_{old}, respectively. In conformity with the foregoing, trace θ gives h its value, refusal set $\mathfrak{R}$ gives R its value, $\mathfrak{O}$ and $\mathfrak{Q}$ do so for O, respectively Q, and the environment γ interprets the logical variables. The meaning of assertions defined in Definition G.2 can easily be adapted for transformation expressions by adding the clauses

- $\mathcal{T}[\![h_{old}]\!](\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma)=\theta_0$,
- $\mathcal{RF}[\![R_{old}]\!](\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma)=\mathfrak{R}_0$,
- $\mathcal{OC}[\![O_{old}]\!](\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma)=\mathfrak{O}_0$ and
- $\mathcal{RQ}[\![Q_{old}]\!](\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma)=\mathfrak{Q}_0$.

We write $(\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma)\models\psi$ to denote that ψ holds for 9-tuple $(\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma)$. Since the terms h_{old}, R_{old}, O_{old}, and Q_{old} do not occur in assertions, the following lemma is trivial.

Lemma 6.19 (Correspondence) For an assertion $\varphi(h,R,O,Q)$ and for all θ_0, $\mathfrak{R}_0$, $\mathfrak{O}_0$, and $\mathfrak{Q}_0$ it is the case that $(\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma)\models\varphi$ if, and only if, $(\theta,\mathfrak{R},\mathfrak{O},\mathfrak{Q},\gamma)\models\varphi$. ○

The following lemma is easy to prove by structural induction.

Lemma 6.20 (Substitution) For the transformation expression ψ,

(a) $(\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma)\models\psi(texp,h,R_{old},R,O_{old},O,Q_{old},Q)$
iff $(\mathcal{T}[\![texp]\!](\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma),\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma)\models\psi$;

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

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

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

(e) $(\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma)\models\psi(h_{old},h,R_{old},R,ohxp,O,Q_{old},Q)$
iff $(\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathcal{OC}[\![ohxp]\!](\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma),\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma)\models\psi$;

(f) $(\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma)\models\psi(h_{old},h,R_{old},R,O_{old},ohxp,Q_{old},Q)$
iff $(\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathcal{OC}[\![ohxp]\!](\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma),\mathfrak{Q}_0,\mathfrak{Q},\gamma)\models\psi$;

(g) $(\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma)\models\psi(h_{old},h,R_{old},R,O_{old},O,rhxp,Q)$
iff $(\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathcal{RQ}[\![rhxp]\!](\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma),\mathfrak{Q},\gamma)\models\psi$;

(h) $(\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma)\models\psi(h_{old},h,R_{old},R,O_{old},O,Q_{old},rhxp)$ iff
$(\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathcal{RQ}[\![rhxp]\!](\theta_0,\theta,\mathfrak{R}_0,\mathfrak{R},\mathfrak{O}_0,\mathfrak{O},\mathfrak{Q}_0,\mathfrak{Q},\gamma),\gamma)\models\psi$. ○

Definition 6.21 (Validity of a transformation expression) A transformation expression ψ is *valid*, notation $\models \psi$, if, and only if, for all θ_0, θ, $\mathfrak{R}_0$, $\mathfrak{R}$ and γ, $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \psi$. $\diamond$

The observation channels that appear in a transformation expression are as defined in Definition G.1 with the extra clauses

- $chan(h_{old}) = CHAN$,
- $chan(R_{old}) = CHAN$,
- $chan(O_{old}) = \emptyset$, and
- $chan(Q_{old}) = \emptyset$.

Definition 6.22 (Failure hypothesis) A *failure hypothesis* χ is a transformation expression which represents a reflexive relation on the normal behaviour, to guarantee that the normal behaviour is part of the acceptable behaviour:

- $\models \chi\,(h_{old}, h_{old}, R_{old}, R_{old}, O_{old}, O_{old}, Q_{old}, Q_{old})$.

Furthermore, a failure hypothesis of failure prone multiprocess *FP* does not impose restrictions on communications along channels not in $chan(FP)$:

- $chan(\chi) \subseteq chan(FP)$. $\diamond$

As already mentioned in the previous chapter a failure hypothesis should be handled with care. Note that for networks of multiprocesses, for which the occupation and request history are irrelevant, a failure hypothesis has the form $\chi(h_{old}, h, R_{old}, R)$.

Example 6.23 (Reset) Consider the process C introduced in Example 6.15. Suppose that, due to a reset, C does not complete its current task but starts processing the next input. Then, where h_{old} recorded an *out* communication, h does not. Such a reset typically coincides with the conclusion of the execution of an atomic statement, that is, it occurs at the end of a block. After a reset occurs, the resource is only needed to prepare for a subsequent *in* communication — a single block with length K_α. Assuming that the environment does not offer input too frequently, the timing of this input does not differ from the one recorded in h_{old}.

$$
\begin{array}{rl}
\mathit{Reset} \equiv & h{\uparrow}\{in, out\} \trianglelefteq h_{old}{\uparrow}\{in, out\} \\
& \wedge\ h{\uparrow}\{in\} \ = \ h_{old}{\uparrow}\{in\} \\
& \wedge\ \forall t \cdot \exists v \cdot (t, out, v) \in h_{old} \ \wedge\ (t, out, v) \in h \\
& \qquad \rightarrow \exists t_1, t_2 \cdot \quad t_1 = \mathbf{max}\ t_3 < t \cdot \exists v \cdot (t_3, in, v) \in h \\
& \qquad\qquad\qquad \wedge\ t_2 = \mathbf{min}\ t_3 > t \cdot \exists v \cdot (t_3, in, v) \in h \\
& \qquad\qquad\qquad \wedge\ O{\uparrow}[t_1, t_2] = O_{old}{\uparrow}[t_1, t_2] \\
& \qquad\qquad\qquad \wedge\ Q{\uparrow}[t_1, t_2] = Q_{old}{\uparrow}[t_1, t_2] \\
& \qquad\qquad\qquad \wedge\ R{\uparrow}[t_1, t_2] = R_{old}{\uparrow}[t_1, t_2] \\
& \wedge\ \forall t \cdot \exists v \cdot (t, out, v) \in h_{old} \ \wedge\ (t, out, v) \notin h \\
& \qquad \rightarrow \exists t_1, t_2, t_3, t_4, t_5, t_6 \cdot \\
& \qquad\quad t_1 = \mathbf{max}\ t_7 < t \cdot \exists v \cdot (t_7, in, v) \in h \\
& \qquad\quad \wedge\ t_2 = \mathbf{min}\ t_7 > t \cdot \exists v \cdot (t_7, in, v) \in h \\
& \qquad\quad \wedge\ t_3 = \mathbf{min}\ t_7 > t_1 \cdot \exists p, t_8 \cdot (t_8, p, t_7) \in O_{old} \\
& \qquad\quad \wedge\ t_4 = \mathbf{max}\ t_7 < t \cdot \exists p, t_8 \cdot (t_7, p, t_8) \in O_{old} \\
& \qquad\quad \wedge\ t_3 \leq t_5 < t_4 \ \wedge\ \exists p, t_7 \cdot (t_7, p, t_5) \in O_{old} \\
& \qquad\quad \wedge\ t_5 \leq t_6 < t_2 - K_\alpha \\
& \qquad\quad \wedge\ O{\uparrow}[t_1, t_2] = O_{old}{\uparrow}[t_1, t_5] \cup \{(t_6, \Pi(t_6 - t_5), t_6 + K_\alpha)\} \\
& \qquad\quad \wedge\ Q{\uparrow}[t_1, t_2] = Q_{old}{\uparrow}[t_1, t_5] \cup \{(t_5, t_6)\} \\
& \qquad\quad \wedge\ R{\uparrow}[t_1, t_2] \\
& \qquad\qquad = R_{old}{\uparrow}[t_1, t_5] \cup [t_5, t_6 + K_\alpha] \times \{in\} \cup [t_5, t_2] \times \{out\}.
\end{array}
$$

In this expression t is the timestamp of the output (as recorded in h_{old}) under discussion; t_1 and t_2 are the timestamps of the inputs preceding, respectively succeeding, that output. In the last of the four conjuncts t_3 is the end of the first block after t_1, and t_4 is the start of the last block before t. The reset occurs at t_5. At t_6 the resource is obtained to prepare for the input. △

The construct $P \wr \chi$ enables us to specify *failure prone multiprocesses*, with typical representative FP. Using P to denote a multiprocess as defined in Table 6.1, Table 6.3 gives the syntax of our extended programming language. Since we have abstracted from the internal state of a process, we allow only constants in priority assignments.

Table 6.3: Extended syntax of the programming language

Failure prone multiProcess	FP	$::=$	$P \mid FP_1 /\!/ FP_2 \mid \mathbf{prio}\ \pi\ (FP) \mid FP \wr \chi$
Failure prone Network	FN	$::=$	$\ll FP \gg \mid$ $FN_1 \parallel FN_2 \mid FN \wr \chi \mid FN \setminus cset$

From Definition 6.22 we obtain $chan(\chi) \subseteq chan(FP)$. Then, $chan(FP \wr \chi) = chan(FP) \cup chan(\chi) = chan(FP)$. Also, $chan(FN \wr \chi) = chan(FN)$. As before, define $chan(FP_1 /\!/ FP_2) = chan(FP_1) \cup chan(FP_2)$, $chan(\mathbf{prio}\ \pi\ (FP)) =$

$chan(FP)$, $chan(FN_1 \| FN_2) = chan(FN_1) \cup chan(FN_2)$ and $chan(FN \setminus cset) = chan(FN) - cset$.

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

- From the definition of $\mathcal{M}[\![P_1 /\!/ P_2]\!]$ given in Section 6.3 we obtain:

$$\begin{array}{l} \mathcal{O}[\![FP_1 /\!/ FP_2]\!] = \\ \{\ (\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}) \mid \text{for } i = 1,2 \text{ there exist } (\theta_i, \mathfrak{R}_i, \mathfrak{O}_i, \mathfrak{Q}_i) \in \mathcal{O}[\![FP_i]\!] \text{ such} \\ \qquad\qquad\qquad \text{that } NoConflict(\mathfrak{O}_1, \mathfrak{O}_2),\ \mathfrak{O} = \mathfrak{O}_1 \cup \mathfrak{O}_2,\ \mathfrak{Q} = \mathfrak{Q}_1 \cup \mathfrak{Q}_2, \\ \qquad\qquad\qquad Respect(\mathfrak{O}, \mathfrak{Q}),\ \theta\!\uparrow chan(FP_i) = \theta_i, \\ \qquad\qquad\qquad \theta\!\uparrow chan(FP_1 /\!/ FP_2) = \theta, \text{ and } \mathfrak{R} = \mathfrak{R}_1 \cup \mathfrak{R}_2\ \}. \end{array}$$

- From the definition of $\mathcal{M}[\![\mathbf{prio}\ e\ (P)]\!]$ given in that section we obtain:

$$\begin{array}{l} \mathcal{O}[\![\mathbf{prio}\ \pi\ (FP)]\!] \ = \\ \{\ (\theta, \mathfrak{R}, IncPr(\pi, \mathfrak{O}), IncPr(\pi, \mathfrak{Q})) \mid (\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}) \in \mathcal{O}[\![FP]\!]\ \}. \end{array}$$

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

$$\begin{array}{l} \mathcal{O}[\![FP \wr \chi]\!] \ = \\ \{\ (\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}) \mid \text{there exists a } (\theta_0, \mathfrak{R}_0, \mathfrak{O}_0, \mathfrak{Q}_0) \in \mathcal{O}[\![FP]\!] \text{ such that,} \\ \qquad\qquad\qquad \text{for all } \gamma,\ (\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \mathfrak{O}_0, \mathfrak{O}, \mathfrak{Q}_0, \mathfrak{Q}, \gamma) \models \chi, \\ \qquad\qquad\qquad \theta\!\uparrow chan(FP) = \theta, \text{ and } \mathfrak{R}\!\uparrow chan(FP) = \mathfrak{R}\ \}. \end{array}$$

The timed observations of the failure prone network *FN* are as defined in Section 5.5 with the extra clause:

- From the definition of $\mathcal{O}[\![\ll P \gg]\!]$ given in Section 6.3 we easily obtain:

$$\begin{array}{rcll} \mathcal{O}[\![\ll FP \gg]\!] & = & \{\ (\theta \setminus io(FP), \mathfrak{R} \setminus io(FP)) \mid & (\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}) \in \mathcal{O}[\![FP]\!] \\ & & & \wedge\ NoStrike(\mathfrak{O}, \mathfrak{Q}) \\ & & & \wedge\ ASAP(\mathfrak{R}, io(FP))\ \}. \end{array}$$

Definition 6.24 (Composite transformation expression) For transformation expressions ψ_1 and ψ_2, the *composite transformation expression* $\psi_1 \wr \psi_2$ is defined as follows:

$$\begin{array}{rcll} \psi_1 \wr \psi_2 & \equiv & \exists s, N, K, L \cdot & \psi_1(h_{old}, s, R_{old}, N, O_{old}, K, Q_{old}, L) \\ & & & \wedge\ \psi_2(s, h, N, R, K, O, L, Q), \end{array}$$

where s, N, K, and L must be fresh. ◇

We will also use this operator to compose assertions and transformation expressions, e.g. $\varphi \wr \psi \equiv \exists s, N, K, L \cdot \varphi(s, N, K, L) \wedge \psi(s, h, N, R, K, O, L, Q)$. Observe that, since φ is an assertion, h_{old}, O_{old}, Q_{old}, and R_{old} do not appear in φ, and hence also the composite expression $\varphi \wr \chi$ is an assertion.

Since the interpretation of assertions has not changed, the validity of the correctness formula FP **sat** φ is as defined in Definition 6.18, with P replaced by FP.

Definition 6.25 (Validity of a correctness formula) For process FP and assertion φ correctness formula FP **sat** φ is *valid*, notation $\models FP$ **sat** φ, if, and only if, for all γ and all $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}) \in \mathcal{O}[\![FP]\!]$, $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \phi$. ◇

The validity of the correctness formula FN **sat** ϕ follows from Definition 5.35.

6.6 A compositional network proof theory

In this section we give a compositional network proof system for our correctness formulae.

6.6.1 A proof theory for failure prone multiprocesses

The proof system for failure prone multiprocesses contains the following two general rules.

Rule 6.26 (Consequence)

$$\frac{FP \textbf{ sat } \varphi_1 \ , \quad \varphi_1 \rightarrow \varphi_2}{FP \textbf{ sat } \varphi_2}$$

Rule 6.27 (Conjunction)

$$\frac{FP \textbf{ sat } \varphi_1 \ , \quad FP \textbf{ sat } \varphi_2}{FP \textbf{ sat } \varphi_1 \ \wedge \ \varphi_2}$$

For interleaving we have the following inference rule.

Rule 6.28 (Interleaving)

$$\frac{FP_1 \textbf{ sat } \varphi_1(h, R, O, Q) \ , \quad FP_2 \textbf{ sat } \varphi_2(h, R, O, Q)}{\begin{aligned} FP_1 /\!/ FP_2 \textbf{ sat } \exists K_1, K_2, L_1, L_2, N_1, N_2 \cdot \ & \mathit{NoConflict}(K_1, K_2) \\ & \wedge O = K_1 \cup K_2 \\ & \wedge Q = L_1 \cup L_2 \\ & \wedge \mathit{Respect}(O, Q) \\ & \wedge R = N_1 {\uparrow} \mathit{chan}(FP_1) \cup N_2 {\uparrow} \mathit{chan}(FP_2) \\ & \wedge \varphi_1(h {\uparrow} \mathit{chan}(FP_1), N_1, K_1, L_1) \\ & \wedge \varphi_2(h {\uparrow} \mathit{chan}(FP_2), N_2, K_2, L_2) \end{aligned}}$$

The following rule characterizes priority assignment.

Rule 6.29 (Priority assignment)

$$\frac{FP \textbf{ sat } \varphi(h, R, \mathit{IncPr}(\pi, O), \mathit{IncPr}(\pi, Q))}{\textbf{prio } \pi \ (FP) \textbf{ sat } \varphi(h, R, O, Q)}$$

For the introduction of a failure hypothesis we have

Rule 6.30 (Failure hypothesis introduction)

$$\frac{FP \ \mathbf{sat} \ \varphi}{FP \wr \chi \ \mathbf{sat} \ \varphi \wr \chi}$$

Example 6.31 (Calculator) In Example 6.15 we saw that after each input C does not accept subsequent input for $RO(K_{C_1} + K_{C_2})$ time units. Consequently, if failure prone process $C \wr Reset$ refuses in communications during the $RO(K_{C_1} + K_{C_2})$ time units following a previous input, then we can conclude that no reset has occurred while processing that previous input and, hence, output must have been produced.

$$\begin{array}{ll} C \wr Reset \ \mathbf{sat} & \forall t, v \cdot (t, in, v) \in h \\ & \qquad \rightarrow in \ \mathbf{refused\ for\ precisely} \ RO(K_{C_1} + K_{C_2}) \\ & \qquad\quad \rightarrow \exists t_1 \cdot \ RO(K_{C_1}) \leq t_1 < RO(K_{C_1} + K_{C_2}) \\ & \qquad\qquad\quad \wedge (t_1, out, f(v)) \in h. \end{array}$$

△

6.6.2 A proof theory for failure prone networks

The following rule establishes the correspondence between the model for multiprocesses and that for networks. Internal channels are hidden.

Rule 6.32 (Processor closure)

$$\frac{FP \ \mathbf{sat} \ (NoStrike(O, Q) \wedge ASAP(R, io(FP))) \rightarrow \phi(h \setminus io(FP), R \setminus io(FP))}{\ll FP \gg \ \mathbf{sat} \ \phi(h, R)}$$

Example 6.33 (Processor closure) Provided the environment offers subsequent inputs at least $2K_{C_1}$ time units apart, C enables output within $2K_{C_1}$ time units after the environment started offering input.

$$\begin{array}{lll} \ll C_1 // C_2 \gg \ \mathbf{sat} & \forall t, \hat{t} \cdot & in_1 \ \mathbf{enabled\ precisely\ for} \ \hat{t} \\ & & \quad \rightarrow \mathbf{after} \ \hat{t} : in_1 \ \mathbf{refused\ for} \ 2K_{C_1} \\ & & \wedge \ in_2 \ \mathbf{enabled\ precisely\ for} \ \hat{t} \\ & & \quad \rightarrow \mathbf{after} \ \hat{t} : in_2 \ \mathbf{refused\ for} \ 2K_{C_1} \\ & \rightarrow & \\ & \forall t, \hat{t} \cdot & in_1 \ \mathbf{enabled\ for} \ \hat{t} \rightarrow \hat{t} \leq 2K_{C_1} \\ & & \wedge \ in_2 \ \mathbf{enabled\ for} \ \hat{t} \rightarrow \hat{t} \leq 2K_{C_1}, \end{array}$$

and because a process that was reset requires the resource even less we still have:

$$\begin{array}{ll}
\multicolumn{2}{l}{\ll (C_1 \wr \mathit{Reset}) /\!\!/ (C_2 \wr \mathit{Reset}) \gg} \\
\multicolumn{2}{l}{\mathbf{sat}} \\
\forall t, \widehat{t} \cdot & \mathit{in}_1 \textbf{ enabled precisely for } \widehat{t} \rightarrow \textbf{ after } \widehat{t} : \mathit{in}_1 \textbf{ refused for } 2K_{C_1} \\
& \wedge\ \mathit{in}_2 \textbf{ enabled precisely for } \widehat{t} \rightarrow \textbf{ after } \widehat{t} : \mathit{in}_2 \textbf{ refused for } 2K_{C_1} \\
\multicolumn{2}{l}{\rightarrow} \\
\forall t, \widehat{t} \cdot & \mathit{in}_1 \textbf{ enabled for } \widehat{t} \rightarrow \widehat{t} \leq 2K_{C_1} \\
& \wedge\ \mathit{in}_2 \textbf{ enabled for } \widehat{t} \rightarrow \widehat{t} \leq 2K_{C_1}.
\end{array}$$

△

Once at the network level, we can use the proof theory of Section 5.6.

6.7 Soundness and relative network completeness

In this section we show that the proof system of Section 6.6 is sound and relatively network complete.

Theorem 6.34 (Soundness) The proof system of Section 6.6 is sound.

Proof. See Appendix H.1.

Axiom 6.35 (Relative completeness assumption) For an assertion φ,

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

○

For multiprocesses we need an adapted notion of (relative) preciseness.

Definition 6.36 (Relative preciseness) An assertion φ is *relatively precise* for failure prone multiprocess *FP* if, and only if,

i) $\models FP$ **sat** φ;

ii) if $chan(\theta) \subseteq chan(FP)$, if $chan(\mathfrak{R}) \subseteq chan(FP)$, and if, for some γ, $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \varphi$, then $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}) \in \mathcal{O}[\![FP]\!]$;

iii) $\varphi(h, R, O, Q) \rightarrow \varphi(h \uparrow chan(FP), R \uparrow chan(FP), O, Q)$. ◇

Lemma 6.37 (Preciseness preservation) Assume that for any multiprocess P there exists an assertion φ which is precise for P and $\vdash P$ **sat** φ. Then, for any failure prone multiprocess FP there exists an assertion η which is precise for FP and $\vdash FP$ **sat** η.

Proof. See Appendix H.2.

Completeness is proved by analogy with Section 5.8.

Theorem 6.38 (Relative network completeness) The proof system of Section 6.6 is relatively network complete.

6.8 Discussion

The programming language for multiprogramming, especially the construct **prio** e (P) and the processor closure operator $\ll\gg$, was inspired by Hooman [Hooman92]. The same goes for the special purpose communication hardware assumption. In Hooman's theory, the priority of a process is determined by the closest surrounding priority assignment. Consequently, priority assignments may not be nested, since such a nesting might alter the relative process priorities. For instance, by assigning the priority 2 to the multiprocess $P_1 /\!/$ **prio** 1 (P_2) the process P_1 becomes the most important component. In our model the nesting of priority assignments has a cumulative effect. Furthermore, the model presented in this chapter enables us to take the time spent waiting for the resource into account.

