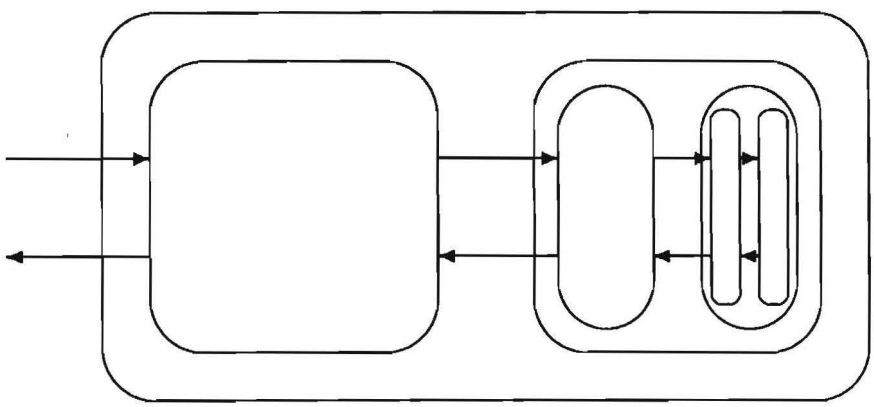
Compositionality, concurrency and partial correctness : proof theories for networks of processes, and their connection

Compositionality, Concurrency and Partial Correctness:

Proof theories for networks of processes, and their connection



J. Zwiers

Compositionality, Concurrency and Partial Correctness:

Proof theories for networks of processes, and their connection

Voor mijn ouders

Compositionality, Concurrency and Partial Correctness:

Proof theories for networks of processes, and their connection

PROEFSCHRIFT

TER VERKRIJGING VAN DE GRAAD VAN DOCTOR
AAN DE TECHNISCHE UNIVERSITEIT EINDHOVEN,
OP GEZAG VAN DE RECTOR MAGNIFICUS, PROF. DR. F.N. HOOGE,
VOOR EEN COMMISSIE AANGEWEZEN DOOR HET COLLEGE VAN
DEKANEN IN HET OPENBAAR TE VERDEDIGEN OP
VRIJDAG 12 FEBRUARI 1988 TE 16.00 UUR

DOOR

Jakob Zwiers

GEBOREN TE LEIDEN

Dit proefschrift is goedgekeurd
door de promotoren

prof. dr. Willem-P. de Roever
en
dr. Peter van Emde Boas

Corrigenda

Replace "$\mathit{Kern}(t_0, \psi)$" by "$\mathit{Kern}(\mathbf{c}, t_0, \psi)$" on pages 141,142,169,173,202,249. Also omit the line "Let $\mathbf{c} = \mathit{hchan}(\psi)$" at the bottom of page 141.

Add on page 279 the following case:

- Kernel

$$\frac{(\varphi)\ m\ (\psi)}{(\varphi \wedge h|\mathbf{c} = t_0|\mathbf{c})\ \mathit{Kern}(m)\ (\mathit{Kern}(\mathbf{c}, t_0, \psi))}\ (\text{Kern})$$

transforms into:

$$\frac{\dfrac{\dfrac{\psi[\bot] : \{\varphi[\top]\}\ m\ \{\psi[\top]\}}{\mathit{Kern}(\mathbf{c}, t_0, \psi[\bot])\ :\ \{\varphi[\top] \wedge h|\mathbf{c} = t_0|\mathbf{c}\}\ \mathit{Kern}(m)\ \{\psi[\top]\}}\ (\text{Kern})}{\mathit{Kern}(\mathbf{c}, t_0, \psi[\bot])\ :\ \{\varphi[\top] \wedge h|\mathbf{c} = t_0|\mathbf{c}\}\ \mathit{Kern}(m)\ \{\psi[\top] \wedge \mathit{Kern}(\mathbf{c}, t_0, \psi[\bot])\}}\ (\text{Clos. adap.})}{\mathit{Kern}(\mathbf{c}, t_0, \psi)[\bot]\ :\ \{(\varphi \wedge h|\mathbf{c} = t_0|\mathbf{c})[\top]\}\ \mathit{Kern}(m)\ \{\mathit{Kern}(\mathbf{c}, t_0, \psi)[\top]\}}\ (\text{Consequence})$$

ACKNOWLEDGEMENTS

I thank Willem-Paul de Roever for his cooperation and his many constructive comments.

I thank Peter van Emde Boas, Jozef Hooman and Ernst-Rudiger Olderog, for carefully reading the manuscript.

In Dr. Ir. Henk Bosma I thank the management of Philips Research Laboratories for generously allowing me to finish this thesis.

And finally I thank my wife Elisabeth for typing the manuscript, and, most of all, for providing me all the support I needed.

Contents

Chapter 1

Introduction

1.1 Summary and perspective

The hierarchical decomposition of programs into smaller ones is generally considered imperative to master the complexity of large programs. The impact of program decomposition on the specification and verification of parallel executing programs is the subject of this thesis. Two important yardsticks for verification methods, viz. those of compositionality and modularity, are made precise. Within this context, three methods for specifying the observable behavior of communicating processes, and their associated proof systems are considered, and proven sound and complete in various senses, as discussed below.

The problem of verifying large programs was already recognized by Alan Turing in 1949. His paper "Checking a large routine" [Turing] opens with the following sentences:

> "How can one check a routine in the sense of making sure that it is right? In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows."

Turing's idea is to reduce a *global* statement, about the whole program, in one step to a number of *local* checks for the atomic actions that constitute the program. The idea is embodied in the Floyd/Naur method for program verification, named after its inventors R. Floyd and P. Naur, cfr. [Floyd], [Naur].

In 1965, E.W. Dijkstra [Dijk] improved upon this by proposing a way of *gradually* decomposing the verification problem of a program. He suggests

to develop a program in a top down fashion, where a program is decomposed into smaller *programs*, i.e. possibly *composed* entities, rather than into atomic actions. The specification for the whole program is to be verified on the basis of specifications for the programs that are the constituent components of the whole. The development and verification of subprograms then proceeds in essentially the same way, until no further decomposition is necessary. This top down development results in hierarchically structured programs. The idea of hierarchical program structure already occurs in [Wijn].

A hierarchical program structure can result in several ways, not only by a top down strategy, but also as the result of bottom up composing small programs into larger ones, or by a mixed development strategy, or even from a posteriori decomposing an already existing program for the sake of its analysis. [Dijk2]

Regardless of *how* a certain hierarchical structure has been achieved, the principle of *compositional program verification* asserts that:

> the specification of a program should be verified on the basis of specifications of its constituent components, without knowledge of the interior construction of those components.

An important step forwards was made when the idea arose to describe the decomposition of a program into subprograms by means of its syntactic phrase structure. This idea formed the basis for a *syntax directed* reformulation of the Floyd/Naur method for a class of simple sequential programs, by Hoare in 1969 [Hoare].

Section 1.5 contains a rigorous mathematical characterization of the compositionality principle for programs with a hierarchical structure that follows their syntactic phrase structure. According to this definition, Hoare's system is compositional, whereas the Floyd/Naur method is not.

Among the first proof systems for *parallel* programs are the systems by Owicki and Gries [OG], and by Apt, Francez and de Roever [AFR]. The rules for parallel composition of processes in these systems are *not* compositional, since in [OG] a so called "interference freedom test", and in [AFR] a "cooperation test" must be applied to the proof outlines of the components of a parallel construct, that is, to a text that contains the program text of those components, thereby revealing their internal structure.

Around 1979, T. Janssen and P. van Emde Boas already stressed the importance of the compositionality principle in the context of program semantics and verification [JanEmBo].

The "Concurrent Hoare Logic" published by Lamport in 1980 [Lam1] *is* compositional. Unfortunately it has the disadvantage that the verification of a program against a specification is reduced to verifying the *same* specification for the components of the program. So there is no reduction in complexity of specifications. Moreover, it seems to enforce a strict top down development strategy for programs, for it would be rather surprising if, for a bottom up approach, two different and independently developed program modules would have the *same* specification. And if we want to combine those modules into a larger program, then, for Lamport's system, they *must* have the same specifications.

Early compositional proof methods for communication based parallel programming were formulated by Zhou Chao-Chen and Hoare [ZH], and by Misra and Chandy [MC]. Both approaches have in common that the idea of a *communication history* plays a central role in the specification of processes. This is also the case in Hailpern's work [HailOw]. Such a history essentially describes which communications occurred in which order up to some given moment of time. Communication histories, or traces as they are sometimes called, are also used in [NDO], [Jon], [JoPa], [OH], [Olderog2], [Pratt], [Rem], [SouDa], [Widom], and, already in 1977, by Yonezawa [Yon].

A compositional trace based proof system in the style of Misra and Chandy was published in 1983 by Arie de Bruin, Willem Paul de Roever, and myself, in [ZBR]. The system consists of rules for a type of process specifications closely related to those used by Misra and Chandy in [MC]. The language constructs axiomatized were those of the language **DNP** ("Dynamic Networks of Processes"), which is a simplification of a language with the same name studied by de Bruin, Böhm in [BrBö]. (The original **DNP** had to be described by means of continuation semantics, see [Bruin], which complicates a compositional style of reasoning considerably). In essence the language concept goes back to Kahn and McQueen [KaQu].

An important notion in this context is that of the *compositional completeness* of a proof system, roughly described as the requirement that whenever a program satisfies a certain specification it should be possible to *infer* the validity of that specification, by means of the axioms and rules of the proof system, in a compositional way.

At first it was thought that the system of [ZBR] would be necessary *in*complete. For **DNP** does possess the essential characteristics of those languages for which E. Clarke had proven that no (relatively) complete axiomatization can exist [Cla]. The side condition on the expressiveness for finite interpretations

of the language used for assertions in specifications was not satisfied, since natural numbers were intrinsically required on account of the "length" operator for histories, and so it seemed that some more work was necessary to prove the incompleteness result. Our aim then became to show completeness for the *finite* processes of **DNP**, as this was not excluded by the results of Clarke. The specifications of [ZBR] are quite complicated. Not only do they contain pre and postconditions in the style of Hoare formulae, but also an assumption and commitment on the communication behavior are included. (The latter was anticipated in [Hailpern]). It must be said however, that, although more complicated, the Misra and Chandy's rule turns out to be superior when it comes to *actual* verification of parallel programs. Now from the interpretation of these specifications, and also from the form of the proof rules in [ZBR], it can be seen that in many aspects the assumption commitment pair acts as an *invariant* of the communication history of a process. The completeness question could be simplified by reformulating the proof system so as to make this invariant character *explicit*. The main difference between the two systems is in the much simpler interpretation of specifications for the Invariant system, and, as a consequence, in the rule for parallel composition. The resulting proof system is the direct ancestor of the so called Invariant system that is one of the three proof systems studied in this thesis. The following results were achieved:

- The Misra Chandy approach has a source of incompleteness, due to the simple fact that what one *assumes* about the communication history can obviously be *committed* too. The axiomatizations by Misra and Chandy, nor the system of [ZBR] did include a corresponding axiom, and so both are incomplete. This fact was observed independently by Van Nguyen [Nguyen]. No such axiom is needed for the Invariant system.

- The axiomatization of parallel composition by means of the Misra Chandy rule as well as the corresponding rule for the Invariant system are not sufficient to obtain a complete system. Rather a new axiom, now called the prefix invariance axiom, has to be added to the system.

- Clarke's result does *not* extend to the system of [ZBR], or to the Invariant system. Rather the Invariant system could be shown to be *arithmetically complete* for the full language, including recursion. These results appear in the report [ZRE2]. The ICALP paper with the same title, [ZRE], contains the part of the report that is concerned with the Invariant system only.

The problem with combining specifications in a bottom up development that we signaled above for the system of Lamport is not exclusive for that system. This can be understood as follows. Many proof systems include rules that enable one to reduce the verification of an a priori given specification for a composed program to the verification of specifications for the constituent components. A well known example is the rule for sequential composition in the Hoare's logic. Such rules favor a top down development, for in this case the specification for the parts is designed only after the specification for the whole is known. For a bottom up approach on the other hand, one must derive a specification for the whole from arbitrary a priori given specifications *for the parts*. If the proof rules do not allow one to combine arbitary specifications for the parts, then the given specifications for the parts have to be *adapted* into another form until the rules do become applicable. Hence one might draw the conclusion that what we need is proof rules that combine arbitrary specifications for parts into a specification for a larger program composed out of these parts. Obviously this favors a bottom up development.

Such bottom up style rules have been given by Barringer, Kuiper and Pnueli in [BKP], for the specification of parallel programs by means of temporal logic. However, the system of [BKP] was obtained only by the introduction of a new temporal operator for each language construct. The result is that specifications constructed for a program in this way can be as complex as the semantics of the program itself. Moreover, the necessity for adaptation of specifications remains, since in a top down development there might not exist appropriate specifications for the parts of a composite program that will result immediately in the desired specification for the whole by applying a bottom up style rule. In such cases one must additionally prove that the specification that has been obtained for the whole can be adapted to the desired one.

The idea of adaptation of specifications turned out to be a key notion for a *modular* approach to program development. By modularity we mean the following. We have some program S, specified by some given specification *spec*, and composed out of parts $P_1, \ldots, P_n$, each specified by a given specification $spec_i(P_i)$. The "modules" $P_1, \ldots, P_n$ are treated as black boxes, that is, without (known) inner structure. We call the given decomposition of the program S correct if the black box specifications $spec_i(P_i)$ logically imply the correctness of S with respect to its specification *spec*.

A proof system is called *modular complete* if, for *any* correct decomposition as described, one can *formally deduce* the specification *spec* for S under the

hypotheses that the black boxes satisfy their corresponding specification.

The modular completeness notion is applicable for a top down as well as for a bottom up development. For a top down approach, the inner structure of the black boxes is literally unknown, since they are not implemented at that stage. For a bottom up approach, the black box mechanism is used to abstract from the innards of the already implemented modules.

Why is modular completeness different from compositional completeness? Compositional completeness requires, for a given specification for the whole program, the *existence* of appropriate specifications for the parts, such that the specification for the whole can be deduced from them. Modular completeness asserts that such a deduction can be found for *a priori given* specifications for the parts.

A third completeness notion is that of *adaptation completeness*. It asserts that if some given specification $spec(P)$ for a black box P logically implies another specification $spec'(P)$for that same black box, then the proof system admits a formal deduction of that fact. Adaptation completeness can be seen as a special, restricted form of modular completeness. In section 1.5 it is proven that the combination of compositional completeness together with adaptation completeness implies modular completeness of a proof system.

The necessity of adapting specifications is well known from the investigations concerning the completeness of Hoare style proof systems dealing with (sequential) recursive procedures cfr. [Apt], [Bakker], [Gorelick]. Gorelick [Gorelick], succeeded in proving the completeness for a Hoare style proof system by adding various substitution rules and an invariance rule to the usual Hoare system for while-programs. These extra rules were used to adapt Hoare style specifications of a restricted form, called *most general formulae* in [Gorelick]. A special rule, called the Rule of Adaptation, was proposed by Hoare in [Hoare4], also for a proof system dealing with recursion. By means of this rule one can "adapt" *arbitrary* Hoare specifications. Olderog investigated the Rule of Adaptation and proved essentially that the rule can replace the "extra" rules employed by Gorelick, but also that it is *not* sufficiently strong to achieve adaptation completeness [Olderog3].

An interesting topic in this context is the type of adaptation rules that are based on certain closure properties of the semantic domain of denotations for processes. For communication based languages, *prefix closedness* with respect to communication histories is such a property, that is always satisfied by **TNP** processes, but not by specifications for such processes. The adaptation problem caused by this is treated extensively in the work of J.

Widom [Widom], [WiPa]. The solution proposed by Widom is based on an enrichment of the specification language by means of certain temporal logic operations. With respect to our proof systems we remark the following. First of all, our systems are axiomatizations, not of a programming language, but of a so called *mixed formalism*, where programs and specifications appear on the same footing. As a consequence, our processes need *not* denote prefix closed sets. And so it turns out that the corresponding adaptation problem has vanished. Now this is only half of the story, since one may object that we have blurred the distinction between "real" programs and specifications instead of having solved anything. Therefore we included a so called *kernel* operation within our mixed formalism. This operation allows for the distinction between those black boxes satisfying the prefix closedness property and those that need not. The adaptation problem is then solved by the inclusion of (ordinary) proof rules for the kernel operation. We could do so without extending our language of specifications.

For the proof systems considered here, the focus of attention is whether these systems are compositional complete, adaptation complete or even modular complete. The origin and characteristics of the three proof systems are as follows. As already stated above, our original aim was to prove what we now call compositional completeness for the Misra/Chandy and Invariant systems, specialized for the programming language **DNP**. Soon it was realized that such completeness considerations could be much clearer if the language **DNP** was simplified. The result was the language **TNP** , for "Theoretical Networks of Processes", which is simpler, but at the same time more powerful than **DNP**. **TNP** is much in the spirit of languages like CCS [Mil] or TCSP [OH] except that it combines a state based approach with a communication based approach to programming.

For TCSP several proof systems have been developed by Olderog and Hoare [OH], [Olderog1], [Olderog2]. Some of these deal with program properties, such as absence of deadlock, that are not considered in this thesis. The system dealing with communication histories (only) seemed a good candidate for a comparison with our own Invariant system. To this end we developed a new proof system for **TNP**, for a type of specifications called SAT formulae. This type of formulae can be seen as the generalization of the formulae used by [Olderog2] to the combined state and communication based approach. Essentially they express the inclusion between a program and its specification. We discovered how to represent formulae of the Hoare system by means of equivalent formulae of the SAT system, and vice versa, and even how to transform complete deductions within one system to corresponding

deductions in the other system in a canonical way. In fact it proved to be simpler to introduce a third proof system as a sort of intermediate system in between the SAT system and the Invariant system that must be seen as an alternative formulation of the Invariant system. We named it the Hoare system, since the form and interpretation of the formulae of this system is exactly that of classical Hoare style formulae for sequential programs, except that the pre- and postconditions of our formulae are not assertions on states, but rather on the combination of communication histories and states. (An idea going back to V. Pratt [Pratt]).

The proof transformations yield much insight in the structure and interrelationship of the three proof systems. Maybe the most important difference between SAT style and Hoare style reasoning is in the number and complexity of proof rules for adaptation of specifications. The SAT system is clearly superior in this respect, since it contains very few and only very simple adaptation rules. The transformation of Hoare style deductions into SAT style deductions reveals why no Gorelick type adaptation rules are needed for the SAT system: All such rules from the Hoare system transform essentially into applications of the consequence rule of the SAT system. Another important conclusion that follows from this proof transformation is that it is feasible to treat Hoare style specifications and Hoare style reasoning as *abbreviating* certain SAT style specifications and reasoning. Thus we can embed the Hoare system as a subsystem of the SAT system. The same can said about the relation between the Hoare and Invariant systems: The Invariant specifications and reasoning abbreviate corresponding Hoare style specifications and reasoning.

One should not draw the conclusion from this that the SAT system is also superior when it comes to the actual verification of concrete programs. For instance, the absence of certain adaptation rules in the SAT system only means that where the Hoare style proof applies one of these adaptation rules, the SAT style proof applies the consequence rule, and in fact both rely on one and the same underlying logic principle. A second, equally important difference is the treatment of the classical sequential programming constructs, which is more complicated within the SAT system than in the Hoare and Invariant system. The Invariant system is in fact a generalization of classical Hoare logic, and for purely sequential, non communicating **TNP** programs, the system simply coincides with Hoare style logic.

We end with a description of the organization of this monograph.

Chapter 1 starts, after this summary, with an example of the development of

a parallel sorting algorithm, thereby introducing many language constructs, specification methods and verification principles of later chapters. Section 1.5 treats the concepts of compositional and modular completeness.

TNP is introduced in chapter 2, after the definition of **DNP**. We indicate how to translate **DNP** programs into **TNP**, whereafter **DNP** is no longer used in this thesis.

The (denotational) semantics of **TNP** is developed in chapter 3. An interesting point here is that there are remarkable similarities between the treatment of state transformations and communications. Another point that deserves attention is the treatment of parallelism by means of a *projection* operation. Although equivalent to the usual interleaving semantics, the description by means of projections is mathematically more elegant.

Chapter 4 introduces the *assertion language*, used to express properties of communication histories and states, and the three types of process specifications. The properties of, and relationship between, the three specification methods is treated in depth. Also the language of *mixed terms* is defined here.

Chapter 5 introduces the three proof systems. The soundness of the SAT system is proven directly from the semantic definitions. The Hoare system is shown to be sound essentially by transforming Hoare style deductions into SAT style deductions. Similarly, the Invariant system is proven sound by transforming Invariant style deductions into Hoare style deductions.

Chapter 6 addresses the completeness question for the SAT system. The system is shown to be compositionally complete and adaptation complete, and therefore modular complete.

Chapter 7 proves the compositional completeness of the Hoare system by showing that SAT style deductions can be transformed, in a canonical way, into corresponding Hoare style deductions. The Gorelick type of adaptation rules are used extensively in this proof. A so called *strong* adaptation rule is introduced in chapter 5 which, contrary to Hoare's Rule of Adaptation, results in adaptation completeness. The proof can be found in chapter 7. Finally, we prove the compositional completeness of the Invariant system, by means of a proof transformation from Hoare style deductions into Invariant deductions.

1.2 Specification and Construction of Processes

The object of study of this thesis is the specification, construction and verification of parallel executing processes. In this introductory section we would like to give an overview by showing the development of a parallel sorting algorithm. This gives us the opportunity to introduce the main programming language constructs, various specification methods, and some of the verification rules in an informal setting.

If one wants to reason in a secure way about processes then the first step is to introduce *formalized languages* to describe processes and specifications. Our main formal language to describe processes is called **TNP**. Its syntax and semantics are provided in the chapters 2 and 3. Thereafter, in chapter 4, we introduce formal languages to describe and specify process behavior.

The language **TNP** , for Theoretical Networks of Processes, evolved from an earlier language called **DNP** , for Dynamic Networks of Processes. The starting point for **DNP** was the concept of a dynamically changing networks of processes. A parallel network consists of a number of *processes* executing in parallel and *communicating* messages along interconnecting channels. A process is a sequential program that can *expand* temporarily into parallel subnetworks. This can happen recursively since the so formed subnetworks can contain new copies of the original (expanded) process.

By generalizing, and at the same time simplifying, **DNP** the language **TNP** was designed. Whereas **DNP** resembles a procedure based parallel programming language, **TNP** is much more in the style of languages such as TCSP [OH].

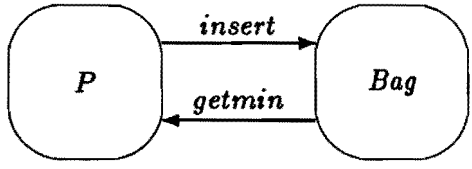
We shall introduce the main language constructs of **TNP** as we need them in the development of a parallel sorting algorithm, also known as "priority queue" or, as we shall call it, the "sorted bag".

The parallel execution of two processes P_1 and P_2 is denoted by $P_1 \parallel P_2$. The CSP notation $c!e$ is used to denote the *sending* of the message denoted by e along the channel named c. Similarly, $c?x$ is a command that *requests* some message along c and stores it into variable x. Communication is *synchronous*, that is, the sender and receiver of a message must cooperate and, conceptually, a message is received at the same time as it was sent.

Example 1.1

We give a picture of the network $P \parallel Bag$, consisting of two processes named P and Bag. The two component processes are connected by means of chan-

nels named *insert* and *getmin*. Process P can send messages to *Bag* along channel *insert* by executing *insert*!*e* commands. For such communications to occur the *Bag* process must execute corresponding commands of the form *insert*?x. Similarly a communication along *getmin* occurs if P executes a *getmin*?x command in cooperation with a *getmin*!*e* command of *Bag*.



□

We want to *specify* the intended behavior of processes and to *verify* that the specification is met by some proposed implementation. For instance a description of the intended behavior of the *Bag* process could be the following:

Example 1.2

" *Bag* behaves as a so called sorted bag of values. Bags are also called multisets since they are like sets except that multiple copies of the same value can be member of a bag. New values can be inserted by sending them to *Bag* via the *insert* channel. A value can be requested via the *getmin* channel and this results in sending back and removal from the bag of the smallest element of the bag."

□

As is well known, a *Bag* process as described can be used to sort a given list of values. This is done by first inserting all elements of the list, followed by requesting an equal number of values.

We want to specify the *Bag* behavior in a rigorous way, rather than informally as in the English description above. Our formal description method is based upon the notions of *communication events* and *sequences* of such events called *traces* or *communication histories*.

A communication is an event that is described by a channel name and a communicated value. For instance, the event of communicating a value v via channel *insert* is described by the pair $(insert, v)$. A communication history h then, can be seen as a description of which values have been communicated along which channels in which order at some particular moment of

time. For instance, the history $h \stackrel{\text{def}}{=} < (insert, v), (insert, w), (getmin, v) >$ corresponds to the state where values v and w were sent, in that order, via *insert* followed by the communication of value v via *getmin*.

In essence a process specification can be seen as a formula of a (many sorted, first order) predicate logic that expresses the desired properties of the communication history h that should hold for all possible executions of the process, at any moment during the execution. Such a predicate is called an *assertion* about the behavior of the process.

We would like to give an impression of what a formal specification for the *Bag* process looks like. To this end we discuss a few aspects of the *language* of assertions.

Fundamental is the class of trace expressions. Examples are the *empty trace* ε, denoting the empty sequence of communications, and the *communication history* h, denoting the communications that have been performed by the process up to some moment during some execution of the process. The history h must be distinguished from ordinary trace valued *variables* t that denote some arbitrary sequence of communications, not especially related to those communications actually performed by the process. Operations on trace expressions te include the important *channel projection* operation $te|\{c_1, \ldots, c_n\}$. This denotes the subsequence of te consisting of all communications via the channels $\{c_1, \ldots, c_n\}$. The special case $h|\{c\}$ is often abbreviated as c, that is, a channel name, used as a trace expression, denotes the sequence of all communications via that particular channel. Operations that we shall use in the example below are $last(te)$, denoting the last communication of te, and $rest(te)$, denoting the sequence of all *but* the last communication of te.

The expression $last(te)$ is not a trace expression since it denotes a single communication rather than a sequence of communications. The channel name and the communicated value of a single communication α are referred to by the expressions $chan(\alpha)$ and $val(\alpha)$.

Assertions are many sorted first order predicate formulae. Here we only mention one such assertion, denoted by $te_1 \leq te_2$, which expresses that the sequence te_1 is an *initial prefix* of the sequence te_2.

The fact that one requires an assertion χ to hold for *all possible* executions of a process P is not expressed within the assertion as such. Rather this is the interpretation of the satisfaction relation between processes and assertions. This is expressed by the following formula:

P *sat* χ.

We call such formulae SAT specifications, to distinguish them from other types of specifications that we shall later on.

Example 1.3

We use the following notation for bags:

- The empty bag is denoted by $\emptyset$, the bag containing elements $e_1, \ldots, e_n$ by $[e_1, \ldots, e_n]$.
- The union and difference of bags B_1 and B_2 are denoted by $B_1 \oplus B_2$ and $B_1 \ominus B_2$.
- The least element of a bag B of ordered values is denoted by $min(B)$.

Let "$bag(te, c)$" denote the bag of all values of all communications via channel c that occur in the sequence denoted by trace expression te. Instead of $bag(h|\{c\}, c)$ we use the abbreviation $bag(c)$. Also we use $cont(c_1, c_2)$ as an abbreviation for $bag(c_1) \ominus bag(c_2)$, and $cont(te, c_1, c_2)$ as an abbreviation for $bag(te, c_1) \ominus bag(te, c_2)$, where te is some trace expression other than h.

Let us fix some arbitrary moment during the *Bag* execution. We want to express that for all *getmin* communications that occur in the sequence $h|\{insert, getmin\}$ reached thus far, the correct value, that is, the "current" least element at the moment in question, was sent back by the *Bag* process. Now assume that t is some prefix of $h|\{insert, getmin\}$ that ends with a *getmin* communication. We can express this assumption by means of the assertion

$$t \leq h|\{insert, getmin\} \ \wedge \ chan(last(t)) = getmin.$$

The "current contents" of the bag just before that last communication is given by the expression $cont(rest(t), insert, getmin)$. Therefore, the desired property is expressed by the assertion $\chi_{bag}(insert, getmin)$, defined as:

$$\chi_{bag}(insert, getmin) \stackrel{\text{def}}{=}$$

$$\forall t\Big((t \leq h|\{insert, getmin\} \wedge chan(last(t)) = getmin) \rightarrow$$

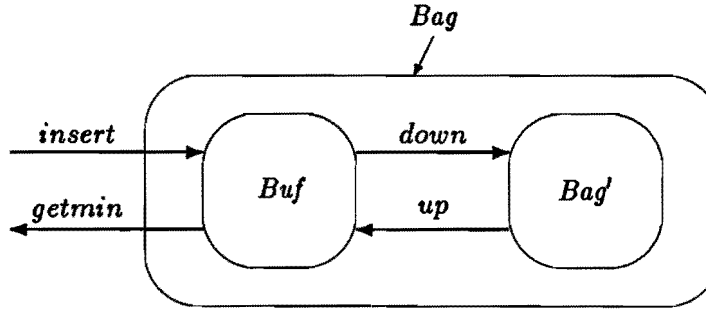
$$val(last(t)) = min(cont(rest(t), insert, getmin))\Big).$$

Finally, the *Bag* specification is the following SAT formula:

$$Bag \ sat \ \chi_{bag}(insert, getmin).$$

□

Apart from operating as a sequential communicating program a process can be built up as a subnetwork. For instance we might implement the *Bag* process as a buffer process *Buf*, keeping a few *Bag* elements amongst which the current least element, executing in parallel with a process *Bag'* keeping the rest of the *Bag* elements. This is shown in the picture below.



As can be seen the channels *insert* and *getmin* are connected to the *Buf* component of the network. The two components themselves are connected by means of channels *down* and *up*. The idea is that *Buf* on request will send its least element along *getmin*. The *Buf* process can also send elements "down" to, or request the least element contained in, the *Bag'* process. So the *Bag'* process must behave as a copy of the *Bag* process except that its channels have different names.

We would like to turn the intuitive descriptions of *Bag'* and *Buf* into formal specifications, and then *verify* that the parallel network *Buf* || *Bag'* does conform to the *Bag* specification. To this end we first discuss the verification principles for parallel processes.

A well known problem for the verification of parallel programs is that some specification that would be correct for a given process viewed in isolation, might be invalidated by the actions performed by other processes executing in parallel. In particular this is the case when a specification for a process *P* refers to a channel that can be modified by other processes without the cooperation of *P*. For instance, if a specification for the *Bag'* process would refer to the *insert* or *getmin* channel, then communications on these channels performed by the parallel executing *Buf* process, without cooperation

with Bag', could be in conflict with this specification. We shall avoid such specifications, for only then the soundness is implied of the following simple proof rule for parallelism:

Let P_1 *sat* χ_1 and P_2 *sat* χ_2 be specifications of the communication behavior of P_1 and P_2 that obey the restriction that the assertion χ_i only refers to communications via channels *connected to* process P_i, where $i = 1, 2$. Then from P_1 *sat* χ_1 and P_2 *sat* χ_2 one can infer the following specification for the parallel composition of the two processes:

$$P_1 \parallel P_2 \textit{ sat } (\chi_1 \wedge \chi_2).$$

We took care that the *Bag* specification does obey this restriction, since the only reference to the communication history is by means of the *projection* of this history onto the channels *insert* and *getmin*. A communication performed via some other channel than those two by some other process *does* affect the value of h, but it does *not* affect the value of $h|\{insert, getmin\}$.

To verify the correctness of the parallel composition $Buf \parallel Bag'$, one has essentially to prove that the *Buf* process preserves $\chi_{bag}(insert, getmin)$. For, during the expansion, the *insert* and *getmin* channels are connected to the *Buf* process. Of course the fact that *Buf* conforms to this behavior depends on the *assumption* that the process Bag', executing in parallel, behaves correctly as a bag with respect to the channels *down* and *up*, that is, satisfies the specification $\chi_{bag}(down, up)$. Therefore we choose the following *Buf* specification:

$$Buf \textit{ sat } (\chi_{bag}(down, up) \rightarrow \chi_{bag}(insert, getmin)).$$

By the rule for parallel composition we then infer that:

$$Buf \parallel Bag' \textit{ sat } ((\chi_{bag}(down, up) \rightarrow \chi_{bag}(insert, getmin)) \wedge \chi_{bag}(down, up)).$$

Since the assertion of this last formula clearly implies $\chi_{bag}(insert, getmin)$, We have shown that $Buf \parallel Bag'$ does satisfy the required assertion.

The fact that the Bag' process shows the behavior of a *Bag* process, except that its channels have different names, suggests that the Bag' process can be implemented as a parallel network itself and so on, ad infinitum. However, instead of such an infinitely deep nested static network, we prefer a *dynamic* network in which *Bag* starts as a *sequential* process and *expands* into a subnetwork only after elements have been inserted into the bag. Moreover, we can construct the Bag' process from a *recursive copy* of the *Bag* process itself, and so we will obtain a *Bag* process that has a variable but finite degree of nesting, dependent on the number of elements contained in it. To

denote that, within the program S that we shall use to implement *Bag*, recursive copies of *Bag* are allowed we use the *recursion construct* $\mu_z Bag.S$. Apart from recursion we need a few more language constructs. First of all, we must *rename* the channels of the recursive *Bag* copy into *up* and *down*. This can be denoted in our language **TNP** as $Bag\langle down/insert, up/getmin\rangle$. The effect is that messages sent along *getmin* by *Bag* appear from outside to be messages sent along the *up* channel. Similarly, messages sent to $Bag\langle down/insert, up/getmin\rangle$ via *down* are received by the *Bag* process via its *insert* channel. Our first approach to implement *Bag* can now be given:

$$\mu_z Bag \cdot\ insert?x\ ;\ (Buf \parallel Bag\langle down/insert, up/getmin\rangle)$$

The semicolon, as usual, denotes *sequential composition*. So first a value has to be received before the process expands into a subnetwork as indicated. The *Buf* process has access to the x variable in which the received value is stored. For instance it could send it back via the *getmin* channel to the outside world. But now a problem arises, for the recursive *Bag* copy *also* accesses the x variable. To avoid such "clashes" the so called *variable hiding* construct must be used to turn x into a *local* variable of the *Bag* process. This construct has the form $S\backslash x$ and it indicates that x is a local variable of process S. We have the same problem with channel names: the *up* and *down* channels *connected to* the recursive *Bag* copy have nothing to do with the channels of the same name *within* this copy. The problem is solved by using the *channel hiding* construct of the form $S\backslash c$ that denotes that c is an internal channel of S, not visible from outside.

We arrive at the following implementation for the *Bag* process:

$$\mu_z Bag \cdot \Big(insert?x\ ;(Buf \parallel Bag\langle down/insert, up/getmin\rangle)\backslash up, down\Big)\backslash x$$

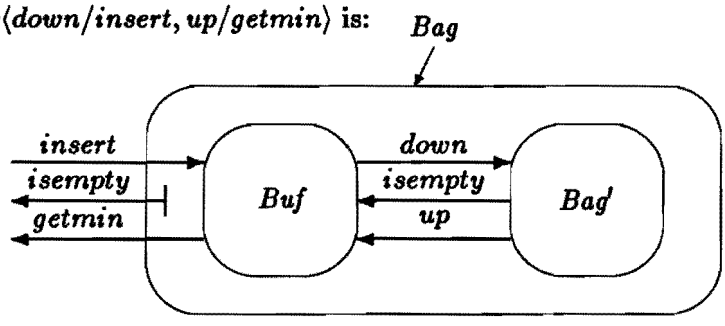
Our solution still has one defect. Incarnations of the *Buf* and the *Bag* processes never terminate, and so although new incarnations are created when necessary these incarnations do not disappear when they are no longer needed. Ideally a parallel network $Buf \parallel Bag\langle down/insert, up/getmin\rangle$ should vanish as soon as neither the *Buf* process nor the *Bag* process stores anymore values. So we cannot simply design the processes such that they terminate as soon as they store no value. To this end we *synchronize* the *Buf* and the *Bag* processes as follows. We include a new local channel called *isempty* between the two processes. If at any moment a process stores no values, it offers to communicate via *isempty* whereafter it will terminate. That is, if the other process stores no values either, and so is also able to communicate via *isempty*, then the parallel combination can terminate as a whole. On the other hand if some new value is received then the process stores it

and is no longer willing to communicate via *isempty*. In our program the commands *isempty*! and *isempty*? denote communication commands where no values are sent or received. That is, only *synchronization* is required. (Alternatively one might see this as ordinary communication where some immaterial value is passed.) This suggests our final implementation for the *Bag* process:

$$\mu_x Bag.\ \Big(isempty!$$
$$\textbf{or}$$
$$insert?x\ ;$$
$$(Buf \parallel Bag\langle down/insert, up/getmin\rangle)\backslash up, down, isempty\ ;$$
$$Bag$$
$$\Big)\backslash x.$$

The process has the *choice* between sending an *isempty* signal followed by termination and receiving a value via *insert* followed by an expansion. Note that if the subnetwork created by this expansion terminates itself then the whole process starts over again, since the execution of the subnetwork is followed by a recursive incarnation. In fact this second recursive call has the form of a "tail" recursion, and so could have been replaced by a loop construct.

A picture of this process after an expansion, where we have used *Bag'* to denote $Bag\langle down/insert, up/getmin\rangle$ is:



The vertical bar at the end of the outer *isempty* channel indicates that during the expansion this channel is not connected to any of the sub processes. Only *after* the subnetwork has terminated the *Bag* process is (again) able to send an *isempty* signal.

We adapt our *Bag* specification so as to express the fact that the *Bag* process only can terminate *after* it has sent an *isempty* signal, and that the contents of the bag is empty indeed, on termination.

In the assertion language, we use the symbol $\top$ to denote the characteristic predicate for computations that have *terminated*. Computations that do *not* satisfy this $\top$ predicate are called *unfinished*, and correspond to intermediate stages of the execution of a process. Unfinished computations are not the same as nonterminating computations; *every* execution passes through unfinished intermediate stages whether it eventually terminates or not.

The specification $\phi_{bag}(insert, getmin)$ is:

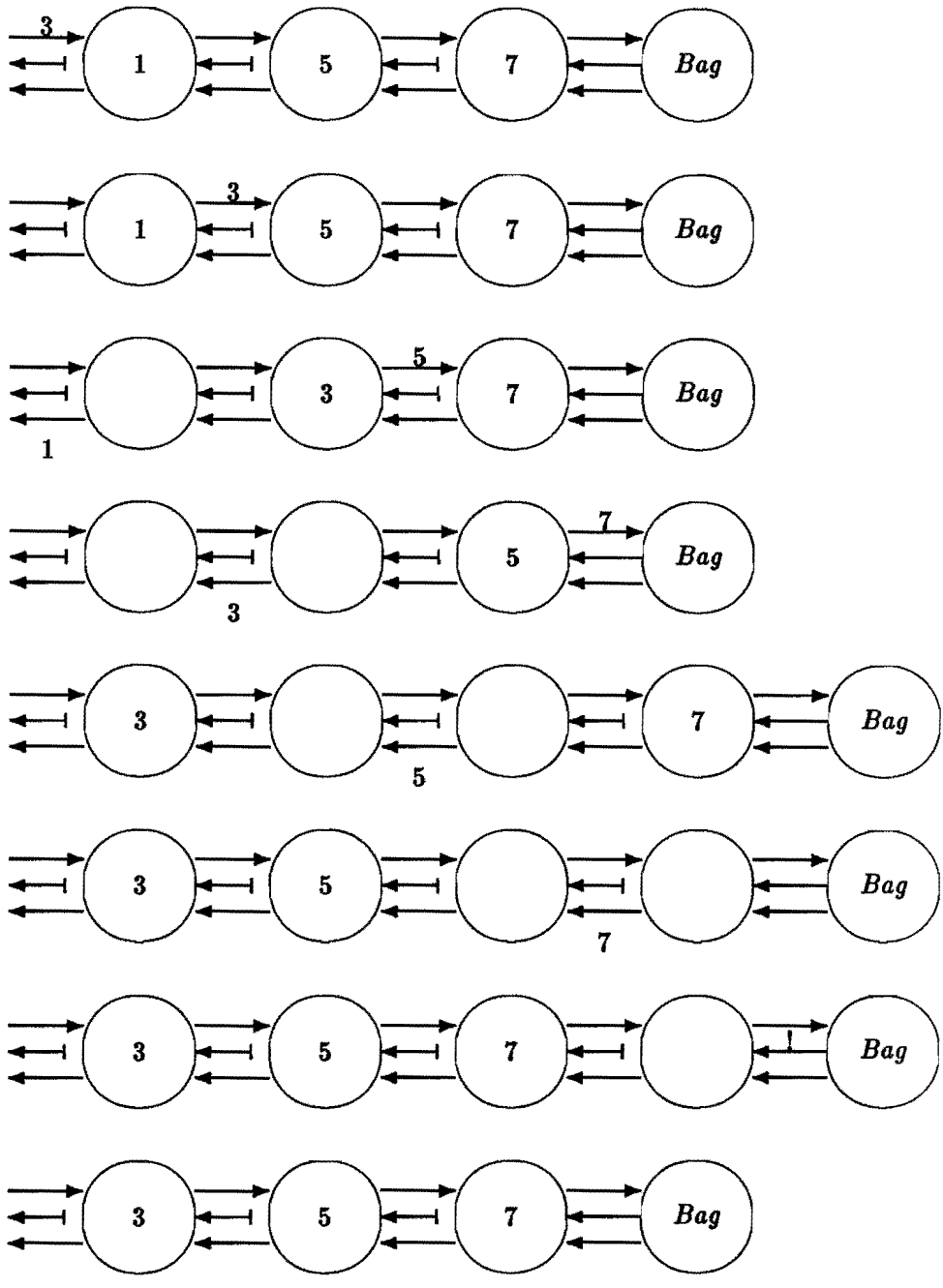
$$\phi_{bag}(insert, getmin) \stackrel{\text{def}}{=}$$

$$\chi_{bag}(insert, getmin) \wedge (\top \rightarrow (cont(insert, getmin) = \emptyset \wedge isempty \neq \varepsilon)).$$

The new specification still requires $\chi_{bag}(insert, getmin)$ to hold at *all* stages, and for terminated computations it requires the *Bag* contents to be empty, and the sequence of communications sent via *isempty* to be nonempty. The $\phi_{bag}(insert, getmin)$ does *not*, and even *cannot*, express that the process *must* terminate after it has sent the *isempty* signal. The reason is that we study specification methods for so called *safety properties*, and the necessity of termination is not one of those properties.

Example 1.4

The picture below shows an execution of the *Bag* process from a certain (unfinished) stage for which $cont(insert, getmin) = [1, 5, 7]$. We have only shown the *Buf* processes and the most deeply nested *Bag* process. We show also how a new value (3) is inserted, and how the current minimum (1) is requested. Note that, as seen from outside the minimum is requested strictly after the value 3 has been inserted, but that internally the *Bag* process is still busy with the insertion process at that time.



(the exclamation mark in the one but last picture indicates a synchronization action via an "*isempty*" channel.)

1.3 Hoare specifications and Invariant specifications

Now that we have proposed a top level design for *Bag*, we would like to verify that it satisfies the bag specification. We sketch how this verification could proceed. The proof systems that we develop in chapter 4 and 5 can be used for a *formal* verification, but for the moment we simply rely on the plausibility of certain verification principles. We shall encounter certain difficulties in connection with the *sequential* structure of the process. To resolve them we introduce new types of process specifications.

A well known verification principle, called Scott's induction rule, implies that, to verify that *Bag* satisfies the specification $\phi_{bag}(insert, getmin)$, it suffices to verify that the *body* of *Bag* does satisfy this specification, where we may assume " by induction" that the two occurrences of *Bag* processes within the body do satisfy $\phi_{bag}(insert, getmin)$. That is, under the hypothesis that *Bag* satisfies $\phi_{bag}(insert, getmin)$, we must verify the same assertion for the process:

($isempty!$
or $insert?x$;
  $(Buf \parallel Bag\langle down/insert, up/getmin\rangle)\backslash\{up, down, isempty\}$;
  Bag
)$\backslash\{x\}$

To do so, we need verification principles for the channel and variable hiding constructs. Now these are fairly simple: if a process satisfies a certain assertion χ, and the assertion *does not refer to* some channel c or variable x, then the assertion remains valid after we hide this channel or variable.

It will be clear that in general such specifications can be obtained only by using appropriate *projections* of the communication history.

Since our specification $\phi_{bag}(insert, getmin)$ does not refer to the variable x, our verification task for the body of *Bag* boils down to the verification of $\phi_{bag}(insert, getmin)$ for both components of the choice construct. We concentrate on the verification for the second component, that is, of:

$(insert?x\ ;\ (Buf \parallel Bag\langle down/insert, up/getmin\rangle)\backslash up, down, isempty;\ Bag)$
sat
$\phi_{bag}(insert, getmin)$.

The process of this last formula has the form of a sequential composition $P_1; P_2; Bag$, where P_1 is the process $insert?x$ and where P_2 is:

$$(Buf \parallel Bag\langle down/insert, up/getmin\rangle)\backslash up, down, isempty$$

First we must establish specifications χ_1 and χ_2, such that P_1 *sat* χ_1 and that P_2 *sat* χ_2.

This brings us to the *state transformer* aspect of our specifications. For instance, the P_1 process does not only *communicate* some value along the *insert* channel, but it also *modifies the process state* by storing the received value in x. As a consequence the specification χ_1 must describe this state transformation together with, and related to, the communication behavior. Actually this relationship is very simple; it is: $x = val(last(insert))$. This relationship between x and the value communicated via the *insert* channel exists only *after the* P_1 *process has terminated.* Therefore we must use again the $\top$ predicate, indicating a terminated computation. The specification for the P_1 process then becomes: P_1 *sat* $(\top \rightarrow x = val(last(insert)))$.

The last specification expresses a certain relationship between the communication behavior and the *final* state of computations. But in general one must also refer to the initial state in which the computation starts. For instance, for the process $down!x; up?x$ one must be able to specify that the value communicated via *down* is the *initial state value* of x, whereas the *final state value* of x is the value communicated via *up*. We use a "°" superscript to indicate an initial state value of some variable. For example,

$$(down!x; up?x) \textit{ sat } (val(last(down)) = x^\circ \wedge val(last(up)) = x)$$

Our next problem is to invent a specification for the parallel construct. Because this parallel process starts executing only after a value has been sent via *insert*, the original bag specification is no longer applicable.

If our goal would be the verification of the *Bag* version that we considered in the previous section just before the introduction of the *isempty* channel, then we would know how to proceed. In this case, any communication history of $Buf \parallel Bag\langle down/insert, up/getmin\rangle$, prefixed by a single communication $(insert, v)$ where v is arbitrary, should satisfy the *Bag* assertion $\phi_{bag}(insert, getmin)$. This means that the appropriate assertion for the parallel construct would be:

$$\forall v\Big(\phi_{bag}(insert, getmin)[(< (insert, v) > h)/h]\Big),$$

where $\cdots[(< (insert, v) > h)/h]$ indicates that the concatenation of the one element trace $< (insert, v) >$ with the history h must be substituted for h.

This simple verification principle is well known from the literature, see for instance [ZH].

However, for the final version of *Bag*, we must not only account for the single communication via insert *before* the parallel construct starts, but also for the communications by the recursive *Bag* call *after* the parallel construct has terminated. With respect to the communication history of the recursive *Bag* call we may assume, by Scott's induction rule, that it satisfies the bag specification. Unfortunately there is no elegant way to combine two assertions that describe the components of a sequential composition of processes. The reason why we suceeded in the situation above was that the communication histories of one of the two components were fairly simple.

In general one must construct a formula χ that describes the execution of our sequential process $P_1; P_2; Bag$. Informally speaking, this formula is:

" There are intermediate values x' and x'' for x, and subsequences h_1, h_2, h_3 of the complete communication history h, such that h is the concatenation of these three histories, χ_1 holds for the communications in h_1 and the transition of the x value from x° to x', χ_2 holds for h_2 and the transition from x' to x'', and $\phi_{bag}(insert, getmin)$ holds for h_3 and the transition from x'' to x ."

In the assertion language, one can write down a predicate logic formula with the intended interpretation as above. This formula is abbreviated as $\chi_1 \hat{\circ} \chi_2 \hat{\circ} \phi_{bag}(insert, getmin)$.

The second and final step in the verification of:

$$P_1; P_2; Bag \ sat \ \phi_{bag}(insert, getmin)$$

is then to show that:

$$(\chi_1 \hat{\circ} \chi_2 \hat{\circ} \phi_{bag}(insert, getmin)) \rightarrow \phi_{bag}(insert, getmin)$$

is a valid implication.

Let us compare this with the well known rule for sequential composition in Hoare's logic for sequential, i.e. non parallel, programs. The specifications for programs S are in the form of pre- and postconditions, denoted by $\{pre\}\, S\, \{post\}$. The pre- and postconditions are assertions on the initial and final state of the computation. To verify $\{pre\}\, S_1; S_2\, \{post\}$ one must invent an *intermediate* assertion *int* and verify $\{pre\}\, S_1\, \{int\}$ and $\{int\}\, S_2\, \{post\}$.

Since there is no complicated verification condition to validate, like the implication $(\chi_1 \hat{\circ} \chi_2 \hat{\circ} \phi_{bag}(insert, getmin)) \rightarrow \phi_{bag}(insert, getmin)$, we prefer this type of reasoning over the approach sketched above.

The observation that formulae of the form P *sat* χ are not very appropriate for the sequential composition construct formed the incentive for studying a new style of process specifications. These specifications are based upon the idea that if some process S_2 starts after termination of some other process S_1, then at that moment of time there is already some communication history. We call this the *initial trace* for the execution of S_2. By introducing this notion of an initial communication history, we can view a process as a transformer from *initial* histories and - process states to *final* histories and process states. We use the phrase "generalized state" for a combination of some history and some process state. The outer form of our specifications is the same as that of "classical" Hoare style formulae, with a pre- and postcondition. These pre- and postconditions are assertions on *generalized* states, however. To indicate this, we use the notation $(\varphi)\ P\ (\psi)$ for Hoare specifications, where the φ and ψ are assertions on generalized states. With our Hoare formulae, one can essentially use the same type of reasoning for sequential programs as within Hoare's logic. In particular the proof rule for sequential composition of processes has the same form as the classical Hoare rule mentioned above.

Our Hoare style specifications have in common with SAT specifications the uniform treatment of terminated and unfinished computations. That is, the assertions φ and ψ of the Hoare formula $(\varphi)\ P\ (\psi)$ and the assertion χ of the SAT formula P *sat* χ are interpreted for both terminated and unfinished computations. The characteristic predicate $\top$ is used within these assertions to distinguish between the two types of computations.

In practice it turns out to be easier to *separate* assertions into pre- and postconditions on generalized states that are interpreted for terminated computations only, and invariants on communication histories, that must hold continuously. This leads to a third type of formulae, called "Invariant specifications". They have the following form:

$$I : \{pre\}\ P\ \{post\},$$

where I is an assertion on histories only, and where *pre* and *post* are assertions on generalized states, i.e. on histories and states together. The characteristic predicate $\top$ is not used within I, *pre* or *post*. The distinction between terminated and unfinished computations is made in the interpretation of the specification. Informally, the meaning of the specification is:

" If P is started in an initial state and with an initial history for which *pre* holds, then the invariant I will hold for the communication history of P at any moment during execution of P, and if and when P terminates, *post* will hold for the final history and state of P."

Remarks

- By "the communication history of P" we mean the complete history, that is, the initial history extended with communications performed by P at some moment during execution, *not* just the communications performed by P itself.
- It is understood that, if P terminates, the invariant holds up to *and including* the moment of termination.

We proceed with the *Bag* example, turning over to the Invariant specification style. First we transform the SAT specification *Bag sat* $\phi_{bag}(insert, getmin)$ into an equivalent Invariant specification. A straightforward transformation is obtained by choosing a precondition that requires the initial trace to be *empty*, for then the role of the invariant is essentially the same as an assertion of a SAT formula. This would result in the following Invariant specification:

$\chi_{bag}(insert, getmin)$:

$\{insert = \varepsilon \wedge getmin = \varepsilon \wedge isempty = \varepsilon\}$

Bag

$\{cont(insert, getmin) = \emptyset \wedge isempty \neq \varepsilon\}$.

We have not included the assertion $\chi_{bag}(insert, getmin)$ in the postcondition, although it would have made no difference since the invariant requires it anyhow.

We are not satisfied with this specification for the following reason. Its precondition does not hold at those moments where the *inner recursive calls* of the *Bag* process start executing. So the given specification, although it correctly specifies the desired process behavior, will not fit into the correctness proof that we have in mind. As is not unusual with induction proofs, we can only prove a *stronger* specification. An Invariant specification can be made stronger by weakening its precondition, and by strengthening its postcondition and invariant. So the following formula is seen to be stronger than the specification above:

$\chi_{bag}(insert, getmin)$:

$\{cont(insert, getmin) = \emptyset \wedge \chi_{bag}(insert, getmin)\}$

Bag

$\{cont(insert, getmin) = \emptyset \wedge isempty \neq \varepsilon \wedge \chi_{bag}(insert, getmin)\}.$

To prove this specification we give a so called *annotated program.* It consists of the program text of the *Bag* process with assertions on generalized states attached to control points of the process. Similar to the usual program annotations for Hoare's logic we enclose these assertions within set braces. The annotated text is preceded by a clause, called the invariant of the annotation, that has the form $\chi_{bag}(insert, getmin)$: . Such an annotated text is to be understood as follows. If S is some piece of process text that occurs within an annotation that is prefixed by invariant I, and S is enclosed between the assertions p and q, then

$$I \ : \ \{p \wedge I\}\ S\ \{q \wedge I\}$$

is claimed to be a valid specification for S. (The fact that the invariant I is implicitly attached as a conjunct to all assertions within annotations is just a notational convenience.)

The actual annotation for *Bag* is:

$\chi_{bag}(insert, getmin)$:

$\{cont(insert, getmin) = \emptyset\}$

$($ $isempty!$

$\{cont(insert, getmin) = \emptyset \wedge isempty \neq \varepsilon\}$

or

($insert?x$;

$\{cont(insert, getmin) = [x]\}$

$(Buf \parallel Bag\langle down/insert, up/getmin\rangle)\backslash\{up, down, isempty\}$;

$\{cont(insert, getmin) = \emptyset\}$

Bag

) $\{cont(insert, getmin) = \emptyset \wedge isempty \neq \varepsilon\}$

$)\backslash\{x\}$

$\{cont(insert, getmin) = \emptyset \wedge isempty \neq \varepsilon\}$

Note that the specification for the second inner call of the *Bag* process equals the specification for the whole. This can be formally justified by applying

Scott's induction rule. The annotation does not indicate how to prove the specification that it claims to be valid for the parallel network within the *Bag* process. This specification is:

$\chi_{bag}(insert, getmin)$:

$\{cont(insert, getmin) = [x] \wedge \chi_{bag}(insert, getmin)\}$

$(Buf \parallel Bag\langle down/insert, up/getmin\rangle)\backslash\{up, down, isempty\}$

$\{cont(insert, getmin) = \emptyset \wedge \chi_{bag}(insert, getmin)\}$

We sketch how to prove this.

Again we may assume that the *Bag* specification holds for the recursive call within this network, and this implies the following for the *Bag* process with renamed channels:

$$\chi_{bag}(down, up) : \{cont(down, up) = \emptyset \wedge \chi_{bag}(down, up)\}$$
$$Bag\langle down/insert, up/getmin\rangle$$
$$\{cont(down, up) = \emptyset \ \wedge \ isempty \neq \varepsilon \wedge \chi_{bag}(down, up)\}.$$

By strengthening the precondition and weakening the postcondition we obtain:

$$\chi_{bag}(down, up) : \{down = up = \varepsilon\}$$
$$Bag\langle down/insert, up/getmin\rangle$$
$$\{cont(down, up) = \emptyset \wedge \chi_{bag}(down, up)\}.$$

The next task is to determine an appropriate specification for the *Buf* process. In our top down development, the most natural thing to do is to choose this specification such that it suits the verification of the parallel construct. Therefore we take the following one:

$$\chi_{bag}(down, up) \rightarrow \chi_{bag}(insert, getmin) :$$
$$\{cont(insert, getmin) = [x] \wedge \chi_{bag}(insert, getmin) \wedge \ up = down = \varepsilon\}$$
$$Buf$$
$$\{cont(down, up) = \emptyset \rightarrow cont(insert, getmin) = \emptyset \wedge$$
$$\chi_{bag}(down, up) \rightarrow \ \wedge \chi_{bag}(insert, getmin)\}$$

We now have available specifications for the constituent components of the parallel construct. They satisfy the restriction that the channels and variables referred to within these specifications are the channels and variables of the corresponding process. Under these conditions the following rule is applicable:

Let $I_1 : \{pre_1\}\ P_1\ \{post_1\}$ and $I_2 : \{pre_2\}\ P_2\ \{post_2\}$ be Invariant specifications such that the channels and variables of I_i, pre_i and $post_i$ are contained within the channels and variables used by P_i, for $i = 1, 2$. Then the following formula can be inferred from these two specifications:

$$(I_1 \wedge I_2)\ :\ \{pre_1 \wedge pre_2\}\ P_1\ \|\ P_2\ \{post_1 \wedge post_2\}.$$

We can use this rule to conjoin the *Buf* and the $Bag\langle down/insert, up/getmin\rangle$ specifications. Then, by weakening the invariant and postcondition we obtain the following specification for the parallel construct:

$\chi_{bag}(insert, getmin)$:

$\{cont(insert, getmin) = [x] \wedge \chi_{bag}(insert, getmin) \wedge up = down = \varepsilon\}$

$(Buf\ \|\ Bag\langle down/insert, up/getmin\rangle)$

$\{cont(insert, getmin) = \emptyset \wedge \chi_{bag}(insert, getmin)\}$

Let us omit the conjunct $up = down = \varepsilon$ from the precondition. That is, we no longer assume that the local channels *up* and *down* are initially empty. Now, unlike the initial state of *variables*, the initial state of *channels* cannot be sensed in any way by a process, and so the possible communication histories are the same as far as the projection onto channels *other* than *up* and *down* is concerned. And because the invariant and postcondition *do not refer to the up and down channels,* they still remain valid for this larger set of possible communication histories. We conclude that the following formula is valid:

$\chi_{bag}(insert, getmin)$:

$\{cont(insert, getmin) = [x] \wedge \chi_{bag}(insert, getmin)\}$

$(Buf\ \|\ Bag\langle down/insert, up/getmin\rangle)$

$\{cont(insert, getmin) = \emptyset \wedge \chi_{bag}(insert, getmin)\}.$

Since this specification does not refer anymore to the local channels *up*, *down* or *isempty*, it remains valid if we hide these local channels. Therefore, we have shown the validity of:

$\chi_{bag}(insert, getmin)$:

$\{cont(insert, getmin) = [x] \wedge \chi_{bag}(insert, getmin)\}$

$(Buf \parallel Bag\langle down/insert, up/getmin\rangle)\backslash\{up, down, isempty\}$

$\{cont(insert, getmin) = \emptyset \wedge \chi_{bag}(insert, getmin)\}$

This was to be shown.

We have succeeded in verifying the top level design for the *Bag* process. It remains to implement the *Buf* process, and to verify that the implementation does satisfy the specification that we chose for this component. This development does neither introduce new language constructs nor new verification principles. Therefore we simply list the resulting program text and the corresponding proof.

The proposed *Buf* implementation is:

($insert?y$;

$(x, y) := (min(x, y), max(x, y))$; $down!y$; Buf)

or

($getmin!x$;

(($up?x$; Buf) **or** $isempty?$))

Similar to the verification of the *Bag* process it is necessary to prove a *stronger* specification for *Buf* than the one that we required above. Again, the reason is that the given specification is inappropriate as an induction hypothesis for Scott's induction rule.

The proof is in the form of an annotated program text:

Proof outline for the *Buf* process:

Invariant $\chi_{bag}(down, up) \rightarrow \chi_{bag}(insert, getmin)$:

$\{cont(insert, getmin) = cont(down, up) \oplus [x] \ \wedge$
$\chi_{bag}(down, up) \rightarrow x = min(cont(insert, getmin))\}$

($insert?y$;

$\{cont(insert, getmin) = cont(down, up) \oplus [x, y] \ \wedge$
$\chi_{bag}(down, up) \rightarrow min(x, y) = min(cont(insert, getmin))\}$

$(x, y) := (min(x, y), max(x, y))$;

$\{cont(insert, getmin) = cont(down, up) \oplus [x, y] \ \wedge$
$\chi_{bag}(down, up) \rightarrow x = min(cont(insert, getmin))\}$

$down!y$;

$\{cont(insert, getmin) = cont(down, up) \oplus [x] \ \wedge$
$\chi_{bag}(down, up) \rightarrow x = min(cont(insert, getmin))\}$

Buf

) $\{cont(insert, getmin) = cont(down, up)\}$

or

($getmin!x$;

$\{cont(insert, getmin) = cont(down, up)\}$

($up?x$;

$\{cont(insert, getmin) = cont(down, up) \oplus [x] \ \wedge$
$\chi_{bag}(down, up) \rightarrow x = min(cont(insert, getmin))\}$

Buf

) $\{cont(insert, getmin) = cont(down, up)\}$

or

$isempty?$

$\{cont(insert, getmin) = cont(down, up)\}$

)

$\{cont(insert, getmin) = cont(down, up)\}$

1.4 Compositionality and Modularity

For the task of program specification, construction and verification, programs are not to be considered as monolithic entities.

Rather a program is built up from parts that are specified, implemented and verified independently. The whole program is then verified on the basis of *specifications* for the parts, that is, without (using) knowledge of the inner structure of the parts.

There are several ways in which one can interpret this. One view is that "built up from parts" refers to the *syntactic phrase structure* of the program. According to this structure a program is either an *atomic action*, a *program variable*, ranging over program meanings, or else it is built up from smaller programs by means of some syntactic operator C. We assume here that each of these operators has a *fixed*, finite arity, that is, combines a fixed number of programs into one larger program.

Examples of atomic statements for the case of **TNP** are the assignment statement, and the communication commands. The set of syntactic operators includes for instance the sequential and parallel composition operators. Program variables coincide with process identifiers. Within **TNP** such variables can occur *free* or *bound*. Note that the recursion construct in **TNP** does *bind* program variables.

Proofs of program specifications should be *compositional,* that is, *syntax directed.* To explain this, assume that, according to this syntax, some program S is composed out of the programs $S_1, \ldots, S_n$. That is, S is of the form $C(S_1, \ldots, S_n)$ where C is some syntactic constructor. Then according to the statement made above a specification *spec* for S should be proven from specifications $spec_1, \ldots, spec_n$ for the parts $S_1, \ldots, S_n$. And, most importantly, the proof must *not* refer to the inner syntactic structure of these parts.

In a historical perspective, the first methods for program verification , invented by Floyd and Naur [Floyd],[Naur], were not compositional. The reason is that programs were represented as *flowcharts*, and the usual syntactic structure of a flowchart, that has essentially the form of a labeled graph, does not decompose a program into smaller *programs*, but rather into *atomic actions*. Moreover, the number of atomic actions that constitute a flowchart is not bounded from above. Therefore, we do not have a syntactic operator with fixed arity.

A major improvement was made by C.A.R. Hoare [Hoare2] who reformulated the Floyd Naur method as a compositional proof system.

Hoare studied a very tiny language, often referred to as the class of "while programs". The method was extended by several people to encompass most of the usual sequential programming constructs. But when the first proof systems for *parallel* programs were created, these systems did *not* adhere to the syntax directed style.

For instance, a proof rule that violates this constraint is the one for parallel composition proposed by Owicki and Gries in [OG]. To verify a specification for a parallel program $S_1 \parallel S_2$ one must apply a so called *interference freedom test* to *proof outlines* that contain the *program text* of the parts S_1 and S_2.

Another well known proof system, for the verification of CSP programs, is the system by Apt, Francez and de Roever [AFR]. Here a so called *cooperation test* must be applied, again to proof outlines containing the program texts of the parts. So this system is not compositional either.

One of the first publications containing a proof rule for parallel programs *without* reference to the inner structure of the constituent components was by Misra and Chandy in "Proofs of Hierarchical Networks of Processes" [MC]. It must be noted that Misra and Chandy did not study a language with a clear cut syntax. Rather they used a "picture language" for statically nested parallel processes communicating via channels. Therefore, strictly speaking their system is not compositional in the sense as defined above, because of the rather trivial reason that they have no syntactic operators.

Misra and Chandy's rule is based on an interesting new type of process specifications. Essentially these specifications consist of a certain *assumption* and *commitment* about the *communication history* of a process. It is especially this formulation in terms of communication histories that made it unnecessary to refer to the inner structure of processes.

The Misra and Chandy rule formed the basis for a compositional proof system for the language **DNP** ("Dynamic Networks of Processes") by Zwiers, de Bruin and de Roever [ZBR]. As already mentioned in the summary, one of the proof systems that we consider in this thesis evolved from the system of [ZBR].

Before we go on discussing several aspects of compositionality in more detail, we would like to contrast this concept with a second interpretation of the statement made at the start of this section.

As one might have noticed already when we mentioned Misra and Chandy's rule, a context free grammar is *not* a presupposition for considering programs as "built up from parts". It suffices that there is some well defined notion of "black boxes" from which a program can be constructed, and which can later on be replaced by implementations in the form of concrete programs. Exactly *how* a program is built up from black boxes can be left open.

A clear advantage of this view is that, since it is more liberal with respect to program structure, many more interesting systems can be discussed in this setting. Typical examples are flowchart languages and transition systems. For instance, although a flowchart does not have a nice syntax described by a context free grammar, one *can* allow named black boxes in it. If only black boxes with a single entry and exit are allowed, then it is possible to *specify* these black boxes in the same way as complete flowcharts are specified, i.e. by means of an entry and an exit condition. Essentially this is the structure of the language FORTRAN, where subroutine bodies play the role of flowcharts and subroutine calls the role of black boxes.

Now our language **TNP** *does* have a neat context free grammar, so why are we so interested in this black box idea? The reason is that we want to make the distinction between what is called "programming in the small" versus "programming in the large". An often heard explanation of these terms is that programming in the large considers the program structure in terms of subroutines or procedures, whereas programming in the small refers to the internal structure of these subroutines and procedures. The terms "small" and "large" suggest that the only difference is in the *scale* of the program, and that mathematically speaking there is no real difference at all. We present here a different point of view. Programming "in the small" is done by one person, designing a program and its correctness proof hand in hand. As a consequence, when a specification is designed for a certain program part it is already clear in which program context it must function, and how the specification must "fit" into the correctness proof of the whole. On the other hand, programming "in the large" is not an activity of one person at one time. Therefore specifications for program parts are designed, and proven correct, *without* knowing exactly the context in which the part is to be placed. An (extreme) example is the design of modules for a program library where there is literally no contact between the designer and users of a module. The price for this division of labor is partly paid by the *user* of modules. For he must treat those modules as black boxes for which only an *a priori given* specification is known, and these specifications might *not* suit the correctness proof of the whole. This implies that, in general, a priori

given specifications must be *adapted* to a form that *does* fit into the proof. In our opinion, it is this extra step of specification adaptation that makes the difference between correctness proofs "in the small" and "in the large".

Let $spec(S)$ denote that some program S satisfies a certain specification *spec*. Specification adaptation means that a given specification $spec(\varsigma)$ for a black box ς has to be transformed into an alternative specification $spec'(\varsigma)$. That is, it must be shown that $spec'(\varsigma)$ is a valid formula on the basis of the given formula $spec(\varsigma)$. Such adaptations do not exclusively occur within the context of programming "in the large". For instance, for proving the correctness of a *recursive* program, it is necessary to give a correctness proof for the "body" of a recursive construct on the basis of an induction hypothesis that has the form of a specification for recursive calls. In general this given hypothesis must be adapted for each occurrence of a recursive call within the body. Usually the adaptation is performed with the aid of the well known *consequence rule* and various *substitution rules*. An example is given in [de Bakker], where a (two page long!) proof is given, essentially showing that if ς satisfies the Hoare style partial correctness formula

$$\{x = z\}\ \varsigma\ \{x = z\},$$

and we also know, on the basis of syntactic considerations, that the variable z cannot be read or modified by ς, then it is also true that ς satisfies:

$$\{x = z - 1\}\ \varsigma\ \{x = z - 1\}$$

Although specification is needed for programming "in the small", there is a major difference with programming "in the large". In the first case, when a specification is designed it is already known *whether* it has to be adapted, and *for which contexts*. By a proper choice of this specification one can ensure that all adaptations that are necessary can actually be proven correct within the given proof system at hand. We can contrast this with the situation for the second case, where *a priori given* specifications have to be adapted.

Example 1.5

We consider again the *Bag* process, introduced in the examples above.

On the one hand we have the syntactic phrase structure of the *Bag* process, showing for instance that it is a recursive process, and that the body of this recursive process consists of a choice construct, and so on. On the other hand one sees occurrences of black boxes named *Buf* and *Bag*. In the top down style verification that we gave for the *Bag* process, a certain specification was chosen for the *Buf* process that suited the verification for *Bag*. Thereafter we verified the *Buf* specification. Now consider the following different situation.

Assume that actually the *Buf* process did already exist, due to some earlier design activity. That is, it was already constructed, specified and verified by someone else before, *not* with the intention to use it for our *Bag* design. It is unlikely that the specification would have been chosen the same as our *Buf* specification. For instance, it could have been as follows.

- $bcont \stackrel{\text{def}}{=} bag(insert) \oplus bag(up) \ominus bag(getmin) \ominus bag(down)$,
- $bcont[rest(down)/down]$ is *bcont* with $rest(down)$ substituted for *down*,
- $lastchan \stackrel{\text{def}}{=} chan(last(h|\{insert, getmin, up, down\}))$,
- $lastval \stackrel{\text{def}}{=} val(last(h|\{insert, getmin, up, down\}))$,

Let the buffer invariant be the assertion:

$$\chi_{buf} \stackrel{\text{def}}{=} (lastchan = down \rightarrow lastval = max(bcont[rest(down)/down])) \wedge$$
$$(lastchan = getmin \rightarrow lastval = min(bcont[rest(getmin)/getmin]))$$

Then the alternative *Buf* specification is:

$$\chi_{buf} \ : \ \{bcont = [x]\}\ Buf\ \{bcont = \emptyset\}.$$

Since we want to treat *Buf* as a black box, *redoing* the verification task to see that *our Buf* specification is satisfied (too) is not possible. Rather the already existing specification must be *adapted*, that is, we must prove that the last specification *implies* the one that we used for the verification of the *Bag* process.

However, simply strengthening of the precondition and weakening of the postcondition and the invariant of the alternative specification will not suffice to prove this implication between specifications. For it is *not* the case that the invariant of the alternative specification implies that of the original *Buf* specification. The reason is that the alternative invariant guarantees something about the *last* communication of the history *h*, at least if it is a communication via *down* or *getmin*. But it does *not* guarantee anything about earlier communications in *h*. This is to be contrasted with the original *Buf* invariant, that has the form

$$\chi_{bag}(down, up) \rightarrow \chi_{bag}(insert, getmin).$$

The assertion $\chi_{bag}(c_1, c_2)$ has the form:

$$\forall t(t \leq h|\{c_1, c_2\} \cdots),$$

The result of this is that the original invariant makes a certain commitment about *all down* and *getmin* communications occurring in h. This indicates why the alternative invariant cannot imply the original one.

One might question whether the alternative specification actually *is* as strong as the original one, since its invariant is logically weaker than the other invariant. The intuitive reason why this is nevertheless the case is that the invariant of an Invariant specification is required to hold, not only for all possible executions, but also *at all moments during such executions.*

□

1.5 Compositional and Modular Completeness

For any formal proof system one of the main questions is whether sufficiently many axioms and rules have been collected. For instance, given a specification *spec* and a program S that satisfies this specification we might ask whether one can always *prove* that fact. Or we might ask whether we can always carry through the necessary specification adaptations. Such problems are referred to as the *completeness* question for the formal system.

First we summarize some of the standard concepts in this respect.

As before, $spec(S)$ denotes that specification *spec* is satisfied by program S.

If S is a program that does not contain black boxes, this means that, under a given interpretation for programs and specifications, the formula $spec(S)$ is interpreted as "true". If S does contain black boxes, then this truth value depends on the interpretation of the black boxes. If $spec(S)$ is interpreted as true for all possible interpretations for black boxes within S, then the formula is called *valid*. This is denoted by

$$\models spec(S).$$

If for certain black boxes $\varsigma_1, \ldots, \varsigma_n$ and specifications $spec_1, \ldots, spec_n$ it is the case that for any interpretation of black boxes that makes the formulae $spec_1(\varsigma_1), \ldots, spec_n(\varsigma_n)$ true the formula $spec(S)$ is also true, we say that the last formula is *semantically implied* by the former ones. We denote this as usual by

$$spec_1(\varsigma_1), \ldots, spec_n(\varsigma_n) \models spec(S).$$

As is well known, *validity* of some formula is only one side of the coin, where the other side is its *provability*. A formula is provable within a certain formal

system if there is a formal deduction from axioms by means of proof rules that has the formula in question as its conclusion. If the formula $spec(S)$ is provable this is denoted by

$$\vdash spec(S)$$

We say that $spec(S)$ is provable *from hypotheses* $spec_1(\varsigma_1), \ldots, spec_n(\varsigma_n)$ if $spec(S)$ can be deduced by means of proof rules from axioms together with the given hypotheses. This is denoted by:

$$spec_1(\varsigma_1), \ldots, spec_n(\varsigma_n) \vdash spec(S)$$

It is always required that a formal system is *sound,* meaning that if some formula is provable, then it is valid. If the reverse holds too, that is, every valid formula is also provable, then the system is called *complete.*. Although completeness is a desirable property, one does not, and cannot, always insist on it. For the famous results of Gödel [Gödel] showed that for many mathematical systems, including the system of natural numbers, a complete axiomatization by means of first order predicate logic is *impossible.* The impact of this on our work is the following. Instead of proving completeness we shall prove *relative* completeness. By this we mean that we assume that we have a so called oracle to decide whether some assertion of our assertion language $\mathcal{A}ssn$, that we define in chapter 4, is valid or not. Relative completeness means that every valid specification can be formally deduced where we may call upon the oracle to decide the validity of assertions. The typical case where the oracle is used is for application of one of the *consequence rules* that we introduce in chapter 5.

A rather subtle point is the following. When we come to the definition of (the meaning of) the assertion language $\mathcal{A}ssn$, we shall include the natural numbers, together with the usual operations of addition and multiplication. The interpretation for these is by means of the standard model of arithmetic. Such an interpretation is called *strongly arithmetical* [Cla2]. The reason for doing so is that we insist on a *specification* language that is sufficiently expressive in the sense as defined in section 6.2, and this can be achieved only if we can express within assertions the *length* of communication histories. Moreover, even if a finite interpretation is chosen for the domain of data that processes act on, one still can construct processes with three channels that are such that the length the history via one of these channels is essentially the sum or product of the lengths of the histories via the other two channels. This explains why we need a strongly arithmetical interpretation.

Since we are relying on a strongly arithmetical interpretation for assertions, our notion of completeness is better called *arithmetical completeness* [Harel].

To avoid cumbersome terminology we shall usually omit the adjectives "relative" and "arithmetical" when we discuss completeness questions. That is, these qualifications are always implicitly understood.

The "classical" completeness question in the field of program correctness is whether for *closed* programs one can prove the program correct with respect to every valid specification. This notion of completeness is not appropriate for "programming in the large", since there the normal situation is that some formula $spec(S)$ has to be proven correct from specifications $spec_1(\varsigma_1), \ldots, spec_n(\varsigma_n)$ for the black boxes $\varsigma_1, \ldots, \varsigma_n$ in S.

The natural notion of completeness in this context is that of *modular completeness*:

Let SPEC be a given class of program specifications. A formal proof system is called modular complete for this class if the following holds. If $spec \in$SPEC is some specification and S is some program such that the formula $spec(S)$ is semantically implied by specifications $spec_1, \ldots, spec_n$ for the black boxes $\varsigma_1, \ldots, \varsigma_n$, then it is possible to *prove* $spec(S)$ from the hypotheses $spec_1(\varsigma_1), \ldots, spec_n(\varsigma_n)$.

In practice there is one minor technical problem with this definition. A specification *spec* is *satisfiable* if there exists at least one program that satisfies it, else *spec* is *unsatisfiable*. Now if one of the hypotheses $spec_i(\varsigma_i)$ is actually unsatisfiable, then the semantic implication is always valid, vacuously. Then, in general, a *proof* of $spec(S)$ from the given hypotheses requires the logical principle of reductio ad absurdum in one form or another. Since this principle plays no role in proving programs correct except for the situation as described, we would like to avoid the introduction of (yet) another proof rule that expresses this principle on the level of specifications. This motivates the following more accurate definition of modular completeness:

Definition 1.6 (Modular Completeness)

Let SPEC be some class of specifications for some given class of programs PROG. A formal proof system for these specifications is modular complete if, for all $spec \in$SPEC, all satisfiable $spec_1, \ldots, spec_n \in$SPEC and all $S \in$PROG,

$$spec_1(\varsigma_1), \ldots, spec_n(\varsigma_n) \models spec(S).$$

implies

$$spec_1(\varsigma_1), \ldots, spec_n(\varsigma_n) \vdash spec(S).$$

□

For the logical systems that we shall consider, the specification language SPEC is closed under the usual logical operations such as conjunction and implication. (We shall assume that these operations do have their usual interpretation, and that the usual logical rules for these operations have been included in the system.)

Let us call a formula $mspec(S)$, of the form

$$\Big(\bigwedge_{i=1,n} spec_i(\varsigma_i)\Big) \rightarrow spec(S) \qquad (1)$$

a *modular* specification of S. Since our logical formalisms can represent a finite conjunction of the form

$$\bigwedge_{j=1,m} spec_j(\varsigma)$$

by means of a *single* specification $spec'(\varsigma)$ we shall assume that the black boxes ς_i in the premisse of a modular specification of the form (1) are all distinct names. It is understood that if $n = 0$ the conjunction $\left(\bigwedge_{i=1,n} spec_i(\varsigma_i)\right)$ is the formula "**true**". Identifying the formula **true** $\rightarrow spec(S)$ with $spec(S)$ one sees that specifications *spec* $\in$SPEC are a special case of the form (1). Let MSPEC be the class of all such modular specifications, corresponding to the given class of (ordinary) specifications SPEC.