We have not explicitly considered the scheduler. Instead we have concentrated on the effects of a scheduler on the observable process behaviour. Consequently, failures of the scheduler have been ignored. However, the case that the scheduler does not grant the resource to a requesting process as soon as it becomes available corresponds to not applying the processor closure rule. Also, the case that the scheduler does not respect the priorities can be considered using a version of the interleaving rule in which the $Respect(O, Q)$ clause does not appear.

Chapter 7

Concluding remarks

In a fault tolerant system, three forms of behaviour are distinguished: normal, exceptional and catastrophic. Normal behaviour is the behaviour that conforms to the specification. The discriminating factor between exceptional and catastrophic behaviour is the failure hypothesis which stipulates how faults affect the normal behaviour. The exceptional behaviour together with the normal behaviour constitutes the acceptable behaviour. Another important fault tolerant system feature is the fault hypothesis which, in fact, determines the collection of components that must function correctly during any interval of operation.

In this thesis we develop formal frameworks to specify and verify fault tolerant real-time distributed systems. In these theories, the failure hypothesis of a process is formalized as a relation between the normal and acceptable observable input and output behaviour of that process. Such a relation enables us to abstract from the precise nature of a fault and to focus on the abnormal behaviour it causes. We abstract from the sequential details of programs and formalize fault tolerance in relation to concurrency.

The formalisms are compositional to support top-down program design where, to master the complexity, a program is decomposed into a number of smaller ones. In a compositional theory the specification of a composite process can be inferred from the specifications of its components without reference to the internal structure of those parts. Consequently, each design step can be individually verified. Our approach allows a general treatment of paradigms for fault tolerance because it supports a modular treatment of acceptable behaviour: the acceptable behaviour of the process P under the failure hypothesis χ is the normal behaviour of the failure prone process $P \wr \chi$. The possibility of expressing the failure hypothesis of a subsystem enables the formalization of a fault hypothesis.

The basic formalism is the untimed trace-based approach presented in Chapter 3. Two interesting applications, namely the classification of the processes that, given a particular failure hypothesis, satisfy a given specification, and the determination of the least restrictive failure hypothesis such that a given process still satisfies a given specification, are illustrated in Chapter 4.

To describe the real-time behaviour of distributed systems we introduce in Chapter 5 a primitive to express when a process refuses to communicate. The resulting formalism, which assumes maximal parallelism, is generalized in Chapter 6 for systems whose limited resources are shared by several processes and scheduling takes place on the basis of dynamic priorities. This is achieved using primitives to denote when a process occupies the resource and when it is requesting to do so.

Comparing our proof system with trace-based formalisms for normal behaviour (see e.g. [Zwiers89]), only one new rule, viz. the failure hypothesis introduction rule, has been added to capture acceptable executions. Apart from a number of smaller examples, we illustrate our method by proving the correctness of a triple modular redundant system (in Chapters 3 and 5) and the alternating bit protocol (in Chapter 3), using only the specifications of the components. An analysis, purely in terms of the model presented in Section 3.2, of a stable disk can be found in [Schepers93b].

The formalization of a failure hypothesis as a relation does not hang on our particular representation of the process behaviour. Consider, for instance, a system S whose state consists of two integers x and y, that is, $STATE_S = \{ \sigma \mid \sigma : \{x,y\} \rightarrow \mathbb{N} \}$. Assume that in a sequence s of states a new state is recorded whenever the value of x or y changes. If we allow transient memory faults to occur, then it is easy to formalize that we might observe the sequence $s = (0,0),(3,0),(10,0),\ldots$ instead of some intended sequence $s_{old} = (0,0),(10,0),\ldots$

Finding a logic to express failure hypotheses more elegantly, e.g. using the classification of failures that appears in [Cristian91], is a subject of future investigation. It is advisable to investigate whether there is any benefit in relating an acceptable observation to a number of normal observations instead of just one. This will certainly be the case when taking the sequential details of programs into account, which is another continuation of the research described in this thesis.

An interesting subject of future investigation is the incorporation of failure rates in our formalisms. The qualitative theories presented in this thesis do not allow us to express that a transmission medium corrupts a message in only 5% of the cases, or at most once every hour. The fault tolerance of a system is also a quantitative matter, for instance because some failure hypothesis holds in only 99.9% of the cases.

A system is adaptive fault tolerant if it continues to provide its specified service even when the circumstances, for instance the weather, change. To reason about adaptive fault tolerance our theories need to be extended with a mechanism to weave failure hypotheses. Finally, note that our formalisms are very suitable to reason about (multi-level) security where most properties are expressed purely in terms of the observable behaviour (e.g. [SMcD93]). Of particular interest is the determination of the least restrictive failure hypothesis such that a given system is still secure.

Appendix A

Paradigms for fault tolerance

A.1 Consistency check

Consistency check paradigms apply to those cases where the output of a component is checked with respect to its specified functionality. Such paradigms are used especially when a component performs a mathematical function, for instance by verifying whether the result conforms to the specified format (*syntax checking*), by verifying whether the result lies in the specified range (*range checking*) or by verifying whether the application of the reverse function to the result yields the input again (*reversal checking*).

A.2 Duplication with comparison

If consistency checks are not feasible, then the most rigorous way to detect the failure of a component is to duplicate that component. Both components receive the same input and perform the same tasks. Their output is compared and only passed on if there is a match (see Figure A.1). Such a design leads to a fail-silent system: if one component fails the system does not output anything. The (subtle) difference with a fail-stop system [SS83] is that the components do not halt, i.e., still accept input. Under the fault hypothesis that if both components fail they do not output identical erroneous values, the system always delivers correct output or none at all. When this paradigm is used to design fault tolerant hardware, for which it is very popular, the components are usually synchronized. This synchronization is less stringent when used to design fault tolerant software.

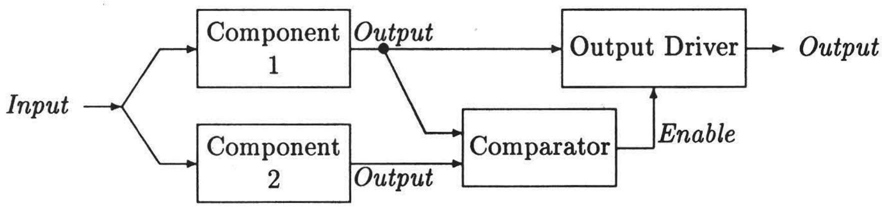

Figure A.1: Duplication with comparison

A.2.1 Analysis of duplication with comparison

This method requires the use of an extra component, a comparator, and an output driver. Since a component sends its output via one link only, there is no distinction between the failure of a link and the failure of the component using it. Furthermore, the failure of the comparator or the output driver results in the failure of the system. It may seem as if the system has become merely less reliable because of the larger number of components, but because of the relative simplicity of both the comparator and the output driver, their failure is far less likely than the failure of one of the duplicated components.

A.3 Triple modular redundancy

Duplication with comparison is capable of preventing the failure of a system, but if one of the duplicated components fails the system outputs nothing. If the component is triplicated and another component acts as a *voter*, which passes the majority vote of the outputs of the individual components, the system can still produce correct output even when one of the triplicated components fails: its failure can be masked. This is known as the triple modular redundancy paradigm which is illustrated in Figure A.2. Again, the synchronization is less stringent when used to design fault tolerant software (for instance the SIFT system [WL+78]).

The triple modular redundancy paradigm can be generalized to N-modular redundancy ($N \geq 3$). In case the output of an N-modular redundant system is used as input for an M-modular redundant system, M voters process the M-fold output of the N components. The class of faults that cause a component to send conflicting output to the M voters is the well-known class of *Byzantine* faults [LSP82].

It should be noted that instead of N identical components, N *similar* components can be used. Using N different implementations of the same specification is a popular method to protect against faults in the software. The

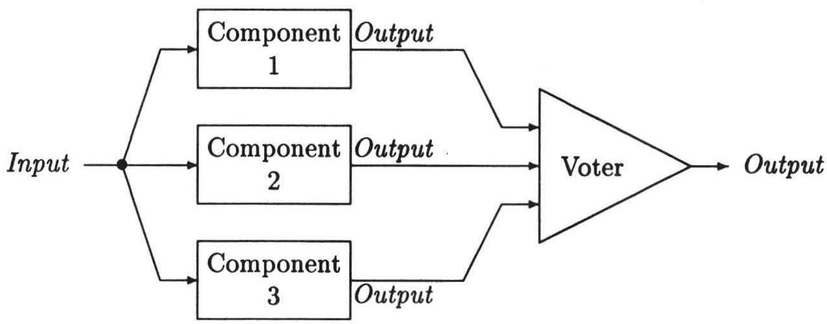

Figure A.2: A triple modular redundant component

recovery block scheme [HLMR74] is a well-known paradigm to minimize the consequences of programmer faults. The reasonableness of the result calculated by the primary version, or module, is checked by an acceptance test. If the result is unacceptable, an alternate module is executed. The result returned by this alternate module is checked by the same acceptance test, and if the test is negative yet another module is executed. Typically, a checkpoint is established and recorded before executing the primary module. A major strength of the recovery block paradigm is that graceful degradation can easily be incorporated by invoking ever simpler modules. A major weakness is the acceptance test. If all versions are invoked then voting can determine the presumably correct result calculated by the majority of the modules — the N-version programming paradigm [CA78]. However, especially in case of real number arithmetic, different algorithms can lead to small discrepancies between valid results. To identify a consensus in such a case so-called *inexact* voting is needed.

A.3.1 Analysis of triple modular redundancy

Here the redundancy consists of the two replicas of the given component, plus the voter. The voter uses a majority vote on the outputs of the three components; this is possible as long as the outputs of at least two components are identical. The voter is usually designed to output nothing if no two of its inputs are identical. Clearly, when two components produce identical incorrect values, incorrect output is produced. Hence, the fault hypothesis typically stipulates that "the voter does not fail and no two components fail in similar fashion".

A.4 Coding

A popular and effective method to protect data against corruption during transmission is the use of coding: a dataword is transformed into a codeword which contains some redundant bits. Besides its application for reliable communication, coding has been used for decades to realize fault tolerant data storage.

For two (binary) codewords of the same length, the *Hamming distance* [Hamming50] is the number of bit positions in which the two codewords differ, i.e., the number of single bit errors that are needed to convert one codeword into another. For example, the Hamming distance between the codewords 0000000000 and 1111111111 is ten. The Hamming distance of a complete code is equal to the minimum Hamming distance of all pairs of codewords in the code. For example, the Hamming distance of the code consisting of the codewords 0000000000, 0000011111, 1111100000 and 1111111111 is five.

Now, if a code has Hamming distance h, $h-1$ single bit errors cannot transform one codeword into another codeword. This code is thus capable of detecting up to $h-1$ single bit errors. Furthermore, if no more than $\lfloor \frac{1}{2}(h-1) \rfloor$ single bit errors occur, the original codeword is still closer than any other codeword. Hence, up to $\lfloor \frac{1}{2}(h-1) \rfloor$ single bit errors can be corrected. For the above given code, up to 4 single bit errors can be detected, and up to 2 single bit errors can be corrected.

The following sections present Hamming coding, the well-known error correcting coding paradigm, and cyclic redundancy coding, which is a very popular error detecting coding paradigm, especially to detect corruption of data stored on disks.

A.4.1 Hamming coding

The positions of the bits in a codeword can be numbered, where the leftmost bit position has number 1. To be able to correct a single bit error in a codeword, the code bits at the positions with numbers that are powers of 2 are used as check bits. A dataword is converted into a codeword by inserting the data bits at the remaining positions [Hamming50].

Every bit position can be written as a sum of powers of 2, e.g. $5 = 2^0 + 2^2$. The check bit at position 2^i ensures the parity of those code bits whose position contains a term 2^i, thus including itself, to be odd or even. For example, the dataword 11011011 is converted into the codeword $1_c1_c11_c1011_c1011$, where the subscript c denotes a check bit and even parity is used.

A.4.1.1 Analysis of Hamming coding

As mentioned before, Hamming coding can only be used to correct single bit errors. In an n-bit Hamming codeword there are $\lceil {}^2\log(n+1) \rceil$ redundant bits, that is, the relationship between the number m of data bits and the n bits of the codeword is $n = m + \lceil {}^2\log(n+1) \rceil$. To correct single bit errors, the Hamming distance of the code must be 2, or, in other words, the n-bit bit strings at Hamming distance 1 from a legal codeword are illegal. Since there are n such bit strings, there are $n+1$ n-bit bit strings 'dedicated' to each m-bit dataword. Because there are 2^m m-bit datawords and there are 2^n n-bit bit strings, it is necessary that $2^n \geq (n+1)2^m$. Thus, the Hamming coding method achieves the lower bound.

Now, assume that a single bit error has occurred. The check bits at the positions that occur as a term of the position of the corrupted bit disagree with the parity. For instance, the corruption of the bit at position 5 results in incorrect check bits at positions 1 and 4. It can easily be seen that the sum of the positions of the incorrect check bits equals the position of the corrupted bit.

A.4.2 Cyclic redundancy coding

An n-bit dataword can be regarded as the list of coefficients, where the coefficients are 0 or 1, of a polynomial $M(x)$ with n terms, ranging from x^{n-1} to x^0. The basic idea of cyclic redundancy coding is to append a checksum to the end of the dataword, such that the polynomial $C(x)$ represented by the checksummed dataword is divisible, using modulo 2 arithmetic, by a generator polynomial $G(x)$ [PB61].

Let g be the degree of $G(x)$. The algorithm for computing the checksummed dataword consists of three steps:

1. Append g zero bits to the end of the dataword, resulting in a bit string of $n+g$ bits which represents the polynomial $x^g M(x)$.

2. Divide the bit string from Step 1 by the generator polynomial $G(x)$ using modulo 2 division. This can easily be implemented in hardware, i.e., by repeatedly shifting and exclusive or-ing. The remainder is a bit string consisting of at most g bits.

3. Subtract the remainder generated in Step 2 from the bit string of Step 1 using modulo 2 subtraction. Again, this can easily be implemented in hardware, i.e., by exclusive or-ing. The result is the checksummed dataword which is divisible by the generator.

Consider the dataword 11010 and the generator 101. Step 1 produces the bit string 1101000. Step 2 yields the remainder 01. Subtracting 01 from 1101000 results in 1101001 being transmitted.

A.4.2.1 Analysis of cyclic redundancy coding

Suppose that instead of a bit string representing $C(x)$, a bit string representing $C(x)+E(x)$ is received, where $E(x)$ is the error polynomial. $E(x)$ has the same degree as $C(x)$ and a coefficient equal to 1 means that the corresponding bit is inverted, that is, incorrect.

In the case of a single bit error, $E(x) = x^i$, where i determines which bit is in error. If $G(x)$ contains more than one term, it does not divide $E(x)$ and hence it does not divide $C(x) + E(x)$. Thus, if $G(x)$ contains more than one term, a single bit error is always detected.

In the case of a double bit error, $E(x) = x^i + x^j$ $(i > j)$, or $E(x) = x^j(x^{i-j} + 1)$. If we assume that $G(x)$ does not contain a factor x — which is simply satisfied if the lowest order bit of the generator is 1 — all double bit errors are detected if $G(x)$ does not divide $x^{i-j} + 1$ for any $i - j$, i.e., for $i - j$ up to the length of $C(x)$.

In the case of an odd number of errors, $E(x)$ contains an odd number of terms. Evaluating $E(1)$ thus yields 1 (modulo 2). Since $E(1)$ would be zero if $E(x)$ contained a factor $(x + 1)$, an odd number of errors is detected if $G(x)$ has a factor $x + 1$.

In the case of a burst error of length b, $E(x) = x^{i+b-1} + ... + x^i$, or $E(x) = x^i(x^{b-1} + ... + 1)$. Under the assumption that $G(x)$ does not contain a factor x and that the coefficient of its lowest order term, x^0, is 1, $G(x)$ cannot divide $E(x)$ if the degree of $G(x)$ is greater than the degree of $E(x)$, i.e., if $g > b - 1$, or $b < g + 1$. If $b = g + 1$ then $G(x)$ can only divide $E(x)$ if $E(x) = G(x)$. The most and the least significant bit of a burst are 1 by definition, so that, assuming that 0 and 1 have equal probability, the probability that a burst error of length $g + 1$ is not detected is $\frac{1}{2}^{g-1}$. If $b > g + 1$ then $G(x)$ can only divide $E(x)$ if $E(x) = A(x)G(x)$. Because the least significant bit of both $E(x)$ and $G(x)$ is 1, the least significant bit of $A(x)$ is 1. Since the degree of $A(x)$ is $b - 1 - g$, there are 2^{b-2-g} different undetectable burst errors. Because the total number of different burst errors of length b is 2^{b-2}, the probability that a burst error of length b is not detected is 2^{-g}. Thus, if $G(x)$ does not contain a factor x and the coefficient of x^0 is 1, the fraction of burst errors of length b that is not detected is 0 if $b < g + 1$, $\frac{1}{2}^{g-1}$ if $b = g + 1$ and $\frac{1}{2}^{g}$ if $b > g + 1$.

Appendix B

Definitions from Chapter 3

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

- $var(\mu) = \emptyset$;
- $var(x) = \{x\}$;
- $var(f(e_1, \ldots, e_n)) = \cup_{i=1}^{n} var(e_i)$;
- $var(e_1 = e_2) = var(e_1 < e_2) = var(e_1) \cup var(e_2)$;
- $var(\neg b) = var(b)$;
- $var(b_1 \vee b_2) = var(b_1) \cup var(b_2)$;
- $var(\mathbf{skip}) = \emptyset$;
- $var(x := e) = \{x\} \cup var(e)$;
- $var(c!e) = var(e)$;
- $var(c?x) = \{x\}$;
- $var(P_1 \,;\, P_2) = var(P_1) \cup var(P_2)$;
- $var([\ \|_{i=1}^{n}\ b_i \rightarrow P_i\ ]) = \cup_{i=1}^{n} var(b_i)\ \cup\ \cup_{i=1}^{n} var(P_i)$;
- $var(*G) = var(G)$;
- $var(P_1 \parallel P_2) = var(P_1) \cup var(P_2)$;
- $var(P \backslash cset) = var(P)$. $\diamond$

Definition B.2 (Observable input channels of a process) The set of visible, or observable, *input channels* of process P, notation $in(P)$, is defined inductively as follows:

- $in(\mathbf{skip}) = in(x := e) = in(c!e) = \emptyset$;
- $in(c?x) = \{c\}$;
- $in(P_1 \,;\, P_2) = in(P_1) \cup in(P_2)$;

- $in([\ \|_{i=1}^{n}\ b_i \to P_i\ ]) = \cup_{i=1}^{n} in(P_i)$;
- $in(*G) = in(G)$;
- $in(P_1 \parallel P_2) = in(P_1) \cup in(P_2)$;
- $in(P \backslash cset) = in(P) - cset$. ◇

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

- $out(\textbf{skip}) = out(x := e) = \emptyset$;
- $out(c!e) = \{c\}$;
- $out(c?x) = \emptyset$;
- $out(P_1\ ;\ P_2) = out(P_1) \cup out(P_2)$;
- $out([\ \|_{i=1}^{n}\ b_i \to P_i\ ]) = \cup_{i=1}^{n} out(P_i)$;
- $out(*G) = out(G)$;
- $out(P_1 \parallel P_2) = out(P_1) \cup out(P_2)$;
- $out(P \backslash cset) = out(P) - cset$. ◇

Definition B.4 (History channels of an assertion) For assertion ϕ the set $chan(\phi)$ is inductively defined as the union of the sets of channels used to restrict references to h in ϕ.

- $chan(0) = chan(1) = chan(i) = \emptyset$;
- $chan(iexp_1 + iexp_2) = chan(iexp_1 \times iexp_2) = chan(iexp_1) \cup chan(iexp_2)$;
- $chan(len(texp)) = chan(texp)$;
- $chan(c) = \emptyset$;
- $chan(ch(rexp)) = chan(rexp)$;
- $chan(\mu) = chan(v) = \emptyset$;
- $chan(val(rexp)) = chan(rexp)$;
- $chan(f(vexp_1, \ldots, vexp_n)) = \cup_{i=1}^{n} chan(vexp_i)$;
- $chan((cexp, vexp)) = chan(cexp) \cup chan(vexp)$;
- $chan(texp(iexp)) = chan(texp) \cup chan(iexp)$;
- $chan(s) = \emptyset$;

- $chan(h) = CHAN$;
- $chan(\langle\rangle) = \emptyset$;
- $chan(\langle rexp \rangle) = chan(rexp)$;
- $chan(texp_1 {}^\frown texp_2) = chan(texp_1) \cup chan(texp_2)$;
- $chan(texp \upharpoonright cset) = chan(texp) \cap cset$;
- $chan(texp[iexp]) = chan(texp) \cup chan(iexp)$;
- $chan(iexp_1 = iexp_2) = chan(iexp_1 < iexp_2) = chan(iexp_1) \cup chan(iexp_2)$;
- $chan(cexp_1 = cexp_2) = chan(cexp_1) \cup chan(cexp_2)$;
- $chan(vexp_1 = vexp_2) = chan(vexp_1 < vexp_2) = chan(vexp_1) \cup chan(vexp_2)$;
- $chan(texp_1 = texp_2) = chan(texp_1) \cup chan(texp_2)$;
- $chan(\phi_1 \wedge \phi_2) = chan(\phi_1) \cup chan(\phi_2)$;
- $chan(\neg\phi) = chan(\exists i \cdot \phi) = chan(\exists v \cdot \phi) = chan(\exists s \cdot \phi) = chan(\phi)$. $\diamond$

Appendix C

Proofs from Chapter 3

C.1 Proof of the prefix closedness lemma

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

(a) Assume that $\theta \in \mathcal{H}[\![FP_1 \parallel FP_2]\!]$, that is, assume that, for $i = 1, 2$,

$$\theta \uparrow chan(FP_i) \in \mathcal{H}[\![FP_i]\!] \tag{C.1}$$

and

$$\theta \uparrow chan(FP_1 \parallel FP_2) = \theta. \tag{C.2}$$

Consider any $\theta' \preceq \theta$. Since $\theta' \preceq \theta$, we have $\theta' \uparrow chan(FP_i) \preceq \theta \uparrow chan(FP_i)$, for $i = 1, 2$. By (C.1) and the induction hypothesis, we conclude that, for $i = 1, 2$,

$$\theta' \uparrow chan(FP_i) \in \mathcal{H}[\![FP_i]\!]. \tag{C.3}$$

By (C.2), $chan(\theta) \subseteq chan(FP_1 \parallel FP_2)$. Since $\theta' \preceq \theta$, $chan(\theta') \subseteq chan(\theta)$. Consequently, $chan(\theta') \subseteq chan(FP_1 \parallel FP_2)$ from which we infer that

$$\theta' \uparrow chan(FP_1 \parallel FP_2) = \theta'. \tag{C.4}$$

From (C.3) and (C.4) we conclude that $\theta' \in \mathcal{H}[\![FP_1 \parallel FP_2]\!]$.

(b) Assume $\theta \in \mathcal{H}[\![FP \setminus cset]\!]$, that is, assume there exists a $\theta_0 \in \mathcal{H}[\![FP]\!]$ such that $\theta_0 \setminus cset = \theta$. Consider any $\theta' \preceq \theta$. Obviously, there exists a $\theta'_0 \preceq \theta_0$ such that $\theta'_0 \setminus cset = \theta'$. By the induction hypothesis, $\theta'_0 \in \mathcal{H}[\![FP]\!]$. Hence $\theta' \in \mathcal{H}[\![FP \setminus cset]\!]$.

(c) Assume $\theta \in \mathcal{H}[\![FP \wr \chi]\!]$, that is, assume that there exists a $\theta_0 \in \mathcal{H}[\![FP]\!]$ such that, for all γ, $(\theta_0, \theta, \gamma) \models \chi$. Consider $\theta' \preceq \theta$. Using $\widehat{\gamma} = (\gamma : s \mapsto \theta')$, s fresh, we have $(\theta_0, \theta, \widehat{\gamma}) \models \chi$. Since $\theta' \preceq \theta$, we have $(\theta_0, \theta, \widehat{\gamma}) \models s \preceq h$. Consequently, $(\theta_0, \theta, \widehat{\gamma}) \models \chi \wedge s \preceq h$. By the syntactic restriction on χ, we obtain $(\theta_0, \theta, \widehat{\gamma}) \models \exists s_{old} \preceq h_{old} \cdot \chi[s/h, s_{old}/h_{old}]$. Thus there exists a θ'' such that $(\theta_0, \theta, (\widehat{\gamma} : s_{old} \mapsto \theta'')) \models s_{old} \preceq h_{old} \wedge \chi[s/h, s_{old}/h_{old}]$. Consequently, $\theta'' \preceq \theta_0$ and hence $(\theta_0, \theta, (\widehat{\gamma} : s_{old} \mapsto \theta'')) \models \chi[s/h, s_{old}/h_{old}]$. Then, by substitution lemma 3.50, $(\theta'', \widehat{\gamma}(s), (\widehat{\gamma} : s_{old} \mapsto \theta'')) \models \chi$. Since $\widehat{\gamma}(s) = \theta'$ and s and s_{old} do not occur in χ, we obtain $(\theta'', \theta', \gamma) \models \chi$. As $\theta_0 \in \mathcal{H}[\![FP]\!]$ and $\theta'' \preceq \theta_0$, the induction hypothesis yields $\theta'' \in \mathcal{H}[\![FP]\!]$, which proves $\theta' \in \mathcal{H}[\![FP \wr \chi]\!]$.

□

C.2 Proof of the composite failure hypothesis lemma

Assume that $\theta \in \mathcal{H}[\![FP \wr (\chi_1 \wr \chi_2)]\!]$, or, equivalently, assume that there exists a $\theta_0 \in \mathcal{H}[\![FP]\!]$ such that, for any γ, $(\theta_0, \theta, \gamma) \models (\chi_1 \wr \chi_2)$. By Definition 3.57, $(\theta_0, \theta, \gamma) \models \exists s \cdot \chi_1[s/h] \wedge \chi_2[s/h_{old}]$, that is, there exists a θ_1 such that, for $\widehat{\gamma} = (\gamma : s \mapsto \theta_1)$, $(\theta_0, \theta, \widehat{\gamma}) \models \chi_1[s/h] \wedge \chi_2[s/h_{old}]$. Observe that $\mathcal{T}[\![s]\!](\theta_0, \theta, \widehat{\gamma}) = \theta_1$. By substitution lemma 3.50, $(\theta_0, \theta, \widehat{\gamma}) \models \chi_1[s/h] \wedge \chi_2[s/h_{old}]$ iff $(\theta_0, \theta_1, \widehat{\gamma}) \models \chi_1$ and $(\theta_1, \theta, \widehat{\gamma}) \models \chi_2$. Then, $\theta \in \mathcal{H}[\![FP \wr (\chi_1 \wr \chi_2)]\!]$ iff there exists a $\theta_0 \in \mathcal{H}[\![FP]\!]$ such that, for any γ, there exists a θ_1 with $(\theta_0, \theta_1, \gamma) \models \chi_1$ and $(\theta_1, \theta, \gamma) \models \chi_2$. Hence, $\theta \in \mathcal{H}[\![FP \wr (\chi_1 \wr \chi_2)]\!]$ iff there exists some $\theta_1 \in \mathcal{H}[\![FP \wr \chi_1]\!]$ such that $(\theta_1, \theta, \gamma) \models \chi_2$. Equivalently, $\theta \in \mathcal{H}[\![FP \wr (\chi_1 \wr \chi_2)]\!]$ iff $\theta \in \mathcal{H}[\![(FP \wr \chi_1) \wr \chi_2]\!]$.

□

C.3 Proof of the persistency lemma

By induction on the length of h. (*Base Step*) The case $h = \langle\rangle$ is trivial. (*Induction Step*) Assume that the lemma holds for s, that is,

$$Val(RDAck(s \uparrow a_{out})) \preceq^1 Val(RDAck(s \uparrow a_{in})) \tag{C.5}$$

and

$$Dat(RDMsg(s \uparrow m_{out})) \preceq^1 Dat(RDMsg(s \uparrow m_{in})). \tag{C.6}$$

Four cases need examination:

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

   By (3.10), $len(RDAck(h \uparrow a_{out})) \leq^1 len(RDMsg(h \uparrow m_{in}))$. Since $s \prec h$, (3.10) yields $len(RDAck(s \uparrow a_{out})) \leq^1 len(RDMsg(s \uparrow m_{in}))$. Then, because in this case $h = s^\wedge(m_{in}, (v, b))$, we may conclude that

$$len(RDAck(s\uparrow a_{out})) = len(RDMsg(s\uparrow m_{in})). \tag{C.7}$$

Since $s \prec h$, (3.12) yields $Val(RDAck(s\uparrow a_{in})) \preceq^1 Bit(RDMsg(s\uparrow m_{out}))$. Then, by (C.5), $Val(RDAck(s\uparrow a_{out})) \preceq Bit(RDMsg(s\uparrow m_{out}))$. Consequently, $len(Val(RDAck(s\uparrow a_{out}))) \leq len(Bit(RDMsg(s\uparrow m_{out})))$, from which we conclude

$$len(RDAck(s\uparrow a_{out})) \leq len(RDMsg(s\uparrow m_{out})). \tag{C.8}$$

By (C.6), $len(RDMsg(s\uparrow m_{out})) \leq^1 len(RDMsg(s\uparrow m_{in}))$, i.e., by (C.8), $len(RDAck(s\uparrow a_{out})) \leq len(RDMsg(s\uparrow m_{out})) \leq^1 len(RDMsg(s\uparrow m_{in}))$. Finally, by (C.7), $len(RDMsg(s\uparrow m_{out})) = len(RDMsg(s\uparrow m_{in}))$, which, by (C.6), yields $Dat(RDMsg(s\uparrow m_{out})) = Dat(RDMsg(s\uparrow m_{in}))$. Then, obviously, $Dat(RDMsg(h\uparrow m_{out})) \prec^1 Dat(RDMsg(h\uparrow m_{in}))$, from which the lemma follows.

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

   Since $s \prec h$, (3.12) yields $Val(RDAck(s\uparrow a_{in})) = Bit(RDMsg(s\uparrow m_{out}))$. Then, by (C.5), $Val(RDAck(s\uparrow a_{out})) \preceq^1 Bit(RDMsg(s\uparrow m_{out}))$, from which we conclude

$$len(RDAck(s\uparrow a_{out})) \leq^1 len(RDMsg(s\uparrow m_{out})). \tag{C.9}$$

   Since $s \prec h$, we infer, using (3.10), that

$$len(RDAck(s\uparrow a_{out})) \leq^1 len(RDMsg(s\uparrow m_{in})). \tag{C.10}$$

   Since $s \prec h$, (3.13) yields $len(RDMsg(s\uparrow m_{out})) \leq len(RDMsg(s\uparrow m_{in}))$. By (C.9) and (C.10),

$$len(RDMsg(s\uparrow m_{out})) \leq^1 len(RDMsg(s\uparrow m_{in})). \tag{C.11}$$

   Suppose that $len(RDMsg(s\uparrow m_{out})) = len(RDMsg(s\uparrow m_{in}))$. Since in this case $h = s^\wedge(m_{out}, (v, b))$, with $b \neq bit(val(last(s\uparrow m_{out})))$, we obtain $len(RDMsg(h\uparrow m_{out})) = len(RDMsg(h\uparrow m_{in})) + 1$, which conflicts with (3.13). Then, by (C.11), $len(RDMsg(s\uparrow m_{out})) <^1 len(RDMsg(s\uparrow m_{in}))$, which, by (C.6), yields $Dat(RDMsg(s\uparrow m_{out})) \prec^1 Dat(RDMsg(s\uparrow m_{in}))$. By (3.14), we obtain that $v = msg(val(last(h[len(h)]\uparrow m_{in})))$, or, equivalently, $v = msg(val(last(s\uparrow m_{in})))$. Hence, $Dat(RDMsg(h\uparrow m_{out})) = Dat(RDMsg(h\uparrow m_{in}))$, from which we conclude that the lemma holds.

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

   By (3.12), $len(RDAck(h \uparrow a_{in})) \preceq^1 len(RDMsg(h \uparrow m_{out}))$. Since $s \prec h$, we obtain, by (3.12), that $len(RDAck(s \uparrow a_{in})) \preceq^1 len(RDMsg(s \uparrow m_{out}))$. Then, we conclude

$$len(RDAck(s \uparrow a_{in})) <^1 len(RDMsg(s \uparrow m_{out})). \tag{C.12}$$

   By (C.6), $len(RDMsg(s \uparrow m_{out})) \preceq^1 len(RDMsg(s \uparrow m_{in}))$; i.e., by (C.12), we conclude that

$$len(RDAck(s \uparrow a_{in})) < len(RDMsg(s \uparrow m_{in})). \tag{C.13}$$

   Since $s \prec h$, (3.15) yields $len(RDAck(s \uparrow a_{out})) \leq len(RDAck(s \uparrow a_{in}))$, which leads, using (C.13), to

$$len(RDAck(s \uparrow a_{out})) \leq len(RDAck(s \uparrow a_{in})) < len(RDMsg(s \uparrow m_{in})) \tag{C.14}$$

   Since $s \prec h$, (3.10) yields $len(RDAck(s \uparrow a_{out})) \preceq^1 len(RDMsg(s \uparrow m_{in}))$, which, by (C.14), leads to $len(RDAck(s \uparrow a_{out})) = len(RDAck(s \uparrow a_{in}))$. Hence, by (C.5), we obtain $Val(RDAck(s \uparrow a_{out})) = Val(RDAck(s \uparrow a_{in}))$. Then, we have, obviously, $Val(RDAck(h \uparrow a_{out})) \prec^1 Val(RDAck(h \uparrow a_{in}))$, from which we conclude that the lemma holds.

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

   Since $s \prec h$, (3.10) yields $Val(RDAck(s \uparrow a_{out})) \preceq^1 Bit(RDMsg(s \uparrow m_{in}))$. Hence, $Val(RDAck(s \uparrow a_{out})) \prec^1 Bit(RDMsg(s \uparrow m_{in}))$, from which we can conclude that

$$len(RDAck(s \uparrow a_{out})) <^1 len(RDMsg(s \uparrow m_{in})). \tag{C.15}$$

   By (C.6), we have $len(RDMsg(s \uparrow m_{out})) \preceq^1 len(RDMsg(s \uparrow m_{in}))$. Then, by (C.15), we conclude

$$len(RDAck(s \uparrow a_{out})) \preceq^1 len(RDMsg(s \uparrow m_{out})). \tag{C.16}$$

   Since $s \prec h$, we obtain, using (3.12), that

$$len(RDAck(s \uparrow a_{in})) \preceq^1 len(RDMsg(s \uparrow m_{out})). \tag{C.17}$$

   Since $s \prec h$, (3.15) yields $len(RDAck(s \uparrow a_{out})) \leq len(RDAck(s \uparrow a_{in}))$. Then, by (C.16) and (C.17), we conclude

$$len(RDAck(s \uparrow a_{out})) \preceq^1 len(RDAck(s \uparrow a_{in})). \tag{C.18}$$

Suppose $len(RDAck(s\uparrow a_{out})) = len(RDAck(s\uparrow a_{in}))$. Since in this case $h = s^\wedge(a_{out}, b)$, where $b \neq val(last(s\uparrow a_{out}))$, we may conclude that $len(RDAck(h\uparrow a_{out})) = len(RDAck(h\uparrow a_{in})) + 1$, conflicting with (3.15). Then, by (C.18), $len(RDAck(s\uparrow a_{out})) <^1 len(RDAck(s\uparrow a_{in}))$, which, using (C.5), yields $Val(RDAck(s\uparrow a_{out})) \prec^1 Val(RDAck(s\uparrow a_{in}))$. Finally, since, by (3.16), we have that $b = val(last(h[len(h)]\uparrow a_{in}))$, or, equivalently, $b = val(last(s\uparrow a_{in}))$, we obtain $Val(RDAck(h\uparrow a_{out})) = Val(RDAck(h\uparrow a_{in}))$, from which we conclude that the lemma holds.

□

C.4 Proof of the soundness theorem

C.4.1 Soundness of the consequence and conjunction rules

Trivial.

C.4.2 Soundness of the invariance rule

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

C.4.3 Soundness of the parallel composition rule

Assume that

$$chan(\phi_1) \subseteq chan(FP_1)\ ,\quad chan(\phi_2) \subseteq chan(FP_2). \tag{C.19}$$

Assume further

$$\models FP_1 \ \mathbf{sat}\ \phi_1\ ,\quad \models FP_2 \ \mathbf{sat}\ \phi_2. \tag{C.20}$$

We prove $\models FP_1 \| FP_2 \ \mathbf{sat}\ \phi_1 \wedge \phi_2$. Consider any γ. Let $\theta \in \mathcal{H}[\![FP_1\|FP_2]\!]$. By the definition of the semantics, for $i = 1, 2$, $\theta\uparrow chan(FP_i) \in \mathcal{H}[\![FP_i]\!]$ and $\theta\uparrow chan(FP_1\|FP_2) = \theta$. Hence, by (C.20), we obtain $(\theta\uparrow chan(FP_i), \gamma) \models \phi_i$. By projection lemma 3.51(a), we have $((\theta\uparrow chan(FP_i))\uparrow chan(\phi_i), \gamma) \models \phi_i$, thus

$$(\theta\uparrow(chan(FP_i) \cap chan(\phi_i)), \gamma) \models \phi_i. \tag{C.21}$$

By (C.19), $(\theta\uparrow chan(\phi_i), \gamma) \models \phi_i$, and hence, by projection lemma 3.51(a), $(\theta, \gamma) \models \phi_i$. This establishes that $\models FP_1\|FP_2 \ \mathbf{sat}\ \phi_1 \wedge \phi_2$. □

C.4.4 Soundness of the hiding rule

Assume

$$\models FP \ \mathbf{sat}\ \phi \tag{C.22}$$

and

$$chan(\phi) \cap cset = \emptyset. \tag{C.23}$$

We show $FP \setminus cset$ **sat** ϕ. Consider any γ. Let $\theta \in \mathcal{H}[\![FP \setminus cset]\!]$. Then there exists a $\theta_1 \in \mathcal{H}[\![FP]\!]$ with $\theta = \theta_1 \setminus cset$. By (C.22), we conclude $(\theta_1, \gamma) \models \phi$. By (C.23), $chan(\phi) \subseteq CHAN - cset$, and, hence, projection lemma 3.51(a) leads to $(\theta_1 \uparrow (CHAN - cset), \gamma) \models \phi$, and consequently, by definition, $(\theta_1 \setminus cset, \gamma) \models \phi$. Hence, $(\theta, \gamma) \models \phi$. □

C.4.5 Soundness of the failure hypothesis introduction rule

Assume

$$\models FP \ \mathbf{sat}\ \phi. \tag{C.24}$$

Consider any γ. Let $\theta \in \mathcal{H}[\![FP \wr \chi]\!]$. Then there exists a $\theta_0 \in \mathcal{H}[\![FP]\!]$ such that, for all γ, $(\theta_0, \theta, \gamma) \models \chi$. By (C.24), for any θ_0', $(\theta_0', \theta_0, \gamma) \models \phi$, thus also $(\theta_0, \theta_0, \gamma) \models \phi$. Let, for fresh s, $\widehat{\gamma} = (\gamma : s \mapsto \theta_0)$. Since s does not occur in ϕ, $(\theta_0, \theta_0, \widehat{\gamma}) \models \phi$. Note that $\mathcal{T}[\![s]\!](\theta_0, \theta, \widehat{\gamma}) = \theta_0$, thus $(\theta_0, \mathcal{T}[\![s]\!](\theta_0, \theta, \widehat{\gamma}), \widehat{\gamma}) \models \phi$. By substitution lemma 3.50(a) we obtain $(\theta_0, \theta, \widehat{\gamma}) \models \phi[s/h]$, or, by correspondence lemma 3.49,

$$(\theta, \widehat{\gamma}) \models \phi[s/h]. \tag{C.25}$$

Since $(\theta_0, \theta, \widehat{\gamma}) \models \chi$, we have $(\mathcal{T}[\![s]\!](\theta_0, \theta, \widehat{\gamma}), \theta, \widehat{\gamma}) \models \chi$. Applying substitution lemma 3.50(b) leads to $(\theta_0, \theta, \widehat{\gamma}) \models \chi[s/h_{old}]$. Since h_{old} does not occur in $\chi[s/h_{old}]$, correspondence lemma 3.49 leads to

$$(\theta, \widehat{\gamma}) \models \chi[s/h_{old}]. \tag{C.26}$$

From (C.25) and (C.26) we obtain $(\theta, (\gamma : s \mapsto \theta_0)) \models \phi[s/h] \wedge \chi[s/h_{old}]$, from which we may conclude that $(\theta, \gamma) \models \exists s \cdot \phi[s/h] \wedge \chi[s/h_{old}]$ □

C.5 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 that the lemma holds for *FP*:

(a) Assume $\vdash FP_1$ **sat** ϕ_1 and $\vdash FP_2$ **sat** ϕ_2, with ϕ_1 and ϕ_2 precise for FP_1 and FP_2, respectively. By the preciseness of ϕ_1 for FP_1, we have

$$chan(\phi_1) \subseteq chan(FP_1). \tag{C.27}$$

Similarly,

$$chan(\phi_2) \subseteq chan(FP_2). \tag{C.28}$$

Thus, by applying parallel composition rule 3.64, we obtain

$$\vdash FP_1 \parallel FP_2 \ \mathbf{sat}\ \phi_1 \wedge \phi_2. \tag{C.29}$$

We show that $\phi_1 \wedge \phi_2$ is precise for $FP_1 \parallel FP_2$.

(*i*) By (C.29) and soundness, we obtain $\models FP_1 \parallel FP_2$ **sat** $\phi_1 \wedge \phi_2$.

(*ii*) Let

$$chan(\theta) \subseteq chan(FP_1 \parallel FP_2), \tag{C.30}$$

and assume $(\theta, \gamma) \models \phi_1 \wedge \phi_2$. By (C.27) and projection lemma 3.51(a), $(\theta \uparrow chan(FP_1), \gamma) \models \phi_1$. Consequently, by the preciseness of ϕ_1 for FP_1, we conclude

$$\theta \uparrow chan(FP_1) \in \mathcal{H}[\![FP_1]\!]. \tag{C.31}$$

Similarly,

$$\theta \uparrow chan(FP_2) \in \mathcal{H}[\![FP_2]\!]. \tag{C.32}$$

Finally, by (C.30),

$$\theta \uparrow chan(FP_1 \parallel FP_2) = \theta. \tag{C.33}$$

Then, by (C.31) – (C.33), we conclude that $\theta \in \mathcal{H}[\![FP_1 \parallel FP_2]\!]$.

(*iii*) By (C.27) & (C.28), $chan(\phi_1) \cup chan(\phi_2) \subseteq chan(FP_1) \cup chan(FP_2)$. Hence, by definition, we have $chan(\phi_1 \wedge \phi_2) \subseteq chan(FP_1 \parallel FP_2)$.