We call $mspec(S)$ *regular* if all the premisses $spec_i(\varsigma_i)$ are satisfiable. Modular completeness for SPEC is in fact nothing else but completeness for the regular formulae of MSPEC. That is, for regular *mspec* $\in$MSPEC, if $\models mspec(S)$ then $\vdash mspec(S)$. This follows from the following facts:

Since an implication $f_1 \rightarrow f_2$ is valid if and only if f_1 *semantically* implies f_2, formula (1) is valid if and only if

$$spec_1(\varsigma_1), \ldots, spec_n(\varsigma_n) \models spec(S).$$

Also, for our logical formalism a (formal) implication $f_1 \rightarrow f_2$ is provable if and only if f_2 can be proven from the hypothesis f_1. So

$$spec_1(\varsigma_1), \ldots, spec_n(\varsigma_n) \vdash spec(S)$$

amounts to the same as

$$\bigwedge_{i=1,n} spec_i(\varsigma_i) \rightarrow spec(S).$$

(We remark that, although all this might seem fairly straightforward, the standard treatment of Hoare's logic as for example in [Apt] or [Bakker] defines implication between correctness formulae in a more complicated way, due to the presence of *implicit* universal quantifications for both sides of the implication sign.)

Spelled out in detail the alternative formulation of modular completeness is as follows: for specifications and programs as indicated in the definition, if

$$\models \Big(\bigwedge_{i=1,n} spec_i(\varsigma_i) \Big) \rightarrow spec(S),$$

then

$$\vdash \Big(\bigwedge_{i=1,n} spec_i(\varsigma_i) \Big) \rightarrow spec(S).$$

A property that is implied by modular completeness is what we call *adaptation completeness*. For this case we have a given specification $spec(\varsigma)$ that we want to adapt into another specification $spec'(\varsigma)$. Of course this adaptation is legal only when the first specification logically implies the second one. So we are interested in formulae of the following form:

$$spec(\varsigma) \rightarrow spec'(\varsigma)$$

Proving *tautologies* for black boxes, by which we mean proving formulae of the form $spec'(\varsigma)$ that are universally valid, shall be considered as a special case of this. To this end we allow $spec(\varsigma)$ to be the formula **true**. Adaptation completeness means simply completeness with respect to this particular class of formulae.

Definition 1.7 (Adaptation completeness)

A formal proof system is adaptation complete for a given class of specifications SPEC if for all $spec, spec' \in$SPEC, where $spec$ must be satisfiable, if

$$\models spec(\varsigma) \rightarrow spec'(\varsigma)$$

then

$$\vdash spec(\varsigma) \rightarrow spec'(\varsigma)$$

It is understood here that $spec(\varsigma)$ can be the formula **true**. □

Remark

A from an intuitive point of view less attractive, but nevertheless equivalent definition of adaptation completeness would be to state that it is modular

completeness for formulae of the restricted form $mspec(\xi)$, i.e. where the program on the right hand side of the implication is itself a black box ξ. That adaptation completeness according to this alternative definition implies adaptation completeness as defined above will be clear. To see that the reverse hold too, assume that we have a adaptation complete system, in the sense of the original definition, and assume that a universally valid formula

$$\Big(\bigwedge_{i=1,n} spec_i(\varsigma_i)\Big) \rightarrow spec(\xi) \quad (*)$$

has been given. If $spec(\xi)$ is a tautology, it is provable by the assumed adaptation completeness, and with the usual logical rules $(*)$ is then provable too. Next assume that $spec(\xi)$ is not a tautology. Since all the specifications $spec_i(\varsigma_i)$ are satisfiable and all the ς_i are distinct, it then can be seen that exactly one of the ς_i must be ξ, and moreover that:

$$\models spec'(\xi) \rightarrow spec(\xi)$$

Clearly the last implication is provable by the assumed adaptation completeness, and again the provability of $(*)$ follows from this.

□

Since modular completeness implies adaptation completeness, one might ask whether we can formulate some requirement that together with adaptation completeness implies modular completeness. In general, we can say little about this. But for *compositional* systems, a positive answer can be given.

Assume that our class of programs PROG is described by means of compositional syntax. By this we mean that each program S has either the form of a black box ς which in this context shall be identified with a *variable* ranging over program meanings, or else is of the form $C(S_1, \ldots, S_n)$, where C is some syntactic constructor and where $n \geq 0$. If for a program of the latter form $n = 0$ we call it an *atomic* program. To be able to treat constructs like recursion within this framework we shall allow constructors C that can *bind* black box variables.

For a compositional approach a constructor $C(S_1, \ldots, S_n)$ is regarded as a *function*, where the S_i are the variables of the function. As is usual in mathematical logic, we shall use the term *meta* variables, for variables like S_i. Meta variables range over *program texts*, and must be distinguished from black box variables ς that can occur within actual program texts, and that range over *the domain of interpretation* for programs.

By a *compositional proof rule* we mean a logical inference of the following form:

"From $spec_0(S_0)$ and ... $spec_{n-1}(S_{n-1})$ infer $spec_n(S_n)$"

Here $spec_i$ and S_i are both meta variables, one ranging over specification texts, the other over program texts. That is, when we say "proof rule" we actually mean a proof rule *scheme*. An actual rule is obtained by substituting actual texts for the meta variables in the scheme. Of course one must not substitute a program text for a (meta) specification variable. That is, meta variables are of a certain *type*, and one may only substitute texts of the appropriate type. Syntactic attributes such as the free variables of a text are usually regarded as part of the type of the text. For example, the type of some meta variable S might be "program containing at most ξ_1, ξ_2 as free blackbox variables and at most channels c_1, c_2, c_3 as free channel names"

For a proof rule as above, the types of meta variables are usually put into the form of restrictions attached to the rule. It is important that such restrictions can refer only to the *types* of the meta variables, but *not* to the inner syntactic structure of these variables. For instance the rule for parallel composition by Owicki and Gries is of the following form:

"Let $\{p_1\}\ S_1\ \{q_1\}$ and $\{p_2\}\ S_2\ \{q_2\}$ be Hoare style proof outlines that are interference free. Then the Hoare formula $\{p_1 \wedge p_2\}\ S_1 \parallel S_2\ \{q_1 \wedge q_2\}$ holds."

This is not a compositional proof rule, since the side condition about interference freedom does refer to proof outlines, containing the whole program texts of S_1 and S_2, and not just to the *types* of these texts.

We give a definition of what is called *elementary compositional completeness*.

Definition 1.8 (Elementary compositional completeness)

Assume that S is some arbitrary program of the form $C(S_1, \ldots, S_n)$, that no black box variables occur free in S and that C does not bind a black box variable. (The term elementary refers to these restrictions.) Let *spec* be some specification for S. A proof system is compositionally complete if it is compositional and moreover is complete in the sense that , whenever

$$\models spec(S),$$

there exist specifications $spec_1, \ldots, spec_n$ such that:

(a) $\models spec_i(S_i)$ for $i = 1..n$.

(b) $spec(C(S_1, \ldots, S_n))$ can be proven from the hypotheses $spec_i(S_i)$.

□

The requirement that the system must be compositional guarantees that the proof required for point (b) is *uniform* in that no reference to the internal

structure of the parts $S_1, \ldots, S_n$ is possible. The proof that is required for (b) is in fact a proof *scheme* that can be instantiated by substituting actual program texts for the meta variables $S_1, \ldots, S_n$. Note that this definition does not require that the specification for S can be proven from *a priori given* specifications for the parts. Rather one must be able to *choose* appropriate specifications, such that (a) and (b) are fulfilled.

For the general case, where we allow free black box variables and more importantly, constructs C that bind black box variables, we do not want compositional completeness on the level of *specifications spec* $\in$SPEC, but rather on the level of *modular* specifications *mspec* $\in$MSPEC. Of course this includes the simple case above since SPEC formulae can be regarded as MSPEC formulae. Therefore our general definition is the following one.

Definition 1.9 (Compositional Completeness)

Let PROG be a class of programs. Let SPEC be a class of program specifications and let MSPEC be the corresponding class of modular specifications. A proof system is compositionally complete if it is compositional and moreover is complete in the sense that for all $S \in$PROG of the form $C(S_1, \ldots, S_n)$ and all *mspec* $\in$MSPEC, if

$$\models mspec(S),$$

then there exist $mspec_1, \ldots, mspec_n \in$MSPEC such that:

(a) $\models mspec_i(S_i)$ for $i = 1..n$

(b) $mspec(C(S_1, \ldots, S_n))$ is provable from the hypotheses $mspec_i(S_i)$

□

We show that compositional completeness is supplementary to adaptation completeness in the sense that together they imply modular completeness.

Theorem 1.10

A proof system that is both compositional complete and adaptation complete is modular complete.

□

This is seen as follows: Assume that we have a system that is compositional and adaptation complete. Let some regular modular specification *mspec* and some program S been given, and assume that $mspec(S)$ is *valid*. We show that $mspec(S)$ is *provable* by means of induction on the syntactic structure of the program.

There are two cases to distinguish, according to whether S is a black box ς or is of the form $C(S_1, \ldots, S_n)$. Note that the second case includes the situation where S is an *atomic* program.

If S is a black box then the provability of $mspec(S)$ follows from the adaptation completeness, as shown in the remark following the definition of this type of completeness above.

If on the other hand S is of the form $C(S_1, \ldots, S_n)$, then by compositional completeness we can choose specifications $mspec_1, \ldots, mspec_n$ such that $mspec_i(S_i)$ is valid for $i = 1..n$, and $mspec(C(S_1, \ldots, S_n))$ can be proven from the hypotheses $mspec_1(S_1), \ldots, mspec(S_n)$.
Since the components S_i have a simpler syntactic structure than S, we may conclude by induction that the formulae $mspec(S_i)$ are *provable*. And instantiating the variables $S_1, \ldots, S_n$ in the given proof scheme, we see that $mspec(C(S_1, \ldots, S_n))$ is provable from the hypotheses $mspec(S_i)$. We conclude that $mspec(C(S_1, \ldots, S_n))$ is provable, as was to be shown.

We already remarked that modular completeness includes the "classical" notion of completeness which states: if S is a *closed* program and $spec(S)$ is a valid formula then this formula is provable. By the theorem, a *sufficient* condition for completeness is that the proof system is both compositionally and adaptation complete. However, adaptation completeness is not a necessary condition for completeness, as we will show now.

First we consider the simple case of elementary compositional completeness. That is, for the moment we only consider programs without occurrences of black box variables, either free or bound. It is easily seen that an elementary compositionally complete system is complete for this class of programs and specifications from SPEC. The proof is along the same lines as that for the theorem above, if one reads "specification" for "modular specification", and omits the part of the proof that handles the case where S is a black box.

Next we treat the general case. Even when we want to verify a simple specification *spec* for a closed program S of the form $C(S_1, \ldots, S_n)$, it will in general not be possible to find SPEC formulae $spec_i$ for the parts S_i from which the specification for the whole can be verified. For if C does *bind* some black box variable, say θ, then some of the S_i contain θ as a free variable. And so, for these parts S_i we will only be able to find a valid specification of the form

$$spec_\theta(\theta) \rightarrow spec(S_i).$$

That is, there is no escape from using *modular* specifications in this case, even when the program as a whole has no free occurrences of black box variables.

We make the observation that the specification $spec_\theta$ for the black box θ is not a priori given, but can be chosen by the verificator of the program S. This explains why a proof system can be complete without being adaptation complete. For in this case one must only be able to adapt specifications such as $spec_\theta$, and this can be simpler than being able to adapt arbitrary specifications.

If *spec* is some particular specification, then we call it *adaptable* for a certain proof system if the following holds, for arbitrary specifications *spec'*: If $spec(\varsigma) \rightarrow spec'(\varsigma)$ is a valid implication, then this implication is *provable*. (Clearly a system is adaptation *complete* if and only if *every* specification is adaptable.)

Chapter 2

The languages DNP and TNP.

2.1 Introduction

The object of study is the proof theory of the concept of a dynamically changing network of processes. In such a *network*, primarily sequential programs, called *processes*, execute in parallel and communicate via interconnecting channels. Processes each have their own (private) set of (assignable) variables, i.e. their variables are not shared with other processes executing in parallel. Processes can *expand* temporarily into parallel subnetworks. This can happen recursively since the so formed subnetworks can contain new copies of the original (expanded) process. After termination of all component processes of a subnetwork, the expanded process *contracts* (shrinks) again, and continues its original mode of execution. The wish to model this concept as close as possible led to the design of a small programming language called **DNP** ("Dynamic Networks of Processes"). The language includes *process declarations* similar to conventional procedure declarations. New incarnations of processes can be created by means of (potentially recursive) *process calls*. The syntax of **DNP** is as follows. We leave unspecified the precise definition of the following syntactic classes:

$(x, u \in)$	$\mathcal{V}ar$	–	A set of *assignable variables*
$(c, d \in)$	$\mathcal{C}han$	–	A set of *channel names*
$(p \in)$	$\mathcal{P}id$	–	A set of *process names*
$(e \in)$	$\mathcal{E}xp$	–	The class of *expressions*
$(b \in)$	$\mathcal{B}exp$	–	The class of *boolean expressions*

Lists will be abbreviated using "vector" notation. Similarly, **bold** indicates a *set* of names. E.g. $x_1, x_2 \ldots x_n$ is abbreviated as $\bar{x}$, and $\{\bar{x}\} \stackrel{\text{def}}{=} \{x_1, x_2 \ldots x_n\}$ is denoted by $\mathbf{x}$.

Definition 2.1 (Syntax of **DNP**)

Statements:

$$S ::= \textbf{skip} \mid x := e \mid c!e \mid c?x \mid S_1 ; S_2 \mid S_1 \textbf{ or } S_2 \mid$$

$$\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi} \mid \textbf{while } b \textbf{ do } S \textbf{ od} \mid$$

$$\textbf{cobegin } N \textbf{ coend} \mid P(\bar{d}_i; \bar{d}_o; \bar{y})$$

Networks:

$$N ::= S_1 \parallel S_2$$

Process declarations:

$$Dec ::= P(\bar{c}_i; \bar{c}_o; \bar{x}) \textbf{ begin } S_0 \textbf{ end}$$

Programs:

$$Prog ::= S \mid Dec : Prog$$

□

The intuitive meaning of **skip**, $x := e$ and $S_1 ; S_2$ should be clear. The construct S_1 **or** S_2 stands for nondeterministic choice between S_1 and S_2.

A *network* $S_1 \parallel S_2$ calls for concurrent execution of S_1 and S_2. In such a network, S_1 and S_2 are not allowed to have "shared" assignable variables. The two component processes can communicate with each other (only) along named, directed channels. Communication along a channel, say c, occurs when an output command $c!e$ is executed by one of the component processes simultaneously, i.e. synchronized, with an input command $c?x$ of the other. The value of e is then assigned to the variable x and both processes continue their execution. In **DNP**, channels always connect exactly two processes. That is, two different processes are not allowed to both read from or both write to some common channel. A channel from which some process reads or to which it writes is called an *external* input or output channel of that process. We denote the sets of (external) input and output channels of S by $in(S)$ and $out(S)$. When two processes are bound together into a network, their common channels, along which they communicate, are said to be *internal* channels of that network. That is, the set $intern(S_1 \parallel S_2)$ of internal channels of $S_1 \parallel S_2$ is defined as

$$intern(S_1 \parallel S_2) = (in(S_1) \cap out(S_2)) \cup (in(S_2) \cap out(S_1)).$$

When dealing with "nested" networks it is possible that some subnetwork has an internal channel with the same name as the main network. This

is even unavoidable when the subnetwork and the main network belong to different incarnations of the same process in case of a recursive process call. Such channel name clashes are resolved by introducing a kind of block structure with the **cobegin - coend** construct, which *hides* all internal channels. No internal channel of $S_1 \parallel S_2$ is visible anymore outside the process **cobegin** $S_1 \parallel S_2$ **coend.**

Example

In process S defined as

$$S \equiv \textbf{cobegin } S_1 \parallel \textbf{cobegin } S_2 \parallel S_3 \textbf{ coend coend},$$

where

$$S_1 \equiv c?x,\ S_2 \equiv c!0 \text{ and } S_3 \equiv c?y,$$

the S_2 process communicates with S_3 along the c channel internal to $S_2 \parallel S_3$, and not with S_1. The c channel of S_1 is rather an external channel of the whole process S.

Note that, if channel name clashes arise, parallel composition including this hiding is not associative.

To obtain a modular character for **DNP**, we have for recursive process declarations a scope concept different from that of Algol like languages. All variables used in some process body, bracketed by **begin - end**, are assumed to be *local* variables, i.e. there are no references to any kind of global variables possible. (Correspondingly, there is no explicit variable declaration mechanism needed in **DNP**.) The parameter list of a process consists of *input channels*, followed by *output channels*, followed by *value/result variable parameters*. To simplify matters technically, we impose the restriction that all names in a (formal or actual) parameter list be distinct. This avoids any kind of "aliasing" that could introduce unwanted sharing of assignable variables or channel names by two processes.

2.2 The language TNP

Process declaration in **DNP** is a rather elaborate construct, for it comprises all of the following programming concepts:

- process naming,
- recursion,

- local variables,
- channel renaming, and
- a call by value/result parameter mechanism.

For the sake of a simple mathematical description of processes, we introduce the language **TNP** (for Theoretical **DNP**). In **TNP**, the **DNP** process concept has been disassembled into the more fundamental concepts above. For the same reasons, the parallel construct of **DNP** has been generalized in **TNP**, in the sense that no distinction is made in **TNP** between *networks* and *statements*. As a result the four syntactic classes of **DNP** could be replaced by the single class **TNP** of process terms. **TNP** is similar to a class of terms for a predicate logic, except that the variables of **TNP** have the following structure:

Definition 2.2 (the class *Pvar* of process variables)

$$(\beta \in) \quad \mathit{Base} \stackrel{\text{def}}{=} \mathcal{P}(\mathit{Chan}) \times \mathcal{P}(\mathit{Var})$$

$$(p \in) \quad \mathit{Pvar} \stackrel{\text{def}}{=} \mathit{Pid} \times \mathit{Base}$$

□

The *base* of a process is a pair $(\mathbf{c}, \mathbf{x})$, denoting the channels $\mathbf{c}$ via which the process can communicate and the variables $\mathbf{x}$ that the process can read and write. Each process variable p consists of a process *name* P with such a pair β attached as a subscript, that is, p is of the form P_β $(\equiv P_{(\mathbf{c},\mathbf{x})})$.

The (context free) syntax of **TNP** is given in definition 1.3.

Definition 2.3 (syntax of **TNP**)

Process terms $S \in$ **TNP** :

$$S ::= \mathbf{skip} \mid \mathbf{abort} \mid x := e \mid b \mid c.x{:}b \mid S_1 ; S_2 \mid$$

$$S_1 \mathbf{\ or\ } S_2 \mid S_1 \, \beta_1 \| \beta_2 \, S_2 \mid S_1 \backslash \mathbf{x} \mid S_1 \backslash \mathbf{c} \mid S_1 \langle d/c \rangle \mid$$

$$P_\beta \mid \mu_z P_\beta . S_1 \mid P_\beta = S_1 \mathbf{\ in\ } S_2$$

□

We call b a *guard*, $c.x{:}b$ *communication* via channel c, $S_1 \backslash \mathbf{x}$ *hiding* of variables $\mathbf{x}$, $S_1 \backslash \mathbf{c}$ *hiding* of channels $\mathbf{c}$, and $S_1 \langle d/c \rangle$ *channel renaming* of c into d. p is called a *process call*, $\mu_z P_\beta . S_1$ a *recursion*, and finally, $P_\beta = S_1$ **in** S_2 is called *process naming*.

There are a number of (context sensitive) restrictions on the language, summarised in table 1.5 , and formulated with the aid of the functions "*chan*","*var*", and "*pvar*", introduced in definition 1.4.

Definition 2.4 (free variables of process expressions)

For each **TNP** expression S we define:

- the set of free channels $chan(S)$,
- the set of free assignable variables $var(S)$, and
- the set of free process variables $pvar(S)$.

Remarks:

- $base(S)$ denotes $(chan(S), var(S))$.
- We abbreviate $var(S_1) \cup var(S_2)$ as $var(S_1, S_2)$ etc.
- We assume that for $e \in \mathcal{E}xp$ and $b \in \mathcal{B}exp$ the sets $var(e)$ and $var(b)$ have already been defined.
- For $\beta \in \mathcal{B}ase$ of the form (c,x) we define: $var(\beta) = \mathbf{x}$ and $chan(\beta) = \mathbf{c}$.
- *Set operations* applied to bases β are understood as abbreviating the corresponding operations applied to the components of those bases.

S	$chan(S)$	$var(S)$	$pvar(S)$
skip	$\emptyset$	$\emptyset$	$\emptyset$
abort	$\emptyset$	$\emptyset$	$\emptyset$
$x := e$	$\emptyset$	$\{x\} \cup var(e)$	$\emptyset$
b	$\emptyset$	$var(b)$	$\emptyset$
$c.x!b$	$\{c\}$	$\{x\} \cup var(b)$	$\emptyset$
P_β	$chan(\beta)$	$var(\beta)$	$\{P_\beta\}$
$S_1 ; S_2$	$chan(S_1, S_2)$	$var(S_1, S_2)$	$pvar(S_1, S_2)$
S_1 **or** S_2	$chan(S_1, S_2)$	$var(S_1, S_2)$	$pvar(S_1, S_2)$
$S_1 \ {}_{\beta_1}\|\|_{\beta_2} \ S_2$	$chan(\beta_1, \beta_2)$	$var(\beta_1, \beta_2)$	$pvar(S_1, S_2)$
$S_1 \backslash \mathbf{x}$	$chan(S_1)$	$var(S_1) - \mathbf{x}$	$pvar(S_1)$
$S_1 \backslash \mathbf{c}$	$chan(S_1) - \mathbf{c}$	$var(S_1)$	$pvar(S_1)$
$S_1\langle d/c \rangle$	$(chan(S_1) - \{c\}) \cup \{d\}$	$var(S_1)$	$pvar(S_1)$
$\mu_z P_\beta.S_1$	$chan(\beta)$	$var(\beta)$	$pvar(S_1) - \{P_\beta\}$
$P_\beta = S_1 \mathbf{in} S_2$	$chan(S_2)$	$var(S_2)$	$pvar(S_1, S_2) - \{P_\beta\}$

□

Definition 2.5 (context sensitive restrictions for **TNP**)

CSR1 For $S_1\ \beta_1 \| \beta_2\ S_2$: $base(S_i) \subseteq \beta_i$, for $i = 1, 2$

CSR2 For $P_\beta = S_1$ **in** S_2 : $P_\beta \notin pvar(S_1)$

CSR3 For $P_\beta = S_1$ **in** S_2 : $base(S_1) \subseteq \beta$

CSR4 For $\mu_z P_\beta . S_1$: $base(S_1) \subseteq \beta$

□

Remark:

- The definition above implies that $d \in chan(S_1\langle d/c\rangle)$, even when $c \notin chan(S_1)$. Although this causes no harm, it would have been more elegant to define $chan(S_1\langle d/c\rangle) = chan(S_1)$ in this case.

2.3 Intuitive explanation of TNP

The meaning of **skip**, $x := e$, $S_1\ ;\ S_2$ and S_1 **or** S_2 is the same as for **DNP**.

The parallel construct of **TNP**, and its associated communication mechanism, is a generalization of the networks of **DNP**. The rationale behind this generalization is the following one. At first the only goal was to simplify the language structure somewhat by removing the difference between networks and processes. Now the reason for introducing this distinction in **DNP** in the first place was to avoid channel sharing between more than two processes, by enforcing to put the **cobegin - coend** brackets around a parallel combination *before* one could compose it in parallel with a third process. So for the case of **TNP**, where essentially a combination of the form $S_0\ \|\ S_1\ \|\ S_2$ *is* allowed, we must either rule out the possibility of all three processes communicating via some common channel by means of rather complicated context sensitive restrictions, or else give a meaning to such situations. We adopted the second alternative. Apart from the simplification of syntax the resulting generalization is interesting for its own sake. For instance, a sort of "shared variables" can now be described by modelling such variables as processes, as will be shown below. Another application is that, in the context of VLSI design, a synchronous clock of such a design can be modelled by means of "broadcasting" special messages, one for each "tick" of the clock, to all relevant processes.

For **DNP** we determined the set of channels between two parallel processes for which synchronization is required from the *syntax* of the two processes. This was thought nice since it avoids an explicit declaration mechanism. There are essentially two reasons why we changed this for **TNP**.

1. Since the base of processes is statically determined, we prefer to treat bases as *syntax* rather than *semantics*. To determine the semantics of a parallel construct, in the style of **DNP** , one needs to refer to the bases of the two component processes, which is not allowed for a truly *compositional* semantic definition, in which only can be referred to the *semantics* of the components.

2. A **TNP** term $P_\beta = S_1$ **in** S_2 denotes a process S_2 where occurrences of p within S_2 call for execution of S_1. So intuitively one expects this process to be equivalent to S_2 with all occurrences of P_β replaced by S_1, denoted as $S_2[S_1/P_\beta]$. Unfortunately this need not be the case for a **DNP** style parallel construct as is seen by the following example:

   $$S \stackrel{\text{def}}{=} P_{(\{c,d\},\emptyset)} = c!0 \textbf{ in } P_{(\{c,d\},\emptyset)} \parallel (d!0\,;c?x)$$

   $$S' \stackrel{\text{def}}{=} c!0 \parallel (d!0\,;c?x)$$

   For process S, synchronization is enforced for both channel c and d, since they occur both free in the two components of the parallel construct. For process S' on the other hand, only synchronization for channel c is required. Therefore S is not equivalent to S'.

The upshot of this is that $P_\beta = S_1$ **in** S_2 need not be equivalent to $S_2[S_1/P_\beta]$ if $base(S_1)$ is included but not equal to $base(P_\beta)$. We regard this as undesirable.

For **TNP** we resolved these two problems by including explicit bases, within the parallel construct, for which synchronization is required. The semantic definition, to be given in chapter 3, now simply refers to these explicitly indicated bases rather than to syntactic attributes of component processes. And so the problem with compositionality has disappeared. The second problem above is handled by not only allowing for the construct $S_1 \ \beta_1 \parallel \beta_2 \ S_2$ that $base(S_i)$ equals β_i, but *also* the situation that $base(S_i)$ is properly included into β_i for $i = 1, 2$. (This is the restriction CSR1). The synchronization set is in all cases determined by β_1 and β_2. Therefore, replacing S_i by some semantically equivalent process S_i' that differs with respect to bases from S_i does not affect the semantics of the whole.

We adopt the notational convention that

- $S_1 \ \beta_1 \parallel S_2$ abbreviates $S_1 \ \beta_1 \parallel_{base(S_2)} S_2$,
- $S_1 \parallel_{\beta_2} S_2$ abbreviates $S_1 \ {}_{base(S_1)} \parallel_{\beta_2} S_2$, implying that
- $S_1 \parallel S_2$ abbreviates $S_1 \ {}_{base(S_1)} \parallel_{base(S_2)} S_2$.

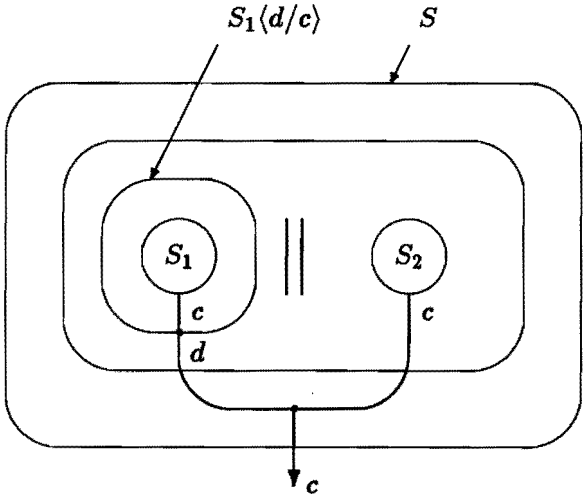
of the different readers (writers) will be between *pairs* of processes, rather than between *all* processes connected to that channel.

Example

Assume that both S_1 and S_2 communicate via channel c. Using some auxiliary channelname d, not among the names in $chan(S_1, S_2)$, we define the process

$$S \equiv (S_1\langle d/c\rangle \parallel S_2)\langle c/d\rangle,$$

visualized as in the picture below.



Both S_1 and S_2 communicate via c, but their communication actions are *not* synchronized since on the level where the parallel binding takes place the c channel of S_1 is renamed, and so has nothing to do with the c channel of S_2 . Nevertheless, still one more level outside, the renamed channel of S_1 is renamed back into c, with the effect that c communications send to $(S_1\langle d/c\rangle \parallel S_2)\langle c/d\rangle$ are divided over the d channel of $S_1\langle d/c\rangle$ and the c channel of S_2 . So any value communicated via c with S will be communicated with either S_1 or S_2, but not by both. It is interesting to see this "pairwise" mechanism easily expressed in terms of the multiple process synchronization mechanism. The other way around seems much more cumbersome.

In contrast with the languages CCS and TCSP, **TNP** includes the concept of program *states*. That is, a process modifies assignable variables x via explicit assignment statements $x := e$ and via communication actions $c.x:b$. Assignable variables x need not to be "declared". However, they can be made *local* to a process by means of the hiding construct $S_1\backslash\mathbf{x}$. This hiding construct is analogous to the block construct in combination with variable

declarations for ALGOL like languages. It differs from these block constructs in that initialization problems have been circumvented by assuming that no state change takes place at all on entry of the scope of the hiding operator $\backslash \mathbf{x}$. Changes made to variables in $\mathbf{x}$ by S_1 are simply undone for the process $S_1 \backslash \mathbf{x}$. As can be seen from the semantic definitions and the proof rules in the following chapters, this results in remarkable similarities between channel hiding on the one hand and variable hiding on the other. Variable hiding is an essential construct in the presence of recursion, for, if some process p using variable x creates a parallel network with x occurring in one parallel component, and a recursive incarnation of p in another component, then clearly the x of this incarnation must be hidden to avoid the sharing of x.

The similarity between channel hiding and variable hiding raised the question whether there is such an analogy for other constructs too. To some extend this is indeed the case. For instance in the next section we define an abbreviation $S\langle \bar{y}/\bar{x} \rangle$ which can be seen as the analogue of $S\langle d/c \rangle$. Another example is the generalization of the parallel construct with respect to assignable variables. Although shared variable concurrency is not described by the semantics of **TNP**, we nevertheless did not impose any restriction on variables for $S_1 \ \beta_1 \parallel \beta_2 \ S_2$. That is, although the semantics has a clear operational explanation *only* for those cases that $var(\beta_1) \cap var(\beta_2) = \emptyset$, we have not made the latter condition one of the context sensitive restrictions on **TNP**. The reason for doing this is twofold: The treatment of channels and variables for parallelism becomes uniform, which is a *technical* advantage. And secondly, we prepare the way for studying systems where it is not possible to determine *statically* appropriate bounds for the part of the state space modified by some process. A simple example of this is the situation where S_1 and S_2 share some array typed variable. It might well be the case that they operate on disjoint elements of the array, but since it is not a decidable question whether this is so or not, our semantics should assign a meaning in *all* cases.

Let us nevertheless give an informal description of the semantics of $S_1 \ \beta_1 \parallel \beta_2 \ S_2$ in case $var(\beta_1) \cap var(\beta_2) \neq \emptyset$. The idea is that each of the two processes execute with its own private copy of the state. That is, assignments made by S_1 to some variable x are not seen at all by S_2 and vice versa. If both S_1 and S_2 have terminated, then the parallel construct terminates, too, and all *modifications* made to the state by S_1 or S_2 are taken over as modifications made by the whole construct, *except* when both S_1 and S_2 have modified the same variable x, but in a different way. In this last case the computation is *aborted*.

Example

For the process

$$x := 2 \parallel (x := 1 \textbf{ or } x := 2)$$

the only computation that terminates is the one for which x is set to 2. Compare this with the process:

$$c.x!(x = 2) \parallel c.x?(x = 1 \vee x = 2)$$

Here only the value 2 can be communicated.

The process **abort** simply aborts the computation in the sense that it neither terminates nor performs any communication. The use of this **abort** term is mainly that it provides a notation, within the programming language, for the least element of the domain of process denotations to be introduced in the next chapter.

Instead of the **if**...**fi** construct of **DNP**, boolean expressions b are incorporated as processes. They function as "guards": whenever b evaluates to **true** the guard can be passed, i.e. it is equivalent to **skip** in this case. When b evaluates to **false** the guard cannot be passed and the computation is aborted. Clearly the guard **false** has the same meaning as **abort**. In chapter 3, we will argue that, as far as the class of so called *safety properties* that we are interested in is concerned, only the *finite observations* of processes have to be taken into account. Consequently, it turns out that a construct as **if** b **then** S_1 **else** S_2 **fi** can be regarded as equivalent to $b\,;S_1$ **or** $\neg b\,;S_2$.

We have an analogous situation for the **while** loop, using the *recursion* construct $\mu_z P_\beta.S_1$ in **TNP**. Intuitively, $\mu_z P_\beta.S_1$ behaves like S_1, except that calls of the form P_β within S_1 result in a new "recursive" incarnations of $\mu_z P_\beta.S_1$. We require that all free channels and assignable variables of S_1 are listed in the base β of the process variable P_β (cf. CSR4).

Now let $S_1{}^*$abbreviate $\mu_z p.(\textbf{skip or } S_1;p)$, where the base of p equals $base(S_1)$. Clearly, the $S_1{}^*$ construct denotes an arbitrary number of repetitions of S_1. Then, as far as safety properties are concerned, **while** b **do** S_1 **od** is equivalent to $(b\,;S_1)^*\,;\,\neg b$.

Similarly, initialization to some "random" value for local variables, if so desired, can be modelled by the process $x :=?$ which is an abbreviation for: $x := 0\,;\,(x := x+1)^*$

Processes can be given a name by means of the *process naming* construct $P_\beta = S_1$ **in** S_2. Here, restriction CSR3 requires that the base β includes all

free channels and assignable variables of the body S_1. Within S_2 , process-*calls* of the form P_β create a new incarnation of S_1. The restriction CSR2 excludes recursive calls of P_β in S_1.

We shall use the abbreviation **rec** $P_\beta = S_1$ **in** S_2 for the expression $P_\beta = \mu_x P_\beta.S_1$ **in** S_2 . (Clearly this is similar to $P_\beta = S_1$ **in** S_2 except that now recursive calls of P_β S_1 *are* allowed in S_1).

Finally we show how "shared variables" can be modelled in **TNP**. Assume that we want a parallel construct $S_1 \parallel S_2$ where S_1 and S_2 share some variable x in the sense that all changes made to x by one process are seen by the other process, as is the case for usual shared variable concurrency. We model this by a **TNP** process

$$(S_1 \parallel S_2)\langle assign/assign', value/value'\rangle \parallel XVAR,$$

where $XVAR$ is the process:

$$((assign?x \textbf{ or } value!x)^*)\backslash\{x\}.$$

If process S_1 wants to assign to the shared variable, then it must execute a *assign!e* command, whereas it must execute *value?y* to read the value of this variable. Process S_2 operates essentially in the same way except that it reads and writes the shared variable via the channels *assign'* and *value'*. Note that the channel renaming after the parallel composition of S_1 and S_2 is the same technique as used above to obtain a *pairwise* communication mechanism. Clearly pairwise communication is the right choice here since read and write actions by one process are to be *interleaved* rather than *synchronized* with read and write actions of the other process.

2.4 Parametrization of TNP processes

Unlike **DNP** there is no parameter mechanism associate with process declarations and -calls in **TNP**. Nevertheless, we can obtain essentially the same sort of parametrization as in **DNP** by using a few abbreviations.

- $S\langle\bar{d}/\bar{c}\rangle$, where $\bar{c}$ and $\bar{d}$ are *lists* rather then single channels denotes *simultaneous* renaming of channels. Clearly this can be expressed within **TNP**, using a list of fresh names $\bar{f}$, as

$$S\langle f_0/c_0\rangle\langle f_1/c_1\rangle \ldots \langle f_{n-1}/c_{n-1}\rangle\ \langle d_0/f_0\rangle \ldots \langle d_{n-1}/f_{n-1}\rangle.$$

- The *simultaneous assignment* $\bar{x} := \bar{e}$ is executed by first evaluating the expressions from the list $\bar{e}$, followed by assigning the values thus obtained to the variables $\bar{x}$ from left to right. It is easily expressed in terms of the simple assignment as follows. Let $z_1, \ldots, z_n$ be a list of fresh variables. Then $\bar{x} := \bar{e}$ is equivalent to:

$$(z_1 := e_1; \ldots; z_n := e_n; x_1 := z_1; \ldots; x_n := z_n) \backslash \{z_1, \ldots, z_n\}.$$

- $S\langle \bar{y}/\bar{x} \rangle$, where $\bar{x}, \bar{y} \in \mathcal{V}ar^*$, abbreviates: $(\bar{x} := \bar{y} \ ; \ S \ ; \ \bar{y} := \bar{x}) \backslash \mathbf{z}$, where $\mathbf{z}$ is defined by $\mathbf{z} = \{\bar{x}\} - \{\bar{y}\}$.

We call this the *parameter transfer* construct. Intuitively, it models the call-by-value/result mechanism of **DNP** process calls, where the $\bar{x}$ play the role of formal parameters and the $\bar{y}$ the role of the actuals. The hiding "$\backslash \mathbf{z}$", ensures that the "formals" are not included into the set of local variables of the calling environment, except for those that happen to have the same name as some "actual".

Using these abbreviations, the **DNP** process declaration and -calls can be translated into **TNP** expressions as follows:

- A *declaration* $P(\bar{c}_i; \bar{c}_o; \bar{x})$ **begin** S_0 **end** translates into:

$$P_{(\mathbf{c},\mathbf{x})} = \mu_z P_{(\mathbf{c},\mathbf{x})} \cdot (\bar{z} := \bar{\omega} \ ; \ S_0) \backslash \mathbf{z} \ \mathbf{in} \ldots,$$

  with $\mathbf{c} = \{\bar{c}_i, \bar{c}_o\}$, $\mathbf{z} = \{\bar{z}\} = var(S_0) - \{\bar{x}\}$, and where ω is some appropriately chosen initialization constant.

- A call $P(\bar{d}_i, \bar{d}_o, \bar{y})$ translates into:

$$P_{(\mathbf{c},\mathbf{x})} \ \langle \bar{d}/\bar{c} \rangle \ \langle \bar{y}/\bar{x} \rangle,$$

  where $\bar{c} = \bar{c}_i, \bar{c}_o$ and $\bar{d} = \bar{d}_i, \bar{d}_o$.

We have included the initialization of the local variables $\mathbf{z}$ of the procedure body S_0 to avoid differences with the definition of **DNP** in [ZRE].

2.5 Translation of DNP into TNP

We summarize how to transform any **DNP** program into an equivalent **TNP** process, by defining a compositional, i.e. syntax directed, translation scheme. Let C denote a **DNP** Statement S, Network N or Program *Prog*. We define a function Tl : **DNP** $\rightarrow$ **TNP** by means of the following table.