(b) Assume

$$\vdash FP \textbf{ sat } \phi, \tag{C.34}$$

with ϕ precise for FP. Define

$$\widehat{\phi} \equiv \exists s \cdot \phi[s/h] \;\wedge\; h\!\uparrow\!(chan(FP) - cset) = s\!\uparrow\!(chan(FP) - cset)$$

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

Lemma C.1 $\models \phi \;\rightarrow\; \widehat{\phi}$

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

By Lemma C.1 and relative completeness axiom 3.74, $\vdash \phi \;\rightarrow\; \widehat{\phi}$. By (C.34) and consequence rule 3.60, we obtain $\vdash FP$ **sat** $\widehat{\phi}$. Note that, by definition, $chan(\exists s \cdot \phi[s/h]) = \emptyset$, thus $chan(\widehat{\phi}) = chan(FP) - cset$, and hence $chan(\widehat{\phi}) \cap cset = \emptyset$. Then, hiding rule 3.65 leads to

$$\vdash FP \setminus cset \textbf{ sat } \widehat{\phi}. \tag{C.35}$$

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

(*i*) By (C.35) and soundness, we have $\models FP \setminus cset$ **sat** $\widehat{\phi}$.

(*ii*) Let

$$chan(\theta) \subseteq chan(FP \setminus cset), \tag{C.36}$$

and, for some γ, $(\theta, \gamma) \models \widehat{\phi}$. There exists a $\widehat{\gamma} = (\gamma : s \mapsto \widehat{\theta}))$ with

$$(\theta, \widehat{\gamma}) \models \phi[s/h] \wedge h\!\uparrow\!(chan(FP) - cset) = s\!\uparrow\!(chan(FP) - cset). \tag{C.37}$$

Then, by substitution lemma 3.50(a), $(\widehat{\theta}, (\gamma : s \mapsto \widehat{\theta})) \models \phi$, and thus $(\widehat{\theta}, \gamma) \models \phi$. By projection lemma 3.51(a), $(\widehat{\theta}\!\uparrow\! chan(\phi), \gamma) \models \phi$. Since, by the preciseness of ϕ for FP, $chan(\phi) \subseteq chan(FP)$, we have $(\widehat{\theta}\!\uparrow\! chan(FP), \gamma) \models \phi$. Obviously, $chan(\widehat{\theta}\!\uparrow\! chan(FP)) \subseteq chan(FP)$, so, by the preciseness of ϕ for FP, we have $\widehat{\theta}\!\uparrow\! chan(FP) \in \mathcal{H}[\![FP]\!]$.

As we have, by (C.36), that $chan(\theta) \subseteq chan(FP) - cset$, and, by (C.37), that $\theta \uparrow (chan(FP) - cset) = \widehat{\theta} \uparrow (chan(FP) - cset)$, we obtain that $\theta = \widehat{\theta} \uparrow chan(FP \setminus cset)$, and thus $\theta = (\widehat{\theta} \uparrow chan(FP)) \setminus cset$. Consequently, $\theta \in \mathcal{H}[\![FP \setminus cset]\!]$.

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

(c) Assume

$$\vdash FP \textbf{ sat } \phi, \tag{C.38}$$

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

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

Then, by failure hypothesis introduction rule 3.66,

$$\vdash FP \wr \chi \textbf{ sat } \widehat{\phi}. \tag{C.39}$$

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

(*i*) By (C.39) and soundness, we have $\models FP \wr \chi \textbf{ sat } \widehat{\phi}$.

(*ii*) Let

$$chan(\theta) \subseteq chan(FP \wr \chi), \tag{C.40}$$

and assume, for some γ, $(\theta, \gamma) \models \widehat{\phi}$. Consequently, there exists a $\widehat{\theta}$ such that

$$(\theta, (\gamma : s \mapsto \widehat{\theta})) \models \phi[s/h] \wedge \chi[s/h_{old}]. \tag{C.41}$$

Then, by substitution lemma 3.50(a), $(\widehat{\theta}, (\gamma : s \mapsto \widehat{\theta})) \models \phi$, and thus, since s does not occur free in ϕ, $(\widehat{\theta}, \gamma) \models \phi$. Since, by the preciseness of ϕ for FP, $chan(\phi) \subseteq chan(FP)$, by projection lemma 3.51(a), $(\widehat{\theta} \uparrow chan(FP), \gamma) \models \phi$. Trivially, $chan(\widehat{\theta} \uparrow chan(FP)) \subseteq chan(FP)$, and hence, because of the preciseness of ϕ for FP, we obtain that

$$\widehat{\theta} \uparrow chan(FP) \in \mathcal{H}[\![FP]\!]. \tag{C.42}$$

By correspondence lemma 3.49 and substitution lemma 3.50(b), (C.41) leads to $(\widehat{\theta}, \theta, (\gamma : s \mapsto \widehat{\theta})) \models \chi$, thus, since s does not occur free in χ, $(\widehat{\theta}, \theta, \gamma) \models \chi$. Since $chan(\chi) \subseteq chan(FP)$, projection lemma 3.51(b) leads to

$$(\widehat{\theta}\uparrow chan(FP), \theta, \gamma) \models \chi. \tag{C.43}$$

Finally, by definition, (C.40) leads to

$$chan(\theta) \subseteq chan(FP). \tag{C.44}$$

Consequently, by (C.42) – (C.44), $\theta \in \mathcal{H}[\![FP \wr \chi]\!]$.

(*iii*) By definition,

$$chan(\widehat{\phi}) = chan(\phi[s/h]) \cup chan(\chi[s/h_{old}]). \tag{C.45}$$

Clearly,

$$chan(\chi[s/h_{old}]) \subseteq chan(\chi). \tag{C.46}$$

It is also obvious that $chan(\phi[s/h]) \subseteq chan(\phi)$, and, since, by the preciseness of ϕ for FP, we have that $chan(\phi) \subseteq chan(FP)$, we conclude

$$chan(\phi[s/h]) \subseteq chan(FP). \tag{C.47}$$

By (C.45) – (C.47), we have $chan(\widehat{\phi}) \subseteq chan(FP) \cup chan(\chi)$, that is, $chan(\widehat{\phi}) \subseteq chan(FP \wr \chi)$.

□

Appendix D

The Influence of the Ancient Greek Philosophers on the Modern Computer Science Community

What struck already the pre-Socratic philosophers was the contradiction between the variety and the mutability of the world of appearances (φαινόμενα) and the stability and unity required by reason (λόγος). Consequently, they tried to find a basic element or a principle of unity beneath the apparent change of the world.

The Ionian philosopher Heracleitus (Ἡράκλειτος, ca. 530–470 B.C.) claimed that nothing is definite: everything is coming into being or passing away. The nature of the world consists of a continuous process of becoming, in a constant flux: πάντα ρεῖ (everything flows). Meanwhile, Parmenides (Παρμενίδης, ca. 540–480 B.C.), the Eleatic philosopher, rejected motion and change as an illusion of the senses and claimed that the world consists in an eternal and immutable being (ὄν). Every substance must be unitary, ungenerated, indestructible, immutable, eternal and indivisible: ἔστιν εἶναι (being is).

The Parmenidean *being* and the Heracleitean *becoming* are the two opposing principles of unity. The doctrine of Parmenides has greatly influenced Archimedes (Ἀρχιμήδης, 287–212 B.C.), whose *On the Equilibrium of Planes* is considered the first important work on statics. In statics one reasons in terms of laws of equilibrium in which temporal aspects play no part. The best known exponent of the doctrine of Heracleitus is Aristotle (Ἀριστοτέλης, 384–322 B.C.). His *Physics* is seen as the first important work on dynamics. In dynamics, reasoning is governed by principles of motion and change which can not be understood without an analysis of time.

Zeno (Ζήνων, ca. 490–430 B.C.), a student of Parmenides, has become famous for the paradoxes that illustrate his teacher's doctrine. The best known paradox is that of Achilles and the tortoise. Before the runner Achilles catches up with the slower tortoise he must first cover the distance already traversed by the tortoise. While doing so, the tortoise advances a yet further distance. Extending this argumentation ad infinitum leads to the conclusion that Achilles will never catch up with the tortoise. Accepting the divisibility of magnitudes thus results in unacceptable consequences because in reality Achilles easily overtakes the tortoise.

A compositional framework allows us to reason with the specification of a process. Using a discrete notion of time, a smallest time unit has to be chosen in such a specification. Consequently, when two independently developed processes are combined it is possible that a new smallest time unit must be defined and that the respective specifications have to be modified accordingly. Therefore, in modern computer science the use of dense and even continuous time domains is popular. Characteristic of such domains is the fact that it is always possible to find a point in between two other points: time is infinitely divisible. Consider, for instance, an infinite loop where the nth execution of the body takes $(\frac{1}{2})^n$ $(n \geq 1)$. During the execution of this loop time does not progress a single unit!

To be able to reason about progress properties, finite variability [BKP86], also called non-Zeno-ness (cf. [AL92]), has to be enforced. In this thesis this is accomplished by ensuring that every action has a fixed minimal duration and that there are only finitely many different actions.

Appendix E

Definitions from Chapter 5

Definition E.1 (Observation channels of an assertion) For assertion ϕ the set $chan(\phi)$ is inductively defined as the union of the sets of channels used to restrict references to h and R in ϕ.

- $chan(0) = chan(1) = chan(i) = \emptyset$;
- $chan(iexp_1 + iexp_2) = chan(iexp_1 \times iexp_2) = chan(iexp_1) \cup chan(iexp_2)$;
- $chan(len(texp)) = chan(texp)$;

- $chan(\tau) = chan(t) = \emptyset$;
- $chan(ts(rexp)) = chan(rexp)$;
- $chan(tixp_1 + tixp_2) = chan(tixp_1) \cup chan(tixp_2)$;

- $chan(c) = \emptyset$;
- $chan(ch(rexp)) = chan(rexp)$;

- $chan(\mu) = chan(v) = \emptyset$;
- $chan(val(rexp)) = chan(rexp)$;
- $chan(f(vexp_1, \ldots, vexp_n)) = \cup_{i=1}^{n} chan(vexp_i)$;

- $chan((tixp, cexp, vexp)) = chan(tixp) \cup chan(cexp) \cup chan(vexp)$;
- $chan(texp(iexp)) = chan(texp) \cup chan(iexp)$;

- $chan(s) = \emptyset$;

- $chan(h) = CHAN$;
- $chan(\langle\rangle) = \emptyset$;
- $chan(\langle rexp \rangle) = chan(rexp)$;
- $chan(texp_1 {}^\wedge texp_2) = chan(texp_1) \cup chan(texp_2)$;
- $chan(texp \uparrow cset) = chan(texp) \cap cset$;

- $chan([tixp_1, tixp_1)) = chan(tixp_1) \cup chan(tixp_2)$;
- $chan(\{tixp\}) = chan(tixp)$;

- $chan(N) = \emptyset$;
- $chan(R) = CHAN$;
- $chan(\emptyset) = \emptyset$;
- $chan(cset \times inxp) = chan(inxp)$;
- $chan(rfxp_1 \cup rfxp_2) = chan(rfxp_1) \cup chan(rfxp_2)$;
- $chan(rfxp \uparrow cset) = chan(rfxp) \cap cset$;

- $chan(iexp_1 = iexp_2) = chan(iexp_1 < iexp_2) = chan(iexp_1) \cup chan(iexp_2)$;
- $chan(tixp_1 = tixp_2) = chan(tixp_1 < tixp_2) = chan(tixp_1) \cup chan(tixp_2)$;
- $chan(cexp_1 = cexp_2) = chan(cexp_1) \cup chan(cexp_2)$;
- $chan(vexp_1 = vexp_2) = chan(vexp_1 < vexp_2) = chan(vexp_1) \cup chan(vexp_2)$;
- $chan(texp_1 = texp_2) = chan(texp_1) \cup chan(texp_2)$;
- $chan(rfxp_1 = rfxp_2) = chan(rfxp_1) \cup chan(rfxp_2)$;
- $chan(\phi_1 \wedge \phi_2) = chan(\phi_1) \cup chan(\phi_2)$;
- $chan(\neg\phi) = chan(\exists i \cdot \phi) = chan(\exists t \cdot \phi) = chan(\exists v \cdot \phi) =$ $chan(\exists s \cdot \phi) = chan(\exists N \cdot \phi) = chan(\phi)$. $\diamond$

Definition E.2 (Meaning of assertions) We define the value of an integer expression *iexp* in the trace θ, refusal $\mathfrak{R}$, and an environment γ, denoted by $\mathcal{I}[\![iexp]\!](\theta, \mathfrak{R}, \gamma)$, as yielding a value in $\mathbb{N} \cup \{\dagger\}$; the value of a time expression *tixp* in the trace θ, refusal $\mathfrak{R}$, and an environment γ, denoted by $\mathcal{TI}[\![tixp]\!](\theta, \mathfrak{R}, \gamma)$, as 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

$\mathcal{C}[\![cexp]\!](\theta, \mathfrak{R}, \gamma)$, as 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 $\mathcal{V}[\![vexp]\!](\theta, \mathfrak{R}, \gamma)$, as 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 $\mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma)$, as yielding a value in $(TIME \times CHAN \times VAL) \cup \{\dagger\}$; the value of a trace expression $texp$ for trace θ, refusal $\mathfrak{R}$, and an environment γ, denoted by $\mathcal{T}[\![texp]\!](\theta, \mathfrak{R}, \gamma)$, as 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 $\mathcal{IN}[\![inxp]\!](\theta, \mathfrak{R}, \gamma)$, as 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 $\mathcal{RF}[\![rfxp]\!](\theta, \mathfrak{R}, \gamma)$, as yielding a value in $REF \cup \{\dagger\}$,

- $\mathcal{I}[\![0]\!](\theta, \mathfrak{R}, \gamma) = 0$;
- $\mathcal{I}[\![1]\!](\theta, \mathfrak{R}, \gamma) = 1$;
- $\mathcal{I}[\![i]\!](\theta, \mathfrak{R}, \gamma) = \gamma(i)$;
- $\mathcal{I}[\![iexp_1 + iexp_2]\!](\theta, \mathfrak{R}, \gamma) =$
$$\begin{cases} \dagger & \text{if } \mathcal{I}[\![iexp_1]\!](\theta, \mathfrak{R}, \gamma) = \dagger \text{ or} \\ & \mathcal{I}[\![iexp_2]\!](\theta, \mathfrak{R}, \gamma) = \dagger, \\ \mathcal{I}[\![iexp_1]\!](\theta, \mathfrak{R}, \gamma) + \mathcal{I}[\![iexp_2]\!](\theta, \mathfrak{R}, \gamma) & \text{otherwise;} \end{cases}$$
- $\mathcal{I}[\![iexp_1 \times iexp_2]\!](\theta, \mathfrak{R}, \gamma) =$
$$\begin{cases} \dagger & \text{if } \mathcal{I}[\![iexp_1]\!](\theta, \mathfrak{R}, \gamma) = \dagger \text{ or} \\ & \mathcal{I}[\![iexp_2]\!](\theta, \mathfrak{R}, \gamma) = \dagger, \\ \mathcal{I}[\![iexp_1]\!](\theta, \mathfrak{R}, \gamma) \times \mathcal{I}[\![iexp_2]\!](\theta, \mathfrak{R}, \gamma) & \text{otherwise;} \end{cases}$$
- $\mathcal{I}[\![len(texp)]\!](\theta, \mathfrak{R}, \gamma) = \begin{cases} \dagger & \text{if } \mathcal{T}[\![texp]\!](\theta, \mathfrak{R}, \gamma) = \dagger, \\ len(\mathcal{T}[\![texp]\!](\theta, \mathfrak{R}, \gamma)) & \text{otherwise;} \end{cases}$
- $\mathcal{TI}[\![\tau]\!](\theta, \mathfrak{R}, \gamma) = \tau$;
- $\mathcal{TI}[\![t]\!](\theta, \mathfrak{R}, \gamma) = \gamma(t)$;
- $\mathcal{TI}[\![ts(rexp)]\!](\theta, \mathfrak{R}, \gamma) = \begin{cases} \dagger & \text{if } \mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma) = \dagger, \\ \tau & \text{if there exist } c \text{ and } \mu \text{ such that} \\ & \mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma) = (\tau, c, \mu); \end{cases}$
- $\mathcal{TI}[\![iexp_1 + iexp_2]\!](\theta, \mathfrak{R}, \gamma) =$
$$\begin{cases} \dagger & \text{if } \mathcal{I}[\![iexp_1]\!](\theta, \mathfrak{R}, \gamma) = \dagger \text{ or} \\ & \mathcal{I}[\![iexp_2]\!](\theta, \mathfrak{R}, \gamma) = \dagger, \\ \mathcal{I}[\![iexp_1]\!](\theta, \mathfrak{R}, \gamma) + \mathcal{I}[\![iexp_2]\!](\theta, \mathfrak{R}, \gamma) & \text{otherwise;} \end{cases}$$
- $\mathcal{C}[\![c]\!](\theta, \mathfrak{R}, \gamma) = c$;
- $\mathcal{C}[\![ch(rexp)]\!](\theta, \mathfrak{R}, \gamma) = \begin{cases} \dagger & \text{if } \mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma) = \dagger, \\ c & \text{if there exist } \tau \text{ and } \mu \text{ such that} \\ & \mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma) = (\tau, c, \mu); \end{cases}$

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

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

- $\mathcal{V}[\![val(rexp)]\!](\theta, \mathfrak{R}, \gamma) = \begin{cases} \dagger & \text{iff } \mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma) = \dagger, \\ \mu & \text{if there exist } \tau \text{ and } c \text{ such that} \\ & \mathcal{R}[\![rexp]\!](\theta, \mathfrak{R}, \gamma) = (\tau, c, \mu); \end{cases}$

- $\mathcal{V}[\![f(vexp_1, \ldots, vexp_n)]\!](\theta, \mathfrak{R}, \gamma) =$
$$\begin{cases} \dagger & \text{if } \mathcal{V}[\![vexp_1]\!](\theta, \mathfrak{R}, \gamma) = \dagger, \\ & \text{or } \ldots, \text{ or} \\ & \mathcal{V}[\![vexp_n]\!](\theta, \mathfrak{R}, \gamma) = \dagger, \\ f(\mathcal{V}[\![vexp_1]\!](\theta, \mathfrak{R}, \gamma), \ldots, \mathcal{V}[\![vexp_n]\!](\theta, \mathfrak{R}, \gamma)) & \text{otherwise}; \end{cases}$$

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

- $\mathcal{R}[\![texp(iexp)]\!](\theta, \mathfrak{R}, \gamma) =$
$$\begin{cases} (\tau, c, \mu) & \text{if there exist } \theta_1 \text{ and } \theta_2 \text{ such that} \\ & len(\theta_1) = \mathcal{I}[\![iexp]\!](\theta, \mathfrak{R}, \gamma) - 1 \\ & \text{and } \mathcal{T}[\![texp]\!](\theta, \mathfrak{R}, \gamma) = \theta_1{}^\wedge(\tau, c, \mu)^\wedge\theta_2, \\ \dagger & \text{otherwise}; \end{cases}$$

- $\mathcal{T}[\![s]\!](\theta, \mathfrak{R}, \gamma) = \gamma(s)$;

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

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

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

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

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

- $\mathcal{IN}[\![[iexp_1, iexp_2)]\!](\theta, \mathfrak{R}, \gamma) =$
$$\begin{cases} \dagger & \text{if } \mathcal{I}[\![iexp_1]\!](\theta,\mathfrak{R},\gamma) = \dagger \text{ or} \\ & \mathcal{I}[\![iexp_2]\!](\theta,\mathfrak{R},\gamma) = \dagger, \\ [\mathcal{I}[\![iexp_1]\!](\theta,\mathfrak{R},\gamma), \mathcal{I}[\![iexp_2]\!](\theta,\mathfrak{R},\gamma)) & \text{otherwise;} \end{cases}$$

- $\mathcal{IN}[\![\{iexp\}]\!](\theta,\mathfrak{R},\gamma) = \begin{cases} \dagger & \text{if } \mathcal{I}[\![iexp]\!](\theta,\mathfrak{R},\gamma) = \dagger, \\ \{\mathcal{I}[\![iexp]\!](\theta,\mathfrak{R},\gamma)\} & \text{otherwise;} \end{cases}$

- $\mathcal{RF}[\![N]\!](\theta,\mathfrak{R},\gamma) = \gamma(N)$;

- $\mathcal{RF}[\![R]\!](\theta,\mathfrak{R},\gamma) = \mathfrak{R}$;

- $\mathcal{RF}[\![\emptyset]\!](\theta,\mathfrak{R},\gamma) = \emptyset$;

- $\mathcal{RF}[\![cset \times inxp]\!](\theta,\mathfrak{R},\gamma) =$
$$\begin{cases} \dagger & \text{if } \mathcal{IN}[\![inxp]\!](\theta,\mathfrak{R},\gamma) = \dagger, \\ cset \times \mathcal{IN}[\![inxp]\!](\theta,\mathfrak{R},\gamma) & \text{otherwise;} \end{cases}$$

- $\mathcal{RF}[\![rfxp_1 \cup rfxp_2]\!](\theta,\mathfrak{R},\gamma) =$
$$\begin{cases} \dagger & \text{if } \mathcal{RF}[\![rfxp_1]\!](\theta,\mathfrak{R},\gamma) = \dagger \\ & \text{or } \mathcal{RF}[\![rfxp_2]\!](\theta,\mathfrak{R},\gamma) = \dagger, \\ \mathcal{RF}[\![rfxp_1]\!](\theta,\mathfrak{R},\gamma) \cup \mathcal{RF}[\![rfxp_2]\!](\theta,\mathfrak{R},\gamma) & \text{otherwise;} \end{cases}$$

- $\mathcal{RF}[\![rfxp \uparrow cset]\!](\theta,\mathfrak{R},\gamma) =$
$$\begin{cases} \dagger & \text{if } \mathcal{RF}[\![rfxp]\!](\theta,\mathfrak{R},\gamma) = \dagger, \\ \mathcal{RF}[\![rfxp]\!](\theta,\mathfrak{R},\gamma) \uparrow cset & \text{otherwise.} \end{cases}$$ ◇

Definition E.3 (Satisfaction) We inductively define when an assertion ϕ holds for trace θ, refusal $\mathfrak{R}$, and an environment γ, denoted by $(\theta,\mathfrak{R},\gamma) \models \phi$:

- $(\theta,\mathfrak{R},\gamma) \models iexp_1 = iexp_2$ if, and only if,
$\mathcal{I}[\![iexp_1]\!](\theta,\mathfrak{R},\gamma) = \mathcal{I}[\![iexp_2]\!](\theta,\mathfrak{R},\gamma)$ and $\mathcal{I}[\![iexp_1]\!](\theta,\mathfrak{R},\gamma) \neq \dagger$;

- $(\theta,\mathfrak{R},\gamma) \models iexp_1 < iexp_2$ iff $\mathcal{I}[\![iexp_1]\!](\theta,\mathfrak{R},\gamma) < \mathcal{I}[\![iexp_2]\!](\theta,\mathfrak{R},\gamma)$,
$\mathcal{I}[\![iexp_1]\!](\theta,\mathfrak{R},\gamma) \neq \dagger$ and $\mathcal{I}[\![iexp_2]\!](\theta,\mathfrak{R},\gamma) \neq \dagger$;

- $(\theta,\mathfrak{R},\gamma) \models tixp_1 = tixp_2$ iff
$\mathcal{TI}[\![tixp_1]\!](\theta,\mathfrak{R},\gamma) = \mathcal{TI}[\![tixp_2]\!](\theta,\mathfrak{R},\gamma)$ and $\mathcal{TI}[\![tixp_1]\!](\theta,\mathfrak{R},\gamma) \neq \dagger$;

- $(\theta,\mathfrak{R},\gamma) \models tixp_1 < tixp_2$ iff $\mathcal{TI}[\![tixp_1]\!](\theta,\mathfrak{R},\gamma) = \mathcal{TI}[\![tixp_2]\!](\theta,\mathfrak{R},\gamma)$,
$\mathcal{TI}[\![tixp_1]\!](\theta,\mathfrak{R},\gamma) \neq \dagger$, and $\mathcal{TI}[\![tixp_2]\!](\theta,\mathfrak{R},\gamma) \neq \dagger$;

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

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

- $(\theta, \mathfrak{R}, \gamma) \models vexp_1 < vexp_2$ iff $\mathcal{V}[\![vexp_1]\!](\theta, \mathfrak{R}, \gamma) < \mathcal{V}[\![vexp_2]\!](\theta, \mathfrak{R}, \gamma)$, $\mathcal{V}[\![vexp_1]\!](\theta, \mathfrak{R}, \gamma) \neq \dagger$, and $\mathcal{V}[\![vexp_2]\!](\theta, \mathfrak{R}, \gamma) \neq \dagger$;
- $(\theta, \mathfrak{R}, \gamma) \models texp_1 = texp_2$ 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 $(\theta, \mathfrak{R}, \gamma) \models \phi$;
- $(\theta, \mathfrak{R}, \gamma) \models \exists i \cdot \phi$ iff there exists an integer j for which it is the case that $(\theta, \mathfrak{R}, (\gamma : i \mapsto j)) \models \phi$;
- $(\theta, \mathfrak{R}, \gamma) \models \exists t \cdot \phi$ iff there exists an instant τ such that it is the case that $(\theta, \mathfrak{R}, (\gamma : t \mapsto \tau)) \models \phi$;
- $(\theta, \mathfrak{R}, \gamma) \models \exists v \cdot \phi$ iff there is 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 some refusal set $\widehat{\mathfrak{R}}$ for which it is the case that $(\theta, \mathfrak{R}, (\gamma : N \mapsto \widehat{\mathfrak{R}})) \models \phi$. $\diamond$

Appendix F

Proofs from Chapter 5

F.1 Proof of the soundness theorem

F.1.1 Soundness of the consequence and conjunction rules

Trivial.

F.1.2 Soundness of the invariance rule

If $(\theta, \mathfrak{R}) \in \mathcal{O}[\![FP]\!]$ then, by definition, $\theta \uparrow chan(FP) = \theta$ and $\mathfrak{R} \uparrow chan(FP) = \mathfrak{R}$. Thus, $cset \cap chan(FP) = \emptyset$ implies $\theta \uparrow cset = \langle\rangle$ and $\mathfrak{R} \uparrow cset = \emptyset$.

F.1.3 Soundness of the parallel composition rule

Assume

$$\models FP_1 \textbf{ sat } \phi_1 \ , \quad \models FP_2 \textbf{ sat } \phi_2. \tag{F.1}$$

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

$$(\theta \uparrow chan(FP_i), \mathfrak{R}_i) \in \mathcal{O}[\![FP_i]\!], \tag{F.2}$$

and

$$\mathfrak{R} = \mathfrak{R}_1 \cup \mathfrak{R}_2.$$

By the definition of the semantics, $\mathfrak{R}_i \setminus chan(FP_i) = \emptyset$. Hence,

$$\mathfrak{R} = \mathfrak{R}_1 \uparrow chan(FP_1) \ \cup \ \mathfrak{R}_2 \uparrow chan(FP_2). \tag{F.3}$$

Let, for fresh N_1 and N_2, $\widehat{\gamma} = (\gamma : (N_1, N_2) \mapsto (\mathfrak{R}_1, \mathfrak{R}_2))$. By (F.3), we obtain $(\theta, \mathfrak{R}, \widehat{\gamma}) \models R = N_1 \uparrow chan(FP_1) \ \cup \ N_2 \uparrow chan(FP_2)$, or

$$(\theta, \mathfrak{R}, \gamma) \models \exists N_1, N_2 \cdot R = N_1 \!\uparrow\! chan(FP_1) \;\cup\; N_2 \!\uparrow\! chan(FP_2). \tag{F.4}$$

By (F.1) and (F.2), we have, for all γ', $(\theta \!\uparrow\! chan(FP_i), \mathfrak{R}_i, \gamma') \models \phi_i$, $i = 1, 2$. As a result, $(\theta \!\uparrow\! chan(FP_i), \mathfrak{R}_i, \widehat{\gamma}) \models \phi_i$. Observe that $\mathcal{RF}[\![N_i]\!](\theta, \mathfrak{R}, \widehat{\gamma}) = \mathfrak{R}_i$ and that $\mathcal{T}[\![h \!\uparrow\! chan(FP_i)]\!](\theta, \mathfrak{R}, \widehat{\gamma}) = \theta \!\uparrow\! chan(FP_i)$. Consequently, we have $(\mathcal{T}[\![h \!\uparrow\! chan(FP_i)]\!](\theta, \mathfrak{R}, \widehat{\gamma})\ ,\ \mathcal{RF}[\![N_i]\!](\theta, \mathfrak{R}, \widehat{\gamma})\ ,\ \widehat{\gamma}) \models \phi_i$. Then, by substitution lemma 5.29(b) and (d), we obtain $(\theta, \mathfrak{R}, \widehat{\gamma}) \models \phi_i[\ (h \!\uparrow\! chan(FP_i))/h\ ,\ N_i/R\ ]$, from which we conclude

$$(\theta, \mathfrak{R}, \gamma) \models \exists N_1, N_2 \cdot \phi_i[\ (h \!\uparrow\! chan(FP_i))/h\ ,\ N_i/R\ ]. \tag{F.5}$$

By (F.4) and (F.5) we conclude that parallel composition rule 5.39 is sound.

F.1.4 Soundness of the hiding rule

Assume that

$$\models FP \ \mathbf{sat}\ ASAP(R, cset) \rightarrow \phi(h \backslash cset, R \backslash cset). \tag{F.6}$$

Consider any γ. Let $(\theta, \mathfrak{R}) \in \mathcal{O}[\![FP \backslash cset]\!]$. Then, by the definition of the semantics there exists a

$$(\widehat{\theta}, \widehat{\mathfrak{R}}) \in \mathcal{O}[\![FP]\!] \tag{F.7}$$

for which

$$ASAP(\widehat{\mathfrak{R}}, cset) \tag{F.8}$$

such that

$$\theta = \widehat{\theta} \backslash cset \tag{F.9}$$

and

$$\mathfrak{R} = \widehat{\mathfrak{R}} \backslash cset. \tag{F.10}$$

By (F.7) & (F.6), for all γ, $(\widehat{\theta}, \widehat{\mathfrak{R}}, \gamma) \models ASAP(R, cset) \rightarrow \phi(h \backslash cset, R \backslash cset)$. Then, by (F.8), $(\widehat{\theta}, \widehat{\mathfrak{R}}, \gamma) \models \phi(h \backslash cset, R \backslash cset)$. By substitution lemma 5.29(b) and (d), we obtain $(\widehat{\theta} \backslash cset, \widehat{\mathfrak{R}} \backslash cset, \gamma) \models \phi$. Hence, by (F.9) and (F.10), $(\theta, \mathfrak{R}, \gamma) \models \phi$, from which we conclude that hiding rule 5.40 is sound.

F.1.5 Soundness of the failure hypothesis introduction rule

Assume that

$$\models FP \ \mathbf{sat}\ \phi. \tag{F.11}$$

Consider any γ. Let $(\theta, \mathfrak{R}) \in \mathcal{O}[\![FP \wr \chi]\!]$. By the definition of the semantics, there exists a $(\theta_0, \mathfrak{R}_0) \in \mathcal{O}[\![FP]\!]$, such that, for all γ,

$$(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \gamma) \models \chi. \tag{F.12}$$

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 (F.11), that, for all γ, $(\theta_0, \mathfrak{R}_0, \gamma) \models \phi$. Consequently, $(\theta_0, \mathfrak{R}_0, \widehat{\gamma}) \models \phi$. 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 conclude $(\mathcal{T}[\![s]\!](\theta, \mathfrak{R}, \widehat{\gamma}), \mathcal{RF}[\![N]\!](\theta, \mathfrak{R}, \widehat{\gamma}), \widehat{\gamma}) \models \phi$. Hence, by substitution lemma 5.29(b) and (d),

$$(\theta, \mathfrak{R}, \widehat{\gamma}) \models \phi[\ s/h\ ,\ N/R\ ]. \tag{F.13}$$

By (F.12), $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \widehat{\gamma}) \models \chi$ or $(\mathcal{T}[\![s]\!](\theta, \mathfrak{R}, \widehat{\gamma}), \theta, \mathcal{RF}[\![N]\!](\theta, \mathfrak{R}, \widehat{\gamma}), \mathfrak{R}, \widehat{\gamma}) \models \chi$. By substitution lemma 5.29(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

$$(\theta, \mathfrak{R}, \widehat{\gamma}) \models \chi[s/h_{old}\ ,\ N/R_{old}]. \tag{F.14}$$

By (F.13) and (F.14), $(\theta, \mathfrak{R}, \widehat{\gamma}) \models \phi[\ s/h\ ,\ N/R\ ] \wedge \chi[\ s/h_{old}\ ,\ N/R_{old}\ ]$, 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, failure hypothesis introduction rule 5.42 is sound.

F.2 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 rule 5.39, we obtain

$$\begin{aligned}\vdash FP_1 \parallel FP_2\ \mathbf{sat}\ \exists\ N_1, N_2 \cdot\quad & \\ & R = N_1 \uparrow chan(FP_1) \cup N_2 \uparrow chan(FP_2) \\ & \wedge\ \phi_1(h \uparrow chan(FP_1), N_1) \\ & \wedge\ \phi_2(h \uparrow chan(FP_2), N_2).\end{aligned} \tag{F.15}$$

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

i) By (F.15) and soundness, we obtain

$$\models FP_1 \parallel FP_2 \textbf{ sat } \exists N_1, N_2 \cdot \quad \begin{aligned}[t] & R = N_1 \!\uparrow\! chan(FP_1) \cup N_2 \!\uparrow\! chan(FP_2) \\ \wedge\ & \phi_1(h \!\uparrow\! chan(FP_1), N_1) \\ \wedge\ & \phi_2(h \!\uparrow\! chan(FP_2), N_2). \end{aligned}$$

ii) Let

$$chan(\theta) \subseteq chan(FP_1 \| FP_2), \tag{F.16}$$

and

$$\mathfrak{R} \!\uparrow\! chan(FP_1 \| FP_2) = \mathfrak{R}. \tag{F.17}$$

Assume, for some γ,

$$(\theta, \mathfrak{R}, \gamma) \models \exists N_1, N_2 \cdot \quad \begin{aligned}[t] & R = N_1 \!\uparrow\! chan(FP_1) \cup N_2 \!\uparrow\! chan(FP_2) \\ \wedge\ & \phi_1(h \!\uparrow\! chan(FP_1), N_1) \\ \wedge\ & \phi_2(h \!\uparrow\! chan(FP_2), N_2). \end{aligned}$$

In other words, there exist some $\mathfrak{R}_1$ and some $\mathfrak{R}_2$ such that, using $\widehat{\gamma} = (\gamma : (N_1, N_2) \mapsto (\mathfrak{R}_1, \mathfrak{R}_2))$,

$$(\theta, \mathfrak{R}, \widehat{\gamma}) \models \quad \begin{aligned}[t] & R = N_1 \!\uparrow\! chan(FP_1) \cup N_2 \!\uparrow\! chan(FP_2) \\ \wedge\ & \phi_1(h \!\uparrow\! chan(FP_1), N_1) \\ \wedge\ & \phi_2(h \!\uparrow\! chan(FP_2), N_2). \end{aligned} \tag{F.18}$$

Then, by substitution lemma 5.29(b) and (d),

$$(\theta \!\uparrow\! chan(FP_1), \mathfrak{R}_1, (\gamma : (N_1, N_2) \mapsto (\mathfrak{R}_1, \mathfrak{R}_2))) \models \phi_1,$$

or, since N_1 and N_2 do not occur free in ϕ_1,

$$(\theta \!\uparrow\! chan(FP_1), \mathfrak{R}_1, \gamma) \models \phi_1.$$

By the preciseness of ϕ_1 for FP_1,

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

By applying substitution lemma 5.29(b) and (d), and using the fact that $(\theta \!\uparrow\! chan(FP_1)) \!\uparrow\! chan(FP_1) = \theta \!\uparrow\! chan(FP_1)$, we obtain

$$(\theta \!\uparrow\! chan(FP_1), \mathfrak{R}_1 \!\uparrow\! chan(FP_1), \gamma) \models \phi_1. \tag{F.19}$$

Trivially,

$$chan(\theta\uparrow chan(FP_1)) \subseteq chan(FP_1), \tag{F.20}$$

and

$$chan(\mathfrak{R}_1\uparrow chan(FP_1)) \subseteq chan(FP_1). \tag{F.21}$$

By (F.20), (F.21), and (F.19), the preciseness of ϕ_1 for FP_1 leads to

$$(\theta\uparrow chan(FP_1), \mathfrak{R}_1\uparrow chan(FP_1)) \in \mathcal{O}[\![FP_1]\!]. \tag{F.22}$$

Similarly,

$$(\theta\uparrow chan(FP_2), \mathfrak{R}_2\uparrow chan(FP_2)) \in \mathcal{O}[\![FP_2]\!]. \tag{F.23}$$

By (F.16), trivially,

$$\theta\uparrow chan(FP_1\|FP_2) = \theta. \tag{F.24}$$

By (F.18),

$$\mathfrak{R} \;=\; (\mathfrak{R}_1\uparrow chan(FP_1)) \;\cup\; (\mathfrak{R}_2\uparrow chan(FP_2)). \tag{F.25}$$

By (F.22) – (F.25), $(\theta, \mathfrak{R}) \in \mathcal{O}[\![FP_1\|FP_2]\!]$.

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

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

which is, obviously, equivalent to

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

b) Assume

$$\vdash FP \textbf{ sat } \phi, \tag{F.26}$$

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

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 F.1 $\models \phi \;\rightarrow\; (ASAP(R, cset) \;\rightarrow\; \widehat{\phi}(h \backslash cset, R \backslash cset))$ $\circ$

By Lemma F.1 and relative completeness axiom 5.45,

$$\vdash \phi \;\rightarrow\; (ASAP(R, cset) \;\rightarrow\; \widehat{\phi}(h \backslash cset, R \backslash cset)).$$

Hence, by (F.26) and consequence rule 5.36,

$$\vdash FP \textbf{ sat } ASAP(R, cset) \;\rightarrow\; \widehat{\phi}(h \backslash cset, R \backslash cset).$$

Then, by hiding rule 5.40,

$$\vdash FP \backslash cset \textbf{ sat } \widehat{\phi}. \tag{F.27}$$

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

i) By (F.27) and soundness $\models FP \backslash cset \textbf{ sat } \widehat{\phi}$.

ii) Let

$$chan(\theta) \subseteq chan(FP \backslash cset), \tag{F.28}$$

$$\mathfrak{R} \uparrow chan(FP \backslash cset) = \mathfrak{R}, \tag{F.29}$$

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

$$(\theta, \mathfrak{R}, (\gamma : (s, N) \mapsto (\widehat{\theta}, \widehat{\mathfrak{R}}))) \models \begin{array}{ll} & \phi(s, N) \\ \wedge & ASAP(N, cset) \\ \wedge & h = s \backslash cset \\ \wedge & R = N \backslash cset. \end{array} \tag{F.30}$$

Then,

$$ASAP(\widehat{\mathfrak{R}}, cset), \tag{F.31}$$

$$\theta = \widehat{\theta} \backslash cset, \tag{F.32}$$

and

$$\mathfrak{R} = \widehat{\mathfrak{R}} \backslash cset. \tag{F.33}$$

By (F.30),

$$(\widehat{\theta}, \widehat{\mathfrak{R}}, \gamma) \models \phi. \tag{F.34}$$

By (F.28), $chan(\theta) \subseteq chan(FP \setminus cset)$. Consequently, by (F.32),

$$chan(\widehat{\theta}) \subseteq chan(FP). \tag{F.35}$$

By (F.29) and (F.33), and the fact that $cset \subseteq chan(FP)$, we obtain that

$$\widehat{\mathfrak{R}} \uparrow chan(FP) = \widehat{\mathfrak{R}}. \tag{F.36}$$

By (F.35), (F.36), and (F.34), and the preciseness of ϕ for FP, we have that

$$(\widehat{\theta}, \widehat{\mathfrak{R}}) \in \mathcal{O}[\![FP]\!]. \tag{F.37}$$

By (F.37) and (F.31) – (F.33), $(\theta, \mathfrak{R}) \in \mathcal{O}[\![FP \setminus cset]\!]$.

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

$$\begin{array}{rl} (\theta, \mathfrak{R}, (\gamma : (s, N) \mapsto (\widehat{\theta}, \widehat{\mathfrak{R}}))) \models & \phi(s, N) \\ & \wedge\ ASAP(N, cset) \\ & \wedge\ h = s \setminus cset \\ & \wedge\ R = N \setminus cset \end{array} \tag{F.38}$$

By the preciseness of ϕ for FP,

$$\phi(s, N) \ \rightarrow\ \phi(s \uparrow chan(FP), N \uparrow chan(FP)). \tag{F.39}$$

It is obvious that

$$ASAP(N, cset) \ \rightarrow\ ASAP(N \setminus chan(FP), cset). \tag{F.40}$$

Note that

$$h = s \setminus cset \ \rightarrow\ h \uparrow chan(FP \setminus cset) = (s \uparrow chan(FP)) \setminus cset. \tag{F.41}$$

By (F.38), $R = N \setminus cset$, that is,

$$R \uparrow chan(FP \setminus cset) = (N \uparrow chan(FP)) \setminus cset. \tag{F.42}$$

By (F.38) – (F.42), we obtain, using $\widehat{\gamma} = (\gamma : (s, N) \mapsto (\widehat{\theta}, \widehat{\mathfrak{R}}))$,

$$\begin{array}{rl} (\theta, \mathfrak{R}, \widehat{\gamma}) \models & \phi(s \uparrow chan(FP), N \uparrow chan(FP)) \\ & \wedge\ ASAP(N \uparrow chan(FP), cset) \\ & \wedge\ h \uparrow chan(FP \setminus cset) = (s \uparrow chan(FP)) \setminus cset \\ & \wedge\ R \uparrow chan(FP \setminus cset) = (N \uparrow chan(FP)) \setminus cset, \end{array}$$

from which we may conclude

$$(\theta, \mathfrak{R}, \gamma) \models \widehat{\phi}(h \uparrow chan(FP \setminus cset), R \uparrow chan(FP \setminus cset)).$$

c) Assume

$$\vdash FP \textbf{ sat } \phi, \tag{F.43}$$

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 (F.43) and failure hypothesis introduction rule 5.42,

$$\vdash FP \wr \chi \textbf{ sat } \widehat{\phi}. \tag{F.44}$$

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

i) By (F.44) and soundness, we have $\models FP \wr \chi \textbf{ sat } \widehat{\phi}$.

ii) Let

$$chan(\theta) \subseteq chan(FP \wr \chi), \tag{F.45}$$

$$\mathfrak{R} \uparrow chan(FP \wr \chi) = \mathfrak{R}, \tag{F.46}$$

and assume, for some γ, $(\theta, \mathfrak{R}, \gamma) \models \widehat{\phi}$. Hence, there exist $\widehat{\theta}$ and $\widehat{\mathfrak{R}}$ such that

$$(\theta, \mathfrak{R}, (\gamma : (s, N) \mapsto (\widehat{\theta}, \widehat{\mathfrak{R}}))) \models \begin{array}{l} \phi[\, s/h, N/R\,] \\ \wedge\ \chi[\, s/h_{old}, N/R_{old}\,]. \end{array} \tag{F.47}$$

By substitution lemma 5.29(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 5.29(b) and (d),

$$(\widehat{\theta} \uparrow chan(FP), \widehat{\mathfrak{R}} \uparrow chan(FP), \gamma) \models \phi. \tag{F.48}$$

Trivially,

$$chan(\widehat{\theta} \uparrow chan(FP)) \subseteq chan(FP), \tag{F.49}$$

and

$$chan(\widehat{\mathfrak{R}}\uparrow chan(FP)) \subseteq chan(FP). \tag{F.50}$$

By results (F.49), (F.50), (F.48), and the fact that ϕ is precise for FP, we may conclude that

$$(\widehat{\theta}\uparrow chan(FP), \widehat{\mathfrak{R}}\uparrow chan(FP)) \in \mathcal{O}[\![FP]\!]. \tag{F.51}$$

By (F.47) and correspondence lemma 5.28, 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 5.29(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 5.29(a) and (c)

$$(\widehat{\theta}\uparrow chan(FP), \theta, \widehat{\mathfrak{R}}\uparrow chan(FP), \mathfrak{R}, \gamma) \models \chi. \tag{F.52}$$

By (F.51), (F.52), (F.49), and (F.50), $(\theta, \mathfrak{R}) \in \mathcal{O}[\![FP \wr \chi]\!]$.