C	$Tl(C)$
skip	**skip**
$x := e$	$x := e$
$c!e$	$(c.x : x = e)\backslash\{x\}$, where $x \notin var(e)$
$c?x$	$c.x : true$
$S_1 \,;\, S_2$	$Tl(S_1)\,;\, Tl(S_2)$
S_1 **or** S_2	$Tl(S_1)$ **or** $Tl(S_2)$
if b **then** S_1 **else** S_2 **fi**	$(b\,;Tl(S_1))$ **or** $(\neg b\,;Tl(S_2))$
while b **do** S **od**	$(b\,;\, Tl(S))^*;\neg b$
cobegin N **coend**	$Tl(N)\backslash\mathbf{c}$, where $\mathbf{c} = intern(N)$
$P(\bar{d}_i, \bar{d}_o, \bar{y})$	$P_{(\mathbf{c},\mathbf{x})}\ \langle \bar{d}/\bar{c}\rangle\ \langle \bar{y}/\bar{x}\rangle$ where $\bar{c} = \bar{c}_i, \bar{c}_o$ and $\bar{d} = \bar{d}_i, \bar{d}_o$
$S_1 \parallel S_2$	$Tl(S_1) \parallel Tl(S_2)$
$P(\bar{c}_i; \bar{c}_o; \bar{x})$ **begin** S_0 **end** : $Prog$	$P_{(\mathbf{c},\mathbf{x})} = \mu_{\mathbf{z}} P_{(\mathbf{c},\mathbf{x})} \cdot (\bar{z} := \bar{\omega}; Tl(S_0))\backslash\mathbf{z}$ in $Tl(Prog)$, where $\mathbf{c} = \{\bar{c}_i, \bar{c}_o\}$, $\mathbf{z} = \{\bar{z}\} = var(S_0) - \{\bar{x}\}$

Chapter 3

The semantics for TNP

3.1 Introduction

In this introductory section we discuss certain aspects of semantic definitions for programming languages from a rather general point of view. We take the opportunity to define already some of the domains to be used for our particular semantics for **TNP**, as an illustration of the general concepts.

For CSP-like languages a number of semantic definitions have been given. ([Hoare3], [FHLR], [FLP], [OH]). In general these definitions differ considerably with respect to the degree in which they abstract from behavioural properties of programs. To discuss the idea of "program behaviour" we assume, or better postulate, that with a given programming language some set of observable events "Event" is associated. We acknowledge that, even for a single language, there are several reasonable choices for this set, each leading to a different semantics. To describe our choice for the case of **TNP**, we introduce the following basic domains:

Definition 3.1 (the domains $\mathcal{V}al$,$\mathcal{A}$ and *State*)

$(v \in)$ $\mathcal{V}al$ — Some given countable set of (proper) *values.*

$(a \in)$ $\mathcal{A} \stackrel{\text{def}}{=} Chan \times \mathcal{V}al$ — The *communication alphabet.*

$(s \in)$ $State \stackrel{\text{def}}{=} \mathcal{V}ar \to \mathcal{V}al$ — The set of *proper states.*

□

For **TNP** we postulate that Event consists of the following types of events:

(1) the event of *starting* a computation in some *initial state* s_0,

(2) the events of the form $(c, v) \in \mathcal{A}$. These represent the *communication* of some value v via channel c,

(3) the event of *termination* in some final state s.

Event is almost the smallest reasonable set of observables for **TNP**. For instance, we have not included a deadlocked state as directly observable. Our particular choice is motivated by the type of properties of processes that we can analyse within the proof systems studied here.

For any execution of some given process S, after any finite amount of time, some finite sequence λ of such events will have occurred. The collection of all finite nonempty sequences of events that can arise in this way during execution of processes, is called the set of *finite observations* Λ. If λ $(\in \Lambda)$ can occur when we execute S, we will say that S *admits* the observation λ. We denote the set of observations that some process admits by $\mathcal{O}bs(S)$.

Remark Below we shall introduce the notation $\mathcal{O}bs[\![S]\!]\eta$, which denotes the same set of observations. It has the extra argument η to cope with free occurrences of process variables. For the moment we omit this argument, since it does not affect the present argumentation.

It will be clear that if S admits λ and λ' is some prefix of λ, then S must admit λ' too. For **TNP** we can divide Λ into *unfinished* computations U and finished or *terminated* computations F:

$$U = \{(s_0, t) \mid s_0 \in \mathit{State}, t \in \mathcal{A}^*\}.$$

$$F = \{(s_0, t, s) \mid s_0, s \in \mathit{State}, t \in \mathcal{A}^*\}.$$

U-computation, *started* in initial state s_0, have performed the sequence of *communications* t thus far. F-computations additionally did *terminate* in final state s. To represent unfinished and terminated computations in a uniform way, we introduce a special state "$\perp$" called the *bottom state*. A bottom state indicates an unfinished computation. For the technical development to follow, it is convenient to treat $\perp$ as the least element of a complete partial order (cpo). For the same - technical - reason, we introduce a bottom value $\perp_{\mathit{val}}$ and a bottom channel $\perp_{\mathit{chan}}$.

Definition 3.2 $(\mathit{State}_\perp, \mathit{Val}_\perp, \mathit{Chan}_\perp, \mathit{Trace}, \Delta)$

- $\mathit{State}_\perp, \mathit{Val}_\perp$ and $\mathit{Chan}_\perp$ are defined as the flat cpo's derived from State, Val and Chan.The corresponding least elements are denoted as $\perp, \perp_{\mathit{val}}$ and $\perp_{\mathit{chan}}$.
- $(h, t \in)\ \mathit{Trace} \stackrel{\text{def}}{=} (\mathcal{A}^*, \preceq)$ — The p.o. of communication histories or *traces*, ordered by the prefix order on sequences. The *empty* trace is denoted by ε.

- $(\delta \in)\ \Delta \stackrel{\text{def}}{=} (State \times Trace \times State_\perp) \cup \{(\perp, \epsilon, \perp)\}$ — The set of finite observations or computation triples.
- $Proper(\Delta)$ denotes the subset $State \times Trace \times State_\perp$ of *proper observations* of Δ

□

The triple $(\perp, \epsilon, \perp)$ has been added to the set Δ to allow for a simpler treatment of sequential composition. Informally it can be understood as representing the fact that if a process is not even started, for instance because its sequential predecessor did not terminate, then it will neither communicate nor terminate.

For the purpose of process specification we are interested in those process properties for which the truth or falsity is determined by the set of possible observations of a process. In fact, one should be able to tell for any particular observation whether it violates some given specification or not.

To make this more precise, let us assume that *Prop* is some class of properties of processes, i.e., for each property $\pi \in Prop$ and process S, $\pi(S)$ might hold or might not hold.

A binary predicate $Ref(\pi, \delta)$ on properties π and observations δ is called a *refutation criterion* for *Prop* if the following holds:

$$\forall \pi, S\Big(\exists \delta \in Obs(S) : Ref(\pi, \delta) \Rightarrow \neg\pi(S)\Big) \qquad (1)$$

If $Ref(\pi, \delta)$ holds, we say that the observation δ refutes $\pi(S)$. Equation (1) states simply that Ref is correct in the sense that a refutation of $\pi(S)$ indeed implies that $\pi(S)$ cannot hold. Now consider some hypothetical specification π and process S such that on the one hand $\pi(S)$ does not hold, but on the other hand there is no possible S observation that refutes $\pi(S)$. We regard this specification as uninteresting since a claim that some process satisfies it does not imply any guarantee in terms of observations. We want to exclude such specifications. To this end, a class *Prop* is said to consist of *falsifiable* properties if there is some refutation criterion Ref for *Prop* such that:

$$\forall \pi, \forall S\Big(\neg\pi(S) \Rightarrow \exists \delta \in Obs(S) : Ref(\pi, \delta)\Big) \qquad (2)$$

In words: if π is not valid for S, then there is at least one observation possible about S that refutes $\pi(S)$. Properties that are falsifiable by means of *finite* observations are called *safety properties*. In this thesis the safety properties of processes are the focus of attention.
For a class of falsifiable properties it follows from (1) and (2) that validity

of some property π for processes S can be expressed as follows, in terms of the observations that S admits:

$$\pi(S) \iff \forall \delta \in Obs(S) : \neg Ref(\pi, \delta)$$

Now if we associate with a property π the set of observations that do not refute it, that is, if we define:

$$Obs(\pi) = \{\delta \in \Delta \mid \neg Ref(\pi, \delta)\},$$

then we see that we have the following characterization of validity for safety properties:

$$\forall \pi, S\Big(\pi(S) \iff (Obs(S) \subseteq Obs(\pi))\Big)$$

We mention here the work of [OH] where essentially this last formula was taken as the meaning of specifications.

All this suggests that the meaning of **TNP** processes as well as the meaning of safety properties concerning these processes should be expressed by means of so called Δ-predicates:

Definition 3.3 (Δ-predicates)

Let $\mathcal{P}$ denote the usual powerset operation. The domain of Δ predicates, with typical element ρ, is defined as:

$(\rho \in)\ \mathcal{P}(\Delta)$ — the set of Δ-predicates.

□

3.2 The domain of observations

Up to now we have not taken into account that **TNP** includes *recursive* constructs, and so, that our semantics includes fixed point equations of the form $\rho = f(\rho)$. We adopt the standard solution of denotational semantics to this problem. That is, we will turn $\mathcal{P}(\Delta)$ into a complete partial order (cpo), and rely on the fixed point theorem of Kleene [Kleene], to determine the unique least solution of the equation.

A cpo structure on some set U consists of a partial order $\sqsubseteq$ on U that has a *least element* $\perp_U$ in U, and that is *complete* in the sense that every countable infinite chain $\rho_0 \sqsubseteq \rho_1 \sqsubseteq \cdots$ has a least upper bound (lub) $\bigsqcup_{i \in \mathbf{N}} \rho_i$ in U. This cpo structure is denoted by $(U, \sqsubseteq, \perp_U)$.

A function f from a cpo U to a cpo W is *continuous* if it preserves least upper bounds of countable ascending chains, that is, if for every ascending chain $\langle \rho_i \rangle_{i \in \mathbf{N}}$:

$$f(\bigsqcup_{i \in \mathbf{N}} \rho_i) = \bigsqcup_{i \in \mathbf{N}} f(\rho_i).$$

The theorem of Kleene states that every continuous function φ on a cpo U has a *least fixed point* $\mu(\varphi)$ in U, and moreover that this fixed point can be obtained as the least upper bound (lub) of an ascending chain as follows:

$$\mu(\varphi) = \bigsqcup_{i \geq 0} \varphi^i(\perp_U),$$

where φ^i is defined by:

$\varphi^0 \stackrel{\text{def}}{=} Id$, and

$\varphi^{i+1} \stackrel{\text{def}}{=} \varphi \circ \varphi^i$, for $i \geq 0$.

There is no general agreement as to which type of cpo structure must be used for concurrent programs. In fact this depends very much on which type of observable events one wants to describe. Therefore we let our precise mathematical definitions precede by an intuitive development of the particular cpo structure that we have chosen.

The first task is to determine which set of computations in $\mathcal{P}(\Delta)$ is the appropriate *least element* for our cpo structure. This might sound strange since, as yet, we have not even determined a *partial order* on $\mathcal{P}(\Delta)$. However, in the standard approach to denotational semantics the meaning of the recursive process $\mu_z P_\beta.P_\beta$ is the least solution of an equation of the form $\rho = \rho$, and since clearly it is the case that *every* $\mathcal{P}(\Delta)$ element is a solution, the least cpo element must coincide with the set of computations admitted by this process. Because the process $\mu_z P_\beta.P_\beta$ never terminates nor communicates, but only allows for the observation of starting a computation in any initial state, the appropriate least element is the following one:

$$\mathbf{Z} \stackrel{\text{def}}{=} \{(s_0, \varepsilon, \perp) \mid s_0 \in \mathit{State}_\perp\}.$$

Note that, according to the intuitive explanation of the **TNP** constructs in chapter 2, $\mathbf{Z}$ is exactly the set of computations admitted by the process **abort**. In fact we included the **abort** process to have a denotation within **TNP** for the least element of our domain. Using the **abort** process, we are able to define the so called *syntactic approximations* $S^{[i]}$ for some recursive process $\mu_z P_\beta.S$.

Definition 3.4 (Syntactic approximations)

Let S be some process with possible occurrences of the process variable P_β, and let S' be some process with its base contained in β. Then $S[S'/P_\beta]$ denotes the process obtained from S by replacing all occurrences of P_β in S by S'.

For the recursive process $\mu_z P_\beta.S$, we define:

$S^{[0]} \stackrel{\text{def}}{=} \textbf{abort}$

$S^{[i+1]} \stackrel{\text{def}}{=} S[S^{[i]}/P_\beta]$, for $i \geq 0$.

□

We use syntactic approximations to determine the appropriate cpo structure for our domain. To this end we consider a process $\mu_z P_\beta.S$ where S does contain neither (nested) recursive processes, nor process calls other than of P_β. On the one hand, since syntactic approximations are *finite* processes in this case, one can use operational insight to determine the observations admitted by these. On the other hand, in the standard approach to denotational semantics, the fixed point that is the meaning of a recursion construct can be calculated as the lub of an ascending chain that consists of the interpretations of the syntactic approximations.

It is clear that if some observation δ of $\mu_z P_\beta.S$ corresponds to some execution where the depth of recursive calls of P_β is at most i, then this observation is *also* admitted by the process $S^{[i]}$. And since a *finite* observation corresponds to some finite depth of recursive calls, *every* finite observation of $\mu_z P_\beta.S$ is admitted by some $S^{[i]}$.

Next we argue that every finite observation admitted by $S^{[i]}$ is also admitted by $\mu_z P_\beta.S$. For computations that correspond to a recursion depth of at most i this is clear. About computations that reach a recursion depth *greater* than i we can remark the following. Assume that at the moment where, for the first time, the depth $i+1$ is reached, the sequence of communications t' has already performed. The execution has reached some internal, not observable state s'. For $\mu_z P_\beta.S$, the execution then would proceed from state s', possibly extending the sequence t' by performing new communications, and possibly by reaching an observable final state. But for $S^{[i]}$ an occurrence of an **abort** process is started, implying that no further communications take place and that no final state will be reached. That is, $S^{[i]}$ admits the observation $(s_0, t', \perp)$. The interesting point here is that the same observation is *also* admitted by $\mu_z P_\beta.S$, for it corresponds to the intermediate stage of the

computation where the recursion depth $i+1$ is reached. What we have argued is that every finite observation admitted by $S^{[i]}$, whether it corresponds to some execution where one of the occurrences of **abort** has been reached or not, is also admitted by $\mu_z P_\beta.S$, and vice versa, that every finite observation admitted by $\mu_z P_\beta.S$ is also admitted by some $S^{[i]}$. But this implies that the meaning of $\mu_z P_\beta.S$, as a set of finite observations, simply is the union of the sets that are the meanings of the $S^{[i]}$.

By the reasoning above one sees that the set of observations denoted by $S^{[i]}$ is *included* in the set denoted by $S^{[i+1]}$. Therefore the chain $\langle \mathit{Obs}(S^{[i]})\rangle_i$ is monotone increasing with respect to the set inclusion order on $\mathcal{P}(\Delta)$and the union of the sets $\mathit{Obs}(S^{[i]})$ coincides with the least upper bound of this chain. We conclude that the set inclusion order is the appropriate one for our cpo structure.

For this order it is even the case that *every* collection $\{\rho_i \mid i \in I,\ \rho_i \in \mathcal{P}(\Delta)\}$ has a least upper bound $\bigsqcup_{i\in I} \rho_i$ in $\mathcal{P}(\Delta)$. It is determined by:

$$\bigsqcup_{i\in I} \rho_i = \bigcup_{i\in I} \rho_i,$$

where it is understood that the union of an empty collection of sets is the empty set.

If not only ascending chains, but rather *every* subset of some partial order U has a least upperbound, then U is called a *complete lattice*. A complete lattice has a least element, determined as the lub of the empty set. If f is a function from a complete lattice U to a complete lattice W, then f is called *completely additive* (c.a.), if it preserves least upper bounds of arbitrary subsets of U. Clearly a complete lattice is a cpo, and a completely additive function is continuous. Although a cpo structure suffices for our semantics, we shall develop our domain as a complete lattice since it is technically somewhat more convenient. For the operations on this domain we shall often prove complete additivity rather than continuity, for the same reason. The same notation is used for complete lattices and cpos.

The following structure is a complete lattice:

Definition 3.5 (The domain $\mathcal{P}(\Delta)$)

$(\mathcal{P}(\Delta), \subseteq, \emptyset)$ is the domain of *all* subsets of Δ with the set inclusion order, and the *empty set* as least element.

□

At this point a technical problem arises, for we have defined a domain structure with the empty set as least element, although we argued above that

this least element should be the set $\mathbf{Z}$, and so, that the appropriate domain is the following one:

Definition 3.6 (The domain $\mathcal{P}_{\mathbf{Z}}(\Delta)$)

- A subset ρ of Δ is called *total* if $\mathbf{Z} \subseteq \rho$.
- $(\mathcal{P}_{\mathbf{Z}}(\Delta), \subseteq, \mathbf{Z})$ is the domain of all total subsets with the set inclusion order, and $\mathbf{Z}$ as least element.

□

Here we must be careful. Although we argued that every **TNP** process must have a meaning within the subset $\mathcal{P}_{\mathbf{Z}}(\Delta)$, this is *not* the case for the *specifications* that we shall use for such processes. Such specifications essentially have the form of a predicate logic formula, interpreted as a subset of Δ. Not all such formulae do admit all computations from $\mathbf{Z}$. A simple example is the predicate **"false"** that does not admit *any* computation. This shows that $\mathcal{P}_{\mathbf{Z}}(\Delta)$ does not suffice as the domain of interpretation for such predicates. In chapter 4 we shall introduce the language "Mixed terms", in which **TNP** processes and specifications are within the same syntactic category, implying that the (common) domain of interpretation must be $\mathcal{P}(\Delta)$.

Now the problem is that certain equations of the form $\rho = f(\rho)$ will have *different* least solutions, dependent on whether we solve the equation in the domain $\mathcal{P}(\Delta)$ or in the domain $\mathcal{P}_{\mathbf{Z}}(\Delta)$. For instance, in the former case the equation $\rho = \rho$ has $\emptyset$ as least solution, whereas in the latter case $\mathbf{Z}$ is the least solution.

More interesting examples are processes that need not terminate but rather keep on communicating, like the *Bag* example of chapter 1. A simple example of such a process is $\mu_z P_\beta \,.\, (c!0; P_\beta)$; a process that never terminates but rather communicates forever via c. Hence it admits the following observations:

$$\{(\bot, \epsilon, \bot)\} \cup$$
$$\{(s_0, \epsilon, \bot), (s_0, <(c,0)>, \bot), (s_0, <(c,0)(c,0)>, \bot) \ldots \mid s_0 \in State\} \qquad (*)$$

Now with our semantic definitions the fixedpoint equation for this recursive process is essentially the following one:

$$\rho = \{(s_0, t_1 t_2, s) \mid \exists s_1[(s_1, t_1, s_1) \in Obs(c!0) \wedge (s_1, t_2, s) \in \rho]\} \qquad (**)$$

The set $Obs(c!0)$ of observations admitted by $c!0$ is:

$$\{(\bot, \epsilon, \bot), (s_0, \epsilon, \bot), (s_0, <(c,0)>, \bot), (s_0, <(c,0)>, s_0) \mid s_0 \in State\}.$$

Thus it is seen that (*) is indeed a possible solution of the equation (**). However, it is not the least one within $\mathcal{P}(\Delta)$, for $\emptyset$ is a solution too!

The examples show that solving equations in $\mathcal{P}(\Delta)$ does not always yield the right solution with respect to the intuitive operational semantics of **TNP**. To correct this situation, we first examine the relationship between the two domains in more detail. The underlying idea is that we use insight about the intuitive operational semantics of **TNP** processes to determine the appropriate semantic operations corresponding to the **TNP** operations on the restricted domain $\mathcal{P}_{\mathbf{Z}}(\Delta)$. Then, as the next step, we try to extend these operations to the complete domain $\mathcal{P}(\Delta)$ in a canonical way. Finally we define an embedding that allows us to extend the least fixed point operator μ_z to an operator defined on $[\mathcal{P}(\Delta) \to \mathcal{P}(\Delta)]$ that can be used to determine the semantics of the recursion construct. The embedded operator differs from the μ operator for $[\mathcal{P}(\Delta) \to \mathcal{P}(\Delta)]$ in that as far as **TNP** processes are concerned it yields a meaning that matches the operational intuition.

We start with a straightforward embedding of $\mathcal{P}_{\mathbf{Z}}(\Delta)$ into $\mathcal{P}(\Delta)$by means of the *inclusion function:*

$$i : \mathcal{P}_{\mathbf{Z}}(\Delta) \to \mathcal{P}(\Delta)$$

Since the order on $\mathcal{P}_{\mathbf{Z}}(\Delta)$ is the restriction of the order on $\mathcal{P}(\Delta)$ the inclusion function is completely additive. Moreover, some function $f : D \to \mathcal{P}_{\mathbf{Z}}(\Delta)$ is c.a. iff $i \circ f$ is c.a. Cfr. [Arbib] this means that $\mathcal{P}_{\mathbf{Z}}$ is a *substructure* of $\mathcal{P}(\Delta)$ in the category of complete lattices and c.a. functions.

The inclusion function has a left inverse in the form of a c.a. *projection function* π. That is, as we shall show now, we have a function:

$$\pi : \mathcal{P}(\Delta) \to \mathcal{P}_{\mathbf{Z}}(\Delta),$$

that satisfies the equality:

$$\pi \circ i = Id,$$

where Id is the identity function. From this equality it follows that, for arbitrary $\rho \in \mathcal{P}(\Delta)$,

$$\pi(\rho \cup \mathbf{Z}) = \pi(i(\rho \cup \mathbf{Z})) = \rho \cup \mathbf{Z}.$$

Then, using also the additivity of π, we see that

$$\pi(\rho) = \pi(\rho) \cup \mathbf{Z} = \pi(\rho) \cup \pi(\mathbf{Z}) = \pi(\rho \cup \mathbf{Z}) = \rho \cup \mathbf{Z}.$$

That is, π is determined by the following equality.

$$\pi(\rho) = \rho \cup \mathbf{Z}.$$

This also shows that, although i has no right inverse, the following inequality does hold:

$$i \circ \pi \sqsupseteq Id,$$

where $\sqsubseteq$ denotes the pointwise ordering on operations on $\mathcal{P}(\Delta)$. Because of this, the pair (π, i) is called a *continuous closure*, cfr. [Sanchis].

Remark A continuous closure resembles a *continuous projection pair* as used in D. Scott's domain theory [Scott], except that for such a pair one would have the inequality $i \circ \pi \sqsubseteq Id$ instead of the inequality above. Both continuous closures and projections are instances of what is called a *continuous representation* in [Sanchis]. □

We want to define a similar embedding and projection for *operations* on the two domains. To this end we first extend the injection and projection above to cartesian products for we must be able to treat operations like (the interpretation of) sequential composition, that have more than one argument.

Therefore, for tuples $(\rho_1, \ldots, \rho_n) \in (\mathcal{P}(\Delta))^n$ we define:

$$\pi((\rho_1, \ldots, \rho_n)) \stackrel{\text{def}}{=} (\pi(\rho_1), \ldots, \pi(\rho_n)).$$

The injection function i is extended similarly.

Based upon these functions π and i, we define a corresponding embedding and projection for operations defined on the two domains $\mathcal{P}(\Delta)$and $\mathcal{P}_{\mathbb{Z}}(\Delta)$.

Definition 3.7 (Projection and embedding of operations on $\mathcal{P}(\Delta)$)

For operations f on $\mathcal{P}(\Delta)$ and g on $\mathcal{P}_{\mathbb{Z}}(\Delta)$ we define:

$\Pi(f) \stackrel{\text{def}}{=} \pi \circ f \circ i$ " projection of f"

$\mathrm{E}(g) \stackrel{\text{def}}{=} i \circ g \circ \pi$ "embedding of g "

□

For operations f and g of one argument, we can illustrate this definition by means of commuting diagrams:

$$\begin{array}{ccc} \mathcal{P}(\Delta) & \xrightarrow{f} & \mathcal{P}(\Delta) \\ i \uparrow \downarrow \pi & & i \uparrow \downarrow \pi \\ \mathcal{P}_{\mathbb{Z}}(\Delta) & \xrightarrow[g]{} & \mathcal{P}_{\mathbb{Z}}(\Delta) \end{array}$$

Lemma 3.8

Π and E form a continuous closure, that is:

$$\Pi(\mathrm{E}(g)) = g$$

$$\mathrm{E}(\Pi(f)) \sqsupseteq f$$

□

Proof

$$\Pi(\mathrm{E}(g)) = \pi \circ (i \circ g \circ \pi) \circ i = (\pi \circ i) \circ g \circ (\pi \circ i) = Id \circ g \circ Id = g.$$

$$\mathrm{E}(\Pi(f)) = i \circ (\pi \circ f \circ i) \circ \pi = (i \circ \pi) \circ f \circ (i \circ \pi) \sqsupseteq Id \circ f \circ Id = f.$$

□

Some (n-place) operation f on $\mathcal{P}(\Delta)$ *preserves totality* if:

$$(\bigwedge_{i=1,n} \mathbf{Z} \subseteq \rho_i) \text{ implies } \mathbf{Z} \subseteq f(\rho_1, \ldots, \rho_n).$$

Every embedding $\mathrm{E}(g)$ of some operation g does preserve totality. For totality preserving operations f, the projection $\Pi(f)$ is simply the restriction of f to the domain $\mathcal{P}_{\mathbf{Z}}(\Delta)$. For if $\rho_i \in \mathcal{P}_{\mathbf{Z}}(\Delta)$ for $i = 1..n$, then $\mathbf{Z} \subseteq f(\rho_1, \ldots, \rho_n)$, and so we have the equality:

$$\Pi(f)(\rho_1, \ldots, \rho_n) = f(\rho_1, \ldots, \rho_n) \cup \mathbf{Z} = f(\rho_1, \ldots, \rho_n).$$

In fact $\mathrm{E}(g)$ is an extension of g in the sense that the operation $\mathrm{E}(g)$ restricted to $\mathcal{P}_{\mathbf{Z}}(\Delta)$ coincides with g again. This follows easily from the equality:

$$\mathrm{E}(g)(\rho) = i(g(\pi(\rho))) = g(\rho \cup \mathbf{Z}).$$

So for $\rho \in \mathcal{P}_{\mathbf{Z}}(\Delta)$ we have that $\mathrm{E}(g)(\rho) = g(\rho \cup \mathbf{Z}) = g(\rho)$.

It is easily checked that Π and E preserve continuity and complete additivity of their arguments. For instance, assume that f and g are c.a. The following calculations prove the complete additivity of $\Pi(f)$ and $\mathrm{E}(g)$.

$$\Pi(f)(\bigsqcup_{i \in I} \rho_i) = f(\bigcup_{i \in I} \rho_i) \cup \mathbf{Z} = (\bigcup_{i \in I} f(\rho_i)) \cup \mathbf{Z}$$

$$= \bigcup_{i \in I} (f(\rho_i) \cup \mathbf{Z}) = \bigsqcup_{i \in I} \Pi(f)(\rho_i), \text{ and}$$

$$\mathrm{E}(g)(\bigsqcup_{i \in I} \rho_i) = g((\bigcup_{i \in I} \rho_i) \cup \mathbf{Z}) = g(\bigcup_{i \in I} (\rho_i \cup \mathbf{Z}))$$

$$= \bigcup_{i \in I} g(\rho_i \cup \mathbf{Z}) = \bigsqcup_{i \in I} \mathrm{E}(g)(\rho_i).$$

This allows us to define a projection and an embedding for functionals F and G of the following types:

$F : [\mathcal{P}(\Delta) \to \mathcal{P}(\Delta)] \to \mathcal{P}(\Delta)$, and

$G : [\mathcal{P}_{\mathbb{Z}}(\Delta) \to \mathcal{P}_{\mathbb{Z}}(\Delta)] \to \mathcal{P}_{\mathbb{Z}}(\Delta)$.

Note that the least fixed point operator μ, defined for continuous functions in $\mathcal{P}(\Delta)$, and the analogous operator μ_z, defined for continuous functions in $\mathcal{P}_{\mathbb{Z}}(\Delta)$, are functionals of these types.

Definition 3.9 (Embedding and projection of functionals)

For functionals F and G as above, we define the projection $\Pi(F)$ and the embedding $E(G)$ as follows:

$\Pi(F) \stackrel{\text{def}}{=} \pi \circ F \circ \mathrm{E}$, and

$E(G) \stackrel{\text{def}}{=} i \circ G \circ \Pi$.

□

$$\begin{array}{ccc} [\mathcal{P}(\Delta) \to \mathcal{P}(\Delta)] & \xrightarrow{F} & \mathcal{P}(\Delta) \\ \mathrm{E} \uparrow \downarrow \Pi & & i \uparrow \downarrow \pi \\ [\mathcal{P}_{\mathbb{Z}}(\Delta) \to \mathcal{P}_{\mathbb{Z}}(\Delta)] & \xrightarrow[G]{} & \mathcal{P}_{\mathbb{Z}}(\Delta) \end{array}$$

Similar to above, the projection Π is the left inverse for the embedding E:

$$\Pi(E(G)) = \pi \circ E(G) \circ \mathrm{E} = \pi \circ i \circ G \circ \Pi \circ \mathrm{E} = Id \circ G \circ Id = G,$$

where we use the same notation for the identity functions on $\mathcal{P}_{\mathbb{Z}}(\Delta)$ and $[\mathcal{P}_{\mathbb{Z}}(\Delta) \to \mathcal{P}_{\mathbb{Z}}(\Delta)]$. Again, E has no right inverse, but we do have the following inequality:

$$E(\Pi(F)) = i \circ \Pi(F) \circ \Pi = i \circ \pi \circ F \circ \mathrm{E} \circ \Pi \sqsupseteq Id \circ F \circ Id = F.$$

This time we used "Id" both for the identity on $\mathcal{P}(\Delta)$ and on $[\mathcal{P}(\Delta) \to \mathcal{P}(\Delta)]$.

The following theorem forms the basis for our embedding of the μ_z operator. Point (i) of the theorem states that the least fixed point $\mu_z(g)$ of some operation g on $\mathcal{P}_{\mathbb{Z}}(\Delta)$ is preserved when both the operation g and the μ_z operator are embedded. Point (ii) is important because it relates the extended μ_z operator to the least fixed point operator μ on $[\mathcal{P}(\Delta) \to \mathcal{P}(\Delta)]$.

Theorem 3.10 (Embedding of the μ_z functional)

(i) $E(\mu_z)(\mathrm{E}(g)) = i(\mu_z(g))$

(ii) $E(\mu_z)(f) = \mu(i \circ \pi \circ f)$

□

Proof

Point (i) is simple:

$$E(\mu_z)(\mathrm{E}(g)) = i(\mu_z(\Pi(\mathrm{E}(g)))) = i(\mu_z(g)).$$

The proof of (ii) relies on the continuity, implying monotonicity, of f, i and π.

$$\begin{aligned}
\mu(i \circ \pi \circ f) &= \bigcup_{j\geq 0}(i \circ \pi \circ f)^j(\emptyset) = \bigcup_{j\geq 1}(i \circ \pi \circ f)^j(\emptyset) \\
&= \bigcup_{j\geq 1}(i \circ \pi \circ f)^{j-1}(i(\pi(f(\emptyset)))) = \bigcup_{j\geq 0}(i \circ \pi \circ f)^j(f(\emptyset) \cup \mathbf{Z}) \\
&\supseteq \bigcup_{j\geq 0}(i \circ \pi \circ f)^j(\mathbf{Z}) = \bigcup_{j\geq 0}(i \circ \pi \circ f)^j(i(\mathbf{Z})) \\
&= \bigcup_{j\geq 0} i((\pi \circ f \circ i)^j(\mathbf{Z})) = i(\bigcup_{j\geq 0}(\pi \circ f \circ i)^j(\mathbf{Z})) \\
&= i(\mu_z(\pi \circ f \circ i)) = i(\mu_z(\Pi(f))) = E(\mu_z)(f).
\end{aligned}$$

This proves inclusion from one side. The other side is shown as follows:

$$\begin{aligned}
\mu(i \circ \pi \circ f) &= \bigcup_{j\geq 0}(i \circ \pi \circ f)^j(\emptyset) \subseteq \bigcup_{j\geq 0}(i \circ \pi \circ f)^j(\mathbf{Z}) \\
&= \cdots(\text{as above}) = E(\mu_z)(f).
\end{aligned}$$

□

At last we can explain our strategy for defining semantic operations on $\mathcal{P}(\Delta)$. For each of the (syntactic) **TNP** operators *op* there is a corresponding semantic operation f_{op} *defined on* $\mathcal{P}_{\mathbf{z}}(\Delta)$, that captures the operational intuition. In the semantic definition we interpret the operator *op* as an operation on $\mathcal{P}(\Delta)$ however, and by now it will be obvious that we shall choose the embedding $\mathrm{E}(f_{op})$ as the meaning of *op*. We shall write f_{op} instead of $\mathrm{E}(f_{op})$ to avoid a cumbersome notation. When we come to the actual definitions we shall define f_{op} directly as an operation on $\mathcal{P}(\Delta)$ and we check that it preserves totality.

For interpretation of the recursion construct we shall use the embedded version $E(\mu_z)$ of the least fixed point operator μ_z. Again we write μ_z instead of the more cumbersome $E(\mu_z)$. It follows from the lemma above that we can treat μ_z on $\mathcal{P}(\Delta)$ as a derived operator, defined by:

$$\mu_z(f) = \mu(i \circ \pi \circ f) = \mu(\lambda\rho.(f(\rho) \cup \mathbf{Z})).$$

Remark

Our final definition of $\mu_z(f)$ turns out to be surprisingly simple. Why did we not define it that way in the first place, thereby avoiding the categorical considerations above? Purely technically this is feasible, for apart from the definition of $\mu_z(f)$ we won't need the results above any more. However, it would then have been rather difficult to see why we should not have chosen one of the following alternatives:

alternative 1 $\mu_z(f) = \mu(\lambda\rho.f(\rho \cup \mathbf{Z}))$.

alternative 2 $\mu_z(f) = \mu(\lambda\rho.(f(\rho \cup \mathbf{Z}) \cup \mathbf{Z}))$.

With respect to functions f that preserve totality there is no real difference with the definition we have chosen. But for instance, for our definition, $\mu_z(\lambda\rho.\emptyset) = \mathbf{Z}$ whereas according to alternative 1, $\mu_z(\lambda\rho.\emptyset) = \emptyset$. Alternative 2 can be seen to be equivalent to our definition. At this stage it is not clear which alternative is "best". But in terms of projections and embeddings our choice can be seen to be more natural than the other alternatives, rather than being just some smart guess.

One might think that it is not too important which alternative is chosen. After all, for "real" process the μ_z operator is always applied to functions that preserve totalness, and for such functions the three alternatives yield the same result. However, such subtle differences show up later on when considering proof rules for recursive processes. The rules for recursion that are introduced in chapter 5 for instance, would be sound but *incomplete* when we had chosen alternative 1.

3.3 Prefix closures

A typical property of the meaning of **TNP** processes is the following one. If some **TNP** process S admits some computation δ of the form (s_0, t, s), then it also admits any δ' of the form $(s_0, t', \perp)$ where t' is some initial prefix of the trace t. For $(s_0, t', \perp)$ corresponds to some intermediate unfinished state where the communications in t' have already been performed. Note that this includes the observation $(s_0, t, \perp)$ where *all* communications of t have occurred already, but where the process has not yet terminated. A set of observations with this property is called *prefix closed.* We give a formal definition of prefix closedness, and also introduce two important closure operations.

The relationship between δ and δ' is captured by the following partial order, defined on the domain Δ.

Definition 3.11 (The prefix order on Δ)

The p.o. $\sqsubseteq_\Delta$ on Δ is defined by:

$(s_0', t', s') \sqsubseteq_\Delta (s_0, t, s)$ iff

either $(s_0', t', s') = (s_0, t, s)$ or $s_0' = s_0, t' \preceq t$ and $s' = \perp$.

Remark Usually we drop the index Δ on "$\sqsubseteq$".

□

Definition 3.12 (prefix closures)

- For a p.o. $(U, \sqsubseteq)$, a set $\rho \subseteq U$ is called *downwards closed* iff:

  $(x \in \rho \wedge y \sqsubseteq x) \Rightarrow y \in \rho$.

- For the p.o.Δ, the term *prefix closed* will be used instead of downwards closed.
- A set ρ is called *closed* if it is both total and prefix closed.
- $\mathcal{P}_p(\Delta)$ and $\mathcal{P}_{pz}(\Delta)$ denote the collections of all prefix closed, and all closed subsets of Δ.
- The prefix closure $\mathit{Pref}(\rho)$ of a set ρ is defined as the smallest prefix closed set containing ρ.
- The kernel $\mathit{Kern}(\rho)$ of a set ρ is defined as the largest prefix closed set contained in ρ.
- The closure $\mathit{Close}(\rho)$ of a set ρ is defined as the smallest closed set containing ρ.

□

Lemma 3.13 (Properties of Pref and Kern)

(a) $\mathit{Kern}(\rho) \subseteq \rho \subseteq \mathit{Pref}(\rho)$.

(b) A set ρ is prefix closed if and only if

$\mathit{Pref}(\rho) = \rho$ if and only if

$\mathit{Kern}(\rho) = \rho$.

(c) $\mathit{Pref}(\rho) = \{\delta' \in \Delta \mid \exists \delta \in \rho : \delta' \sqsubseteq \delta\}$.

(d) $\mathit{Kern}(\rho) = \{\delta \in \Delta \mid \forall \delta' \sqsubseteq \delta : \delta' \in \rho\}$.

(e) The *Pref* operation is completely additive, and preserves totality.

(f) The *Kern* operation is continuous and preserves totality.

(g) $Close(\rho) = Pref(\rho \cup \mathbf{Z}) = Pref(\rho) \cup \mathbf{Z}$.

□

Proof

Most cases are clear. We spell out the proof of (e) and (f).

Let $\{\rho_i \mid i \in I\}$ be some arbitrary collection of Δ subsets.

$$Pref(\bigcup_{i \in I} \rho_i) = \{\delta' \mid \exists \delta \in \bigcup_{i \in I} \rho_i : \delta' \sqsubseteq \delta\}$$

$$= \bigcup_{i \in I} \{\delta' \mid \exists \delta \in \rho_i : \delta' \sqsubseteq \delta\}$$

$$= \bigcup_{i \in I} Pref(\rho_i).$$

Preservation of totality is seen as follows. Assume that $\mathbf{Z} \subseteq \rho$. Then:

$$Pref(\rho) = Pref(\rho \cup \mathbf{Z}) = Pref(\rho) \cup Pref(\mathbf{Z}) = Pref(\rho) \cup \mathbf{Z},$$

which implies that $\mathbf{Z} \subseteq Pref(\rho)$.

Next we consider the *Kern* operation. This operation is *not* completely additive, but it *is* continuous. Failure of complete additivity is seen by the following example.

$$Kern\Big(\{(s_0, \epsilon, \bot)\} \cup \{(s_0, < (c,0) >, \bot)\}\Big)$$

$$= Kern\Big(\{(s_0, \epsilon, \bot), (s_0, < (c,0) >, \bot)\}\Big)$$

$$= \{(s_0, \epsilon, \bot), (s_0, < (c,0) >, \bot)\}$$

$$\neq \{(s_0, \epsilon, \bot)\} \cup \emptyset = Kern(\{(s_0, \epsilon, \bot)\}) \cup Kern(\{(s_0, < (c,0) >, \bot)\}).$$

But assume that $\langle \rho_i \rangle_{i \in \mathbf{N}}$ is an *ascending chain* of Δ subsets. Then we have that

$$Kern(\bigcup_{i \in \mathbf{N}} \rho_i) = \{\delta \mid \forall \delta' \sqsubseteq \delta : \delta' \in \bigcup_{i \in \mathbf{N}} \rho_i\}$$

$$= \{\delta \mid \forall \delta' \sqsubseteq \delta \; \exists i \in \mathbf{N} : \delta' \in \rho_i\}. \qquad (*)$$

It is not immediately clear that the universal and existential quantifiers in $(*)$ can be interchanged. However, this follows from the fact that, for any computation δ, there are only *finitely many* Δ elements δ' such that $\delta' \sqsubseteq \delta$. For δ is of the form (s_0, h, s), where the trace h is a sequence of finite length, and if $\delta' \sqsubseteq \delta$ then δ' has the form $(s_0, h', \bot)$, where h' must be one of the finitely many prefixes of h.