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 $chan(FP \wr \chi) = chan(FP)$.

Appendix G

Definitions from Chapter 6

Definition G.1 (Observation channels of an assertion) For assertion φ, the set $chan(\varphi)$ of channels that might affect the validity of φ is as defined in Definition E.1 with the extra clauses:

- $chan(\pi) = chan(p) = \emptyset$;
- $chan(\Pi(tixp)) = chan(tixp)$;
- $chan((tixp_1, prxp, tixp_2)) = chan(tixp_1) \cup chan(tixp_2)$;
- $chan(K) = chan(O) = \emptyset$;
- $chan(\{blxp\}) = chan(blxp)$;
- $chan(ohxp_1 \cup ohxp_2) = chan(ohxp_1) \cup chan(ohxp_2)$;
- $chan((tixp_1, prxp, tixp_2)) = chan(tixp_1) \cup chan(tixp_2)$;
- $chan(L) = chan(Q) = \emptyset$;
- $chan(\{quxp\}) = chan(quxp)$;
- $chan(rhxp_1 \cup rhxp_2) = chan(rhxp_1) \cup chan(rhxp_2)$. $\diamond$

Definition G.2 (Meaning of assertions) The assertion φ is interpreted with respect to a 5-tuple $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma)$. Trace θ gives h its value, refusal set $\mathfrak{R}$ gives R its value, $\mathfrak{O}$ and $\mathfrak{Q}$ do so for O, respectively Q, and environment γ interprets the logical variables of $IVAR \cup VVAR \cup TVAR \cup RVAR \cup PRVAR \cup OVAR \cup QVAR$. The value of an expression that already appears in Table 5.2 is easily obtained from the corresponding definition in Section 5.4, e.g., the value of a time expression $tixp$ in the trace θ, refusal $\mathfrak{R}$, occupation history $\mathfrak{O}$, request history $\mathfrak{Q}$ and an environment γ, follows from $\mathcal{TI}[\![tixp]\!](\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) = \mathcal{TI}[\![tixp]\!](\theta, \mathfrak{R}, \gamma)$.

The value of a priority expression *prxp* in the trace θ, refusal $\mathfrak{R}$, occupation history $\mathfrak{O}$, request history $\mathfrak{Q}$ and an environment γ, denoted $\mathcal{PR}\llbracket prxp \rrbracket(\theta, \mathfrak{R}, \gamma)$, is defined as yielding a value in $\mathbb{N} \cup \{\dagger\}$; the value of a block expression *blxp* in the trace θ, refusal $\mathfrak{R}$, occupation history $\mathfrak{O}$, request history $\mathfrak{Q}$, and an environment γ, denoted by $\mathcal{B}\llbracket blxp \rrbracket(\theta, \mathfrak{R}, \gamma)$, as yielding a value in $(\mathit{TIME} \times \mathbb{N} \times \mathit{TIME}) \cup \{\dagger\}$; the value of an occupation expression *ohxp* in the trace θ, refusal $\mathfrak{R}$, occupation history $\mathfrak{O}$, request history $\mathfrak{Q}$, and an environment γ, denoted by $\mathcal{OC}\llbracket ohxp \rrbracket(\theta, \mathfrak{R}, \gamma)$, as yielding a value in $\mathit{OCC} \cup \{\dagger\}$; the value of a queue expression *quxp* in the trace θ, refusal $\mathfrak{R}$, occupation history $\mathfrak{O}$, request history $\mathfrak{Q}$, and an environment γ, denoted by $\mathcal{Q}\llbracket quxp \rrbracket(\theta, \mathfrak{R}, \gamma)$, as yielding a value in $(\mathit{TIME} \times \mathbb{N} \times \mathit{TIME}) \cup \{\dagger\}$, and the value of a request expression *rhxp* in the trace θ, refusal $\mathfrak{R}$, occupation history $\mathfrak{O}$, request history $\mathfrak{Q}$, and an environment γ, notation $\mathcal{RQ}\llbracket rhxp \rrbracket(\theta, \mathfrak{R}, \gamma)$, as yielding a value in $\mathit{REQ} \cup \{\dagger\}$.

- $\mathcal{PR}\llbracket \pi \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) = \pi$;

- $\mathcal{PR}\llbracket p \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) = \gamma(p)$;

- $\mathcal{PR}\llbracket \Pi(tixp \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) = \Pi(\mathcal{I}\llbracket tixp \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma))$;

- $\mathcal{B}\llbracket (tixp_1, prxp, tixp_2) \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) =$

$$\begin{cases} \dagger & \text{if } \mathcal{I}\llbracket tixp_1 \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) = \dagger, \\ & \mathcal{PR}\llbracket prxp \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) = \dagger, \text{ or} \\ & \mathcal{I}\llbracket tixp_2 \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) = \dagger, \\ (\ \mathcal{I}\llbracket tixp_1 \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma), & \\ \quad \mathcal{PR}\llbracket prxp \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma), & \\ \quad \mathcal{I}\llbracket tixp_1 \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma)\ ) & \text{otherwise;} \end{cases}$$

- $\mathcal{OC}\llbracket K \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) = \gamma(K)$;

- $\mathcal{OC}\llbracket O \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) = \mathfrak{O}$;

- $\mathcal{OC}\llbracket \{blxp\} \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) =$

$$\begin{cases} \dagger & \text{if } \mathcal{B}\llbracket blxp \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) = \dagger, \\ \{\mathcal{B}\llbracket blxp \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma)\} & \text{otherwise;} \end{cases}$$

- $\mathcal{OC}\llbracket ohxp_1 \cup ohxp_2 \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) =$

$$\begin{cases} \dagger & \text{if } \mathcal{OC}\llbracket ohxp_1 \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) = \dagger \text{ or} \\ & \mathcal{OC}\llbracket ohxp_2 \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) = \dagger, \\ \mathcal{OC}\llbracket ohxp_1 \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) & \\ \cup\ \mathcal{OC}\llbracket ohxp_2 \rrbracket(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) & \text{otherwise;} \end{cases}$$

- $\mathcal{Q}[\![(tixp_1, prxp, tixp_2)]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma) =$
$$\begin{cases} \dagger & \text{if } \mathcal{I}[\![tixp_1]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma) = \dagger, \\ & \mathcal{PR}[\![prxp]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma) = \dagger, \text{ or} \\ & \mathcal{I}[\![tixp_2]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma) = \dagger, \\ (\ \mathcal{I}[\![tixp_1]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma), & \\ \quad \mathcal{PR}[\![prxp]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma), & \\ \quad \mathcal{I}[\![tixp_1]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma)\ ) & \text{otherwise;} \end{cases}$$

- $\mathcal{RQ}[\![L]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma) = \gamma(L);$

- $\mathcal{RQ}[\![Q]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma) = \mathfrak{Q};$

- $\mathcal{RQ}[\![\{quxp\}]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma) =$
$$\begin{cases} \dagger & \text{if } \mathcal{RQ}[\![quxp]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma) = \dagger, \\ \{\mathcal{RQ}[\![quxp]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma)\} & \text{otherwise;} \end{cases}$$

- $\mathcal{RQ}[\![rhxp_1 \cup rhxp_2]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma) =$
$$\begin{cases} \dagger & \text{if } \mathcal{RQ}[\![rhxp_1]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma) = \dagger \text{ or} \\ & \mathcal{RQ}[\![rhxp_2]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma) = \dagger, \\ \mathcal{RQ}[\![rhxp_1]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma) & \\ \cup\ \mathcal{RQ}[\![rhxp_2]\!](\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}, \gamma) & \text{otherwise;} \end{cases}$$

◇

Appendix H

Proofs from Chapter 6

H.1 Proof of the soundness theorem

H.1.1 Soundness of the consequence and conjunction rules

Trivial.

H.1.2 Soundness of the interleaving rule

Assume

$$\models FP_1 \ \mathbf{sat}\ \varphi_1\ ,\ \ \models FP_2\ \mathbf{sat}\ \varphi_2. \tag{H.1}$$

Consider any γ. Let $(\theta, \mathfrak{R}, \mathfrak{D}, \mathfrak{Q}) \in \mathcal{O}[\![FP_1 /\!/ FP_2]\!]$. By the definition of the semantics there exist, for $i = 1, 2$, $\mathfrak{R}_i$, $\mathfrak{D}_i$, and $\mathfrak{Q}_i$ such that

$$(\theta \uparrow chan(FP_i), \mathfrak{R}_i, \mathfrak{D}_i, \mathfrak{Q}_i) \in \mathcal{O}[\![FP_i]\!], \tag{H.2}$$

$$NoConflict(\mathfrak{D}_1, \mathfrak{D}_2), \tag{H.3}$$

$$\mathfrak{D} = \mathfrak{D}_1 \cup \mathfrak{D}_2, \tag{H.4}$$

$$\mathfrak{Q} = \mathfrak{Q}_1 \cup \mathfrak{Q}_2, \tag{H.5}$$

$$Respect(\mathfrak{D}, \mathfrak{Q}) \tag{H.6}$$

and

$$\mathfrak{R} = \mathfrak{R}_1 \cup \mathfrak{R}_2.$$

By the definition of the semantics, $\mathfrak{R}_i \setminus chan(FP_i) = \emptyset$. Hence,

$$\mathfrak{R} = \mathfrak{R}_1 \uparrow chan(FP_1) \cup \mathfrak{R}_2 \uparrow chan(FP_2). \tag{H.7}$$

Let, for fresh K_1, K_2, L_1, L_2, N_1 and N_2,

$$\widehat{\gamma} = (\gamma : (K_1, K_2, L_1, L_2, N_1, N_2) \mapsto (\mathfrak{O}_1, \mathfrak{O}_2, \mathfrak{Q}_1, \mathfrak{Q}_2, \mathfrak{R}_1, \mathfrak{R}_2)).$$

By (H.3) – (H.7),

$$\begin{aligned}(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \exists K_1, K_2, L_1, L_2, N_1, N_2 \cdot \\ & \mathit{NoConflict}(K_1, K_2) \\ & \wedge O = K_1 \cup K_2 \\ & \wedge Q = L_1 \cup L_2 \\ & \wedge \mathit{Respect}(O, Q) \\ & \wedge R = N_1 \uparrow chan(FP_1) \cup N_2 \uparrow chan(FP_2).\end{aligned} \tag{H.8}$$

By (H.1) and (H.2), we obtain $(\theta \uparrow chan(FP_i), \mathfrak{R}_i, \mathfrak{O}_i, \mathfrak{Q}_i, \widehat{\gamma}) \models \varphi_i$. By substitution lemma 6.20(b), (d), (f), (h), $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \widehat{\gamma}) \models \varphi_i(h \uparrow chan(FP_i), N_i, K_i, L_i)$:

$$(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \exists K_1, K_2, L_1, L_2, N_1, N_2 \cdot \varphi_i(h \uparrow chan(FP_i), N_i, K_i, L_i). \tag{H.9}$$

By (H.8) and (H.9) we conclude that interleaving rule 6.28 is sound.

H.1.3 Soundness of the priority assignment rule

Assume

$$\models FP \textbf{ sat } \varphi(h, R, \mathit{IncPr}(\pi, O), \mathit{IncPr}(\pi, Q)). \tag{H.10}$$

Consider any γ. Let $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}) \in \mathcal{O}[\![\textbf{prio } \pi \ (FP)]\!]$. By the definition of the semantics there exist $\widehat{\mathfrak{O}}$ and $\widehat{\mathfrak{Q}}$ such that

$$(\theta, \mathfrak{R}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}) \in \mathcal{O}[\![FP]\!], \tag{H.11}$$

$$\mathfrak{O} = \mathit{IncPr}(\pi, \widehat{\mathfrak{O}}), \tag{H.12}$$

and

$$\mathfrak{Q} = \mathit{IncPr}(\pi, \widehat{\mathfrak{Q}}). \tag{H.13}$$

By (H.10) and (H.11), $(\theta, \mathfrak{R}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}, \gamma) \models \varphi(h, R, \mathit{IncPr}(\pi, O), \mathit{IncPr}(\pi, Q))$, i.e, $(\theta, \mathfrak{R}, \mathit{IncPr}(\pi, \widehat{\mathfrak{O}}), \mathit{IncPr}(\pi, \widehat{\mathfrak{Q}}), \gamma) \models \varphi(h, R, O, Q)$, by substitution lemma 6.20(f) and (h). By (H.12) and (H.13), $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \varphi(h, R, O, Q)$. Hence, priority assignment rule 6.29 is sound.

H.1.4 Soundness of the failure hypothesis introduction rule

Assume that

$$\models FP \ \mathbf{sat} \ \varphi. \tag{H.14}$$

Consider any γ. Let $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}) \in \mathcal{O}[\![FP \wr \chi]\!]$. By the definition of the semantics, there exists a $(\theta_0, \mathfrak{R}_0, \mathfrak{O}_0, \mathfrak{Q}_0) \in \mathcal{O}[\![FP]\!]$, such that, for all γ,

$$(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \mathfrak{O}_0, \mathfrak{O}, \mathfrak{Q}_0, \mathfrak{Q}, \gamma) \models \chi. \tag{H.15}$$

Let, for fresh s, K, L and N, $\widehat{\gamma} = (\gamma : (s, K, L, N) \mapsto (\theta_0, \mathfrak{O}_0, \mathfrak{Q}_0, \mathfrak{R}_0))$. Since $(\theta_0, \mathfrak{R}_0, \mathfrak{O}_0, \mathfrak{Q}_0) \in \mathcal{O}[\![FP]\!]$, we know, because of (H.14), that, for all γ, $(\theta_0, \mathfrak{R}_0, \mathfrak{O}_0, \mathfrak{Q}_0, \gamma) \models \varphi$. Consequently, $(\theta_0, \mathfrak{R}_0, \mathfrak{O}_0, \mathfrak{Q}_0, \widehat{\gamma}) \models \varphi$, that is,

$$\begin{array}{l}(\ \mathcal{T}[\![s]\!](\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \widehat{\gamma}) \ , \\ \ \ \mathcal{RF}[\![N]\!](\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \widehat{\gamma}) \ , \\ \ \ \mathcal{OC}[\![K]\!](\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \widehat{\gamma}) \ , \\ \ \ \mathcal{RQ}[\![L]\!](\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \widehat{\gamma}) \ , \\ \ \ \widehat{\gamma} \ ) \models \varphi.\end{array}$$

Hence, by substitution lemma 6.20(b), (d), (f) and (h),

$$(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \widehat{\gamma}) \models \varphi(s, N, K, L). \tag{H.16}$$

By (H.15), $(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \mathfrak{O}_0, \mathfrak{O}, \mathfrak{Q}_0, \mathfrak{Q}, \widehat{\gamma}) \models \chi$, that is,

$$\begin{array}{l}(\ \mathcal{T}[\![s]\!](\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \widehat{\gamma}) \ , \\ \ \ \theta \ , \\ \ \ \mathcal{RF}[\![N]\!](\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \widehat{\gamma}) \ , \\ \ \ \mathfrak{R} \ , \\ \ \ \mathcal{OC}[\![K]\!](\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \widehat{\gamma}) \ , \\ \ \ \mathfrak{O} \ , \\ \ \ \mathcal{RQ}[\![L]\!](\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \widehat{\gamma}) \ , \\ \ \ \mathfrak{Q} \ , \\ \ \ \widehat{\gamma} \ ) \models \chi.\end{array}$$

By substitution lemma 6.20(a), (c), (e) and (g),

$$(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \mathfrak{O}_0, \mathfrak{O}, \mathfrak{Q}_0, \mathfrak{Q}, \widehat{\gamma}) \models \chi(s, h, N, R, K, O, L, Q).$$

Since h_{old} and R_{old} obviously do not appear in $\chi(s, h, N, R, K, O, L, Q)$, correspondence lemma 6.19 leads to

$$(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \widehat{\gamma}) \models \chi(s, h, N, R, K, O, L, Q). \tag{H.17}$$

By (H.16) and (H.17), we conlude

$$(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \exists s, N, K, L \cdot \varphi(s, N, K, L) \ \wedge \ \psi(s, h, N, R, K, O, L, Q).$$

Hence, failure hypothesis introduction rule 6.30 is sound.

H.1.5 Soundness of the processor closure rule

Assume that

$$\begin{aligned} \models FP \ \textbf{sat}\ (\mathit{NoStrike}(O, Q) \ \wedge \ \mathit{ASAP}(R, io(FP))) \\ \rightarrow \ \phi(h \backslash io(FP), R \backslash io(FP)). \end{aligned} \tag{H.18}$$

Consider any γ. Let $(\theta, \mathfrak{R}) \in \mathcal{O}[\![\ll FP \gg]\!]$. Then, by the definition of the semantics there exists a

$$(\widehat{\theta}, \widehat{\mathfrak{R}}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}) \in \mathcal{O}[\![FP]\!] \tag{H.19}$$

with

$$\mathit{NoStrike}(\widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}), \tag{H.20}$$

$$\mathit{ASAP}(\widehat{\mathfrak{R}}, io(FP)), \tag{H.21}$$

$$\theta = \widehat{\theta} \backslash io(FP) \tag{H.22}$$

and

$$\mathfrak{R} = \widehat{\mathfrak{R}} \backslash io(FP). \tag{H.23}$$

By (H.19) and (H.18), for all γ,

$$\begin{aligned} (\widehat{\theta}, \widehat{\mathfrak{R}}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}, \gamma) \models (\mathit{NoStrike}(O, Q) \ \wedge \ \mathit{ASAP}(R, io(FP))) \\ \rightarrow \ \phi(h \backslash io(FP), R \backslash io(FP)). \end{aligned}$$

Then, by (H.20) and (H.21), $(\widehat{\theta}, \widehat{\mathfrak{R}}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}, \gamma) \models \phi(h \backslash io(FP), R \backslash io(FP))$. By substitution lemma 6.20(b) and (d), $(\widehat{\theta} \backslash io(FP), \widehat{\mathfrak{R}} \backslash io(FP), \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}, \gamma) \models \phi(h, R)$. Hence, by (H.22) and (H.23), $(\theta, \mathfrak{R}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}, \gamma) \models \phi(h, R)$. Since O and Q do not appear in $\phi(h, R)$, we conclude $(\theta, \mathfrak{R}, \gamma) \models \phi(h, R)$. Hence, processor closure rule 6.32 is sound.

H.2 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 interleaving rule 6.28, we obtain

$$\begin{aligned}\vdash FP_1 /\!/ FP_2 \ \mathbf{sat}\ \exists K_1, K_2, L_1, L_2, N_1, N_2 \cdot\ & \\ & NoConflict(K_1, K_2) \\ & \wedge O = K_1 \cup K_2 \\ & \wedge Q = L_1 \cup L_2 \\ & \wedge Respect(O, Q) \\ & \wedge R = N_1 \uparrow chan(FP_1) \cup N_2 \uparrow chan(FP_2) \\ & \wedge \varphi_1(h \uparrow chan(FP_1), N_1, K_1, L_1) \\ & \wedge \varphi_2(h \uparrow chan(FP_2), N_2, K_2, L_2).\end{aligned} \tag{H.24}$$

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

i) By (H.24) and soundness, we obtain

$$\begin{aligned}\models FP_1 /\!/ FP_2 \ \mathbf{sat}\ \exists K_1, K_2, L_1, L_2, N_1, N_2 \cdot\ & \\ & NoConflict(K_1, K_2) \\ & \wedge O = K_1 \cup K_2 \\ & \wedge Q = L_1 \cup L_2 \\ & \wedge Respect(O, Q) \\ & \wedge R = N_1 \uparrow chan(FP_1) \cup N_2 \uparrow chan(FP_2) \\ & \wedge \varphi_1(h \uparrow chan(FP_1), N_1, K_1, L_1) \\ & \wedge \varphi_2(h \uparrow chan(FP_2), N_2, K_2, L_2).\end{aligned}$$

ii) Let

$$chan(\theta) \subseteq chan(FP_1 \| FP_2) \tag{H.25}$$

and

$$\mathfrak{R} \uparrow chan(FP_1 \| FP_2) = \mathfrak{R}. \tag{H.26}$$

Assume, for some γ,

$$
\begin{aligned}
(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \exists K_1, K_2, L_1, L_2, N_1, N_2 \cdot\ & \\
& \mathit{NoConflict}(K_1, K_2) \\
& \wedge O = K_1 \cup K_2 \\
& \wedge Q = L_1 \cup L_2 \\
& \wedge \mathit{Respect}(O, Q) \\
& \wedge R = N_1 \uparrow \mathit{chan}(FP_1) \cup N_2 \uparrow \mathit{chan}(FP_2) \\
& \wedge \varphi_1(h \uparrow \mathit{chan}(FP_1), N_1, K_1, L_1) \\
& \wedge \varphi_2(h \uparrow \mathit{chan}(FP_2), N_2, K_2, L_2).
\end{aligned}
$$

In other words, there exist $\mathfrak{O}_1$, $\mathfrak{O}_2$, $\mathfrak{Q}_1$, $\mathfrak{Q}_2$, $\mathfrak{R}_1$ and $\mathfrak{R}_2$ such that, for $\widehat{\gamma} = (\gamma : (K_1, K_2, L_1, L_2, N_1, N_2) \mapsto (\mathfrak{O}_1, \mathfrak{O}_2, \mathfrak{Q}_1, \mathfrak{Q}_2, \mathfrak{R}_1, \mathfrak{R}_2))$,

$$
\begin{aligned}
(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \widehat{\gamma}) \models\ & \mathit{NoConflict}(K_1, K_2) \\
& \wedge O = K_1 \cup K_2 \\
& \wedge Q = L_1 \cup L_2 \\
& \wedge \mathit{Respect}(O, Q) \\
& \wedge R = N_1 \uparrow \mathit{chan}(FP_1) \cup N_2 \uparrow \mathit{chan}(FP_2) \\
& \wedge \varphi_1(h \uparrow \mathit{chan}(FP_1), N_1, K_1, L_1) \\
& \wedge \varphi_2(h \uparrow \mathit{chan}(FP_2), N_2, K_2, L_2).
\end{aligned} \tag{H.27}
$$

Consequently,

$$\mathit{NoConflict}(\mathfrak{O}_1, \mathfrak{O}_2), \tag{H.28}$$

$$\mathfrak{O} = \mathfrak{O}_1 \cup \mathfrak{O}_2, \tag{H.29}$$

$$\mathfrak{Q} = \mathfrak{Q}_1 \cup \mathfrak{Q}_2, \tag{H.30}$$

$$\mathit{Respect}(\mathfrak{O}, \mathfrak{Q}), \tag{H.31}$$

$$\mathfrak{R} = \mathfrak{R}_1 \uparrow \mathit{chan}(FP_1) \cup \mathfrak{R}_2 \uparrow \mathit{chan}(FP_2) \tag{H.32}$$

and, by substitution lemma 6.20(b), (d), (f) and (h),

$$(\theta \uparrow \mathit{chan}(FP_1), \mathfrak{R}_1, \mathfrak{O}_1, \mathfrak{Q}_1, \widehat{\gamma}) \models \varphi_1,$$

or, since K_1, K_2, L_1, L_2, N_1 and N_2 do not occur free in φ_1,

$$(\theta \uparrow \mathit{chan}(FP_1), \mathfrak{R}_1, \mathfrak{O}_1, \mathfrak{Q}_1, \gamma) \models \varphi_1.$$

By the preciseness of φ_1 for FP_1,

$$(\theta\!\uparrow chan(FP_1), \mathfrak{R}_1, \mathfrak{O}_1, \mathfrak{Q}_1, \gamma)$$
$$\models \varphi_1(h\!\uparrow chan(FP_1), R\!\uparrow chan(FP_1), O, Q).$$

By applying substitution lemma 6.20(b) and (d), and using the fact that $(\theta\!\uparrow chan(FP_1))\!\uparrow chan(FP_1) = \theta\!\uparrow chan(FP_1)$, we obtain

$$(\theta\!\uparrow chan(FP_1), \mathfrak{R}_1\!\uparrow chan(FP_1), \mathfrak{O}_1, \mathfrak{Q}_1, \gamma) \models \varphi_1. \quad \text{(H.33)}$$

Trivially,

$$chan(\theta\!\uparrow chan(FP_1)) \subseteq chan(FP_1) \quad \text{(H.34)}$$

and

$$chan(\mathfrak{R}_1\!\uparrow chan(FP_1)) \subseteq chan(FP_1). \quad \text{(H.35)}$$

By (H.34), (H.35), and (H.33), the preciseness of φ_1 for FP_1 leads to

$$(\theta\!\uparrow chan(FP_1), \mathfrak{R}_1\!\uparrow chan(FP_1), \mathfrak{O}_1, \mathfrak{Q}_1) \in \mathcal{O}[\![FP_1]\!]. \quad \text{(H.36)}$$

Similarly,

$$(\theta\!\uparrow chan(FP_2), \mathfrak{R}_2\!\uparrow chan(FP_2), \mathfrak{O}_2, \mathfrak{Q}_2) \in \mathcal{O}[\![FP_2]\!]. \quad \text{(H.37)}$$

Hence, by (H.28) – (H.32), (H.36) and (H.37), we conclude that $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}) \in \mathcal{O}[\![FP_1 /\!/ FP_2]\!]$.

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

$$\begin{aligned}(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \exists K_1, K_2, L_1, L_2, N_1, N_2\cdot\; & NoConflict(K_1, K_2)\\ \wedge\; & O = K_1 \cup K_2\\ \wedge\; & Q = L_1 \cup L_2\\ \wedge\; & Respect(O, Q)\\ \wedge\; & R = N_1\!\uparrow chan(FP_1) \cup N_2\!\uparrow chan(FP_2)\\ \wedge\; & \varphi_1(h\!\uparrow chan(FP_1), N_1, K_1, L_1)\\ \wedge\; & \varphi_2(h\!\uparrow chan(FP_2), N_2, K_2, L_2).\end{aligned}$$

Since $chan(FP_i) \subseteq chan(FP_1 /\!/ FP_2)$, $i = 1, 2$, we may conclude

$$
\begin{array}{l}
(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \\
\models \exists K_1, K_2, L_1, L_2, N_1, N_2 \cdot \\
\qquad NoConflict(K_1, K_2) \\
\quad \wedge\ O = K_1 \cup K_2 \\
\quad \wedge\ Q = L_1 \cup L_2 \\
\quad \wedge\ Respect(O, Q) \\
\quad \wedge\ R\uparrow chan(FP_1/\!/FP_2) = \quad N_1 \uparrow chan(FP_1) \\
\qquad\qquad\qquad\qquad\qquad\qquad \cup\ N_2 \uparrow chan(FP_2) \\
\quad \wedge\ \varphi_1((h\uparrow chan(FP_1/\!/FP_2))\uparrow chan(FP_1), N_1, K_1, L_1) \\
\quad \wedge\ \varphi_2((h\uparrow chan(FP_1/\!/FP_2))\uparrow chan(FP_2), N_2, K_2, L_2).
\end{array}
$$

Hence, $\models \varphi \ \rightarrow\ \varphi(h\uparrow chan(FP_1/\!/FP_2), R\uparrow chan(FP_1/\!/FP_2), O, Q)$.

b) Assume

$$\vdash FP \textbf{ sat } \varphi, \tag{H.38}$$

with φ precise for FP. Define $\widehat{\varphi}(h, R, O, Q) \ \equiv\ \exists K, L \cdot \ \ \varphi(h, R, K, L) \wedge O = IncPr(\pi, K) \wedge Q = IncPr(\pi, L)$.

The following lemma is trivial.

Lemma H.1 $\models \varphi(h, R, O, Q) \ \rightarrow\ \widehat{\varphi}(h, R, IncPr(\pi, O), IncPr(\pi, Q))$ ○

By lemma H.1 and relative completeness axiom 6.35,

$$\vdash \varphi(h, R, O, Q) \ \rightarrow\ \widehat{\varphi}(h, R, IncPr(\pi, O), IncPr(\pi, Q)).$$

Hence, by (H.38) and consequence rule 6.26,

$$\vdash FP \textbf{ sat } \widehat{\varphi}(h, R, IncPr(\pi, O), IncPr(\pi, Q)).$$

Then, by priority assignment rule 6.29,

$$\vdash \textbf{prio } \pi\ (FP) \textbf{ sat } \widehat{\varphi}. \tag{H.39}$$

It remains to be shown that $\widehat{\varphi}$ is precise for **prio** π (FP).

i) By (H.39) and soundness, we obtain

$$\models \textbf{prio } \pi\ (FP) \textbf{ sat } \widehat{\varphi}.$$

ii) Let

$$chan(\theta) \subseteq chan(FP) \tag{H.40}$$

and

$$\mathfrak{R}\!\uparrow chan(FP) = \mathfrak{R}. \tag{H.41}$$

Assume, for some γ,

$$(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \widehat{\varphi}.$$

Then, there exist $\widehat{\mathfrak{O}}$ and $\widehat{\mathfrak{Q}}$ such that

$$(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, (\gamma : (K, L) \mapsto (\widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}))) \models \begin{array}[t]{l} \varphi(h, R, K, L) \\ \wedge\ O = IncPr(\pi, K) \\ \wedge\ Q = IncPr(\pi, L) \end{array} \tag{H.42}$$

Consequently, by substitution lemma 6.20(f) and (h), we obtain that $(\theta, \mathfrak{R}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}, (\gamma : (K, L) \mapsto (\widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}))) \models \varphi$. Since K and L do not occur in φ,

$$(\theta, \mathfrak{R}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}, \gamma) \models \varphi. \tag{H.43}$$

By (H.40), (H.41), and (H.43), the preciseness of φ for *FP* yields

$$(\theta, \mathfrak{R}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}) \in \mathcal{O}[\![FP]\!]. \tag{H.44}$$

By (H.42),

$$\mathfrak{O} = IncPr(\pi, \widehat{\mathfrak{O}}) \tag{H.45}$$

and

$$\mathfrak{Q} = IncPr(\pi, \widehat{\mathfrak{Q}}). \tag{H.46}$$

Hence, by (H.44) – (H.46), $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}) \in \mathcal{O}[\![\textbf{prio}\ \pi\ (FP)]\!]$.

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

$$(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \exists K, L \cdot \begin{array}[t]{l} \varphi(h, R, K, L) \\ \wedge\ O = IncPr(\pi, K) \\ \wedge\ Q = IncPr(\pi, L) \end{array}$$

By the preciseness of φ for *FP*,

$$(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \exists K, L \cdot \begin{array}[t]{l} \varphi(h\!\uparrow chan(FP), R\!\uparrow chan(FP), K, L) \\ \wedge\ O = IncPr(\pi, K) \\ \wedge\ Q = IncPr(\pi, L) \end{array}$$

Since $chan(\mathbf{prio}\ \pi\ (FP)) = chan(FP)$,

$$\begin{array}{l} (\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \\ \models \exists K, L \cdot \begin{array}[t]{l} \varphi(h\!\uparrow chan(\mathbf{prio}\ \pi\ (FP)), R\!\uparrow chan(\mathbf{prio}\ \pi\ (FP)), K, L) \\ \wedge\ O = IncPr(\pi, K) \\ \wedge\ Q = IncPr(\pi, L) \end{array} \end{array}$$

Hence, $\models \varphi \rightarrow \varphi(h\!\uparrow chan(\mathbf{prio}\ \pi\ (FP)), R\!\uparrow chan(\mathbf{prio}\ \pi\ (FP)), O, Q)$.

c) Assume

$$\vdash FP \ \mathbf{sat}\ \varphi, \tag{H.47}$$

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

$$\widehat{\varphi} \ \equiv\ \exists s, K, L, N \cdot \varphi(s, N, K, L) \ \wedge\ \chi(s, h, N, R, K, O, L, Q)$$

Then, by (H.47) and failure hypothesis introduction rule 6.30,

$$\vdash FP \wr \chi \ \mathbf{sat}\ \widehat{\varphi}. \tag{H.48}$$

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

i) By (H.48) and soundness, we have $\models FP \wr \chi \ \mathbf{sat}\ \widehat{\varphi}$.

ii) Let

$$chan(\theta) \subseteq chan(FP \wr \chi), \tag{H.49}$$

$$\mathfrak{R}\!\uparrow chan(FP \wr \chi) = \mathfrak{R}, \tag{H.50}$$

and assume, for some γ, $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \widehat{\varphi}$. Hence, there exist $\widehat{\theta}$, $\widehat{\mathfrak{R}}$, $\widehat{\mathfrak{O}}$ and $\widehat{\mathfrak{Q}}$ such that, for $\widehat{\gamma} = (\gamma : (s, K, L, N) \mapsto (\widehat{\theta}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}, \widehat{\mathfrak{R}}))$,

$$(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \widehat{\gamma}) \models \varphi(s, N, K, L) \wedge \chi(s, h, N, R, K, O, L, Q) \tag{H.51}$$

By substitution lemma 6.20(b), (d), (f) and (h),

$$(\widehat{\theta}, \widehat{\mathfrak{R}}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}, \widehat{\gamma}) \models \varphi.$$

Since s, K, L and N do not occur free in φ, $(\widehat{\theta}, \widehat{\mathfrak{R}}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}, \gamma) \models \varphi$. Because φ is precise for FP, we may conclude that

$$(\widehat{\theta}, \widehat{\mathfrak{R}}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}, \gamma) \models \varphi(h\!\uparrow\! chan(FP), R\!\uparrow\! chan(FP), O, Q).$$

Hence, by substitution lemma 6.20(b) and (d),

$$(\widehat{\theta}\!\uparrow\! chan(FP), \widehat{\mathfrak{R}}\!\uparrow\! chan(FP), \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}, \gamma) \models \varphi. \tag{H.52}$$

Trivially,

$$chan(\widehat{\theta}\!\uparrow\! chan(FP)) \subseteq chan(FP) \tag{H.53}$$

and

$$chan(\widehat{\mathfrak{R}}\!\uparrow\! chan(FP)) \subseteq chan(FP). \tag{H.54}$$

By (H.53), (H.54), (H.52), and the fact that φ is precise for FP, we may conclude that

$$(\widehat{\theta}\!\uparrow\! chan(FP), \widehat{\mathfrak{R}}\!\uparrow\! chan(FP), \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}) \in \mathcal{O}[\![FP]\!]. \tag{H.55}$$

By (H.51) and correspondence lemma 6.19,

$$(\theta_0, \theta, \mathfrak{R}_0, \mathfrak{R}, \mathfrak{O}_0, \mathfrak{O}, \mathfrak{Q}_0, \mathfrak{Q}, \widehat{\gamma}) \models \chi(s, h, N, R, K, O, L, Q),$$

for any θ_0, $\mathfrak{R}_0$, $\mathfrak{O}_0$ and $\mathfrak{Q}_0$. By substitution lemma 6.20(a), (c), (e) and (g), we obtain

$$(\widehat{\theta}, \theta, \widehat{\mathfrak{R}}, \mathfrak{R}, \widehat{\mathfrak{O}}, \mathfrak{O}, \widehat{\mathfrak{Q}}, \mathfrak{Q}, \widehat{\gamma}) \models \chi,$$

and thus, since s, K, L and N do not occur free in χ,

$$(\widehat{\theta}, \theta, \widehat{\mathfrak{R}}, \mathfrak{R}, \widehat{\mathfrak{O}}, \mathfrak{O}, \widehat{\mathfrak{Q}}, \mathfrak{Q}, \gamma) \models \chi.$$

Since χ is a failure hypothesis, we may conclude that

$$\begin{aligned}&(\widehat{\theta}, \theta, \widehat{\mathfrak{R}}, \mathfrak{R}, \mathfrak{O}_0, \mathfrak{O}, \mathfrak{Q}_0, \mathfrak{Q}, \gamma)\\ &\models \chi(h_{old}\!\uparrow\! chan(FP), h, R_{old}\!\uparrow\! chan(FP), R, O_{old}, O, Q_{old}, Q).\end{aligned}$$

By substitution lemma 6.20(a) and (c)

$$(\widehat{\theta}\!\uparrow\! chan(FP), \theta, \widehat{\mathfrak{R}}\!\uparrow\! chan(FP), \mathfrak{R}, \mathfrak{O}_0, \mathfrak{O}, \mathfrak{Q}_0, \mathfrak{Q}, \gamma) \models \chi. \tag{H.56}$$

By (H.55), (H.56), (H.53) and (H.54), $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}) \in \mathcal{O}[\![FP \wr \chi]\!]$.

iii) Follows from the fact that, since φ is precise for *FP*,

$$\varphi \rightarrow \varphi(h\!\uparrow\! chan(FP), R\!\uparrow\! chan(FP), O, Q),$$

the fact that, since χ is a failure hypothesis,

$$\chi \rightarrow \chi(h_{old}, h\!\uparrow\! chan(FP), R_{old}, R\!\uparrow\! chan(FP), O_{old}, O, Q_{old}, Q),$$

and the fact that $chan(FP \wr \chi) = chan(FP)$.

d) Assume

$$\vdash FP \textbf{ sat } \varphi, \tag{H.57}$$

with φ precise for *FP*. Define $\widehat{\varphi}(h, R) \equiv \exists s, K, L, N \cdot \; \varphi(s, N, K, L) \wedge NoStrike(K, L) \wedge ASAP(N, io(FP)) \wedge h = s \setminus io(FP) \wedge R = N \setminus io(FP)$

The following lemma is trivial.

Lemma H.2 $\models \varphi(h, R, O, Q) \rightarrow (NoStrike(O, Q) \wedge ASAP(R, io(FP)) \rightarrow \widehat{\varphi}(h \setminus io(FP), R \setminus io(FP)))$ ○

By lemma H.2 and relative completeness axiom 6.35,

$$\vdash \varphi(h, R, O, Q) \rightarrow (NoStrike(O, Q) \wedge ASAP(R, io(FP)) \rightarrow \widehat{\varphi}(h \setminus io(FP), R \setminus io(FP))).$$

Hence, by (H.57) and consequence rule 6.26,

$$\vdash FP \textbf{ sat } NoStrike(O, Q) \wedge ASAP(R, io(FP)) \rightarrow \phi(h \setminus io(FP), R \setminus io(FP)).$$

Then, by priority assignment rule 6.29,

$$\vdash \ll FP \gg \textbf{ sat } \widehat{\varphi}. \tag{H.58}$$

It remains to be shown that $\widehat{\varphi}$ is precise for $\ll FP \gg$.

i) By (H.58) and soundness, we have $\models \ll FP \gg \textbf{ sat } \widehat{\varphi}$.

ii) Let

$$chan(\theta) \subseteq chan(\ll FP \gg), \tag{H.59}$$

$$chan(\mathfrak{R}) \subseteq chan(\ll FP \gg), \tag{H.60}$$

and assume, for some γ, $(\theta, \mathfrak{R}, \gamma) \models \widehat{\varphi}$. Since O and Q do not occur in $\widehat{\varphi}$, for all $\mathfrak{O}$ and $\mathfrak{Q}$, $(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \gamma) \models \widehat{\varphi}$. Hence, there exist $\widehat{\theta}$, $\widehat{\mathfrak{R}}$, $\widehat{\mathfrak{O}}$ and $\widehat{\mathfrak{Q}}$ such that, for $\widehat{\gamma} = (\gamma : (s, K, L, N) \mapsto (\widehat{\theta}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}, \widehat{\mathfrak{R}}))$,

$$\begin{aligned}(\theta, \mathfrak{R}, \mathfrak{O}, \mathfrak{Q}, \widehat{\gamma}) \models \quad & \varphi(s, N, K, L) \\ & \wedge\ NoStrike(K, L) \\ & \wedge\ ASAP(N, io(FP)) \\ & \wedge\ h = s \backslash io(FP) \\ & \wedge\ R = N \backslash io(FP).\end{aligned} \tag{H.61}$$

By substitution lemma 6.20(b), (d), (f) and (h),

$$(\widehat{\theta}, \widehat{\mathfrak{R}}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}, \gamma) \models \varphi \tag{H.62}$$

By (H.61),

$$\theta = \widehat{\theta} \backslash io(FP). \tag{H.63}$$

Hence, $chan(\widehat{\theta}) = chan(\theta) \cup io(FP)$, that is, by (H.59)

$$chan(\widehat{\theta}) \subseteq chan(FP). \tag{H.64}$$

By (H.61),

$$\mathfrak{R} = \widehat{\mathfrak{R}} \backslash io(FP). \tag{H.65}$$

Hence, $chan(\widehat{\mathfrak{R}}) = chan(\mathfrak{R}) \cup io(FP)$, that is, by (H.60)

$$chan(\widehat{\mathfrak{R}}) \subseteq chan(FP). \tag{H.66}$$

By (H.64), (H.66), and (H.62), the preciseness of φ for FP yields

$$(\widehat{\theta}, \widehat{\mathfrak{R}}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}) \in \mathcal{O}[\![FP]\!]. \tag{H.67}$$

By (H.61),

$$NoStrike(\widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}) \tag{H.68}$$

and

$$ASAP(\widehat{\mathfrak{R}}, io(FP)). \tag{H.69}$$

By (H.67) – (H.69), (H.63) and (H.65),

$$(\theta, \mathfrak{R}) \in \mathcal{O}[\![\ll FP \gg]\!].$$

iii) Assume $(\theta, \mathfrak{R}, \gamma) \models \widehat{\varphi}$. Then, there exist $\widehat{\theta}$, $\widehat{\mathfrak{O}}$, $\widehat{\mathfrak{Q}}$ and $\widehat{\mathfrak{R}}$ such that, for $\widehat{\gamma} = (\gamma : (s, K, L, N) \mapsto (\widehat{\theta}, \widehat{\mathfrak{O}}, \widehat{\mathfrak{Q}}, \widehat{\mathfrak{R}}))$,

$$\begin{aligned}(\theta, \mathfrak{R}, \widehat{\gamma}) \models \quad & \varphi(s, N, K, L) \\ & \wedge\ ASAP(N, io(FP)) \\ & \wedge\ h = s \setminus io(FP) \\ & \wedge\ R = N \setminus io(FP).\end{aligned} \tag{H.70}$$

By the preciseness of φ for FP,

$$\varphi(s, N, K, L) \ \rightarrow\ \varphi(s \uparrow chan(FP), N \uparrow chan(FP), K, L). \tag{H.71}$$

It is obvious that

$$ASAP(N, io(FP)) \ \rightarrow\ ASAP(N \setminus chan(FP), io(FP)). \tag{H.72}$$

Note that

$$h = s \setminus io(FP) \rightarrow h \uparrow chan(\ll FP \gg) = (s \uparrow chan(FP)) \setminus io(FP). \tag{H.73}$$

By (H.70), $R = N \setminus io(FP)$, that is,

$$R \uparrow chan(\ll FP \gg) = (N \uparrow chan(FP)) \setminus io(FP). \tag{H.74}$$

By (H.70) – (H.74), we obtain

$$\begin{aligned}(\theta, \mathfrak{R}, \widehat{\gamma}) \models \quad & \phi(s \uparrow chan(FP), N \uparrow chan(FP)) \\ & \wedge\ ASAP(N \uparrow chan(FP), io(FP)) \\ & \wedge\ h \uparrow chan(\ll FP \gg) = (s \uparrow chan(FP)) \setminus io(FP) \\ & \wedge\ R \uparrow chan(\ll FP \gg) = (N \uparrow chan(FP)) \setminus io(FP),\end{aligned}$$

from which we may conclude

$$(\theta, \mathfrak{R}, \gamma) \models \widehat{\varphi}(h \uparrow chan(\ll FP \gg), R \uparrow chan(\ll FP \gg)).$$

References

[AL92] 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-Verlag, 1992) 1–27.