So, if δ is such that for all $\delta' \sqsubseteq \delta$ there is some number i with $\delta' \in \rho_i$, then we take the maximum m of these numbers and, by the assumption that $\langle \rho_i \rangle_{i \in \mathbf{N}}$ is an ascending chain, all δ' are contained in ρ_m. This shows that:

$$
\begin{aligned}
(*) &= \{\delta \mid \exists i \in \mathbf{N}\, \forall \delta' \sqsubseteq \delta : \delta' \in \rho_i\} \\
&= \bigcup_{i \in \mathbf{N}} \{\delta \mid \forall \delta' \sqsubseteq \delta : \delta' \in \rho_i\} \\
&= \bigcup_{i \in \mathbf{N}} \mathit{Kern}(\rho_i).
\end{aligned}
$$

It is clear that the *Kern* operation is monotone. And since $\mathit{Kern}(\mathbf{Z}) = \mathbf{Z}$ this also shows that totality is preserved by *Kern*.

□

Lemma 3.14

$\mathcal{P}_{pz}(\Delta)$ is a subcpo of $\mathcal{P}_{\mathbf{z}}(\Delta)$.

□

Proof

Clearly the least element $\mathbf{Z}$ of $\mathcal{P}_{\mathbf{z}}(\Delta)$ is prefix closed. It is obvious that the union of a collection prefix closed sets is again prefix closed. That proves that the lub of a collection of prefix closed sets is prefix closed itself.

□

If some prefix closed set ρ only contains computations with traces of bounded length, then it is often more transparent to define the set as $\mathit{Pref}(\rho_m)$ where ρ_m contains the maximal elements of ρ. Moreover, if in fact all *unfinished* computations of ρ_m are of the form $(s_0, \epsilon, \bot)$, then ρ can be defined as $\mathit{Close}(\rho_F)$, where ρ_F contains all finished computations of ρ. We have done this in section 3.8 in the definition of the meanings of atomic processes.

The purpose of the kernel operation is the following. A process specification is essentially an inequality of the following form:

$$\mathit{Obs}(S) \subseteq \rho,$$

where ρ is determined by some assertion, in a way to be described in the next chapter. Unlike the process denotation $Obs(S)$, ρ need not be prefix closed. However, ρ can always be replaced by the prefix closed set $Kern(\rho)$. For $Kern(Obs(S)) = Obs(S)$, and since $Kern$ is a monotone operation it is seen that the inequality above is equivalent to the following one:

$Obs(S) \subseteq Kern(\rho)$.

This fact forms the basis for the kernel rule that is discussed in chapter 5.

3.4 Semantic operations

To facilitate the definition of the semantics of **TNP**, we define a series of operations on our domains.

Definition 3.15 (Operations on states)

For $s \in State, \bar{x} \in Var^*$ of the form $x_0, \cdots, x_{n-1}$, and $\bar{v} \in Val^*$ of the form $v_0, \cdots, v_{n-1}$, the variant $s|\bar{x} : \bar{v}$ is defined as usual:

$$(s|\bar{x} : \bar{v})(y) = v_i \text{ if, for some } i, y \equiv x_i \text{ and } y \not\equiv x_j \text{ for } i < j < n,$$
$$= s(y) \text{ if } y \not\equiv x_i \text{ for } 0 \leq i < n.$$

For the bottom state we define: $\bot|\bar{x} : \bar{v} = \bot$

□

Definition 3.16 (Operations on traces)

- *Concatenation* of traces t_0 and t_1 is denoted by $t_0{}^\frown t_1$ or $t_0 t_1$. By "*translation* over t_0" we will mean the operation $\lambda t \,.\, t_0 t$. It has a partial inverse, called *prefix chopping* which is defined as follows:

  $t/t_0 = t_1$ if $t = t_0 t_1$, provided such a t_1 exists; otherwise, it is not defined.

  Note the following properties:

  (i) $(t_0 t)/t_0 = t$
  (ii) $t_0(t/t_0) = t$, provided t/t_0 is defined.

- The *projection* $t|\mathbf{c}$, of a trace t onto a set of channels $\mathbf{c}$, is defined as the trace obtained from t by omitting all communications (c, v) with $c \notin \mathbf{c}$.

- A special form of projection is *hiding* $t\backslash\mathbf{c}$, which denotes $t|(Chan - \mathbf{c})$.

- *Renaming* a channel c into d means replacing all occurrences of communications of the form (c, v) by (d, v). It is denoted by $t[d/c]$.
  Note that $t[d/c][c/d] = t[c/d]$, and that this trace expression equals t only if no communications via d occur in t.

□

Definition 3.17 (Operations on Δ)

- Any operation $op(t)$ on traces induces a corresponding operation on Δ, defined by $op\big((s_0, t, s)\big) = (s_0, op(t), s)$.
  E.g $t_0\hat{}(s_0, t, s) = (s_0, t_0\hat{}t, s)$, $(s_0, t, s)|\mathbf{c} = (s_0, t|\mathbf{c}, s)$ etc.
- *Projection onto variables*

  Let $\mathbf{x} \subseteq \mathcal{V}ar$ be some set of variables such that $\mathbf{x} = \{\bar{x}\} = \{x_0, x_1, \cdots\}$. We define the *projection onto the variables* $\mathbf{x}$ as an operation on Δ :

  If $s \neq \perp$, then $(s_0, t, s)|\mathbf{x} = (s_0, t, s_0|\bar{x} : s(\bar{x}))$.

  In all other cases we define $(s_0, t, \perp)|\mathbf{x} = (s_0, t, \perp)$.
- *Hiding of variables*

  $\cdot\backslash\mathbf{x}$ is defined analogous to hiding of channels:

  $$(s_0, t, s)\backslash\mathbf{x} \stackrel{\text{def}}{=} (s_0, t, s)|(\mathcal{V}ar - \mathbf{x})$$

  Provided that $s \neq \perp$, we can rewrite this as:

  $$(s_0, t, s)\backslash\mathbf{x} = (s_0, t, s|\bar{x} : s_0(\bar{x}))$$

  For, in this case also $s_0 \neq \perp$ and so, if $\{\bar{y}\} = \mathcal{V}ar - \mathbf{x}$, we have that $s_0|\bar{y} : s(\bar{y}) = s|\bar{x} : s_0(\bar{x})$. For a bottom final state we have of course that:

  $$(s_0, t, \perp)\backslash\mathbf{x} = (s_0, t, \perp).$$
- *Projection and hiding for bases*

  For a base β of the form $(\mathbf{c}, \mathbf{x})$ we define, for $\delta \in \Delta$:

  $\delta|\beta = \delta|\mathbf{c}|\mathbf{x}$, and

  $\delta\backslash\beta = \delta\backslash\mathbf{c}\backslash\mathbf{x}$.

□

Remark Note that there is no projection or hiding operation defined on states as such.

Next we define operations on $\mathcal{P}(\Delta)$. We show that these operations are completely additive, and preserve totality and prefix closedness.

Definition 3.18 (Pointwise extensions)

Any transformation $op : \Delta \rightarrow \Delta$ induces a corresponding operation $op^{\mathcal{P}} : \mathcal{P}(\Delta) \rightarrow \mathcal{P}(\Delta)$, defined by:

$$op^{\mathcal{P}}(\rho) = \{op(\delta) \mid \delta \in \rho\}$$

We call it the pointwise extension of "op". When no ambiguity arises, we usually omit the $\mathcal{P}$-superscript.

□

Lemma 3.19

The pointwise extension $op^{\mathcal{P}}$, as defined above, is completely additive.

□

Proof

Take some arbitrary collection of $\mathcal{P}(\Delta)$ elements ρ_i, where $i \in I$.
Then one sees that:

$$op^{\mathcal{P}}(\bigcup_{i \in I} \rho_i) = \{op(\delta) \mid \delta \in \bigcup_{i \in I} \rho_i\} =$$

$$\bigcup_{i \in I} \{op(\delta) \mid \delta \in \rho_i\} = \bigcup_{i \in I} op^{\mathcal{P}}(\rho_i)$$

□

Lemma 3.20

Let $\mathbf{c} \subseteq \mathcal{C}han$, $\mathbf{x} \subseteq \mathcal{V}ar$, $c, d \in \mathcal{C}han$.
The following operations preserve totality and prefix closedness:

1. $\lambda\rho.\rho|\mathbf{c}$
2. $\lambda\rho.\rho\backslash\mathbf{c}$
3. $\lambda\rho.\rho|\mathbf{x}$
4. $\lambda\rho.\rho\backslash\mathbf{x}$
5. $\lambda\rho.\rho[d/c]$

□

Proof

Since the hiding operations are defined by means of projections it suffices to treat the cases 1,3 and 5. And since the operations are completely additive a sufficient condition for preservation of totality is that $\mathbf{Z} \subseteq op(\mathbf{Z})$.

Case 1

Preservation of totality:

$$\mathbf{Z}|\mathbf{c} = \{(s_0, \varepsilon, \bot) \mid s_0 \in State_\bot\}\,|\,\mathbf{c} = \{(s_0, \varepsilon|\mathbf{c}, \bot) \mid s_0 \in State_\bot\} = \mathbf{Z}.$$

Preservation of prefix closedness:

Assume that ρ is prefix closed, and that we have some arbitrary δ, δ' such that $\delta \in \rho|\mathbf{c}$ and $\delta' \sqsubseteq \delta$. We must prove that $\delta' \in \rho|\mathbf{c}$. Since this is trivial when $\delta' = \delta$ we only consider the case that $\delta' \sqsubset \delta$. If δ is of the form (s_0, h, s) then δ' is of the form $(s_0, h', \bot)$, for some prefix h' of h. Since $\delta \in \rho|\mathbf{c}$, there is some trace t such that $(s_0, t, s) \in \rho$ and $h = t|\mathbf{c}$. For some, not necessarily uniquely determined, prefix t' of t we have that $h' = t'|\mathbf{c}$. By prefix closedness of ρ it follows that $(s_0, t', \bot) \in \rho$, and so we see that $(s_0, h', \bot) \in \rho|\mathbf{c}$, as was to be shown.

Case 3

Preservation of totality:

$$\mathbf{Z}|\mathbf{x} = \{(s_0, \varepsilon, \bot)|\mathbf{x} \mid s_0 \in State_\bot\} = \{(s_0, \varepsilon, \bot) \mid s_0 \in State_\bot\} = \mathbf{Z}.$$

Preservation of prefix closedness: Obvious because the operation affects only final states and preserves computations with bottom final states.

Case 5

Preservation of totality: similar as for case 1, since $\varepsilon[d/c] = \varepsilon$.

Preservation of prefix closedness: similar as for case 1, but even simpler, because if $h = t[d/c]$ and h' is some prefix of h, then there is some particular prefix t' of t such that $h' = t'[d/c]$.

□

Definition 3.21 (Sequential composition)

Elements of $\mathcal{P}(\Delta)$ can be regarded as a combination of a state transformer relation and a trace set. With this in mind we define the *composition* $\rho_1 \hat{\circ} \rho_2$ of two $\mathcal{P}(\Delta)$ elements as a relation composition w.r.t. the state parts and a trace concatenation w.r.t. the trace sets, thus:

$$\rho_1 \hat{\circ} \rho_2 = \{(s_0, t_1 \hat{} t_2, s_2) \mid \exists s_1 \in State_\bot . (s_0, t_1, s_1) \in \rho_1, (s_1, t_2, s_2) \in \rho_2\}.$$

This operation will be used to define the meaning of sequential composition.

Lemma 3.22

The sequential composition operator is completely additive and preserves totality and closedness.

□

Proof

Complete additivity is shown by the following calculation:

$$(\bigcup_{i\in I} \rho_i) \hat{\circ} (\bigcup_{j\in J} \rho'_j) =$$

$$\{(s_0, t_1 \hat{} t_2, s_2) \mid \exists s_1 : (s_0, t_1, s_1) \in (\bigcup_{i\in I} \rho_i), \text{ and } (s_1, t_2, s_2) \in (\bigcup_{j\in J} \rho'_j)\} =$$

$$\{(s_0, t_1 \hat{} t_2, s_2) \mid \exists s_1, i \in I, j \in J : (s_0, t_1, s_1) \in \rho_i, \text{ and } (s_1, t_2, s_2) \in \rho'_j\} =$$

$$\bigcup_{(i,j)\in I\times J} \{(s_0, t_1 \hat{} t_2, s_2) \mid \exists s_1 : (s_0, t_1, s_1) \in \rho_i, \text{ and } (s_1, t_2, s_2) \in \rho'_j\} =$$

$$\bigcup_{(i,j)\in I\times J} (\rho_i \hat{\circ} \rho'_j).$$

Preservation of totality follows from the complete additivity and the easily verified fact that $\mathbf{Z} \hat{\circ} \mathbf{Z} = \mathbf{Z}$.

Note that it is quite essential here that $\mathbf{Z}$ contains the pseudo computation $(\perp, \varepsilon, \perp)$. For arbitrary sets ρ_1, ρ_2, if, and only if, this pseudo computation is present in ρ_2 we have that every unfinished computation $(s_0, h, \perp)$ of ρ_1 does also occur within $\rho_1 \hat{\circ} \rho_2$.

If ρ_1 and ρ_2 are *closed*, we can prove closedness of $\rho_1 \hat{\circ} \rho_2$. That is, if ρ_1 and ρ_2 are prefix closed and total, then the same holds for the sequential composition of the two sets. Since we already showed the preservation of totality, the preservation of prefix closedness remains. However, we cannot prove prefix closedness of $\rho_1 \hat{\circ} \rho_2$ without the assumption of totality of ρ_2, which guarantees that ρ_2 contains the pseudo computation $(\perp, \varepsilon, \perp)$. This explains why the lemma claims the preservation of closedness, rather than of prefix closedness.

So assume that ρ_1 and ρ_2 are total and prefix closed, and that for certain δ, δ' we have that $\delta' \sqsubset \delta \in \rho_1 \hat{\circ} \rho_2$. By the definition of the operator, we know that δ is of the form $(s_0, h_1 h_2, s)$, where for some intermediate state $s_1 \in \mathit{State}_\perp$, $(s_0, h_1, s_1) \in \rho_1$ and $(s_1, h_2, s) \in \rho_2$. Then δ' must be of the form $(s_0, h', \perp)$, where either h' is a prefix of h_1 or else $h' = h_1 h''$ for some prefix h'' of h_2.

In the first case we infer from the prefix closedness of ρ_1 that $\delta' \in \rho_1$, and, since $(\bot, \epsilon, \bot) \in \rho_2$, it is also true that $\delta' \in \rho_1 \hat{\circ} \rho_2$. In the second case we infer from the prefix closedness of ρ_2 that $(s_1, h'', \bot) \in \rho_2$, and thus it follows that $\delta' \in \rho_1 \hat{\circ} \rho_2$. (As was to be shown).

□

In $\mathcal{P}(\Delta)$ we have the following constant, acting as a unit element for composition:

$$\mathbf{1} \stackrel{\text{def}}{=} \{(s_0, \epsilon, s_0) \mid s_0 \in State_\bot\}.$$

That is, $\mathbf{1} \hat{\circ} \rho = \rho \hat{\circ} \mathbf{1} = \rho$ for all $\rho \in \mathcal{P}(\Delta)$.

Note that $\mathbf{1} \notin \mathcal{P}_\mathbf{z}(\Delta)$! Fortunately the set:

$$\mathbf{1}_\mathbf{Z} \stackrel{\text{def}}{=} \mathbf{1} \cup \mathbf{Z} = \{(s_0, \epsilon, \bot), (s_0, \epsilon, s_0) \mid s_0 \in State_\bot\}$$

is both total and prefix closed, and acts as a unit within $\mathcal{P}_\mathbf{z}(\Delta)$. That is,

$$\mathbf{1}_\mathbf{Z} \hat{\circ} \rho = \rho \hat{\circ} \mathbf{1}_\mathbf{Z} = \rho \text{ for all } \rho \in \mathcal{P}_\mathbf{z}(\Delta).$$

The set $\mathbf{1}_\mathbf{Z}$ consists exactly of the observations admitted by the **skip** process, for this process never communicates, and *when* it terminates, it has not changed the state. As is the case for *any* process, it also admits the observations in $\mathbf{Z}$, corresponding to the intermediate stage where the process has started but not (yet) terminated.

Definition 3.23 (Union and intersection)

Our semantic operation for nondeterministic choice will be *set union*, for the process S_1 **or** S_2 admits exactly all observations of S_1 as well as of S_2. The intersection $\rho_1 \cap \rho_2$ is used below to analyze the parallel composition operator.

Lemma 3.24

The union and intersection operations $\lambda\rho_1.\lambda\rho_2.(\rho_1 \cup \rho_2)$ and $\lambda\rho_1.\lambda\rho_2.(\rho_1 \cap \rho_2)$ are completely additive and preserves totality and prefix closedness.

□

Proof

As the proof for the union operation is almost trivial, we only treat intersection here.

Complete additivity follows immediately from the distributivity of set unions over intersections:

$$(\bigcup_{i\in I} \rho_i) \cap (\bigcup_{j\in J} \rho'_j) = \bigcup_{(i,j)\in I\times J} \rho_i \cap \rho'_j.$$

Preservation of totality is clear: if both ρ_1 and ρ_2 contain $\mathbf{Z}$ as a subset, than the same holds for $\rho_1 \cap \rho_2$. Finally, preservation of prefix closedness is also clear, for if some δ occurs in the intersection of two prefix closed sets, then any δ' such that $\delta' \sqsubseteq \delta$, occurs in both sets too.

□

Similar to the unit element for composition, we have a zero element for set union. In the case of $\mathcal{P}(\Delta)$ this is of course the empty set. For $\mathcal{P}_{\mathbf{Z}}(\Delta)$ we can take the element $\mathbf{Z}$, since we have that, for any set $\rho \in \mathcal{P}_{\mathbf{Z}}(\Delta)$:

$$\mathbf{Z} \cup \rho = \rho \cup \mathbf{Z} = \rho \qquad (*)$$

Obviously $\mathbf{Z}$ is total and prefix closed. As already explained before, this set is the denotation for **abort**, since it is the least element of $\mathcal{P}_{\mathbf{Z}}(\Delta)$. It consists of the observations admitted by a process that never terminates and never performs any communication. Clearly this is the appropriate denotation for a *diverging* process, like **while** true **do skip od**. However, $\mathbf{Z}$ will also be the denotation of a *deadlocked* process like, e.g.

$$c!0 \,;\, d!0 \parallel d?x \,;\, c?x.$$

The reason for this is that we did not include "deadlock" in our list of observable events, and so deadlock and divergence cannot be distinguished by means of finite observations.

From $(*)$ it can already be seen that if a process has the choice between, on the one hand, divergence (or deadlock) and, on the other hand, termination or communication, then the divergence (or deadlock) possibility is not represented any more in our semantics. So e.g. $c!0$ **or abort** turns out to be equivalent to $c!0$. Such identifications are justified since within our class of safety properties there is no such property that could distinguish between the two processes. (In the sense that it would be valid for one process but not for the other).

We have discussed semantic operations corresponding to all **TNP** operators except for parallel composition. The semantics of parallelism is easier to describe after the next section.

3.5 Process Bases

In chapter 2 we defined the (syntactic) base of a process. The definition is such that if $base(S) = (\mathbf{c}, \mathbf{x})$ then S has the following properties:

1. it communicates exclusively via channels in $\mathbf{c}$,
2. it can read only the variables in $\mathbf{x}$,
3. it can assign only to the variables in $\mathbf{x}$.

One of the purposes of this section is to define a corresponding *semantic* notion of bases, and to relate it to the syntactic base of processes.

In the next chapter we define the class of assertions that, in the terminology of section 3.1, play the role of negations of refutation criteria. That is, for a given assertion χ and a computation δ it is possible to determine whether the computation satisfies χ or not. To each assertion we assign a so called *assertion base*, that is the counterpart of the base of a process. The assertion base of assertion χ is denoted by $abase(\chi)$. It has the following properties. If $abase(\chi) = (\mathbf{c}, \mathbf{x})$ then:

1. the truthvalue of χ does not depend on communications via channels outside the set $\mathbf{c}$,
2. the truthvalue of χ does not depend on the values of variables outside the set $\mathbf{x}$.

Similar to the semantic base that we mentioned above, we define the notion of the *semantic* assertion base of some given set of computations. In chapter 4 we consider the exact relationship between the syntactic assertion base of an assertion and the semantic assertion base of the set of all computations that satisfy the assertion.

A third goal of this section is to introduce an operation called *chaotic closure*, that is in some sense the dual of the projection operation. We need this operation to characterize assertion bases, but it plays also a major role in the definition of parallelism. We start with the definition of this operation.

3.5.1 Chaotic closures

From the properties of the projection operations it follows immediately that $\lambda\rho.\rho|\beta$, where β is a base, is a completely additive operation that preserves totality and prefix closedness. We define an operation $\lambda\rho.\rho \uparrow \beta$, called "chaotic closure".

Definition 3.25 (Chaotic closure)

$\rho\uparrow\beta$ is the largest set ρ' such that $\rho'|\beta = \rho|\beta$.

□

Whereas projection onto β *removes* all communications and modifications of channels and variables outside β, the chaotic closure *inserts arbitrary communications and "random assignments" for channels and variables outside* β.

It is clear that $\rho\uparrow\beta$ equals the following set $\{\delta \in \Delta \mid \delta|\beta \in \rho|\beta\}$.

Using this fact, the following lemma is proven easily.

Lemma 3.26

$\lambda\rho.\rho\uparrow\beta$ is completely additive and preserves totality and prefix closedness.

□

Proof

$$(\bigcup_{i\in I}\rho_i)\uparrow\beta =$$

$$\{\delta \in \Delta \mid \delta|\beta \in (\bigcup_{i\in I}\rho_i)|\beta\} = \text{(by c.a. of projection)}$$

$$\{\delta \in \Delta \mid \exists i \text{ such that } \delta|\beta \in \rho_i|\beta\} = \bigcup_{i\in I}\{\delta \in \Delta \mid \delta|\beta \in \rho_i|\beta\}$$

$$= \bigcup_{i\in I}(\rho_i\uparrow\beta).$$

We conclude that the operation is completely additive. Preservation of totality is rather obvious. To see that prefix closedness is preserved, assume that ρ is prefix closed and $\delta' \sqsubseteq \delta \in \rho\uparrow\beta$. Then $\delta'|\beta \sqsubseteq \delta|\beta$ and $\delta|\beta \in \rho|\beta$. Since projection preserves prefix closedness we may conclude that $\delta'|\beta \in \rho|\beta$, which proves that $\delta' \in \rho\uparrow\beta$.

□

We list a few properties of projection and chaotic closure, all of which can be checked fairly straightforward.

(i) $\rho \subseteq \rho\uparrow\beta$

(ii) $\rho\uparrow\beta$ is the largest set ρ' such that $\rho'|\beta \subseteq \rho|\beta$

(iii) $\rho|\beta_1|\beta_2 = \rho|(\beta_1 \cap \beta_2)$

(iv) $\rho|\beta|\beta = \rho|\beta$

(v) $\rho\uparrow\beta_1\uparrow\beta_2 = \rho\uparrow(\beta_1 \cap \beta_2)$

(vi) $\rho\uparrow\beta\uparrow\beta = \rho\uparrow\beta$

(vii) $\rho|\beta\uparrow\beta = \rho\uparrow\beta$

(viii) $\rho\uparrow\beta|\beta = \rho|\beta$

(ix) $\rho|\beta \subseteq \rho\uparrow\beta$

(x) $\delta \in \rho\uparrow\beta$ iff $\delta|\beta \in \rho|\beta$

□

We now come to the definition of the semantic base and the semantic assertion base of a set of computations. After the definition and the proof of a few simple properties we relate these semantic notions to their corresponding syntactic counterparts.

Definition 3.27 (Semantic base and semantic assertion base)

- For a set $\rho \in \mathcal{P}(\Delta)$ we define:

  $base(\rho)$ is the smallest base β such that $\rho|\beta = \rho$, and

  $abase(\rho)$ is the smallest base β such that $\rho\uparrow\beta = \rho$.

- The set Δ_β is defined as the set of all computations with their base contained in β, that is:

  $$\Delta_\beta \stackrel{\text{def}}{=} \{\delta \in \Delta |\ \delta|\beta = \delta\}.$$

- The domains $\mathcal{P}(\Delta_\beta)$, $\mathcal{P}_{\mathbf{z}}(\Delta_\beta)$ and $\mathcal{P}_{p\mathbf{z}}(\Delta_\beta)$ are derived from the corresponding domains $\mathcal{P}(\Delta)$, $\mathcal{P}_{\mathbf{z}}(\Delta)$ and $\mathcal{P}_{p\mathbf{z}}(\Delta)$ by restricting their elements to subsets of Δ_β.

□

To prove that the base and assertion base of a set ρ are well defined, we remark the following.

- There exists at least one β such that $\rho|\beta = \rho$ and $\rho\uparrow\beta = \rho$, viz. the base $(\mathcal{C}han, \mathcal{V}ar)$.
- If for certain bases β and β' it happens that $\rho|\beta = \rho$ and also $\rho|\beta' = \rho$, then $\rho|(\beta \cap \beta') = \rho|\beta|\beta' = \rho$. Similarly, if both $\rho\uparrow\beta = \rho$ and $\rho\uparrow\beta' = \rho$ then also $\rho\uparrow(\beta \cap \beta') = \rho\uparrow\beta\uparrow\beta' = \rho$.

From this it follows that $base(\rho)$ always exists and is uniquely determined as the intersection of all bases β such that $\rho|\beta = \rho$. Similarly, $abase(\rho)$ is the intersection of all β for which $\rho \uparrow \beta = \rho$.

The base of some set of computations can be determined easily from the channelset and assignsets of these computations, that we define as follows:

Definition 3.28 (channelset and assignset of computations)

- $chan\Big((s_0, t, s)\Big) \stackrel{\text{def}}{=} chan(t) \stackrel{\text{def}}{=}$
  $\{c \in Chan \mid t \text{ contains some communication via } c\}$.
- $assign\Big((s_0, t, s)\Big) \stackrel{\text{def}}{=} \{x \in Var \mid s, s_0 \neq \perp \text{ and } s_0(x) \neq s(x)\}$.
- For $\rho \in \mathcal{P}(\Delta)$ we define:

  $$chan(\rho) = \bigcup \{chan(\delta) \mid \delta \in \rho\},$$

  $$assign(\rho) = \bigcup \{assign(\delta) \mid \delta \in \rho\}.$$

□

Lemma 3.29

$$base(\rho) = (chan(\rho), assign(\rho)).$$

□

Proof

Let $base(\rho) = \beta = (\mathbf{c}, \mathbf{x})$, and let

$(chan(\rho), assign(\rho)) = \beta_\rho = (\mathbf{c}_\rho, \mathbf{x}_\rho)$.

Let $(s_0, h, s) \in \rho$. The trace h of this computation contains only communications via channels in $\mathbf{c}_\rho$,by the definition of $\mathbf{c}_\rho$. Therefore $h|\mathbf{c}_\rho = h$. If $s \neq \perp$ then $s_0|\bar{x}_\rho : s(\bar{x}_\rho) = s$, since, by definition, $\mathbf{x}_\rho$ contains all variables for which s_0 and s could possibly differ. We see that $(s_0, h, s)|\beta_\rho = (s_0, h, s)$, and thus, that β_ρ has the property that $\rho|\beta_\rho = \rho$.

It remains to show that β_ρ is the *smallest* base with this property. Assume to the contrary that for some base β', $\rho|\beta' = \rho$ and that there is some channel d or variable y such that $d \notin \mathbf{c}_\rho$ or $y \notin \mathbf{x}_\rho$. By definition of $\mathbf{c}_\rho$ there is at least one computation in ρ that contains d communications, or that ends in a non bottom final state for which y has been modified. This contradicts the assumption that $\rho|\beta' = \rho$, and so we conclude that β_ρ is contained in every β' that has the property that $\rho|\beta' = \rho$.

□

Lemma 3.30

$(\mathcal{P}(\Delta_\beta), \subseteq, \emptyset)$ and $(\mathcal{P}_\mathbf{z}(\Delta_\beta), \subseteq, \mathbf{Z})$

are *complete lattices.*

□

Proof

Let $\{\rho_i \mid i \in I, \rho_i \subseteq \Delta_\beta\}$ be some *nonempty* collection of $\mathcal{P}(\Delta)$ elements. We consider the base of the lub (in $\mathcal{P}(\Delta)$) of this collection:

$$base(\bigsqcup_{i \in I} \rho_i) = base(\bigcup_{i \in I} \rho_i) = \bigcup_{i \in I} base(\rho_i) \subseteq \beta.$$

The lub of an *empty* collection is the least element of a domain. Since it is clear that $base(\emptyset) = base(\mathbf{Z}) = \emptyset$, we see that the lub of *any* collection of sets with base contained in β is again a set with base contained in β. This shows that $\mathcal{P}(\Delta_\beta)$ and $\mathcal{P}_\mathbf{z}(\Delta_\beta)$ are complete lattices.

□

By the characterization of semantic bases above it will be clear that there is a close relationship between the semantic and syntactic base of some process, but that in general the two will not be identical. The reason for this is twofold:

- First of all, the fact that in the program text of S some communication action or assignment occurs doesn't imply that the action or assignment is actually executed for some computation. For instance, the process **if false then** $c!0$ **else** $d!0$ **fi** never communicates via channel c although this channel occurs in the syntactic base of the process. Even when some assignment *is* executed its effect can be undone by assignments executed thereafter. Our notion of the semantic base of a process captures only those channels for which actual communication occurs in at least one computation, and those variables for which there is a difference between the initial and final state value for at least one computation.
- The difference between initial and final state values or the occurrence of a communication can be established for each computation in isolation. But it is *not* possible to establish *for some computation in isolation* whether it has actually read some variable or not. This shows that statements about the read set of some process cannot be *falsifiable.* (This follows from the characterization of falsifiable properties on page 3 of this chapter.) Since the correctness formulae that we study are concerned with falsifiable properties only, we can simplify matters

by defining the semantic base of a process such that it captures assignments to variables and communications, but not read actions for variables. Due to this simplification, we could lay a simple and useful connection between semantic bases and the projection operator.

The second reason raises the question why we did not define the syntactic base along the same lines, i.e., why the syntactic base *does* include the variables that are read at some point. The reason for this is rather indirect. In chapter 6 we associate with each process S an assertion $A(S)$ that is called the characteristic assertion for S. It has the important property that its assertion base equals the (syntactic) base of the process it describes. However, the assertion base of the characteristic assertion $A(S)$ consists of those channels via which the process can communicate and those variables that it can write to *or read from,* and therefore this equality holds only because we included the variables that a process can read in its syntactic base.

We formalize our claims concerning the relation between syntactic and semantic bases in the form of a lemma. This lemma should be compared with the clauses for the corresponding syntactic operations in definition 2.4.

Lemma 3.31

(i) $base(\mathbf{Z}) = \emptyset$

(ii) $base(\mathbf{1}) = \emptyset$

(iii) $base(\rho|\beta) = base(\rho) \cap \beta$

(iv) $base(\rho\backslash\beta) = base(\rho) - \beta$

(v) $base(\rho[d/c]) \subseteq (base(\rho) - (\{c\}, \emptyset)) \cup (\{d\}, \emptyset)$

(vi) $base(\rho_1 \hat{\circ} \rho_2) \subseteq base(\rho_1) \cup base(\rho_2)$

(vii) $base(\rho_1 \cup \rho_2) = base(\rho_1) \cup base(\rho_2)$

□

For most cases the proof is almost trivial. We consider a few interesting cases.

case (iii)

This is a straightforward consequence from the fact that:

$chan((s_0, h|\mathbf{c}, s_0|\bar{x} : s(\bar{x}))) = chan(h|\mathbf{c})$

$= chan(h) \cap \mathbf{c} = chan((s_0, h, s)) \cap \mathbf{c}$, and

$$assign((s_0, h|\mathbf{c}, s_0|\bar{x} : s(\bar{x}))) = assign((s_0, h, s)) \cap \{\bar{x}\}.$$

case (iv)

$$base(\rho\backslash\beta) = base(\rho|((Chan, \mathcal{V}ar) - \beta))$$
$$= base(\rho) \cap ((Chan, \mathcal{V}ar) - \beta) = base(\rho) - \beta.$$

case (v)

This is obvious, except maybe that we claim *containment* rather than equality between left and right hand side. The reason for this is that if ρ does not contain computations that include c communications, then $\rho[d/c]$ need not contain computations with d communications. So it is not always the case that $d \in base(\rho[d/c])$.

case (vi)

If $\delta \in \rho_1 \hat{\circ} \rho_2$ then it is of the form $(s_0, h_1 h_2, s)$, where, for some state s_1, $(s_0, h_1, s_1) \in \rho_1$ and $(s_1, h_2, s) \in \rho_2$. Obviously we have that

$$chan(\delta) = chan(h_1 h_2) = chan(h_1) \cup chan(h_2) \subseteq chan(\rho_1) \cup chan(\rho_2).$$

Also, if $s \neq \perp$ and $s_0(x) \neq s(x)$ for some variable x, then $s_1 \neq \perp$ and $s_0(x) \neq s_1(x)$ or $s_1(x) \neq s(x)$. That is, $assign(\delta) \subseteq assign(\rho_1) \cup assign(\rho_2)$.

□

3.6 Parallel composition

For the semantics of a *parallel composition*, $S_1 \ \beta_1 \| \beta_2 \ S_2$, of processes, we introduce a corresponding semantic operation $\lambda\rho_1.\lambda\rho_2.(\rho_1 \ \beta_1 \| \beta_2 \ \rho_2)$.

Let $i = 1$ or 2, let β_i be $(\mathbf{c}_i, \mathbf{x}_i)$. The context sensitive restriction CSR 1 of the language definition in chapter 2 requires that $base(S_i) \subseteq (\mathbf{c}_i, \mathbf{x}_i)$. This means that we must define a semantic operation that matches the operational intuition for arguments ρ_i that contain only computations with communications via channels within $\mathbf{c}_i$ and modifications of variables within $\mathbf{x}_i$. Note that for such ρ_i the projection $\rho_i|\beta_i$ equals ρ_i, for removing communications or variable modifications that are not present has no effect.

Our handshaking communication protocol states that if $c \in \mathbf{c}_i$, then any communication (of $S_1 \ \beta_1 \| \beta_2 \ S_2$) via c must be synchronized with a c communication by S_i. In particular, if $c \in \mathbf{c}_1 \cap \mathbf{c}_2$, both S_1 and S_2 must perform the c communication simultaneously. This implies that if t is a possible trace for $S_1 \ \beta_1 \| \beta_2 \ S_2$, than $t|\mathbf{c}_i$ must be admitted by S_i.

As to the state transformation we have the following situation: if (s_0, t, s) is admitted by $S_1\ \beta_1 \| \beta_2\ S_2$, and $s \neq \perp$, then s must equal s_0 except possibly for the variables in $\mathbf{x}_1 \cup \mathbf{x}_2$. Moreover, all reads and assignments to a variable $x \in \mathbf{x}_i$ have been performed by S_i, which means that $s_0|\bar{x}_i : s(\bar{x}_i)$, where $\{\bar{x}_i\} = \mathbf{x}_i$, is a possible final state for the execution of S_i.

Altogether one sees that if (s_0, t, s) is admitted by the parallel composition $S_1\ \beta_1 \| \beta_2\ S_2$ then the component S_i must admit $(s_0, t, s)|\beta_i$. So for sets ρ_i that satisfy the equality $\rho_i|\beta_i = \rho_i$ our parallel composition operator must satisfy:

$$(\rho_1\ \beta_1 \| \beta_2\ \rho_2)|\beta_i \subseteq \rho_i \text{ for } i = 1, 2.$$

An elegant generalization into a requirement for *arbitrary* sets ρ_i is the following:

$$(\rho_1\ \beta_1 \| \beta_2\ \rho_2)|\beta_i \subseteq \rho_i|\beta_i \text{ for } i = 1, 2.$$

These inequalities determine the parallel composition as far as the channels and variables in $\beta_1 \cup \beta_2$ are concerned. Note that $\beta_1 \cup \beta_2$ is the base of the parallel construct, and that outside this base neither communications are performed nor assignments are made by the parallel network. This is equivalent to the requirement that the following equality holds:

$$(\rho_1\ \beta_1 \| \beta_2\ \rho_2)|(\beta_1 \cup \beta_2) = (\rho_1\ \beta_1 \| \beta_2\ \rho_2).$$

This equality states that removing communications and modifications of channels and variables outside the base $(\beta_1 \cup \beta_2)$ has no effect on the set $\rho_1\ \beta_1 \| \beta_2\ \rho_2$, which amounts to the same as stating that there *are* no such communications or modifications in the set. More formally, the equality states that the semantic base of the set is contained within $(\beta_1 \cup \beta_2)$, and by lemma 3.29 this implies that all communications and modifications are within $(\beta_1 \cup \beta_2)$.

We define the parallel composition as follows:

Definition 3.32 (Parallel composition)

$\rho_1\ \beta_1 \| \beta_2\ \rho_2$ is the largest set ρ such that:

(i) $\rho|\beta_1 \subseteq \rho_i|\beta_i$, for $i = 1, 2$, and

(ii) $\rho|(\beta_1 \cup \beta_2) = \rho$.

□

Lemma 3.33

The parallel composition operator is completely additive, and preserves totality and prefix closedness.

□

Instead of proving this directly from the definition we show that the parallel composition operator can be expressed in terms of operations introduced above. The lemma then follows from the corresponding properties for these operations. First we prove a simple lemma.

Lemma 3.34 (Distributivity of "↑" and "|")

If, for $i = 1, 2$, $base(\rho_i) \subseteq \beta$, or, for $i = 1, 2$, $abase(\rho_i) \subseteq \beta$, then:

$$(\rho_1 \cap \rho_2)|\beta = (\rho_1|\beta) \cap (\rho_2|\beta),$$

$$(\rho_1 \cap \rho_2)\uparrow\beta = (\rho_1\uparrow\beta) \cap (\rho_2\uparrow\beta).$$

□

Proof

First we assume that $base(\rho_i) \subseteq \beta$ for $i = 1, 2$.

Projection is fairly simple in this case since it is clear that

$$base(\rho_1 \cap \rho_2) \subseteq base(\rho_1) \cup base(\rho_2) \subseteq \beta.$$

Consequently,

$$(\rho_1 \cap \rho_2)|\beta = (\rho_1 \cap \rho_2) = (\rho_1|\beta) \cap (\rho_2|\beta).$$

Using this we prove distributivity for the $\uparrow$ operator, under the given condition for β.

$$(\rho_1 \cap \rho_2)\uparrow\beta = \{\delta \mid \delta|\beta \in (\rho_1 \cap \rho_2)|\beta\}$$
$$= \{\delta \mid \delta|\beta \in \rho_1|\beta \cap \rho_2|\beta\}$$
$$= \{\delta \mid \delta|\beta \in \rho_1|\beta\} \cap \{\delta \mid \delta|\beta \in \rho_1|\beta\}$$
$$= \rho_1\uparrow\beta \cap \rho_2\uparrow\beta.$$

Next we assume that $abase(\rho_i) \subseteq \beta$, for $i = 1, 2$.

Distributivity for the two operators in this case can be reduced to distributivity for sets with their base, rather than their assertion base, contained inβ. From the assumption on ρ_i it follows that $\rho_i = \rho_i\uparrow\beta = \rho_i|\beta\uparrow\beta$.

For projection:

$$(\rho_1 \cap \rho_2)|\beta = (\rho_1|\beta\uparrow\beta \cap \rho_2|\beta\uparrow\beta)|\beta =$$

$(\rho_1|\beta \cap \rho_2|\beta)\uparrow\beta|\beta = (\rho_1|\beta \cap \rho_2|\beta)|\beta =$

$rho_1|\beta|\beta \cap \rho_2|\beta|\beta = \rho_1|\beta \cap \rho_2|\beta.$

And for chaotic closure:

$(\rho_1 \cap \rho_2)\uparrow\beta = (\rho_1|\beta\uparrow\beta \cap \rho_2|\beta\uparrow\beta)\uparrow\beta =$

$(\rho_1|\beta \cap \rho_2|\beta)\uparrow\beta\uparrow\beta = (\rho_1|\beta \cap \rho_2|\beta)\uparrow\beta =$

$rho_1|\beta\uparrow\beta \cap \rho_2|\beta\uparrow\beta = \rho_1\uparrow\beta \cap \rho_2\uparrow\beta.$

□

Lemma 3.35 (Parallel composition as derived operation)

Let $\beta = \beta_1 \cup \beta_2$.

$$\rho_1\ {}_{\beta_1}\|_{\beta_2}\ \rho_2 = (\rho_1\uparrow\beta_1|\beta) \cap (\rho_2\uparrow\beta_2|\beta)$$
$$= (\rho_1\uparrow\beta_1 \cap \rho_2\uparrow\beta_2)|\beta.$$

□

Proof

Let $\tilde{\rho} = (\rho_1\uparrow\beta_1|\beta) \cap (\rho_2\uparrow\beta_2|\beta)$.

First we show that $\tilde{\rho}$ satisfies clauses (i) and (ii) of the definition of the parallel composition operator. We tacitly use the monotonicity, and the properties (i) - (viii) listed in section 3.5.1, for the projection and chaotic closure operators.

$\tilde{\rho}|\beta_i = ((\rho_1\uparrow\beta_1|\beta) \cap (\rho_2\uparrow\beta_2|\beta))|\beta_i$

$\subseteq (\rho_i\uparrow\beta_i|\beta)|\beta_i = \rho_i\uparrow\beta_i|\beta_i = \rho_i|\beta_i$ (This shows (i)).

Since $base(\rho_i\uparrow\beta_i|\beta) \subseteq \beta$, projection onto β distributes over the intersection $(\rho_1\uparrow\beta_1|\beta) \cap (\rho_2\uparrow\beta_2|\beta)$. And since $(\rho_i\uparrow\beta_i|\beta)|\beta = (\rho_i\uparrow\beta_i|\beta)$ this implies that $\tilde{\rho}|\beta = \tilde{\rho}$. (This shows (ii)).

Next we show that $\tilde{\rho}$ is the *largest* set that satisfies clauses (i) and (ii). Assume that ρ' also satisfies these two clauses, that is:

(i') $\rho'|\beta_i \subseteq \rho_i|\beta_i$, for $i = 1, 2$ and

(ii') $\rho'|\beta = \rho'$.

Using (i') one sees that, for $i = 1, 2$:

$\rho' \subseteq \rho'\uparrow\beta_i = (\rho'|\beta_i)\uparrow\beta_i \subseteq \rho_i\uparrow\beta_i.$

Then, using (ii'), it follows that

$$\rho' = \rho'|\beta \subseteq \rho_i \uparrow \beta_i|\beta,$$

from which it immediately follows that

$$\rho' \subseteq (\rho_1 \uparrow \beta_1|\beta) \cap (\rho_2 \uparrow \beta_2|\beta).$$

We have shown that $\tilde{\rho}$ is the largest set satisfying clauses (i) and (ii), and by definition this equals the parallel composition $\rho_1\ \beta_1 \| \beta_2\ \rho_2$.

Finally we remark that, since

$$abase(\rho_i \uparrow \beta_i) \subseteq \beta_i \subseteq \beta,$$

it follows by lemma 3.34 that:

$$(\rho_1 \uparrow \beta_1|\beta) \cap (\rho_2 \uparrow \beta_2|\beta) = (\rho_1 \uparrow \beta_1 \cap \rho_2 \uparrow \beta_2)|\beta.$$

□

Finally we prove a result that amounts to the associativity of parallel composition as mentioned in section 2.3. There we claimed that:

$$(S_1\ \beta_1 \| \beta_2\ S_2)\ \| \beta_3\ S_3 = S_1\ \beta_1 \|\ (S_2\ \beta_2 \| \beta_3\ S_3),$$

where, by convention, the omitted base for the left hand side is $\beta_1 \cup \beta_2$, and for the right hand side is $\beta_2 \cup \beta_3$.

Lemma 3.36 (Associativity of parallel composition)

Let $\beta_{12} = \beta_1 \cup \beta_2$ and $\beta_{23} = \beta_2 \cup \beta_3$.

$$(\rho_1\ \beta_1 \| \beta_2\ \rho_2)\ \beta_{12} \| \beta_3\ \rho_3 = \rho_1\ \beta_1 \| \beta_{23}\ (\rho_2\ \beta_2 \| \beta_3\ \rho_3).$$

□

Proof

Let $\beta = \beta_1 \cup \beta_2 \cup \beta_3$. Since the commutativity of parallel composition is clear, it suffices to prove the following:

$$(\rho_1\ \beta_1 \| \beta_2\ \rho_2)\ \beta_{12} \| \beta_3\ \rho_3 =$$
$$(\rho_1 \uparrow \beta_1|\beta) \cap (\rho_2 \uparrow \beta_2|\beta) \cap (\rho_3 \uparrow \beta_3|\beta).$$

We use the two lemmata above.

$$(\rho_1\ \beta_1 \| \beta_2\ \rho_2)\ \beta_{12} \| \beta_3\ \rho_3 =$$
$$(\rho_1 \uparrow \beta_1|\beta_{12} \cap \rho_2 \uparrow \beta_2|\beta_{12}) \uparrow \beta_{12}|\beta \cap \rho_3 \uparrow \beta_3|\beta =$$
$$\rho_1 \uparrow \beta_1|\beta_{12} \uparrow \beta_{12}|\beta \cap \rho_2 \uparrow \beta_2|\beta_{12} \uparrow \beta_{12}|\beta \cap \rho_3 \uparrow \beta_3|\beta =$$
$$(\rho_1 \uparrow \beta_1|\beta) \cap (\rho_2 \uparrow \beta_2|\beta) \cap (\rho_3 \uparrow \beta_3|\beta).$$

(As was to be shown).

□

3.7 Process environments

Now that we have established the domain $\mathcal{P}(\Delta)$, we are able to define the meaning of **TNP** processes S. This meaning, $\mathcal{O}bs[\![S]\!]\eta$, is the set of observations that S admits, where η is a *process environment* determining the meaning of free occurrences of process variables in S. With this in mind it will be clear that η itself must be a function from the set of process variables $\mathcal{P}var$ to $\mathcal{P}(\Delta)$. Then, if we denote the domain of process environments by H, $\mathcal{O}bs$ is a function of the form:

$$\mathcal{O}bs : \mathbf{TNP} \rightarrow (H \rightarrow \mathcal{P}(\Delta))$$

Actually, a process variable $P_{(\mathbf{c},\mathbf{x})}$ denotes a process with a base contained in $(\mathbf{c},\mathbf{x})$. Therefore we define the domain of process environments as follows.

Definition 3.37 (Process environments)

$$(\eta \in)\ H \stackrel{\text{def}}{=} \{\eta \in \mathcal{P}var \rightarrow \mathcal{P}(\Delta) \mid \text{for all } P_\beta,\ \eta(P_\beta) \in \mathcal{P}(\Delta_\beta)\}.$$

$$(\eta \in)\ H_{p\mathbf{z}} \stackrel{\text{def}}{=} \{\eta \in \mathcal{P}var \rightarrow \mathcal{P}_{p\mathbf{z}}(\Delta) \mid \text{for all } P_\beta,\ \eta(P_\beta) \in \mathcal{P}_{p\mathbf{z}}(\Delta_\beta)\}.$$

The *variant* $\eta[\rho_\beta/P_\beta]$, where $\rho_\beta \in \mathcal{P}(\Delta_\beta)$, is defined as usual:

$$(\eta[\rho_\beta/P_\beta])(P_\beta) = \rho_\beta$$

$$(\eta[\rho_\beta/P_\beta])(Q_{\beta'}) = \eta(Q_{\beta'}) \text{ if } Q_{\beta'} \not\equiv P_\beta.$$

□

Lemma 3.38

(i) H and $H_{p\mathbf{z}}$, with the usual function space order, are complete lattices.

(ii) For any $P_\beta \in \mathcal{P}var$:

$$\lambda\eta.\lambda\rho_\beta.\eta[\rho_\beta/P_\beta] \in [H \rightarrow [\mathcal{P}(\Delta_\beta) \rightarrow H]]\,.$$

(iii) Similar to (ii), with H replaced by $H_{p\mathbf{z}}$.

Proof

(i) Take any collection $\{\eta_i | i \in I\}$ in H. It is clear that $\bigsqcup_i \eta_i$ exists as an element of $\mathcal{P}var \rightarrow \mathcal{P}(\Delta)$. It remains to show that $(\bigsqcup_i \eta_i)(P_\beta) \in \mathcal{P}(\Delta_\beta)$. This follows directly from:

$$(\bigsqcup_i \eta_i)(P_\beta) \;=\; \bigsqcup_i (\eta_i(P_\beta)),$$

and the fact that $\mathcal{P}(\Delta_\beta)$is a complete lattice. The proof for $H_{p\mathbf{z}}$ is very similar.

(ii) First we show that $\lambda\rho_\beta.\eta[\rho_\beta/P_\beta]$ is a completely additive function from $\mathcal{P}(\Delta_\beta)$ to H, for each particular $\eta \in H$.

Let $\{\rho_i | i \in I\}$ be some collection in $\mathcal{P}(\Delta_\beta)$.

$$\Big(\eta[(\bigcup_{i\in I}\rho_i)/P_\beta]\Big)(P_\beta) = \bigcup_{i\in I}\rho_i = \bigcup_{i\in I}\Big(\eta[\rho_i/P_\beta](P_\beta)\Big).$$

For process variables $Q_{\beta'} \not\equiv P_\beta$:

$$\Big(\eta[(\bigcup_{i\in I}\rho_i)/P_\beta]\Big)(Q_{\beta'}) = \eta(Q_{\beta'}) =$$

$$\bigcup_{i\in I}\eta(Q_{\beta'}) = \bigcup_{i\in I}\Big(\eta[\rho_i/P_\beta](Q_{\beta'})\Big).$$

So we have indeed:

$$\eta[(\bigcup_{i\in I}\rho_i)/P_\beta] = \bigsqcup_{i\in I}\eta[\rho_i/P_\beta].$$

Secondly, we show that $\lambda\eta.\lambda\rho.\eta[\rho/P_\beta]$ is *completely additive in its η argument*. Let $\{\eta_j | j \in J\}$ be some collection of environments,

$$(\lambda\eta.\lambda\rho_\beta.\eta[\rho_\beta/P_\beta])(\bigsqcup_{j\in J}\eta_j) = \lambda\rho_\beta.(\bigsqcup_{j\in J}\eta_j)[\rho_\beta/P_\beta]$$

$$= \lambda\rho_\beta.\bigsqcup_{j\in J}(\eta_j[\rho_\beta/P_\beta]) = \bigsqcup_{j\in J}\lambda\rho_\beta.\eta_j[\rho_\beta/P_\beta] =$$

$$\bigsqcup_{j\in J}(\lambda\eta.\lambda\rho_\beta.\eta[\rho_\beta/P_\beta](\eta_j)).$$

(iii) Very similar to case (ii).

□

3.8 The definition of the semantics *Obs*

Finally we come to the semantics of **TNP**, that we define by means of the following function:

$$Obs \ : \ \mathbf{TNP} \rightarrow [H_{p\mathbf{z}} \rightarrow P_{p\mathbf{z}}(\Delta)].$$

We do this by first defining a function with the same name *Obs* that has the functionality:

$$Obs \ : \ \mathbf{TNP} \rightarrow [H \rightarrow \mathcal{P}(\Delta)].$$

Then we prove that the restriction of this latter function to the subset H_{pz} of H has the correct functionality, that is, always yields *prefix closed, total* sets of computations. This way of defining *Obs* facilitates the definition of the semantics of mixed terms that we introduce in chapter 4.

We assume that we have available two interpretation functions, $\mathcal{E}$ and $\mathcal{B}$, for expressions and boolean expressions:

$\mathcal{E} : \mathcal{E}xp \rightarrow (State \rightarrow \mathcal{V}al)$,

$\mathcal{B} : \mathcal{B}exp \rightarrow (State \rightarrow \{true, false\})$.

The definition of *Obs* is by induction on the structure of processes.

Definition 3.39 (Semantics of **TNP**)

$$Obs[\![\mathbf{skip}]\!]\eta = \mathbf{1_Z}$$

$$Obs[\![\mathbf{abort}]\!]\eta = \mathbf{Z}$$

$$Obs[\![x := e]\!]\eta = Close\Big(\{(s_0, \epsilon, s_0|x : (\mathcal{E}[\![e]\!]s_0)) \mid s_0 \in State\}\Big)$$

$$Obs[\![b]\!]\eta = Close\Big(\{(s_0, \epsilon, s_0) \mid s_0 \in State,\ \mathcal{B}[\![b]\!]s_0\}\Big)$$

$$Obs[\![c.x{:}b]\!]\eta =$$

$$Close\Big(\{(s_0, < (c, v) >, s_0|x : v) \mid s_0 \in State,\ v \in \mathcal{V}al, \mathcal{B}[\![b]\!](s_0|x : v)\}\Big)$$

$$Obs[\![P_\beta]\!]\eta = \eta(P_\beta)$$

$$Obs[\![S_1 \backslash \mathbf{x}]\!]\eta = (Obs[\![S_1]\!]\eta)\backslash \mathbf{x}$$

$$Obs[\![S_1 \backslash \mathbf{c}]\!]\eta = (Obs[\![S_1]\!]\eta)\backslash \mathbf{c}$$

$$Obs[\![S_1\langle d/c\rangle]\!]\eta = (Obs[\![S_1]\!]\eta)[d/c]$$

$$Obs[\![S_1 \,;\, S_2]\!]\eta = Obs[\![S_1]\!]\eta \,\hat{\circ}\, Obs[\![S_2]\!]\eta$$

$$Obs[\![S_1 \ \mathbf{or}\ S_2]\!]\eta = Obs[\![S_1]\!]\eta \cup Obs[\![S_2]\!]\eta$$

$$Obs[\![S_1 \ {}_{\beta_1}\|_{\beta_2}\ S_2]\!]\eta = Obs[\![S_1]\!]\eta \ {}_{\beta_1}\|_{\beta_2}\ Obs[\![S_2]\!]\eta$$

$$Obs[\![\mu_z P_\beta.S_1]\!]\eta = \mu_z\Big(\lambda\rho_\beta.Obs[\![S_1]\!](\eta[\rho_\beta/P_\beta])\Big)$$

$$= \mu\Big(\lambda\rho_\beta.(Obs[\![S_1]\!](\eta[\rho_\beta/P_\beta]) \cup \mathbf{Z})\Big)$$

$$Obs[\![P_\beta \ = \ S_1 \ \mathbf{in}\ S_2]\!]\eta = Obs[\![S_2]\!](\eta[Obs[\![S_1]\!]\eta/P_\beta])$$

□

Theorem 3.40 (Well definedness of *Obs*)

The semantics *Obs* is well defined, and in fact:

$\lambda\eta.\mathcal{O}bs[\![S]\!]\eta \in [H \to \mathcal{P}(\Delta_\beta)]$, where $\beta = base(S)$.

□

Proof

Let $S \in \mathbf{TNP}, \beta = base(S) = (\mathbf{c}, \mathbf{x})$.
We must show:

(a) $\mathcal{O}bs[\![S]\!]\eta \in \mathcal{P}(\Delta)$, i.e., $\mathcal{O}bs[\![S]\!]\eta$ is a well defined subset of Δ.

(b) $chan(\mathcal{O}bs[\![S]\!]\eta) \subseteq \mathbf{c}$

(c) $assign(\mathcal{O}bs[\![S]\!]\eta) \subseteq \mathbf{x}$

(d) $\lambda\eta.\mathcal{O}bs[\![S]\!]\eta$ is a *continuous* function.

We prove this simultaneously by induction on the structure of the process S.

Case 1 $S \in \{\mathbf{skip}, \mathbf{abort}, b, x := e, c.x:b\}$

(a) Follows immediately from the definitions of $\mathbf{1_Z}, \mathbf{Z}$ and the *Close* operator.

(b) Is obvious for the first four cases, since here $chan(\mathcal{O}bs[\![S]\!]\eta) = \emptyset$.
Also, from the semantic definition and definition 1.4, it follows that $chan(\mathcal{O}bs[\![c.x:b]\!]\eta) = \{c\} = chan(c.x:b)$.

(c) Is obvious for the first three cases, since here $assign(\mathcal{O}bs[\![S]\!]\eta) = \emptyset$.
Further:

$$assign(\mathcal{O}bs[\![x := e]\!]\eta) \subseteq \{x\} \cup var(e) \;=\; var(x := e),$$

and similarly:

$$assign(\mathcal{O}bs[\![c.x:b]\!]\eta) \subseteq \{x\} \cup var(b) = var(c.x:b).$$

(d) Is satisfied since $\lambda\eta.\mathcal{O}bs[\![S]\!]\eta$ is a constant function in all five subcases.

Case 2 $S \equiv P_\beta$.

Clearly $\mathcal{O}bs[\![P_\beta]\!]\eta = \eta(P_\beta) \in \mathcal{P}(\Delta_\beta)$, by the definition of H.
Moreover, function application is continuous in its first argument.
Hence, $\lambda\eta.\eta(P_\beta)$ is continuous.

Case 3 $S \in \{S_1\backslash\mathbf{x}, S_1\backslash\mathbf{c}, S_1\langle d/c\rangle\}$.

Let $\beta_1 = base(S_1)$, and let $\beta = base(S)$. By induction,

$$\lambda\eta.\mathcal{O}bs[\![S_1]\!]\eta \in [H \to \mathcal{P}(\Delta_{\beta_1})].$$

It has been proven above that the corresponding semantic operations are all continuous. Moreover, by comparing the results of lemma 3.31 and definition 2.4 of base(S) in chapter 2, one sees that these semantic operations transform a set with its base contained in β_1 into a set with base contained in β. That is, the semantic operations have the functionality:

$$\backslash\mathbf{x},\ \backslash\mathbf{c},\ \langle d/c\rangle \in [\mathcal{P}(\Delta_{\beta_1})\rightarrow\mathcal{P}(\Delta_\beta)].$$

But then, by function composition we obtain semantic functions with the following functionality:

$$\lambda\eta.\mathit{Obs}[\![S]\!]\eta \in [H\rightarrow\mathcal{P}(\Delta_\beta)].$$

Case 4 $S \in \{S_1\,;S_2,\ S_1\mathbf{or}S_2,\ S_1\ {}_{\beta_1}\|_{\beta_2}\ S_2\}$

Let $\beta_i = \mathit{base}(S_i)$ for $i = 1,2$. Then $\beta = \mathit{base}(S) = \beta_1 \cup \beta_2$ by definition 1.4.

By induction we may assume that, for $i = 1,2$:

$$\lambda\eta.\mathit{Obs}[\![S_i]\!]\eta \in [H\rightarrow\mathcal{P}(\Delta_{\beta_i})].$$

The semantic operations for these three operators have been shown to be continuous. Lemma 3.31 states that, for sequential composition and choice, sets ρ_1, ρ_2 with bases contained in β_1, β_2 are mapped into a set with base contained in $\beta_1 \cup \beta_2$. The same is clear for parallel composition, from the definition of this operation. This means that:

$$\hat{\circ}, \cup,\ {}_{\beta_1}\|_{\beta_2} \in [\mathcal{P}(\Delta_{\beta_1}) \times \mathcal{P}(\Delta_{\beta_2})\rightarrow\mathcal{P}(\Delta_\beta)].$$

So, by function composition, we obtain the function

$$\lambda\eta.\mathit{Obs}[\![S]\!]\eta \in [H\rightarrow\mathcal{P}(\Delta_\beta)]$$

Case 5 $S \equiv \mu_z P_\beta.S_1$

In chapter 2 the definition of bases and the context sensitive restrictions guarantee that $\mathit{base}(S) = \beta \supseteq \beta_1 \stackrel{\text{def}}{=} \mathit{base}(S_1)$.
From the theory of cpos it is known that the fixed point operator is a continuous operator itself, that is, μ is an element of:

$$[[\mathcal{P}(\Delta_\beta)\rightarrow\mathcal{P}(\Delta_\beta)]\rightarrow\mathcal{P}(\Delta_\beta)].$$

Therefore, it suffices to show that :

$$\lambda\eta.\lambda\rho_\beta.\mathit{Obs}[\![S_1]\!](\eta[\rho_\beta/P_\beta]) \cup \mathbf{z}$$

is an element of $[H\rightarrow[\mathcal{P}(\Delta_\beta)\rightarrow\mathcal{P}(\Delta_\beta)]]$. Now this is the case, since:

(i) by induction,

$$\lambda\eta.Obs[\![S_1]\!]\eta \in [H \to \mathcal{P}(\Delta_{\beta_1})], \text{ and so}$$

$$\lambda\eta.Obs[\![S_1]\!]\eta \cup \mathbf{Z} \in [H \to \mathcal{P}(\Delta_\beta)].$$

(ii) by lemma 3.37, $\lambda\eta.\lambda\rho_\beta.\eta[\rho_\beta/P_\beta] \in [H \to [\mathcal{P}(\Delta_\beta) \to H]]$.

Composition of these two functions yields the desired result.

Case 6 $S \equiv P_\beta = S_1 \text{ in } S_2$

Let $\beta_i = base(S_i)$, for $i = 1, 2$. Then $\beta = base(S) = \beta_2$.

By induction we have that

$\lambda\eta.Obs[\![S_i]\!]\eta \in [H \to \mathcal{P}(\Delta_{\beta_i})]$, for $i = 1, 2$.

By lemma 3.37,

$\lambda\eta.\lambda\rho_{\beta_1}.\eta[\rho_{\beta_1}/P_{\beta_1}] \in [H \to [\mathcal{P}(\Delta_{\beta_1}) \to H]]$.

From this we infer that:

$\lambda\eta.\eta[Obs[\![S_1]\!]\eta/P_\beta] \in [H \to H]$,

and hence that:

$\lambda\eta.Obs[\![S_2]\!](\eta[Obs[\![S_1]\!]\eta/P_\beta]) \in [H \to \mathcal{P}(\Delta_{\beta_2})]$.

Since $\beta_2 = \beta$ we conclude that

$Obs[\![S]\!]\eta \in [H \to \mathcal{P}(\Delta_\beta)]$.

□

Next we show that the restriction of *Obs* to the domain $H_{p\mathbf{z}}$ is a function that yields total prefix closed sets.

Lemma 3.41

The restriction of *Obs* to $H_{p\mathbf{z}}$ has the following functionality:

$$Obs : \mathbf{TNP} \to [H_{p\mathbf{z}} \to \mathcal{P}_{p\mathbf{z}}(\Delta)].$$

□

Proof

The proof is by a straightforward induction on the structure of processes. For process variables P_β we have that $Obs[\![P_\beta]\!]\eta \in \mathcal{P}_{p\mathbf{z}}(\Delta_\beta)$ by the assumption that $\eta \in H_{p\mathbf{z}}$. In section 3.4 we have shown that the denotations for

the atomic processes **skip** and **abort** are total prefix closed sets, and it is clear from the definition that the denotations for the other atomic processes are total and prefix closed because we used the *Close* operator for the corresponding clauses. We have also shown that the semantic operations that we have used in the definition of *Obs* all *preserve* totality and prefix closedness. Finally, in section 3.3 it was shown that $\mathcal{P}_{pz}(\Delta)$ is a subcpo of $\mathcal{P}_{z}(\Delta)$, and so the μ_z operator yields a prefix closed set for functions in the subspace $[\mathcal{P}_{pz}(\Delta) \rightarrow \mathcal{P}_{pz}(\Delta)]$.

□

3.9 An alternative representation

Up to now we have regarded the meaning of a process as a set of computations. This is rather different from the usual semantics for sequential programs, where one defines the meaning of a program as a transformer of states or, in the case of nondeterministic programs, as a transformer of sets of states. We show in this section that our semantics is in fact isomorphic to such a transformer semantics, based upon the notion of so called *generalized states*. A generalized state is a pair (h, s), consisting of a trace component h and a state component s. Intuitively, such pairs can be used to describe, for some moment during the execution of some process S, the current state of the variables and the sequence of communications performed thus far. If this moment corresponds to the *start* of some process S, then we call (h, s) the *initial trace and state* for the execution of S. If and when S terminates, some generalized state (h', s') has been reached, where h' includes the initial trace h as a prefix. A generalized state of the form $(h, \perp)$ shall be used to describe those moments during execution where no observable final state has been reached yet. As one may expect, when we consider the semantics of a sequential composition $S_1; S_2$, a final generalized state (h, s), with an *observable*, i.e. non bottom, state s for some process S_1, is converted into a pair $(h, \perp)$ to indicate an intermediate stage of the computation, and also functions as an initial generalized state for the execution of S_2.

The first step is to treat a set of computations as the *graph of a function*. To this end we define the following domains.

Definition 3.42 ()

- $(\sigma \in)\ \Sigma \stackrel{\text{def}}{=} \mathcal{T}race \times \mathcal{S}tate_{\perp}$ – (Generalized states)

- The partial order $\sqsubseteq_\Sigma$ on Σ is defined by:

  $(t, s) \sqsubseteq_\Sigma (t', s')$ iff either $(t, s) = (t', s')$ or $s = \perp$ and $t \preceq t'$.

  $\preceq$ is the prefix order on traces. We omit the Σ subscript when no ambiguity arises.

- A subset X of Σ is called prefix closed if it is downwards closed with respect to $\sqsubseteq_\Sigma$. This is the case iff for every generalized state (h, s) in X all pairs $(h', \perp)$ are also in X, where h' is an arbitrary prefix of h.
- A subset of Σ is called *total* if it contains the pair $(\varepsilon, \perp)$.
- $\mathcal{P}(\Sigma)$ denotes the collection of all subsets of Σ. $\mathcal{P}_p(\Sigma)$, $\mathcal{P}_t(\Sigma)$ and $\mathcal{P}_{pt}(\Sigma)$ denote the collections of all *prefix closed subsets,* all *total subsets* and all *total prefix closed subsets* of Σ.
- $(\phi \in)\ \mathcal{F}un \stackrel{\text{def}}{=} \{\phi \in (State_\perp \to \mathcal{P}(\Sigma)) \mid \phi(\perp) \subseteq \{(\varepsilon, \perp)\}\}$
- A functions $\phi : State_\perp \to \mathcal{P}(\Sigma)$ is called total (prefix closed) if, for each s_0, $\phi(s_0)$ is nonempty (prefix closed).

□

There is an obvious bijection between the domains $\mathcal{P}(\Delta)$ and $\mathcal{F}un$. The bijection $\mathcal{F}$: $\mathcal{P}(\Delta) \to \mathcal{F}un$, and its inverse $\mathcal{G}$ are defined by:

$$\mathcal{F}(\rho) = \lambda s_0.\{(t, s) \mid (s_0, t, s) \in \rho\} \text{ and:}$$

$$\mathcal{G}(\phi) = \{(s_0, t, s) \mid (t, s) \in \phi(s_0)\}.$$

Note that the strictness requirement for $\mathcal{F}un$ elements corresponds to the property of the domain Δ that $(\perp, \varepsilon, \perp)$ is the *only* computation starting with a bottom initial state.

The bijection $\mathcal{F} : \mathcal{P}(\Delta) \to \mathcal{F}un$ preserves prefix closures and totality. That is:

- $\rho \in \mathcal{P}(\Delta)$ is prefix closed iff $\mathcal{F}(\rho) \in \mathcal{F}un$ is prefix closed.
- $\mathbf{Z} \subseteq \rho$ iff $\mathcal{F}(\rho)$ is total.

This follows directly from the definitions. In fact the complete cpo structure of the domains $\mathcal{P}(\Delta)$ and $\mathcal{P}_\mathbf{Z}(\Delta)$ is inherited, if we define the following cpo structures:

Definition 3.43 (Cpos based on $\mathcal{P}(\Sigma)$)

We introduce the following cpos:

- $(\mathcal{P}(\Sigma), \subseteq, \emptyset)$
- $(\mathcal{P}_t(\Sigma), \subseteq, \{(\epsilon, \perp)\})$
- $(\mathcal{P}_{pt}(\Sigma), \subseteq, \{(\epsilon, \perp)\})$
- The function space $\mathcal{F}un$ with the usual pointwise order on functions.
- The subspaces $\mathcal{F}un_t$ and $\mathcal{F}un_{pt}$ of $\mathcal{F}un$ where the range of functions is restricted to $\mathcal{P}_t$ or $\mathcal{P}_{pt}$.

□

It is straightforward to check that F and G are *monotone* functions. From general cpo theory it is known that if a bijection and its inverse are both monotone, then they are both continuous bottom preserving functions. Therefore, since F *is* such a bijection, one sees that $\mathcal{P}(\Delta)$ and $\mathcal{F}un$ are isomorphic as cpos.

The next step in our development is to associate with each $\mathcal{F}un$ element ϕ a transformer of sets of generalized states.

Definition 3.44 (Generalized predicate transformers)

Define, for $\phi \in State_\perp \to \mathcal{P}(\Sigma)$, the functions:

$\phi' : \Sigma \to \mathcal{P}(\Sigma)$ and

$\phi^\dagger : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$, as follows.

$$\phi'\Big((t_0, s_0)\Big) \stackrel{\text{def}}{=} t_0 \hat{} \phi(s_0)$$

$$\phi^\dagger(X) \stackrel{\text{def}}{=} \bigcup_{(t_0,s_0)\in X} \phi'\Big((t_0, s_0)\Big) = \bigcup_{(t_0,s_0)\in X} t_0 \hat{} \phi(s_0)$$

□

By analogy with "classical" forward predicate transformer semantics, we call $\mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$ elements generalized predicate transformers. We combine the $\dagger$ mapping with the isomorphism F, and define a mapping $\mathcal{T}r$.

Definition 3.45 (The mapping $\mathcal{T}r$)

$$\mathcal{T}r : \mathcal{P}(\Delta) \to (\mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma))$$

is defined by:

$$\mathcal{T}r(\rho) \stackrel{\text{def}}{=} (\mathcal{F}(\rho))^\dagger.$$

□

Obviously this associates a state transformer semantics $\mathcal{T}r(\mathcal{O}bs[\![S]\!]\eta)$ with process S.

The $\mathcal{T}r$-operator enjoys the following property:

$$\mathcal{T}r(\rho_1 \hat{\circ} \rho_2) = \mathcal{T}r(\rho_2) \circ \mathcal{T}r(\rho_1)$$

Thus we see that a *sequential composition* of processes corresponds to a simple *function composition* of the associated transformers:

$$\mathcal{T}r(\mathcal{O}bs[\![S_1 ; S_2]\!]\eta) = \mathcal{T}r(\mathcal{O}bs[\![S_2]\!]\eta) \circ \mathcal{T}r(\mathcal{O}bs[\![S_1]\!]\eta)$$

We want to formulate our semantics directly in terms of Σ-transformers. First, we collect some general facts about isomorphic cpo's.

Definition 3.46 (Isomorphic representations of environments)

Assume that $\alpha : \mathcal{P}(\Delta) \to D$ is some isomorphism of cpo's. Define:

$$D_\beta \stackrel{\text{def}}{=} \alpha(\mathcal{P}(\Delta_\beta)) \quad \text{(for each } \beta \in \mathcal{B}ase\text{)}$$

$$Z \stackrel{\text{def}}{=} \{\varsigma \in \mathcal{P}var \to D \mid \varsigma = \alpha \circ \eta \text{ for some } \eta \in H\}$$

□

Clearly the following holds:

- $\varsigma(P_\beta) \in D_\beta$ for $\varsigma \in Z$ and
- $\lambda\eta.\alpha \circ \eta$ is an isomorphism from H to Z.

Definition 3.47 (Isomorphic copy of semantics)

With α, D, Z as above, define $\mathcal{M} : \mathbf{TNP} \to (Z \to D)$ such as to make the following diagram commute:

$$\begin{array}{ccc} H & \xrightarrow{\mathcal{O}bs[\![S]\!]} & \mathcal{P}(\Delta) \\ {\scriptstyle \lambda\eta.\alpha\circ\eta}\downarrow & & \downarrow{\scriptstyle \alpha} \\ Z & \xrightarrow[\mathcal{M}[\![S]\!]]{} & D \end{array}$$

That is, $\mathcal{M}[\![S]\!]$ is determined by: $\mathcal{M}[\![S]\!](\alpha \circ \eta) = \alpha \circ \mathcal{O}bs[\![S]\!]\eta$.

This canonical way of defining semantic functions $\mathcal{M}$ becomes applicable to our domain of generalized predicate transformers if we can turn the mapping $\mathcal{T}r$ into an isomorphism of cpos. To this end we have:

Definition 3.48 (Domain $\mathcal{T}ra$ of Σ-transformers)

$\mathcal{T}ra$is defined as the set of all functions $\psi \in \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$ that satisfy the following conditions:

(i) $\psi(\perp_\Sigma) \subseteq \perp_\Sigma$ (strictness)

(ii) $\psi(\bigcup_{i \in I} X_i) = \bigcup_{i \in I} \psi(X_i)$ (complete additivity)

(iii) $\psi(t_0 \hat{} X) = t_0 \hat{} \psi(X)$ (prefix preservation)

□

Lemma 3.49

The mapping $\lambda\phi.\phi^\dagger$ is an isomorphism between $\mathcal{F}un$ and $\mathcal{T}ra$. Its inverse is: $\lambda\psi.\lambda s_0.\psi(\{(\epsilon, s_0)\})$.

□

Proof

First we prove that the mappings between $\mathcal{F}un$ and $\mathcal{T}ra$ are well defined.

Assume that $\phi \in \mathcal{F}un$. We check that $\phi^\dagger$ satisfies properties (i) - (iii) of the definition of Σ transformers.

Property (i)

$$\phi^\dagger(\perp_\Sigma) = \phi^\dagger(\{(\epsilon, \perp)\}) = \phi(\perp) \subseteq \perp_\Sigma.$$

Property (ii)

$$\phi^\dagger(\bigcup_{i \in I} X_i) = \bigcup\{\phi'((t_0, s_0)) \mid (t_0, s_0) \in \bigcup_{i \in I} X_i\} =$$

$$\bigcup\{\phi'((t_0, s_0)) \mid (t_0, s_0) \in X_i \mid i \in I\} =$$

$$\bigcup_{i \in I}(\bigcup\{\phi'((t_0, s_0)) \mid (t_0, s_0) \in X_i\}) = \bigcup_{i \in I} \phi^\dagger(X_i).$$

Property (iii)

$$\phi^\dagger(t_0 \hat{} X) = \bigcup\{t \hat{} \phi(s) \mid (t, s) \in t_0 \hat{} X\} =$$

$$\bigcup\{(t_0 t_1) \hat{} \phi(s) \mid (t_1, s) \in X\} =$$

$$\bigcup t_0{}^\frown\{t_1{}^\frown\phi(s) \mid (t_1,s)\in X\} =$$

$$t_0{}^\frown\bigcup\{t_1{}^\frown\phi(s) \mid (t_1,s)\in X\} = t_0{}^\frown\phi^\dagger(X).$$

This proves the well definedness of the mapping $\lambda\phi.\phi^\dagger$. The well definedness of $\lambda\psi\,.\,\lambda s_0\,.\,\psi(\{(\varepsilon,s_0)\})$ follows from:

$$\lambda s_0\,.\,\psi(\{(\varepsilon,s_0)\})\,(\bot) = \psi(\{(\varepsilon,\bot)\}) = \psi(\bot_\Sigma) \subseteq \bot_\Sigma.$$

Next we prove that the two mappings are inverses of each other:

$$\lambda s_0.\phi^\dagger(\{(\varepsilon,s_0)\}) = \lambda s_0.\bigcup\{t{}^\frown\phi(s) \mid (t,s)\in\{(\varepsilon,s_0)\}\} =$$

$$\lambda s_0.\phi(s_0) = \phi.$$

Vice versa, we have:

$$\big(\lambda s_0.\psi(\{(\varepsilon,s_0)\})\big)^\dagger =$$

$$\lambda X.\bigcup\{t{}^\frown(\lambda s_0.\psi(\{(\varepsilon,s_0)\})(s)) \mid (t,s)\in X\} =$$

$$\lambda X.\bigcup\{t{}^\frown\psi(\{(\varepsilon,s)\}) \mid (t,s)\in X\} =$$

$$\lambda X.\bigcup\{\psi(\{(t,s)\}) \mid (t,s)\in X\} =$$

$$\lambda X.\psi(\bigcup\{\{(t,s)\} \mid (t,s)\in X\}) =$$

$$\lambda X.\psi(X) = \psi.$$

The monotonicity of both mappings is clear.

□

So we have the following cpo isomorphisms:

$$\mathcal{P}(\Delta) \xrightarrow{\mathcal{F}} \mathcal{F}un \xrightarrow{\lambda\phi.\phi^\dagger} \mathcal{T}ra$$

Consequently, we define the state transformer semantics for **TNP** as follows:

$\mathcal{M} : \mathbf{TNP} \to (Z \to \mathcal{F}un)$,

$\mathcal{M}^\dagger : \mathbf{TNP} \to (Z^\dagger \to \mathcal{T}ra)$, defined by

$\mathcal{M}[\![S]\!](\mathcal{F}\circ\eta) \stackrel{\text{def}}{=} \mathcal{F}(\mathcal{O}bs[\![S]\!]\eta)$,

$\mathcal{M}^\dagger[\![S]\!](\mathcal{T}r\circ\eta) \stackrel{\text{def}}{=} \mathcal{T}r(\mathcal{O}bs[\![S]\!]\eta)$,

where $Z = \{\mathcal{F}\circ\eta \mid \eta\in H\}, Z^\dagger = \{\mathcal{T}r\circ\eta \mid \eta\in H\}$.

□

Chapter 4

Correctness formulae

4.1 Introduction

Our aim is the study of formal systems for reasoning about **TNP** programs, and so our first task is now to introduce the formulae of these systems. Corresponding to the different semantic domains, we have a number of different classes of formulae. Those formulae corresponding to the domains $\mathcal{T}race$, $\mathcal{S}tate_\perp$, Σ and Δ are called *assertions*, and those corresponding to program denotations are called *correctness formulae*.

Our language for assertions deviates in several ways from what is the custom for Hoare style proof systems:

- It is a first order *typed* predicate language. That is, there are four types of terms, all called expressions, denoting *values, channelnames, natural numbers* and *traces.*
- We distinguish between *assignable* variables, denoting values that can be changed by program execution, and *logical* variables, denoting values that remain fixed throughout program execution.
  Assignable variables coincide with the variables in programs, while logical variables cannot occur within program text.
- Sometimes we also distinguish between the value of an assignable variable in an *initial* state and in a *final* state.
  Therefore we end up with four types of variables:

  1. $(x \in)\mathcal{V}ar$ — *Assignable variables*, ranging over $\mathcal{V}al_\perp$. $\mathcal{V}ar$ was already introduced with the semantics. For assertions interpreted in both an initial and final state, a term x will denote the value of variable x in the *final* state.