[ANSI83] American National Standards Institute. The programming language ADA reference manual, *Lecture Notes in Computer Science* **155** (Springer-Verlag, 1983).

[AL86] A. Avižienis and J.C. Laprie. Dependable computing: From concepts to design diversity, *Proceedings of the IEEE* **74** 5 (May 1986) 629–638.

[BKP86] H. Barringer, R. Kuiper, and 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.

[BSW69] K.A. Bartlett, R.A. Scantlebury, and P.T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links, *Communications of the ACM* **12**(5) (1969) 260–261.

[CdeR93] A. Cau and W.-P. de Roever. Specifying fault tolerance within Stark's formalism, in: *Proc. 23rd Symposium on Fault-Tolerant Computing* (IEEE Computer Society Press, 1993) 392–401.

[CA78] L. Chen and A. Avižienis. N-Version programming: A fault tolerance approach to reliability of software operation, in: *Proc. 8th Symposium on Fault-Tolerant Computing* (IEEE Computer Society Press, 1978) 3–9.

[Coenen93] J. Coenen. Top-down development of layered fault tolerant systems and its problems — A deontic perspective, *Annals of Mathematics and Artificial Intelligence* **9** (1993) 133–150.

[CH93] 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.

[Cook78] S.A. Cook. Soundness and completeness of an axiom system for program verification, *SIAM Journal on Computing* **7**(1) (February 1978) 70–90.

[Cristian85] F. Cristian. A rigorous approach to fault-tolerant programming, *IEEE Transactions on Software Engineering* **SE-11**(1) (January 1985) 23–31.

[Cristian91] F. Cristian. Understanding fault tolerant distributed systems, *Communications of the ACM* **34**(2) (1991) 56–78.

[GB87] 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-Verlag, 1987) 95–114.

[Hamming50] R.W. Hamming. Error detecting and error correcting codes', *The Bell System Technical Journal* **29**(4) (April 1950) 147–160.

[HP85] D. Harel and A. Pnueli. On the development of reactive systems, *Logics and Models of Concurrent Systems* (Springer-Verlag, 1985) 477–498.

[Hoare69] C.A.R. Hoare. An axiomatic basis for computer programming, *Communications of the ACM* **12**(10) (1969) 576–580,583.

[Hoare78] C.A.R. Hoare. Communicating Sequential Processes, *Communications of the ACM* **21**(8) (1978) 666–677.

[Hooman92] J. Hooman. Specification and compositional verification of real-time systems, *Lecture Notes in Computer Science* **558** (Springer-Verlag, 1992).

[HLMR74] J.J. Horning, H.C. Lauer, P.M. Melliar-Smith and B. Randell. A program structure for error detection and recovery, in: *Operating Systems*, Lecture Notes in Computer Science **16** (Springer-Verlag, 1974) 171–187.

[INMOS88] INMOS Limited. occam 2 *Reference Manual* (Prentice Hall, 1988).

[JH87] H. Jifeng and C.A.R. Hoare. Algebraic specification and proof of a distributed recovery algorithm, *Distributed Computing* **2** (1987) 1–12.

[JMS87] M. Joseph, A. Moitra, and N. Soundararajan. Proof rules for fault tolerant distributed programs, *Science of Computer Programming* **8** (1987) 43–67.

[KSdRGA88] R. Koymans, R.K. Shyamasundar, W.-P. de Roever, R. Gerth and S. Arun-Kumar. Compositional semantics for real-time distributed computing, *Information and Computation* **79**(3) (1988) 210–256.

[Krol91] Th. Krol. A generalization of fault tolerance based on masking, Ph.D. Thesis, Eindhoven University of Technology, 1991.

[Lamport78] L. Lamport. Time, clocks, and the ordering of events in a distributed system, *Communications of the ACM* **21**(7) (July 1978) 558–565.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem, *ACM Transactions on Programming Languages and Systems* **4**(3) (July 1982) 382–401.

[Lamport83] L. Lamport. What good is temporal logic, in: R.E. Manson (ed.), *Information Processing* (North-Holland, 1983) 657–668.

[Laprie85] J.C. Laprie. Dependable computing and fault tolerance: Concepts and terminology, in: *Proc. 15th Symp. on Fault-Tolerant Computing* (IEEE Computer Society Press, 1985) 2–11.

[LA90] P.A Lee and T. Anderson *Fault tolerance: Principles and practice* (Springer-Verlag, 1990).

[LJ93] Z. Liu and M. Joseph. Specification and verification of recovery in asynchronous communicating systems, in: J. Vytopil (ed.), *Formal Techniques in Real-Time and Fault Tolerant Systems* (Kluwer Academic Publishers, 1993) 137–165.

[Lomet77] D.B. Lomet. Process structuring, synchronization and recovery using atomic actions, *ACM SIGPLAN Notices* **12**(3) (March 1977) 128–137.

[Nordahl93] 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-Verlag, 1993) 65–89.

[Olderog91] E.R. Olderog. Nets, terms, and formulas, *Cambridge Tracts in Computer Science* **23** (Cambridge University Press, 1991).

[PS91] K. Paliwoda and J.W. Sanders. An incremental specification of the sliding window protocol, *Distributed Computing* **5** (1991) 83–94.

[PB61] W.W. Peterson and D.T. Brown. Cyclic codes for error detection, *Proceedings of the IRE* **49**(1) (January 1961) 228–235.

[PJ93] D. Peled and M. Joseph. A compositional approach for fault tolerance using specification transformation, in: *Proc. Parallel Architectures and Languages Europe (PARLE) '93*, Lecture Notes in Computer Science **694** (Springer-Verlag, 1993) 173–184.

[Peleska91] J. Peleska. Design and verification of fault tolerant systems with CSP, *Distributed Computing* **5** (1991) 95–106.

[RLT78] B. Randell, P.A. Lee, and P.C. Treleaven. Reliability issues in computing system design, *ACM Computing Surveys* **10**(2) (June 1978) 123–165.

[RR86] G. Reed, A. Roscoe. A timed model for Communicating Sequential Processes, in: *Proc. 13th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science **226** (Springer-Verlag, 1986) 314–323.

[Schepers93a] 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.

[Schepers93b] 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-Verlag, 1993) 91–110.

[Schepers94] H. Schepers. Compositional reasoning about responsive systems with limited resources, *Real-Time Systems* (to appear).

[SC94] H. Schepers and J. Coenen. Trace-based compositional refinement of fault tolerant distributed systems, in: *Proc. 4th IFIP Int. Working Conference on Dependable Computing for Critical Applications* (Springer-Verlag, to appear).

[SG93] H. Schepers and R. Gerth. A compositional proof theory for fault tolerant real-time distributed systems, in: *Proc. 12th Symposium on Reliable Distributed Systems* (IEEE Computer Society Press, 1993) 34–43.

[SH93a] 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-Verlag, 1993) 197–208.

[SH93b] H. Schepers and J. Hooman. A trace-based compositional proof theory for fault tolerant distributed systems, *Theoretical Computer Science* (to appear).

[SS83] R.D. Schlichting and F.B. Schneider. Fail-stop processors: An approach to designing fault tolerant computing systems, *ACM Transactions on Computer Systems* **1(3)** (1983) 222–238.

[Schneider90] F.B. Schneider. Implementing fault tolerant services using the state machine approach: A tutorial, *ACM Computing Surveys* **22**(4) (1990) 299–319.

[Scott70] D. Scott. Outline of a mathematical theory of computation, in: *Proc. 4th Annual Princeton Conference on Information Sciences and Systems* (1970) 169–176.

[SMcD93] Q. Shi and J.A. McDermid. Constructing secure distributed systems using components, in: *Proc. 12th Symp. on Reliable Distributed Systems* (IEEE Computer Society Press, 1993) 143–152.

[Weber87] D.G. Weber. Formal specification of fault-tolerance and its relation to computer security, *ACM Software Engineering Notes* **14**(3) (1989) 273–277.

[WL+78] J.H. Wensley, L. Lamport, J. Goldberg, M.W. Green, K.N. Levitt, P.M. Melliar-Smith, R.E. Shostak and C.B. Weinstock. SIFT: Design and analysis of a fault-tolerant computer for aircraft control, *Proceedings of the IEEE* **66**(10) (October 1978) 1240–1255.

[WGS92] J. Widom, D. Gries, and F. Schneider. Trace-based network proof systems: Expressiveness and completeness, *ACM Transactions on Programming Languages and Systems* **14(3)** (July 1992) 396–416.

[Zwiers89] J. Zwiers. Compositionality, concurrency and partial correctness, *Lecture Notes in Computer Science* **321** (Springer-Verlag, 1989).

[ZCdR91] J. Zwiers, J. Coenen, and W.-P. de Roever. A Note on Compositional Refinement, *5th BCS-FACS Refinement Workshop* (Springer-Verlag, 1991) 342–366.

Glossary

$\Box$	end of proof
$\triangle$	end of example
$\diamond$	end of definition
$\circ$	end of axiom or lemma
$=$	semantic equality
$\equiv$	syntactic equality
$\emptyset$	the empty set
$\mathbb{N}$	the natural numbers (including 0)
$\mathbb{Q}$	the rational numbers
$\mathbb{R}$	the real numbers
$\lceil r \rceil$	the greatest integer smaller than or equal to real number r
$\lfloor r \rfloor$	the smallest integer greater than or equal to real number r
$A \cup B$	the union of the sets A and B
$A \cap B$	the intersection of the sets A and B
$A - B$	the difference of the sets A and B, the elements of A that are not an element of B
A^*	the finite sequences of elements of the set A
A^ω	the infinite sequences of elements of the set A
$\mathcal{P}(A)$	the powerset (the set of all subsets) of the set A
$\upharpoonright$	the projection operator
$\backslash$	the hiding operator

$\curvearrowleft$	the time shift operator
$\wr$	the failure hypothesis introduction operator
$:=$	assignment
$!$	output
$?$	input
$;$	the sequential composition operator
$\parallel$	the parallel composition operator
$/\!/$	the interleaving composition operator
$\ll . \gg$	the processor closure operator
$\perp$	the bottom state
$\dagger$	the symbol denoting undefinedness
$\prec$	prefix
$\trianglelefteq$	subsequence
T	the global time
t	the base time
x_0	the initial state value of program variable x
$\exists t \cdot \phi$	there exists a t such that ϕ holds
$\forall t \cdot \phi$	for all t it is the case that ϕ holds
$\models \phi$	the assertion ϕ is valid
$(\theta, \gamma) \models \phi$	the assertion ϕ holds for trace θ and environment γ
$\models P$ **sat** ϕ	the correctness formula P **sat** ϕ is valid
$\vdash \phi$	the assertion ϕ is derivable
$\vdash P$ **sat** ϕ	the correctness formula P **sat** ϕ is derivable
$\circ$	the relational composition operator
$[\,.\,]$	the weakest precondition operator
$\leadsto$	the leads-to operator
iff	if, and only if
min	minimum
max	maximum
sat	satisfies

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

Samenvatting

Het is niet eenvoudig om eigenschappen te bewijzen van gedistribueerde systemen bestaande uit componenten die mogelijkerwijs defect raken. Immers, dergelijke bewijzen moeten er rekening mee houden dat een onbetrouwbare component op ieder willekeurig moment mankementen kan vertonen. Nu gedistribueerde systemen steeds meer gebruikt worden voor kritieke toepassingen, zoals het besturen van vliegtuigen en het bewaken van patiënten, worden de betrouwbaarheidseisen echter zwaarder en zwaarder.

Een component die niet aan zijn specificatie voldoet, dat wil zeggen faalt, veroorzaakt fouten. Een systeem wordt foutentolerant genoemd als het aan zijn specificatie voldoet ook al falen er componenten. Omdat het falen van willekeurig veel componenten in het algemeen niet getolereerd kan worden, geeft een fouthypothese aan van welke componenten het falen te tolereren is.

In dit proefschrift formuleren we een bewijsmethode voor foutentolererende gedistribueerde systemen. Deze methode is compositioneel in de zin dat de specificatie van een samengesteld systeem afgeleid kan worden uit de specificaties van de componenten, zonder dat daarbij de interne structuur van die componenten bekend hoeft te zijn. Een dergelijke methode maakt het mogelijk dat delen van een systeem afzonderlijk ontworpen worden.

We modelleren een gedistribueerd systeem als een netwerk van processen. Processen hebben geen gemeenschappelijke variabelen maar communiceren via kanalen. Voor het formaliseren van foutentolerantie abstraheren we van de interne toestand van processen en concentreren ons op het van buitenaf te observeren communicatie gedrag. Om te specificeren dat een proces P aan een eigenschap ϕ voldoet gebruiken we formules van de vorm P **sat** ϕ.

In het geval van foutentolerantie worden drie soorten gedrag onderscheiden: normaal, exceptioneel en catastrofaal. Normaal gedrag is het gedrag dat voldoet aan de specificatie. Een faalhypothese, die stipuleert hoe fouten het gedrag beïnvloeden, verdeelt het abnormale gedrag in exceptioneel en catastrofaal gedrag. In tegenstelling tot catastrofaal gedrag vertoont exceptioneel gedrag afwijkingen die getolereerd dienen te worden. Het normale en het exceptionele gedrag vormen het acceptabele gedrag.

In dit proefschrift formaliseren we de faalhypothese van een proces als een relatie tussen zijn normale en zijn acceptabele gedrag. We introduceren de constructie $P \wr \chi$ (lees "P onder χ") om aan te geven dat we voor het proces P

de faalhypothese χ beschouwen. Deze constructie stelt ons in staat onbetrouwbare processen te specificeren. Op deze manier kunnen we ook onbetrouwbare netwerken specificeren en zodoende fouthypotheses formaliseren. Onze aanpak is geschikt voor het bestuderen van willekeurige foutentolerantietechnieken omdat deze het mogelijk maakt modulair te redeneren over acceptabel gedrag: het abnormale gedrag van proces P dat acceptabel is voor wat betreft de faalhypothese χ is het normale gedrag van het onbetrouwbare proces $P \wr \chi$.
Het basisformalisme is het formalisme van Hoofdstuk 3. Om de essentie van onze formalisatie van foutentolerantie te benadrukken, laten we voor deze theorie de factor tijd buiten beschouwing. In Hoofdstuk 4 laten we zien hoe deze theorie gebruikt kan worden om het proces te classificeren dat, gegeven een faalhypothese, aan een zekere specificatie voldoet. Ook leiden we af onder welke faalhypotheses een gegeven proces nog steeds aan zijn specificatie voldoet.
In Hoofdstuk 5 breiden we de bewijstheorie van Hoofdstuk 3 uit om over het gedrag in de tijd te redeneren. Dit is nodig omdat voor tijdkritische, zogenaamde real-time, systemen betrouwbaarheid ook inhoudt dat een systeem tijdig reageert. Het uitgebreide formalisme gaat uit van maximaal parallellisme, dat wil zeggen, de aanname dat elk proces een eigen processor heeft.
In de praktijk hebben we vaak te maken met meerdere processen per processor. De executievolgorde wordt in zo'n geval bepaald door de prioriteiten van de diverse acties. In Hoofdstuk 6 generaliseren we het model van Hoofdstuk 5 naar de situatie waarin meerdere processen op één processor uitgevoerd worden. In dit formalisme hoeven prioriteiten niet constant te zijn; in het bijzonder mogen ze afhangen van de tijd gedurende welke een proces al op zijn beurt wacht.
We bewijzen de geldigheid en de volledigheid van de diverse bewijstheoriën. Een bewijstheorie is geldig als elke afleidbare formule waar is; een bewijstheorie is volledig als elke ware formule afleidbaar is. Vergelijken we ons formalisme met formalismen voor louter normaal gedrag, dan valt op dat er slechts één bewijsregel, te weten de faalhypothese introductie regel

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

nodig is om het acceptabele gedrag van het onbetrouwbare proces FP te karakteriseren.

Stellingen

behorende bij het proefschrift

Fault Tolerance and Timing of Distributed Systems

Compositional specification and verification

van

Henk Schepers

1. Voor foutentolerantie zijn drie soorten gedrag van belang: normaal, exceptioneel en catastrofaal. Normaal gedrag is het gedrag dat voldoet aan de specificatie. Een faalhypothese, die stipuleert hoe fouten het gedrag beïnvloeden, verdeelt het abnormale gedrag in exceptioneel en catastrofaal gedrag. In tegenstelling tot catastrofaal gedrag vertoont exceptioneel gedrag afwijkingen die getolereerd dienen te worden. Het normale en het exceptionele gedrag vormen het acceptabele gedrag.

   De in dit proefschrift voorgestelde methode is geschikt voor het formaliseren van foutentolerantie technieken in het algemeen omdat het acceptabele gedrag op een modulaire manier gespecificeerd kan worden: het abnormale gedrag van proces P dat acceptabel is voor wat betreft de faalhypothese χ is het normale gedrag van het onbetrouwbare proces $P \wr \chi$.

2. Het is een bekend gegeven dat, als men het gedrag van een systeem in de tijd beschouwt, een compositionele semantiek alleen gedefinieerd kan worden als het onderliggende model informatie bevat omtrent de bereidheid van een proces tot communiceren. Als men geïnteresseerd is in foutentolerantie is het cruciaal dat men ook kan redeneren over een dergelijke bereidheid van de omgeving van een proces. Hiervoor hoeft het onderliggende model echter niet uitgebreid te worden.

   Hoofdstuk 5 en 6 van dit proefschrift.

3. In [PS91] wordt de correctheid van het Alternating Bit Protocol bewezen in het geval de factor tijd buiten beschouwing gelaten wordt. Door een onnatuurlijke specificatie van de ontvanger, waarin ervan wordt uitgegaan dat de reeks van ontvangen berichten aan zekere condities voldoet, hoeft in [PS91] de persistentie eigenschap van het Alternating Bit Protocol niet bewezen te worden. In Hoofdstuk 3 van dit proefschrift wordt deze eigenschap expliciet bewezen.

[PS91] K. Paliwoda and J.W. Sanders. An incremental specification of the sliding window protocol, *Distributed Computing* **5** (1991) 83–94.

4. In [Hooman92] wordt, in het geval één processor meerdere processen executeert, de prioriteit van een actie bepaald door de direct omvattende prioriteitstoekenning. Omdat zodoende de toekenning van een prioriteit aan een samengesteld proces de relatieve belangrijkheid van de delen kan verstoren mogen dergelijke toekenningen in [Hooman92] niet worden genest.

   Deze onpraktische beperking is niet nodig als het nesten van prioriteitstoekenningen een cumulatief effect heeft, zoals in Hoofdstuk 6 van dit proefschrift beschreven is.

   [Hooman92] J. Hooman. Specification and compositional verification of real-time systems, *Lecture Notes in Computer Science* **558** (Springer-Verlag, 1992).

5. Veel producenten menen nog steeds foutentolererende apparatuur te kunnen leveren zonder een bewijs van correctheid te overleggen.

6. Het veiligheidsverhogende effect van een anti-blokkeer remsysteem gaat in veel gevallen verloren doordat de bestuurder krappere marges in acht neemt. Helaas is dit vaker het geval met betrouwbaarheidsverhogende maatregelen.

7. Het door het International Standardization Organization (ISO) gedefinieerde Basic Reference Model voor Open Systems Interconnection (OSI) staat relaying boven de Network Layer niet toe. Dat ISO zichzelf niet serieus neemt blijkt uit het feit dat zij in het kader van OSI twee niet-compatibele Network Services tot standaard verheven heeft, te weten de Connection-Oriented Network Service en de ConnectionLess Network Service.

   H. Schepers, O. Rikkert de Koe, G. Havermans and D. Hammer. LAN/WAN interworking in the OSI environment, *Computer Networks and ISDN Systems* **23** (1992) 253–266.

8. Op de Kluizerweg in Leende staan een aantal autowerende obstakels. Het levensverlengende effect dat hiervan voor fietsers uitgaat wordt echter grotendeels teniet gedaan door de misleidende waarschuwingsborden.

9. De stilte rondom het Europees Monetair Systeem doet vermoeden dat het inderdaad voornamelijk profijtelijk was voor de valutahandelaren.

10. Zolang de naslagwerken bier definieren als alcoholhoudende drank is alcoholvrij bier een contradictio in terminis.

11. In het belang van het milieu dient bij het kopen van nieuwe autobanden het inleveren van de versleten exemplaren niet langer op vrijwillige basis te geschieden.

12. Om te voorkomen dat van de digitale supersnelweg slechts een digitaal karrespoor overblijft is het noodzakelijk dat de politieke besluitvorming op technisch inzicht gebaseerd wordt en niet hopeloos achter de ontwikkelingen aanloopt.