2. $(x^\circ \in)\mathcal{V}ar^\circ$ — Similar to $\mathcal{V}ar$ except that x° denotes the *initial* state value of x.
3. h — the single variable denoting the *trace* or *communication history* of a program. It is similar to assignable variables in that it is affected by program execution.
4. $(v \in)\mathcal{L}var, (n \in)\mathcal{N}var, (t \in)\mathcal{T}var$ — *Logical variables*, ranging over $\mathcal{V}al_\perp, \mathbf{N}$ and $\mathcal{T}race$, where $\mathbf{N}$ denotes the natural numbers. Variables of this type can occur *free* or *bound* by means of a quantifier. Free logical variables are also called *ghost* variables. Logical variables n, ranging over the natural numbers $\mathbf{N}$ are used as indices for trace expressions.

Expressions are built up from constants and the variables above, by means of operations. As for operations on traces we focus the attention on the important *trace-projection* operator "$|$". We found that, in order to formulate a *compositional* rule for parallel composition, one needs to be careful *in which ways* to refer to the communication history. Restricting such references by making them exclusively via projections allows a rule without any kind of interference freedom tests, by imposing some simple syntactic restrictions on the projections occurring in the premisses of the rule.

Other operations on traces include the *length* $|t|$ of a trace t, as well as $chan(t[n])$ and $val(t[n])$, denoting the *channelname* and *communicated value* of the n-th communication of t.

Finally we remark that, unlike what is the custom, we shall not consider an assertion to be undefined in bottom states. The reason is that a bottom state only indicates an *unfinished* computation, and so it makes sense to interpret an assertion for a computation ending in a bottom state. This is even essential to describe *nonterminating* processes. Unlike what is the case for sequential programs such processes are not deemed to be uninteresting, for it is possible to *communicate* with them. Therefore our assertion language must be capable of describing the communication behaviour of a nonterminated process. Within assertions, the symbols $\top$ and $\top^\circ$ denote the characteristic predicates for non-bottom final- and initial states.

4.2 The syntax of assertions

For each of the domains $\mathcal{T}race, \mathcal{S}tate, \Sigma$ and Δ there is a class $\mathcal{A}ssn(\cdot)$. Since $\mathcal{A}ssn(\mathcal{T}race), \mathcal{A}ssn(\mathcal{S}tate)$ and $\mathcal{A}ssn(\Sigma)$ can be obtained as subsets

of $\mathcal{A}ssn(\Delta)$, we first define this latter class. We shall write simply $\mathcal{A}ssn$ instead of $\mathcal{A}ssn(\Delta)$.

Definition 4.1 (Syntax of expressions)

- $(e \in)$ *exp* — The class of $\mathcal{V}al_\perp$ *expressions.*

  (This class is not fixed; we list the minimum requirements.)

  $$e ::= x \mid x^\circ \mid v \mid val(te[ie]) \mid \cdots$$

- $(ie \in)$ *iexp* — *Integer expressions.*

  $$ie ::= 0 \mid 1 \mid n \mid ie_1 + ie_2 \mid ie_1 \cdot ie_2 \mid |te|$$

- $(ce \in)$ *cexp* — *Channelname expressions.*

  (c denotes a *constant* channelname; there are no variables)

  $$ce ::= c \mid chan(te[ie])$$

- $(\mathbf{c} \in)$ *cset* — Fixed *sets* of *channelnames.*

  $$\mathbf{c} ::= \{c_1, \cdots, c_n\} \mid chan - \{c_1, \cdots, c_n\}$$

- $(te \in)$ *texp* — *Trace expressions.*

  $$te ::= \varepsilon \mid <(c, v)> \mid h \mid t \mid te_1 te_2 \mid te|\mathbf{c} \mid te[d/c]$$

□

Definition 4.2 (Syntax of assertions)

$(\chi \in)$ $\mathcal{A}ssn$ — *Assertions.*

$$\chi ::= e_1 = e_2 \mid ie_1 = ie_2 \mid ce_1 = ce_2 \mid te_1 = te_2 \mid$$
$$\top \mid \top^\circ \mid \chi_1 \wedge \chi_2 \mid \neg\chi \mid \exists v(\chi) \mid \exists n(\chi) \mid \exists t(\chi)$$

□

For expressions and assertions the sets $FV(\cdot)$ and $BV(\cdot)$, of *free-* and *bound* variables, are defined as usual. Note that the variables x, x° and h can only occur free, since there are no quantifiers for these types of variables. (Only logical variables can occur bound).

From $FV(\chi)$ we derive the following sets:

$$var(\chi) \stackrel{\text{def}}{=} FV(\chi) \cap \mathcal{V}ar$$

$$var^\circ(\chi) \stackrel{\text{def}}{=} FV(\chi) \cap \mathcal{V}ar^\circ$$

$$lvar(\chi) \stackrel{\text{def}}{=} FV(\chi) \cap \mathcal{L}var$$

$nvar(\chi) \stackrel{\text{def}}{=} FV(\chi) \cap \mathcal{N}var$

$tvar(\chi) \stackrel{\text{def}}{=} FV(\chi) \cap \mathcal{T}var$

$gvar(\chi) \stackrel{\text{def}}{=} lvar(\chi) \cup nvar(\chi) \cup tvar(\chi)$

$var^{(\circ)}(\chi) \stackrel{\text{def}}{=} \{x \in \mathcal{V}ar \mid x \in var(\chi) \text{ or } x^\circ \in var^\circ(\chi)\}$

(And analogously for expressions).

Definition 4.3 (Assertions on $Trace$, $State$ and Σ)

- $(\varphi, \psi \in)\mathcal{A}ssn(\Sigma) \stackrel{\text{def}}{=}$
  $\{\varphi \in \mathcal{A}ssn \mid var^\circ(\varphi) = \emptyset \text{ and } \top^\circ\text{does not occur within } \varphi\}$
- $(I \in)\mathcal{A}ssn(\mathcal{T}race) \stackrel{\text{def}}{=}$
  $\{I \in \mathcal{A}ssn(\Sigma) \mid var(\varphi) = \emptyset \text{ and } \top\text{does not occur within } I\}$
- $(\xi \in)\mathcal{A}ssn(State) \stackrel{\text{def}}{=} \{\xi \in \mathcal{A}ssn(\Sigma) \mid h \notin FV(\xi)\}$

□

4.3 The meaning of assertions

The next step is to *interpret* expressions and assertions. For the interpretation of assertions, we need the domain Υ of truthvalues:

Definition 4.4 (truth values Υ)

The complete lattice of truth values is defined as:

$\Upsilon \stackrel{\text{def}}{=} \{\mathbf{true}, \mathbf{false}\}$ ordered by: $\mathbf{false} \sqsubseteq \mathbf{true}$.

□

Expressions and assertions are interpreted in a computation triple (s_0, h, s), determining the values of variables x°, x and h, and in a *logical variable environment* γ, determining the values of free logical variables v, n and t.

Definition 4.5 (logical variable environments)

$(\gamma \in)\Gamma \stackrel{\text{def}}{=} (\mathcal{L}var \to \mathcal{V}al) \times (\mathcal{N}var \to \mathbf{N}) \times (\mathcal{T}var \to \mathcal{T}race)$

Each γ is a triple $(\gamma_L, \gamma_N, \gamma_T)$. However, we will usually write simply $\gamma(v)$, $\gamma(n)$ and $\gamma(t)$ instead of $\gamma_L(v)$, $\gamma_N(n)$ and $\gamma_T(t)$.

□

The interpretation of expressions is by means of the following functions:

$\mathcal{E} : exp \rightarrow (\Gamma \rightarrow (\Delta \rightarrow \mathcal{V}al_\perp))$

$\mathcal{CE} : cexp \rightarrow (\Gamma \rightarrow (\Delta \rightarrow Chan_\perp))$

$\mathcal{IE} : iexp \rightarrow (\Gamma \rightarrow (\Delta \rightarrow \mathbf{N}))$

$\mathcal{TE} : texp \rightarrow (\Gamma \rightarrow (\Delta \rightarrow \mathcal{T}race))$

Remark

There is no need for any bottom value for trace expressions, since the computation history is always well defined, and all trace operations are total. In particular the expression $< (c, v) >$ is always defined, but only because we do only allow logical variables v, rather than *arbitrary* expressions e, as the value component of the communication. For similar reasons we don't need a bottom value for natural numbers in our setup.

We list the defining clauses for these four functions.

- $\mathcal{E}[\![x]\!]\gamma(s_0, h, \perp) = \perp$
  $\mathcal{E}[\![x]\!]\gamma(s_0, h, s) = s(x)$ for $s \neq \perp$
  $\mathcal{E}[\![x^\circ]\!]\gamma(\perp, h, s) = \perp$
  $\mathcal{E}[\![x^\circ]\!]\gamma(s_0, h, s) = s_0(x)$ for $s_0 \neq \perp$
  $\mathcal{E}[\![v]\!]\gamma(s_0, h, s) = \gamma(v)$ $(i.e. \gamma_L(v))$
  $\mathcal{E}[\![val(te[ie])]\!]\gamma(s_0, h, s)$ = the value part of the element with index $\mathcal{IE}[\![ie]\!]\gamma(s_0, h, s)$ of trace $\mathcal{TE}[\![te]\!]\gamma(s_0, h, s)$, provided this element exists, otherwise it is defined to be $\perp$.

- $\mathcal{IE}[\![0]\!]\gamma(s_0, h, s) = 0$
  $\mathcal{IE}[\![1]\!]\gamma(s_0, h, s) = 1$
  $\mathcal{IE}[\![n]\!]\gamma(s_0, h, s) = \gamma(n)$ (i.e. $\gamma_N(n)$)
  $\mathcal{IE}[\![ie_1 + ie_2]\!]\gamma(s_0, h, s) = \mathcal{IE}[\![ie_1]\!]\gamma(s_0, h, s) + \mathcal{IE}[\![ie_2]\!]\gamma(s_0, h, s)$
  $\mathcal{IE}[\![ie_1 \cdot ie_2]\!]\gamma(s_0, h, s) = \mathcal{IE}[\![ie_1]\!]\gamma(s_0, h, s) \cdot \mathcal{IE}[\![ie_2]\!]\gamma(s_0, h, s)$
  $\mathcal{IE}[\![|te|]\!]\gamma(s_0, h, s)$ = the length of trace $\mathcal{TE}[\![te]\!]\gamma(s_0, h, s)$.

- $\mathcal{CE}[\![c]\!]\gamma(s_0, h, s) = c$
  $\mathcal{CE}[\![chan(te[ie])]\!]\gamma(s_0, h, s)$ = the channel part of the element with index $\mathcal{IE}[\![ie]\!]\gamma(s_0, h, s)$ of trace $\mathcal{TE}[\![te]\!]\gamma(s_0, h, s)$, provided this element exists, otherwise it is defined to be $\perp$.

- $\mathcal{TE}[\![\varepsilon]\!]\gamma(s_0, h, s) = \varepsilon$
  $\mathcal{TE}[\![< (c, v) >]\!]\gamma(s_0, h, s) =< (c, \gamma(v)) >$
  $\mathcal{TE}[\![h]\!]\gamma(s_0, h, s) = h$
  $\mathcal{TE}[\![t]\!]\gamma(s_0, h, s) = \gamma(t)$ $(i.e. \gamma_T(t))$
  $\mathcal{TE}[\![te_1 te_2]\!]\gamma(s_0, h, s) = \mathcal{TE}[\![te_1]\!]\gamma(s_0, h, s)\hat{}\mathcal{TE}[\![te_2]\!]\gamma(s_0, h, s)$
  $\mathcal{TE}[\![te|\mathbf{c}]\!]\gamma(s_0, h, s) = (\mathcal{TE}[\![te]\!]\gamma(s_0, h, s))|\mathbf{c}$

$$\mathcal{T}\mathcal{E}[\![te[d/c]]\!]\gamma(s_0,h,s) = (\mathcal{T}\mathcal{E}[\![te]\!]\gamma(s_0,h,s))[d/c]$$

- We left open the possibility that there are other operators for expressions. All such operators must be interpreted *strictly*.

There is some ambiguity between the function $\mathcal{E}$ as above and, on the other hand, the function $\mathcal{E} : \mathcal{E}xp \to (State \to Val)$ as defined in chapter 3.

However, we shall make the following assumption:

(a) $\mathcal{E}xp \subseteq exp$, i.e. any expression e that can occur in a program is also allowed within assertions.

(b) $\mathcal{E}[\![e]\!]\gamma(s_0,h,s) = \mathcal{E}[\![e]\!]s \quad$ for $e \in \mathcal{E}xp$.

(c) Consequently , if $e \in \mathcal{E}xp$ and $\bar{x}$ is a list such that $\{\bar{x}\} \supseteq var(e)$, then $\mathcal{E}[\![e[\bar{x}^\circ/\bar{x}]]\!]\gamma(s_0,h,s) = \mathcal{E}[\![e]\!]s_0$

Next we interpret *assertions* by means of the function

$$\mathcal{T} : \mathcal{A}ssn \to (\Gamma \to (\Delta \to \Upsilon)).$$

Again we must face the problem of dealing with "undefined" values. As already mentioned, simply defining $\mathcal{T}[\![\chi]\!]\gamma\delta$ to be **"false"** whenever some expressions within χ turns out to have a bottom value is not appropriate. For instance, assume we want to express that some process S cannot terminate unless some value v has been sent along channel c and is stored in x. Using the predicate $\top$, denoting a terminated computation, we can write this formally as:

$$\top \to \Big(|h|\{c\}| \neq 0 \ \wedge \ x = val(h|\{c\}[1])\Big)$$

Obviously this assertion should be interpreted as **"true"** for any nonterminated computation $(s_0,h,\perp)$, despite the fact that x will have a bottom value in this case.

Since we are not eager to introduce the complexity of a three-valued logic, we have adopted the following solution. All *atomic* assertions occurring in $\mathcal{A}ssn$ are interpreted *strictly* with respect to bottom values of expressions, that is, they have the truthvalue **false** as soon as they contain some expression that has a bottom value.

Definition 4.6 (Interpretation of assertions)

$$\begin{array}{lll}
\mathcal{T}[\![e_1 = e_2]\!]\gamma\delta & \text{iff} & \mathcal{E}[\![e_1]\!]\gamma\delta = \mathcal{E}[\![e_2]\!]\gamma\delta \neq \perp \\
\mathcal{T}[\![ie_1 = ie_2]\!]\gamma\delta & \text{iff} & \mathcal{T}\mathcal{E}[\![ie_1]\!]\gamma\delta = \mathcal{T}\mathcal{E}[\![ie_2]\!]\gamma\delta \\
\mathcal{T}[\![ce_1 = ce_2]\!]\gamma\delta & \text{iff} & \mathcal{C}\mathcal{E}[\![ce_1]\!]\gamma\delta = \mathcal{C}\mathcal{E}[\![ce_2]\!]\gamma\delta \neq \perp \\
\mathcal{T}[\![te_1 = te_2]\!]\gamma\delta & \text{iff} & \mathcal{T}\mathcal{E}[\![te_1]\!]\gamma\delta = \mathcal{T}\mathcal{E}[\![te_2]\!]\gamma\delta
\end{array}$$

$\mathcal{T}[\![\top]\!]\gamma(s_0,h,s)$ iff $s \neq \bot$
$\mathcal{T}[\![\top^\circ]\!]\gamma(s_0,h,s)$ iff $s_0 \neq \bot$
$\mathcal{T}[\![\chi_1 \wedge \chi_2]\!]\gamma\delta$ iff $\mathcal{T}[\![\chi_1]\!]\gamma\delta$ and $\mathcal{T}[\![\chi_2]\!]\gamma\delta$
$\mathcal{T}[\![\neg\chi]\!]\gamma\delta$ iff not $\mathcal{T}[\![\chi]\!]\gamma\delta$
$\mathcal{T}[\![\exists v(\chi)]\!]\gamma\delta$ iff there is some $w \in \mathcal{V}al$ such that $\mathcal{T}[\![\chi]\!]\gamma[w/v]\delta$
$\mathcal{T}[\![\exists n(\chi)]\!]\gamma\delta$ iff there is some $m \in \mathbf{N}$ such that $\mathcal{T}[\![\chi]\!]\gamma[m/n]\delta$
$\mathcal{T}[\![\exists t(\chi)]\!]\gamma\delta$ iff there is some $u \in \mathcal{T}race$ such that $\mathcal{T}[\![\chi]\!]\gamma[u/t]\delta$

□

Note that quantifiers range over *proper*, i.e. non bottom, values.

We often use the following abbreviations:

- $ie_1 \leq ie_2$ for $\exists n(ie_1 + n = ie_2)$.
- $te_1 \leq te_2$ for $\exists t(te_1 t = te_2)$.
- a channelname c *used as trace expression*, for $h|\{c\}$.
- $last(te)$ for $te[\![te]\!]$.
- $\bot$ for $\neg\top$.
- $e_1 \neq e_2$ for $\neg(e_1 = e_2)$.
- $\chi_1 \vee \chi_2$ for $\neg(\neg\chi_1 \wedge \neg\chi_2)$.
- $\chi_1 \rightarrow \chi_2$ for $\neg(\chi_1 \wedge \neg\chi_2)$.
- $\forall v(\chi)$ for $\neg\exists v(\neg\chi)$.
- $\forall n(\chi)$ for $\neg\exists n(\neg\chi)$.
- $\forall t(\chi)$ for $\neg\exists t(\neg\chi)$.
- $\forall x(\chi)$ for $\forall v(\chi[v/x])$, where $v \notin FV(\chi)$.
- $\forall h(\chi)$ for $\forall t(\chi[t/h])$, where $t \notin FV(\chi)$.

Remark Defined predicates, such as $\neq$, need not be strict! E.g., for a nonterminated computation δ, an assertion such as $x \neq x$ turns out to be true.

The abbreviations above are "syntax directed" in the sense that the text denoted by an abbreviation does not depend on, nor affects, the textual context in which the abbreviation is placed. This is not the case for the following more complicated abbreviational mechanisms:

- $rest(te)$is often used as a trace expression denoting all communications of te but the last one. That is, $te = rest(te)\hat{}last(te)$. Our language has no trace expression that equals $rest(te)$, but any *assertion* χ containing the abbreviation $rest(te)$ is equivalent to the assertion:

  $$\forall t(t\hat{}last(te) = te \rightarrow \chi'),$$

  where t is some fresh logical trace variable, and where χ' is obtained from χ by substituting t for all occurrences of $rest(te)$.

- In a similar way one can rewrite an assertion containing a trace expression of the form $< (c, e) >$, where e is some *arbitrary* expression, into an equivalent assertion that uses an expression of the form $< (c, v) >$ instead.

We have similar interpretations for $\mathcal{A}ssn(\Sigma)$ etc.:

$$\mathcal{T} : \mathcal{A}ssn(\Sigma) \rightarrow (\Gamma \rightarrow (\Sigma \rightarrow \Upsilon))$$

$$\mathcal{T} : \mathcal{A}ssn(\mathcal{T}race) \rightarrow (\Gamma \rightarrow (\mathcal{T}race \rightarrow \Upsilon))$$

$$\mathcal{T} : \mathcal{A}ssn(State) \rightarrow (\Gamma \rightarrow (State_{\perp} \rightarrow \Upsilon))$$

For $\varphi \in \mathcal{A}ssn(\Sigma), I \in \mathcal{A}ssn(\mathcal{T}race), \xi \in \mathcal{A}ssn(State)$, we define:

$\mathcal{T}[\![\varphi]\!]\gamma(h, s) \stackrel{\text{def}}{=} \mathcal{T}[\![\varphi]\!]\gamma(s_0, h, s)$, where $s_0 \in State$ is arbitrarily chosen.

$\mathcal{T}[\![I]\!]\gamma h \stackrel{\text{def}}{=} \mathcal{T}[\![I]\!]\gamma(s_0, h, s)$, where $s_0, s \in State$ are arbitrarily chosen.

$\mathcal{T}[\![\xi]\!]\gamma s \stackrel{\text{def}}{=} \mathcal{T}[\![\xi]\!]\gamma(s_0, h, s)$, where $s_0 \in State$, $h \in \mathcal{T}race$ are arbitrarily chosen.

On many occasions it is more appropriate to interpret an assertion as a *set of computations:*

Definition 4.7 (Interpretation of assertions as sets)

$$[\![\cdot]\!]_{\Delta} : \mathcal{A}ssn \rightarrow (\Gamma \rightarrow \mathcal{P}(\Delta)),$$

$$[\![\cdot]\!]_{\Sigma} : \mathcal{A}ssn(\Sigma) \rightarrow (\Gamma \rightarrow \mathcal{P}(\Sigma)),$$

defined as follows:

$$[\![\chi]\!]_{\Delta}\gamma \stackrel{\text{def}}{=} \{\delta \in \Delta \mid \mathcal{T}[\![\chi]\!]\gamma\delta\} \text{ and}$$

$$[\![\varphi]\!]_{\Sigma}\gamma \stackrel{\text{def}}{=} \{\sigma \in \Sigma \mid \mathcal{T}[\![\varphi]\!]\gamma\sigma\}$$

□

We omit the Δ or Σ subscripts if these are clear from the context.

4.4 Assertions in normal form

A simple theorem of predicate logic states that if some variable x does not occur free in some formula χ, then the truthvalue of χ does not depend on the value of x. As far as assignable variables and logical variables are concerned we have a similar result for our assertion language. The communication history h needs a more refined theorem, however. It is not so interesting to know whether χ depends on h itself or not. Rather we must be able to determine which *projections* of h χ depends on. To this end we introduce a certain *normal form* for assertions.

Definition 4.8 (normal form and history channels of assertions)

- A trace expression is in normal form iff:
  - (i) Each occurrence of h is contained within an expression of the form $h|\mathbf{c}$. (Possibly with $\mathbf{c}$ equal to $Chan$).
  - (ii) No occurrence of h is within the scope of a projection operator other than required by (i).
- An *assertion* χ is in normal form if all its trace expressions are.
- For χ in normal form we define the set $hchan(\chi)$ as the union of all sets of channels used for the projections as required by (i). We call this the set of *history channels* of χ.

□

Remark

In $[ZRE]$, only trace expressions in normal form are allowed. There, the sub expressions as required by (i) are denoted as $\pi_{\mathbf{c}}$.

□

Next we list algebraic rules for trace expressions that allow one to rewrite any assertion into (some) normal form. By no means we need *all* these laws to obtain normal forms; but the list is of interest in itself. A normal form is achieved by "moving projections inside" and replacing occurrences of h not within the scope of any projection by the term $h|Chan$. Laws (vi), (vii), (viii), (ix), (x) and (xi) are sufficient to perform this transformation.

Algebraic laws for trace expressions

Concatenation laws:

(i) $te\,\varepsilon = \varepsilon\,te = te$

(ii) $(te_1te_2)te_3 = te_1(te_2te_3)$

Projection laws:

(iii) $\varepsilon|\mathbf{c} = \varepsilon$

(iv) $< (c, v) > |\mathbf{c} =< (c, v) >$ if $c \in \mathbf{c}$

(v) $< (c, v) > |\mathbf{c} = \varepsilon$ if $c \notin \mathbf{c}$

(vi) $(te_1te_2)|\mathbf{c} = (te_1|\mathbf{c})(te_2|\mathbf{c})$

(vii) $te|\mathbf{c}'|\mathbf{c} = te|(\mathbf{c} \cap \mathbf{c}')$

(viii) $(te[d/c])|\mathbf{c} = (te[d/c])|(\mathbf{c} - \{c\})$ provided $d \not\equiv c$

(ix) $(te[d/c])|\mathbf{c} = (te|\mathbf{c})[d/c]$ if $c, d \in \mathbf{c}$

(x) $(te[d/c])|\mathbf{c} = (te|\mathbf{c})$ if $c, d \notin \mathbf{c}$

(xi) $te|Chan = te$

(xii) $te|\emptyset = \varepsilon$

Relabeling laws:

(xiii) $\varepsilon[d/c] = \varepsilon$

(xiv) $< (c, v) > [d/c] =< (d, v) >$

(xv) $< (c', v) > [d/c] =< (c', v) >$ if $c \not\equiv c'$

(xvi) $(te_1te_2)[d/c] = (te_1[d/c])(te_2[d/c])$

(xvii) $(te|\mathbf{c})[d/c] = (te[d/c])|(\mathbf{c} \cup \{d\})$ if $c \in \mathbf{c}$

(xviii) $(te|\mathbf{c})[d/c] = (te|\mathbf{c})$ if $c \notin \mathbf{c}$

(xix) $te[c/c] = te$

(xx) $te[d/c][d'/c] = te[d/c]$ if $c \not\equiv d$

(xxi) $te[d/c][d'/d] = te[d'/c][d'/d]$

(xxii) $te[d/c][d'/c'] = te[d'/c'][d/c]$ if $c' \not\equiv c$, $c' \not\equiv d$, $d' \not\equiv c$

In the presence of (viii), we could have chosen instead of (ix) and (x) the following two laws:

(ix') $te[d/c]|\mathbf{c} = te|(\mathbf{c} \cup \{c\})[d/c]$ provided $d \in \mathbf{c}$

(x') $te[d/c]|\mathbf{c} = te|(\mathbf{c} - \{c\})$ provided $d \notin \mathbf{c}$

Although normal forms are not unique, we claim the following.

Lemma 4.9

If te_1 and te_2 are both normal forms of te, then $hchan(te_1) = hchan(te_2)$.

Proof

Assume that $hchan(te_1) \neq hchan(te_2)$, say $c \in hchan(te_1)$, but $c \notin hchan(te_2)$. If $< (c,0) >^m$ denotes a sequence of m communications, then by a straightforward proof with induction on the structure of te_1 and te_2, it is shown that $\lambda m.\mathcal{TE}[\![te_1]\!]\gamma(s_0, < (c,0) >^m, s)$ is, for all γ, s_0 and s, strict monotone increasing, while $\lambda m.\mathcal{TE}[\![te_2]\!]\gamma(s_0, < (c,0) >^m, s)$ is a constant function. This contradicts the fact that te_1 and te_2 have been obtained by rewriting the same trace expression te, so that $te_1 = te = te_2$.
By reduction ad absurdum we conclude that $hchan(te_1) = hchan(te_2)$.

□

Definition 4.10 (history channels of assertions)

For *arbitrary* assertions χ we now define the set $hchan(\chi)$ as $hchan(\chi_N)$, where χ_N is some normal form of χ. (The previous lemma guarantees that this definition is unambiguous.)

□

In practice, one needs not to rewrite assertions into normal form to determine their history channels. In fact it is sufficient to ensure, via rewriting using the laws above, that there are no intervening renaming operations between any projection operation and occurrences of h within the scope of that projection operation. For provided that χ satisfies this restriction, one can calculate $hchan(\chi)$ by means of the following table.

	$FV(\cdot)$	$tvar(\cdot)$	$hchan(\cdot)$
h	$\{h\}$	$\emptyset$	$Chan$
t	$\{t\}$	$\{t\}$	$\emptyset$
$te\|\mathbf{c}$	$FV(te)$	$tvar(te)$	$hchan(te) \cap \mathbf{c}$,
$te_1 te_2$	$FV(te_1 te_2)$	$tvar(te_1, te_2)$	$hchan(te_1, te_2)$
$\exists t(\chi)$	$FV(\chi) - \{t\}$	$tvar(\chi) - \{t\}$	$hchan(\chi)$
$te[d/c]$	$FV(te)$	$tvar(te)$	$hchan(te)$

The other cases follow as usual by induction on the structure of χ. Note that channelnames, occurring in χ, but not used in references to h do *not* count for $hchan(\chi)$. E.g. we have $hchan(h|\{c,d\}) = \{c,d\}$, but, on the other hand, $hchan(t|\{c,d\}) = \emptyset$.

Note that channel *renaming* applied to some trace expression te does *not* influence the history channels of the expression. An example might clarify this. Consider the trace expressions $te_1 \stackrel{\text{def}}{=} h|\{c\}$ and $te_2 \stackrel{\text{def}}{=} (h|\{c\})[d/c]$. Of course it is true that te_1 denotes a sequence consisting exclusively of c communications and te_2 denotes a sequence of d communications. But if we interpret the expressions for some computation (s_0, h, s), then *in both cases*, the result is determined by the c communications of the history (only). It is this dependence on the communication history, rather than the channels occurring in the final result of interpreting a trace expression te, that is formalized by the set $hchan(te)$. Indeed $hchan(te_1) = hchan(te_2) = \{c\}$.

Note also that $te_2|\{d\} = te_2$, and so $hchan(te_2|\{d\})$ must equal $hchan(te_2)$ rather than $hchan(te_2) \cap \{d\}$. This shows that the third clause of the table is incorrect for assertions that do not satisfy the restriction.

We define the syntactic counterpart of the semantic assertion base, introduced in chapter 3.

Definition 4.11 (Assertion bases)

For an assertion χ the (syntactic) assertion base is defined as:

$$abase(\chi) \stackrel{\text{def}}{=} (hchan(\chi), var(\chi)).$$

□

Note that the assertion base $abase(\chi)$ does *not* account for variables in $var^\circ(\chi)$, that is, in general $abase(\chi) \neq (hchan(\chi), var^{(\circ)}(\chi))$.

Finally we can state the main lemma of this section, about the independence of assertions χ from variables and trace projections not occurring (free) in χ. The corollaries of this lemma form the basis for proof rules for parallel composition, channel hiding and variable hiding, as we shall introduce in the next chapter. The lemma itself is used again in chapter 6, to show the appropriateness of the characteristic assertion for sequential composition.

Lemma 4.12

Let χ be some assertion with:

$$var(\chi) \subseteq \{\bar{x}\},\ var^\circ(\chi) \subseteq \{\bar{y}^\circ\},\ hchan(\chi) \subseteq \mathbf{c},\ gvar(\chi) \subseteq \{\bar{g}\}.$$

Assume $\gamma, \gamma' \in \Gamma, s_0, s_0', s, s' \in State_\perp, h, h' \in Trace$ are such that:

- s_0 and s_0' agree on $var^\circ(\chi)$, that is, either $s_0 = s_0' = \perp$ or $s_0(\bar{y}) = s_0'(\bar{y})$,
- s and s' agree on $var(\chi)$, that is, either $s = s' = \perp$ or $s(\bar{x}) = s'(\bar{x})$,

- $h|\mathbf{c} = h'|\mathbf{c}$, and
- $\gamma(\bar{g}) = \gamma'(\bar{g})$.

Then $\mathcal{T}[\![\chi]\!]\gamma(s_0, h, s)$ iff $\mathcal{T}[\![\chi]\!]\gamma'(s_0', h', s')$.

Proof First, rewrite all trace expressions within χ into normal form. Then each occurrence of h in χ, is within a projection of the form $h|\mathbf{c}'$, where $\mathbf{c}' \subseteq \mathbf{c}$. Clearly:

$$\mathcal{T}\mathcal{E}[\![h|\mathbf{c}']\!]\gamma h = h|\mathbf{c}' = h'|\mathbf{c}' = \mathcal{T}\mathcal{E}[\![h|\mathbf{c}']\!]\gamma h'$$

Similarly, we have for any $x, y^\circ, g \in FV(\chi)$ that:

$$\mathcal{E}[\![x]\!]\gamma(s_0, h, s) = \bot = \mathcal{E}[\![x]\!]\gamma'(s_0', h', s'), \text{ if } s = s' = \bot,$$

$$\mathcal{E}[\![x]\!]\gamma(s_0, h, s) = s(x) = s'(x) = \mathcal{E}[\![x]\!]\gamma'(s_0', h', s'), \text{ if } s, s' \neq \bot$$

$$\mathcal{E}[\![y^\circ]\!]\gamma(s_0, h, s) = \bot = \mathcal{E}[\![y]\!]\gamma'(s_0', h', s'), \text{ if } s_0 = s_0' = \bot,$$

$$\mathcal{E}[\![y^\circ]\!]\gamma(s_0, h, s) = s_0(y) = s_0'(y) = \mathcal{E}[\![y]\!]\gamma'(s_0', h', s'), \text{ if } s_0, s_0' \neq \bot,$$

$$\mathcal{E}[\![g]\!]\gamma(s_0, h, s) = \gamma(g) = \gamma'(g) = \mathcal{E}[\![g]\!]\gamma'(s_0', h', s').$$

From these equalities, it follows by a straightforward induction on the structure of expressions that all expressions , occurring in atomic formulae of χ get the same values for both cases. Since atomic assertions are either equalities between expressions, or $\top$ or $\top^\circ$ predicates, it follows that all such atomic assertions within χ have the same truthvalue for both cases. Consequently, χ itself must have the same truthvalue for both cases.

□

Corollary 4.13

Let $abase(\chi) \subseteq \beta$. Then, for any $\gamma \in \Gamma, \rho \in \mathcal{P}(\Delta)$:

$$\rho \subseteq [\![\chi]\!]\gamma \text{ iff } \rho|\beta \subseteq [\![\chi]\!]\gamma.$$

Proof

Let $\beta = (\mathbf{c}, \{\bar{x}\})$ and take some arbitrary computation $\delta = (s_0, h, s) \in \rho$. Then if $s \neq \bot$, we have that $\delta|\beta = (s_0, h|\mathbf{c}, s_0|\bar{x} : s(\bar{x}))$, and else $\delta|\beta = (s_0, h|\mathbf{c}, \bot)$.

We apply the lemma, where we take $s_0 = s_0', s' = s_0|\bar{x} : s(\bar{x})$, and $h' = h|\mathbf{c}$. Clearly, $h|\mathbf{c} = (h|\mathbf{c})|\mathbf{c}$ and if $s \neq \bot$ then $s(\bar{x}) = (s_0|\bar{x} : s(\bar{x}))(\bar{x})$.

Hence, by the lemma:

$$\mathcal{T}[\![\chi]\!]\gamma(s_0, h, s) \text{ iff } \mathcal{T}[\![\chi]\!]\gamma(s_0, h|\mathbf{c}, s_0|\bar{x} : s(\bar{x}))$$

That is:

$$\delta \in [\![\chi]\!]\gamma \text{ iff } \delta|\mathbf{c}|\mathbf{x} \in [\![\chi]\!]\gamma,$$

(as was to be shown).

□

Corollary 4.14

$$abase([\![\chi]\!]\gamma) \subseteq abase(\chi).$$

□

Proof

Let $\rho = [\![\chi]\!]\gamma$ and let $\beta = abase(\chi)$. Since $abase(\rho)$ is defined as the smallest *base* β' *such* that $\rho \uparrow \beta' = \rho$, it suffices to show that $\rho \uparrow \beta = \rho$.

$\rho \uparrow \beta$ *is defined as* the largest ρ' such that $\rho'|\beta = \rho|\beta$. Therefore, to prove that this set *actually* equals ρ, we must show that

(a) $\rho|\beta = \rho|\beta$, which *is trivial*, and

(b) If $\rho'|\beta = \rho|\beta$, then $\rho' \subseteq \rho$.

To prove that (b) is satisfied we apply corollary 4.13 twice:

1. Since clearly $\rho \subseteq \rho$, we see that $\rho|\beta \subseteq \rho$, where we use the given conditions on $abase(\chi)$ to apply the previous corollary.
2. Now assume that $\rho'|\beta = \rho|\beta$. Then, from point 1, $\rho'|\beta \subseteq \rho$. Again using the previous corollary, we see that $\rho' \subseteq \rho$.

□

4.5 Validity and proper validity

In our semantic definition we have chosen for a *uniform* treatment of finished and unfinished computations. The assertion language reflects this approach by admitting assertions that express properties of both finished and unfinished computations. This property of the assertion language is vital for two of the proof systems that we formulate in the next chapter. The first of these two, the so called SAT system, has been set up so as to mimic the observation semantics, and therefore it is only natural to deal explicitly with bottom and top predicates in this case. The second system we have called the Hoare system, since many of the proof rules resemble, or even are of

identical form, as the rules of "classical" Hoare style systems for sequential programs. Nevertheless it should be said that these nice looking rules have been obtained at the price of a more complicated assertion language. Therefore a third proof system is presented, the so called Invariant system, for a new type of correctness formulae. This system differs from the Hoare system in that finished and unfinished compuatations are treated separately, i.e. in a non uniform fashion. As a consequence, assertions for the correctness formulae of the Invariant system are simpler, since they express properties of either finished computations or of communication histories. Formulae of the Invariant system can be regarded as formulae of the Hoare system of a special, standardized form. It is the task of this section to introduce the concepts and notations that allow us to discuss the relationship between the two systems. Besides this relationship there is the related problem how to deal with the concepts of validity and universal closure of assertions in the presence of bottom and top predicates. We distinguish between *strict validity* and *proper validity* of assertions. Informally speaking, strict validity means "true for *all* possible computations", and proper validity means "true for all possible *finished* computations". When no ambiguity arises we say "valid" instead of the somewhat cumbersome "strictly valid" or "properly valid". In practice this means that valid stands for strictly valid except for the Invariant system where it stands for properly valid.

Definition 4.15 (Validity of assertions)

- χ is ***strictly valid*** if: $\forall\gamma\forall\delta \in \Delta : \mathcal{T}[\![\chi]\!]\gamma\delta$.
- An assertion χ is ***properly valid*** if:
  $\forall\gamma\forall(s_0,t,s) \in \Delta, s_0 \neq \bot, s \neq \bot : \mathcal{T}[\![\chi]\!]\gamma(s_0,t,s)$.

□

For "classical" first order predicate logics, the *validity* of some assertion χ boils down to the *truth* of the universal closure of χ. We would like to introduce a generalized version of the universal closure operation, with similar properties. To be precise, we want to define the *universal closure* $\underline{\forall}(\chi)$ as an assertion that is true if and only if χ is properly valid. Similarly, the *strict universal closure* $\forall_\bot(\chi)$ should be an assertion that is true if and only if χ is strictly valid.

In analogy with predicate logic, one first attempts to define $\underline{\forall}(\chi)$ by means of putting universal quantifiers in front of χ, one for each free variable of χ. This however does not take care of the – state dependent – bottom and top predicates. To state the problem more clearly: an assertion that

is closed with respect to free variables does *not* denote a constant truth value, since the truth value of bottom and top predicates still depends on the computation. Therefore, let us first state more precisely what is meant by a "closed" assertion in this context.

Definition 4.16 (Closedness of assertions)

- Closedness of some assertion χ with respect to assignable variables or ghost variables is defined as usual for (many sorted) predicate logic. Here we distinguish between $\mathcal{V}ar$ closedness, meaning that $var(\chi) = \emptyset$, and $\mathcal{V}ar^\circ$ closedness, meaning that $var^\circ(\chi) = \emptyset$,
- χ is closed with respect to final states if it is $\mathcal{V}ar$ closed and moreover does not contain the $\bot$ or $\top$ predicates. Analogously it is closed with respect to initial states if it is $\mathcal{V}ar^\circ$ closed and does not contain the $\bot^\circ$ and $\top^\circ$ predicates.
- An assertion is closed if does contain neither free variables nor any of the $\bot, \bot^\circ, \top$ or $\top^\circ$ predicates.

□

It will be clear that the universal closure operation that we try to define must remove free variables as well as all bottom and top predicates. Whereas free variables can be bound by means of universal quantifiers, we can get rid of top and bottom predicates by means of operations called bottom substitution and top substitution. The idea is that any assertion χ can be rewritten into a (finite) conjunction of assertions obtained from χ by such substitutions. Essentially, a bottom substitution $\chi[\bot]$ is some assertion not containing bottom or top predicates that is equivalent to χ as far as *nonfinished* computations are considered. Also, we define $\chi[\top]$ as an assertion not containing bottom or top predicates that is equivalent to χ for *finished* computations. There are similar operations for removing the versions for initial states of the bottom and top predicates. The assertion $\chi[\bot^\circ]$ for instance is an assertion not containing $\bot^\circ$ or $\top^\circ$ that is equivalent to χ for computations starting in a bottom state. Since there is only one such computation, viz. the triple $(\bot, \varepsilon, \bot)$, the operation includes also substitutions for the history h and for atomic predicates referring to the *final* state.

Definition 4.17 (Bottom and top substitutions)

Let χ be some assertion. The assertions: $\chi[\bot], \chi[\bot^\circ], \chi[\top]$ and $\chi[\top^\circ]$ are obtained from χ as follows:

- $\chi[\bot]$ is obtained by replacing atomic subformulae containing $\mathcal{V}ar$ names, and occurrences of $\top$, by **false**.

- $x[\bot^\circ]$ is obtained by replacing atomic subformulae containing $\mathcal{V}ar$ and/or $\mathcal{V}ar^\circ$ names, and occurrences of $\top, \top^\circ$, by **false**. Finally, ε is substituted for h.
- $\chi[\top]$ is obtained by replacing occurrences of $\top$ by **true**.
- $\chi[\top^\circ]$ is similar, except that occurrences of $\top^\circ$ instead of $\top$ are replaced by **true**.

□

There is an accompanying lemma that states that the operations do have the intended effect as explained above:

Lemma 4.18

Let $(s_0, h, s) \in \Delta$

(a) $\mathcal{T}[\![\chi[\bot]]\!]\gamma(s_0, h, s)$ iff $\mathcal{T}[\![\chi]\!]\gamma(s_0, h, \bot)$

(b) If $s \neq \bot$ then: $\mathcal{T}[\![\chi[\top]]\!]\gamma(s_0, h, s)$ iff $\mathcal{T}[\![\chi]\!]\gamma(s_0, h, s)$

(c) $\mathcal{T}[\![\chi[\bot^\circ]]\!]\gamma(s_0, h, s)$ iff $\mathcal{T}[\![\chi]\!]\gamma(\bot, \varepsilon, \bot)$

(d) If $s_0 \neq \bot$ then: $\mathcal{T}[\![\chi[\top^\circ]]\!]\gamma(s_0, h, s)$ iff $\mathcal{T}[\![\chi]\!]\gamma(s_0, h, s)$

(e) $\bot^\circ \rightarrow (h = \varepsilon \wedge \bot)$ is strictly valid.

□

Proof

For cases (a)..(d) the proofs are by means of induction on the structure of assertions, and are fairly straightforward. It suffices to check that, for the right hand and left hand sides, all expressions obtain the same value, and that the $\top$ and $\top^\circ$ have the same truthvalue. For the $\top$ and $\top^\circ$ predicates this is immediately clear from the definitions. For expressions the equality follows essentially from the strictness of our interpretation.

For example, consider case (a). An expression e either does not contain $\mathcal{V}ar$ names , in which case its value does not depend on the final state component, or it *does* contain such names. In the last case, for the right hand side the value of the $\mathcal{V}ar$ names in e is $\bot$, and by the strictness of all operations, the value of e itself is also $\bot$. The atomic assertion containing e is then interpreted as **false**. By the definition of $\chi[\bot]$, the atomic assertion containing e is for the left hand side replaced by the assertion **false**, that is, it has the same truthvalue.

Property (e) follows directly from the structure of the domain Δ. Note that if we would have defined Δ as $\mathcal{S}tate_\bot \times \mathcal{T}race \times \mathcal{S}tate_\bot$, rather than

as $(State \times Trace \times State_\perp) \cup \{(\perp, \varepsilon, \perp)\}$, property (e) would not have been universally valid. Of course, it would have been a property satisfied by any process, and so, (e) would have been present in the form of an extra axiom within the proof systems of chapter 5.

□

At last we can define the universal closure operations:

Definition 4.19 (Universal closures)

- If $var(\chi) = \{\bar{x}\} = \{x_1, \cdots, x_n\}$ and $\bar{v}$ is a list, of the same length as $\bar{x}$, of fresh logical variables, then the *proper* universal closure with respect to final states is the assertion:

  $$\underline{\forall} x\,(\chi) \stackrel{\text{def}}{=} \forall \bar{v}(\chi[\top][\bar{v}/\bar{x}]),$$

- The *strict* universal closure with respect to final states is:

  $$\forall_\perp x\,(\chi) \stackrel{\text{def}}{=} \underline{\forall} x\,(\chi) \wedge \chi[\perp]$$

- With similar notation the proper and strict version of universal closure with respect to initial states are:

  $$\underline{\forall} x^\circ\,(\chi) \stackrel{\text{def}}{=} \forall \bar{v}(\chi[\top^\circ][\bar{v}/\bar{x}^\circ]),$$

  $$\forall_\perp x^\circ\,(\chi) \stackrel{\text{def}}{=} \underline{\forall} x^\circ\,(\chi) \wedge \chi[\perp^\circ]$$

- Universal closures with respect to the three types of ghost variables are defined as usual for (many sorted) predicate logic. Note that there is no need for bottom or top substitution in these cases. We denote these closures by: $\underline{\forall} v\,(\chi), \underline{\forall} n\,(\chi)$ and $\underline{\forall} t\,(\chi)$.

  The universal closure with respect to *all* ghost variables together is denoted by $\underline{\forall} g\,(\chi)$.

- Finally, the proper and strict version of the *universal closure* of χ are:

  $$\underline{\forall}\,(\chi) \stackrel{\text{def}}{=} \underline{\forall} v\, \underline{\forall} n\, \underline{\forall} t\, \forall h\, \underline{\forall} x\, \underline{\forall} x^\circ\,(\chi)$$

  $$\forall_\perp\,(\chi) \stackrel{\text{def}}{=} \underline{\forall} v\, \underline{\forall} n\, \underline{\forall} t\, \forall h\, \forall_\perp x\, \forall_\perp x^\circ\,(\chi)$$

□

We remark that in the formula defining $\forall_\perp\,(\chi)$ the order of the quantifiers is important: the quantification $\forall_\perp x^\circ$ should occur only after the quantifiers for h and x. The reason is that $\forall_\perp x^\circ$ includes substitutions for h, $\top$ and $\mathcal{V}ar$ variables. For actual calculations and proofs a somewhat simpler formula for $\forall_\perp\,(\chi)$ is offered in the lemma below.

One might think that we should have introduced a new quantifier ranging over $\mathcal{V}al_\perp$ rather than $\mathcal{V}al$. This however would not have allowed us to express the strict closure operations in a more direct way since for e.g. $\forall_\perp x\,(\chi)$, either *all* x denote the bottom value or *none* does, excluding intermediate situations. This is not expressed by means of a sequence of quantifiers ranging over $\mathcal{V}al_\perp$.

Lemma 4.20 (properties of universal closures)

(a) $\underline{\forall} x\,(\chi)$ and $\forall_\perp x\,(\chi)$ are closed with respect to final states.
For $s_0 \neq \perp$:

$\mathcal{T}[\![\underline{\forall} x\,(\chi)]\!]\gamma(s_0,h,s)$ iff $\forall s \in \mathit{State} : \mathcal{T}[\![\chi]\!]\gamma(s_0,h,s)$,

$\mathcal{T}[\![\forall_\perp x\,(\chi)]\!]\gamma(s_0,h,s)$ iff $\forall s \in \mathit{State}_\perp : \mathcal{T}[\![\chi]\!]\gamma(s_0,h,s)$.

(b) $\underline{\forall} x^\circ\,(\chi)$ and $\forall_\perp x^\circ\,(\chi)$ are closed with respect to initial states, and:

$\mathcal{T}[\![\underline{\forall} x^\circ\,(\chi)]\!]\gamma(s_0,h,s)$ iff $\forall s_0 \in \mathit{State} : \mathcal{T}[\![\chi]\!]\gamma(s_0,h,s)$,

$\mathcal{T}[\![\forall_\perp x^\circ\,(\chi)]\!]\gamma(s_0,h,s)$ iff $\mathcal{T}[\![\underline{\forall} x^\circ\,(\chi)]\!]\gamma(s_0,h,s)$ and $\mathcal{T}[\![\chi]\!]\gamma(\perp,\varepsilon,\perp)$.

(c) $\underline{\forall}(\chi)$ and $\forall_\perp\,(\chi)$ are closed assertions and moreover:

χ is properly valid iff $\underline{\forall}\,(\chi)$ is true,

χ is strictly valid iff $\forall_\perp\,(\chi)$ is true.

(d) $\forall_\perp\,(\chi)$ is equivalent to:

$\forall_\perp\,(\chi) \stackrel{\mathbf{def}}{=} \underline{\forall}(\chi) \wedge \underline{\forall}[\chi[\perp]] \wedge \underline{\forall}[\chi[\perp^\circ]]$.

□

The proofs are directly from the definitions and lemma 4.18.

Item (d) of the last lemma indicates that a proof of (strict) validity of some assertion χ in general splits into a threefold case analysis, where the cases correspond to a bottom initial state, an unfinished computation and a finished computation. Such case analyses are less cumbersome if the assertion is of the following form:

$$(\perp^\circ \rightarrow \chi_1) \wedge ((\top^\circ \wedge \perp) \rightarrow \chi_2) \wedge (\top \rightarrow \chi_3),$$

where none of χ_1, χ_2 or χ_3 contains bottom or top predicates, χ_1 does not refer to $\mathcal{V}ar$ or $\mathcal{V}ar^\circ$ variables, and χ_2 does not refer to $\mathcal{V}ar$ variables.

For the validity of χ then boils down to the *proper* validity of χ_1, χ_2 and χ_3.

We call assertions that do not contain bottom or top predicates *proper* assertions. Since this type of assertions forms the basis for the Invariant system, we introduce some notation.

Definition 4.21 (Proper assertions)

The class of *proper* assertions is defined as:

$(r \in)\ \mathcal{A}ssn_p \stackrel{\text{def}}{=}$
$\{r \in \mathcal{A}ssn \mid r \text{ does not contain the } \bot^\circ, \bot, \top^\circ \text{ or the } \top \text{ predicate}\}$

$(p, q, r \in)\mathcal{A}ssn_p(\Sigma) \stackrel{\text{def}}{=} \{p \in \mathcal{A}ssn_p \mid var^\circ(p) = \emptyset\}$

Remark. We do not define $\mathcal{A}ssn_p(\mathcal{T}race)$ as this class would coincide with $\mathcal{A}ssn(\mathcal{T}race)$.

□

The next lemma shows that any assertion can be factorized into three conjuncts as indicated above. A similar factorization into disjuncts is also provided. An important special case is the factorization of assertions φ from $\mathcal{A}ssn(\Sigma)$ since this factorization forms the basis for the formulae of the Invariant system.

Lemma 4.22 (Factorization of assertions)

Let $\chi \in \mathcal{A}ssn$ and $\varphi \in \mathcal{A}ssn(\Sigma)$.
Let $\chi_1 = \chi[\bot^\circ]$, $\chi_2 = \chi[\top^\circ][\bot]$, $\chi_3 = \chi[\top]$, $p = \varphi[\top]$ and $I = \varphi[\bot]$.

1. All of χ_1, χ_2, χ_3, p and I are *proper* assertions.
2. I is even closed with respect to initial and final states, that is $I \in \mathcal{A}ssn(\mathcal{T}race)$.
3. The factorization into conjuncts is:

   $\chi \iff (\bot^\circ \rightarrow \chi_1) \wedge ((\top^\circ \wedge \bot) \rightarrow \chi_2) \wedge (\top \rightarrow \chi_3)$,
4. and the factorization into disjuncts is:

   $\chi \iff (\bot^\circ \wedge \chi_1) \vee ((\top^\circ \wedge \bot) \wedge \chi_2) \vee (\top \wedge \chi_3)$,
5. $\varphi \iff (\bot \rightarrow I) \wedge (\top \rightarrow p)$ (factorization in conjuncts)
6. $\varphi \iff (\bot \wedge I) \vee (\top \wedge p)$ (factorization in disjuncts)

□

Proof

The proofs are all very similar. We give the proof of case 6, as an example.

Let $(h, s) \in \Sigma$ and $\gamma \in \Gamma$ be arbitrary. Then:

$(h, s) \in [\![\varphi]\!]\gamma$ iff

$s = \bot$ and $(h, \bot) \in [\![\varphi]\!]\gamma$ or $s \neq \bot$ and $(h, s) \in [\![\varphi]\!]\gamma$

iff

$(h, s) \in [\![\bot]\!]\gamma$ and $(h, s) \in [\![\varphi[\bot]]\!]\gamma$ or $(h, s) \in [\![\top]\!]\gamma$ and $(h, s) \in [\![\varphi[\top]]\!]\gamma$

iff

$(h, s) \in [\![(\bot \wedge \varphi[\bot]) \vee (\top \wedge \varphi[\top])]\!]\gamma$,

as was to be shown.

□

Assertions of the form $\top \rightarrow \varphi$, $\bot \rightarrow \varphi$, $\top \wedge \varphi$ and $\bot \wedge \varphi$, where $\varphi \in \mathcal{A}ssn(\Sigma)$ shall play an important role. We list some useful properties of these. As before, let $p = \varphi[\top]$ and $I = \varphi[\bot]$

1. $(\top \rightarrow \varphi)[\bot] \iff$ **true** (anti-strictness w.r.t. $\bot$)
   $(\top \wedge \varphi)[\bot] \iff$ **false** (strictness w.r.t. $\bot$)
   $(\bot \rightarrow \varphi)[\top] \iff$ **true** ($\top$-strictness)
   $(\bot \wedge \varphi)[\top] \iff$ **false** ($\top$-antistrictness.)

2. $\top \rightarrow \varphi \iff \top \rightarrow p$
   $\top \wedge \varphi \iff \top \wedge p$
   $\bot \rightarrow \varphi \iff \bot \rightarrow I$
   $\bot \wedge \varphi \iff \bot \wedge I$

3. $(\top \rightarrow p) \rightarrow (\top \rightarrow q) \iff \top \rightarrow (p \rightarrow q)$
   $(\top \rightarrow p) \wedge (\top \rightarrow q) \iff \top \rightarrow (p \wedge q)$
   $(\top \rightarrow p) \vee (\top \rightarrow q) \iff \top \rightarrow (p \vee q)$
   $\neg(\top \rightarrow p) \Rightarrow \top \rightarrow \neg p$
   N.B. $\neg(\top \rightarrow p) \not\Longleftrightarrow \top \rightarrow \neg p$

4. $\forall_\bot x\,[\top \rightarrow p] \iff \underline{\forall} x\,[p]$
   $\exists_\bot x\,[\top \rightarrow p] \iff \underline{\exists} x\,[p]$

5. $\top \rightarrow p$ is strictly valid iff p is properly valid.

4.6 Mixed terms

As already motivated in chapter 3, we have set up our definitions of the semantics of processes and assertions such that a formula of the form $S \subseteq \chi$ can be used to express that some process S satisfies certain safety properties, viz. those formalized by the assertion χ. More specificly, we have taken

care that although processes and assertions belong to different syntactical categories, both denote sets of computations. Therefore a formula $S \subseteq \chi$ makes sense since it can be interpreted as $Obs[\![S]\!]\eta \subseteq [\![\chi]\!]\gamma$. Notwithstanding these facts we shall introduce yet another syntactic category, viz. that of *mixed terms*, that incorporates both processes and assertions, and moreover is closed under the **TNP** operations, such as for instance sequential or parallel composition. One reason for doing this is that it allows a more uniform treatment of several definitions and proofs, especially when we come to the completeness results of chapter 6. For instance, we need counterparts of the **TNP** operations for combining assertions anyhow. Now we prove in chapter 6 that one can always represent assertions containing these counterparts within the already given assertion language $\mathcal{A}ssn$ i.e., without such **TNP** operations. Although this shows that it is *inessential* to have these **TNP** operations for assertions, it is conceptually much clearer to allow them nevertheless. A similar reason for introducing mixed terms is that in chapter 6, we seek for each closed process S a corresponding characteristic assertion χ_S, such that not only $S \subseteq \chi_S$ holds, but even $S = \chi_S$ is the case. It turns out that no such assertion does exist. However, we *can* find an assertion $A(S)$ such that $S = A(S)|base(S)$, where "$|$" denotes the projection operator. Although the term $A(S)|base(S)$ is not an assertion, it *is* one of our mixed terms.

Whereas we have indicated certain *technical* reasons for introducing mixed terms, there is also the advantage that certain methodologies for program development can be studied in an elegant way. For instance, if some process S is to be constructed from a given specification χ then according to one strategy we want to transform χ *gradually* into S. Here mixed terms arise in the intermediate stages of the development of S.

Finally we remark that the idea of mixed terms was much inspired by [O], who considers mixed terms for the language TCSP. Olderog names Dijkstra [Dij] and Wirth [Wi] as the inventors of the original idea of mixing programs with specifications. Also, Back [Ba] and Hehner [He] are referred to as the first ones who formalized this idea for sequential and concurrent programs, respectively.

Definition 4.23 (Syntax of mixed terms)

For any set $\mathcal{A}$ of atomic terms, with typical elements a, we define $\mathbf{TNP}(\mathcal{A})$ as follows.

$m \in \mathbf{TNP}(\mathcal{A})$

$$m ::= a \mid Z \mid X_\beta \mid m_1\backslash \mathbf{c} \mid m_1\backslash \mathbf{x} \mid m_1\langle d/c\rangle \mid Kern(m_1) \mid$$

$m_1 ; m_2 \mid m_1 \textbf{ or } m_2 \mid m_1 \beta_1 \| \beta_2 m_2 \mid$

$X_\beta = m_1 \textbf{ in } m_2 \mid \mu X_\beta.m_1 \mid \mu_z X_\beta.m_1$

□

The classes of atomic terms that we are interested in are the following:

Definition 4.24 (Syntax of atomic processes and of predicative terms)

- $\alpha \in \mathcal{A}tom$

  $\alpha ::= \textbf{skip} \mid \textbf{abort} \mid x := e \mid b \mid c.x!b$

- For assertions $\chi \in \mathcal{A}ssn$ and bases $\beta \in \mathcal{B}ase$ the class of *predicative terms* $\mathcal{P}red$ is:

  $\pi \in \mathcal{P}red$

  $\pi ::= \chi | \beta$ (the vertical bar denoting projection)

  If β is finite, then we call the specification *finitely based.*

- We abbreviate $\mathbf{TNP}(\mathcal{A}tom \cup \mathcal{P}red)$ as $\mathcal{M}ixed$ and call its elements *mixed terms.* If all predicative terms of some mixed term are finitely based we call the mixed term itself finitely based. (Atomic *processes* α are always finitely based).

□

We identify the assertion χ with the mixed term $\chi|(Chan, \mathcal{V}ar)$. That is, we regard $\mathcal{A}ssn$ as a subset of $\mathcal{M}ixed$. In the same way, **TNP** processes are included, but here we must be carefull. For assertions are mixed terms and the assertion **false** is interpreted as an *empty set* of computations. This indicates that the appropriate domain of interpretation for mixed terms is the *complete* set $\mathcal{P}(\Delta)$ rather than for instance the set of all *non empty* sets of computations. As discussed extensively in chapter 3 we run into troubles with the recursion construct for this domain, since the least solution for $\mu X_\beta.X_\beta$ is the empty set. There we also argued that this is not appropriate for the recursion construct in **TNP**. Therefore, our language of mixed terms includes two recursion constructs:

- $\mu X_\beta.m_1$, interpreted as the least solution of the equation $X_\beta = m_1(X_\beta)$,
- $\mu_z X_\beta.m_1$, interpreted as $\mu X_\beta.(m_1 \textbf{ or } \mathbf{Z})$.

We identify the **TNP** term $\mu_z P_\beta.S_0$ with the mixed term $\mu_z P_\beta.S_0$. If we compare our approach with the mixed terms as studied in [O], we see that

a slightly different solution has been adopted for the problem. In our terminology we can rephrase it as follows: An assertion χ is interpreted as equivalent to χ **or** Z. To be precise, Olderog [O] deals with traces only, not with state transformers, and so the set Z does not occur within his theory. However, the set $\{\varepsilon\}$ plays the same role in [O] as our set Z, and it is actually this singleton set that is always included in the traceset denoted by some predicate. Consequently, the domain of interpretation can be restricted to non empty (trace) sets, and on this restricted domain the difference between recursion constructs in the style as for our mixed terms vanishes. The disadvantage of this approach is the unnatural interpretation of specifications. For instance, the fact that "**false**" and "$h = \varepsilon$" are equivalent assertions is somewhat surprising, and so although one would expect that *no* process satisfies the specification **false**, any non communicating and non terminating process does.

We prefer to include $\mathbf{Z}$ only for *recursive* processes, where it is justified on the basis of the intuitive operational semantics.

There is a second difference between **TNP** processes and mixed terms in that the former always denote *prefix closed* sets, whereas the set denoted by assertions, and so mixed terms, need not prefix closed. A typical example is the assertion

$$\chi \stackrel{\text{def}}{=} (h = \varepsilon \vee h =< (c,0)(c,0) >).$$

For this reason we introduced the kernel operation into our language. It is assured that $\mathit{Kern}(m)$ is prefix closed since the kernel operation transforms the set denoted by m into the largest prefix closed set contained within this set. The kernel operation plays an important role when it comes to the question of *modular completeness*. The idea is that if X_β is regarded as a "black box" process that is to be replaced by some actual implementation, then we already know that this implementation will denote a prefix closed set. This is similar to the information provided by the base β of the black box: any legal implementation has a base that is included in β. Now to be able to *use* this implicit knowledge in a formal proof one needs *proof rules* to do so. One of these is the *invariance axiom* that expresses the invariance of all channels and assignable variables outside β. The fact that X_β denotes a *process* rather than some arbitrary mixed term can be expresssed by the equality $X_\beta = \mathit{Kern}(X_\beta)$. If it has been *specified* that X_β satisfies some assertion χ, then this equality can be used, within a formal proof that is, to infer that X_β actually satisfies $\mathit{Kern}(\chi)$.

Example If X_β satisfies the assertion

$$\chi \stackrel{\text{def}}{=} (h = \varepsilon \vee h =< (c,0)(c,0) >),$$

and $X_\beta = Kern(X_\beta)$, then it can be proven that X_β satisfies the assertion $(h = \varepsilon)$. The so called *substitutivity rule* that enables the proof of this fact is introduced in section 4.14.

□

The definition of the semantics of mixed terms is almost the same as for **TNP** processes. In section 3.8 we defined the function:

$$Obs : \mathbf{TNP} \to [H \to \mathcal{P}(\Delta)].$$

We extend this to a function defined for mixed terms. Unlike **TNP** processes, mixed terms can contain free logical variables. Consequently we define our new version of *Obs* with an extra argument γ that determines the meaning of such variables. So the new version has the following functionality:

$$Obs : \mathcal{M}ixed \to (\Gamma \to [H \to \mathcal{P}(\Delta)]).$$

The semantic functions $\mathcal{M}$ and $\mathcal{M}^\dagger$ are adapted in the same way.

For atomic processes α we define:

$$Obs[\![\alpha]\!]\gamma\eta = Obs[\![\alpha]\!]\eta.$$

Here the function *Obs* on the right hand side is the one defined for **TNP** processes, and the the function on the left hand side is the "new" one.

For predicative terms $\chi|\beta$, we derive their semantics from the semantics for assertions.

$$Obs[\![\chi|\beta]\!]\gamma\eta = ([\![\chi]\!]\gamma)|\beta.$$

The semantics of operations is the same as for **TNP**, except for the new recursion construct:

Definition 4.25 (Semantics of $\mathbf{TNP}(\mathcal{A})$)

$$Obs[\![X_\beta]\!]\gamma\eta = \eta(X_\beta)$$

$$Obs[\![m_1\backslash \mathbf{x}]\!]\gamma\eta = (Obs[\![m_1]\!]\gamma\eta)\backslash \mathbf{x}$$

$$Obs[\![m_1\backslash \mathbf{c}]\!]\gamma\eta = (Obs[\![m_1]\!]\gamma\eta)\backslash \mathbf{c}$$

$$Obs[\![m_1\langle d/c\rangle]\!]\gamma\eta = (Obs[\![m_1]\!]\gamma\eta)[d/c]$$

$$Obs[\![m_1 ; m_2]\!]\gamma\eta = Obs[\![m_1]\!]\gamma\eta \mathbin{\hat{\circ}} Obs[\![m_2]\!]\gamma\eta$$

$$Obs[\![m_1 \textbf{ or } m_2]\!]\gamma\eta = Obs[\![m_1]\!]\gamma\eta \cup Obs[\![m_2]\!]\gamma\eta$$

$$Obs[\![m_1 \beta_1 \|_{\beta_2} m_2]\!]\gamma\eta = Obs[\![m_1]\!]\gamma\eta \beta_1 \|_{\beta_2} Obs[\![m_2]\!]\gamma\eta$$

$$Obs[\![\mu X_\beta.m_1]\!]\gamma\eta = \mu\big(\lambda\rho_\beta.Obs[\![m_1]\!]\gamma\eta[\rho_\beta/X_\beta]\big)$$

$$Obs[\![X_\beta = m_1 \textbf{in } m_2]\!]\gamma\eta = Obs[\![m_2]\!](\gamma)(\eta[Obs[\![m_1]\!]\gamma\eta/X_\beta])$$

□

The semantics of the μ_z recursion construct is defined via the abbreviation defined above.

It is seen that for **TNP** processes, viewed as mixed term, we have essentially the *same* semantics, that is:

$$Obs[\![S]\!]\gamma\eta = Obs[\![S]\!]\eta,$$

where again the right hand side function is the semantic function for **TNP**, and the left hand side function is the semantic function for mixed terms.

4.7 Correctness formulae

The overall goal of chapter 4 is the introduction of formulae that are used to *specify* processes. These are called "correctness formulae", or simply "formulae".

Definition 4.26 (Syntax of formulae)

Let $\chi \in \mathit{Assn}, \varphi, \psi \in \mathit{Assn}(\Sigma), m, m_1, m_2 \in \mathit{Mixed}$ and let g denote a ghost variable.

$(f \in)\mathcal{F}orm$ — *Formulae* or correctness formulae.

$$f ::= \chi \mid m_1 \subseteq m_2 \mid (\varphi)m(\psi) \mid \forall g(f) \mid$$

$$f_1 \vee f_2 \mid f_1 \wedge f_2 \mid f_1 \rightarrow f_2 \mid \forall X_\beta(f)$$

□

We are mainly interested in those $m_1 \subseteq m_2$ formulae where the second mixed term is an predicative term χ. We use the notation m_1 *sat* χ for such formulae, and call them "SAT formulae". The SAT system of chapter 5 is in fact an axiomatization of SAT formulae.

Similarly, $(\varphi)\ m\ (\psi)$ is called a "Hoare triple", and the Hoare system of chapter 5 forms the corresponding axiomatization.

The Invariant system of chapter 5 is based on formulae that can be seen as Hoare formulae with a special, standardized, form. This is explained in

section 4.8, and so we do not pay special attention to Invariant formulae anymore in this section.

We freely use other logical connectives than the ones listed in the definition. As is custom, these are formally treated as abbreviations. Also we use $m_1 = m_2$ as an abbreviation for $m_1 \subseteq m_2 \wedge m_2 \subseteq m_1$.

The free logical variables, called ghost variables, and the free process variables of a formula are defined as follows:

	$gvar(f)$	$pvar(f)$
χ	$gvar(f)$	$\emptyset$
$m_1 \subseteq m_2$	$gvar(m_1, m_2)$	$pvar(m_1, m_2)$
$(\varphi)m(\psi)$	$gvar(\varphi, \psi, m)$	$pvar(m)$
$\forall g(f)$	$gvar(f) - \{g\}$	$pvar(f)$
$f_1 \vee f_2$	$gvar(f_1, f_2)$	$pvar(f_1, f_2)$
$f_1 \wedge f_2$	$gvar(f_1, f_2)$	$pvar(f_1, f_2)$
$f_1 \rightarrow f_2$	$gvar(f_1, f_2)$	$pvar(f_1, f_2)$
$\forall X_\beta(f)$	$gvar(f)$	$pvar(f) - \{X_\beta\}$

Correctness formulae do never contain free the variable h denoting the communication history, neither do they contain free *assignable* variables. The (implicit) binding of the assignable variables and the history h is not that surprising, as this is also the case for conventional Hoare logic. But the fact that ghost variables are *not* implicitly bound needs some explanation, especially since in [ZRE] we *did* rely on such an implicit quantification.

Examine the following proof outline.

$\{x = v\}$
$x := x - 1\,;$
$\{x = v - 1\}$
X_β
$\{x = v - 1 \wedge y = (v-1)!\}$
$x := x + 1\,;\ y := y \cdot x$
$\{x = v \wedge y = v!\}$

If we consider a formulae such as $\{x = v\}\ x := x - 1\ \{x = v - 1\}$ as a specification on its own, then clearly the intention is that the formula should be true for *any* value of v, which suggests that we interpret such a formula as follows:

$$\forall v \Big(\{x = v\}\ x := x - 1\ \{x = v - 1\}\Big).$$

Indeed such an universal quantification is introduced, implicitly, when we say that the formula $\{x = v\}\ x := x - 1\ \{x = v - 1\}$ is *valid*, which informally means that the formula is true *for all environments*. (The notion of validity is defined in section 4.10). However, the *proof outline* above is intuitively understood as:

"From:

$\{x = v\}\ x := x - 1\ \{x = v - 1\}$,
$\{x = v - 1\}\ X_\beta\ \{x = v - 1 \wedge y = (v - 1)!\}$ and
$\{x = v - 1 \wedge y = (v - 1)!\}\ x := x + 1\ ;\ y := y \cdot x\ \{x = v \wedge y = v!\}$

it follows that:

$\{x = v\}\ x := x - 1\ X_\beta; x := x + 1; y := y \cdot x\ \{x = v \wedge y = v!\}$,"

rather than:

"From:

$\forall v\big(\{x = v\}\ x := x - 1\ \{x = v - 1\}\big)$,
$\forall v\big(\{x = v - 1\}\ X_\beta\ \{x = v - 1 \wedge y = (v - 1)!\}\big)$ and
$\forall v\big(\{x = v - 1 \wedge y = (v - 1)!\}\ x := x + 1\ ;\ y := y \cdot x\ \{x = v \wedge y = v!\}\big)$

it follows that:

$\forall v\big(\{x = v\}\ x := x - 1; X_\beta; x := x + 1; y := y \cdot x\ \{x = v \wedge y = v!\}\big)$."

We do not claim that the latter proof is *invalid*; only that it cannot be seen as an accurate formalization of intuitive reasoning. For intuitively, as soon as we consider the proof as a whole, all occurrences of the "v" in the outline above are *meant* to denote one and the same logical value, throughout the whole proof outline.

Note that this is *not* the case for assignable variables: occurrences of x in different assertions also denote different values.

We decided that no *implicit* quantification for ghost variables should be used for the interpretation of correctness formulae, but that rather the formula language must provide the means to write such quantifications explicitly. As a result, many definitions and proofs became more natural. For example, when implicit quantification over ghost variables is used, the soundness proof of the Hoare rule for sequential composition reveals that implicit applications of the logical rules for removal and introduction of universal quantifiers are hidden in the Hoare rule. Such complications are not present in our approach.

Since the only free variables of correctness formulae are process variables and ghost variables, it is natural that the semantics can be given as follows:

Definition 4.27 (Semantics of formulae)

Let H denote the domain of process environments, as defined in chapter 3. Then the semantic function

$$\mathcal{T} : \mathit{Form} \rightarrow (H \rightarrow (\Gamma \rightarrow \Upsilon))$$

is defined as follows:

$\mathcal{T}[\![\chi]\!]\gamma\eta$ iff for all $\delta \in \Delta$: $\mathcal{T}[\![\chi]\!]\gamma\delta$

$\mathcal{T}[\![m_1 \subseteq m_2]\!]\gamma\eta$ iff $\mathit{Obs}[\![m_1]\!]\gamma\eta \subseteq \mathit{Obs}[\![m_2]\!]\gamma\eta$

$\mathcal{T}[\![(\varphi)m(\psi)]\!]\gamma\eta$ iff $M^{\uparrow}[\![m]\!]\gamma\eta([\![\varphi]\!]_{\Sigma}\gamma) \subseteq [\![\psi]\!]_{\Sigma}\gamma$

$\mathcal{T}[\![f_1 \vee f_2]\!]\gamma\eta$ iff $\mathcal{T}[\![f_1]\!]\gamma\eta$ or $\mathcal{T}[\![f_2]\!]\gamma\eta$

$\mathcal{T}[\![f_1 \wedge f_2]\!]\gamma\eta$ iff $\mathcal{T}[\![f_1]\!]\gamma\eta$ and $\mathcal{T}[\![f_2]\!]\gamma\eta$

$\mathcal{T}[\![f_1 \rightarrow f_2]\!]\gamma\eta$ iff $\mathcal{T}[\![f_1]\!]\gamma\eta$ implies $\mathcal{T}[\![f_2]\!]\gamma\eta$

$\mathcal{T}[\![\forall X_\beta(f)]\!]\gamma\eta$ iff for all $\rho_\beta \in \mathcal{P}(\Delta_\beta)$: $\mathcal{T}[\![f]\!](\eta[\rho_\beta/X_\beta])\gamma$

$\mathcal{T}[\![\forall g(f)]\!]\gamma\eta$ iff $\mathcal{T}[\![f]\!](\gamma[v/g])\eta$

for all values v of the domain of interpretation for g.

□

4.8 Substitution in correctness formulae

For $f \in \mathit{Form}, \xi \in \mathit{Pvar}$ and $m \in \mathbf{TNP}$, the substitution $f[m/\xi]$ is defined provided that $\mathit{base}(m) \subseteq \mathit{base}(\xi)$. It is defined as usual for predicate logics, except for the following: **TNP** terms can contain bindings of ξ in the form of a recursion or process naming construct. Therefore, the following cases deserve special attention:

(a) $(\mu\xi.m_1)[m/\xi] = \mu\xi.m_1$

(b) $(\mu\varsigma.m_1)[m/\xi] = \mu\varsigma.(m_1[m/\xi])$, provided that $\varsigma \notin \mathit{pvar}(m)$

(c) $(\xi = m_1 \textbf{ in } m_2)[m/\xi] = (\xi = m_1 \textbf{ in } m_2)$

(d) $(\varsigma = m_1 \textbf{ in } m_2)[m/\xi] = (\varsigma = m_1[m/\xi] \textbf{ in } m_2[m/\xi])$,
provided that $\varsigma \notin \mathit{pvar}(m)$

(e) $\forall\xi(f)[m/\xi] = \forall\xi(f)$

(f) $\forall\varsigma(f)[m/\xi] = \forall\varsigma(f[m/\xi])$,
provided $\varsigma \notin pvar(f)$.

(g) If, in case (b),(d) or (f), $\varsigma \in pvar(m)$, then the substitution is applied after renaming ς into θ, for some fresh θ, as is usual for such "name clashes".

Lemma 4.28

$$\mathcal{T}[\![f[m/\xi]]\!]\gamma\eta = \mathcal{T}[\![f]\!](\gamma)(\eta[\mathcal{O}bs[\![m]\!]\gamma\eta/\xi])$$

Proof

The proof is by means of induction on the structure of correctness formulae, and mixed terms. It is very much like similar proofs of substitution lemmata, for instance as in [Bak]. Therefore, we omit the details here.

□

4.9 Predicate transformers

We define a weakest precondition and a strongest postcondition operator. It is custom to define the weakest precondition and strongest postcondition of some assertion with respect to some *program*. However, we define these conditions with respect to some *assertion* $\chi \in \mathcal{A}ssn$. This means that we treat assertions from $\mathcal{A}ssn$ as predicate transformers, where in this context a "predicate" is to be understood as an assertion from $\mathcal{A}ssn(\Sigma)$, i.e. a predicate denotes a set of generalized states. The weakest precondition and strongest postcondition are used to transform SAT specifications into Hoare specifications. The transformation of Hoare formulae into SAT specifications is possible too, by means of the so called "leads to" operator "$\leadsto$".

Apart from the three operators mentioned above, we take the opportunity to introduce a few abbreviations and some more operations on assertions. One of these is the (sequential) composition operator "$\hat{\circ}$." This operator forms the basis for the sequential composition rule of the SAT system. A reason for introducing it here is that there is a close relationship with the strongest postcondition operator.

The other operations that we define correspond to the *closure operations*, introduced in chapter 3. Especially the kernel operation is of interest. We remark that in [Widom] the kernel closure is handled by adding extra temporal logic operators to the assertion language. Within our approach this is

not necessary, for the kernel closure of the set denoted by some assertion χ is denotable by some assertion $\mathit{Kern}(\chi)$ that is obtained from χ as described in this section.

The following operations are defined. Let $\chi, \chi_1, \chi_2 \in \mathit{Assn}$, $\varphi, \psi \in \mathit{Assn}(\Sigma)$.

$\chi_1 \mathbin{\hat{\circ}} \chi_2$ $(\in \mathit{Assn})$ composition of χ_1, χ_2.

$\varphi \lhd \chi$ $(\in \mathit{Assn}(\Sigma))$ strongest postcondition for φ with respect to χ.

$\chi \rhd \psi$ $(\in \mathit{Assn}(\Sigma))$ weakest precondition for ψ with respect to χ.

$\varphi \rightsquigarrow \psi$ $(\in \mathit{Assn})$ characteristic formula determined by φ, ψ.

$\mathbf{1_c}$, $\mathbf{1_x}$, $\mathbf{1}_\beta$, $\mathbf{Z_c}$, $\mathbf{Z}$.

$\mathit{Pref}(\chi)$ prefix closure of χ.

$\mathit{Kern}(\chi)$ Kernel of χ.

$\mathit{Kern}(t_0, \psi)$ Kernel of ψ with respect to initial trace t_0.

$\mathit{Close}(\chi)$ Closure of χ. (Not to be confused with the universal closure operations introduced above.)

First we define the semantic operations corresponding to weakest preconditions, strongest postconditions and the leads to operator.

Definition 4.29 (Let $\rho, \rho_1, \rho_2 \in \mathcal{P}(\Delta)$, $\pi, \pi_1, \pi_2 \in \mathcal{P}(\Sigma)$.)

$$\pi \lhd \rho =$$
$$\Big\{(h, s) \in \Sigma \mid \exists t_1, t_2, s_1 \Big(h = t_1 t_2 \wedge (t_1, s_1) \in \pi \wedge (s_1, t_2, s) \in \rho\Big)\Big\}$$
$$\rho \rhd \pi =$$
$$\Big\{(h, s) \in \Sigma \mid \forall t_1, s_1 \Big((s, t_1, s_1) \in \rho \Rightarrow (ht_1, s_1) \in \pi\Big)\Big\}$$
$$\pi_1 \rightsquigarrow \pi_2 =$$
$$\Big\{(s_0, h, s) \in \Delta \mid \forall t_0 \Big((t_0, s_0) \in \pi_1 \Rightarrow (t_0 h, s) \in \pi_2\Big)\Big\}$$

□

Definition 4.30 (Let $\chi, \chi_1, \chi_2 \in \mathit{Assn}, \varphi, \psi \in \mathit{Assn}(\Sigma)$.)

$$[\![\chi_1 \mathbin{\hat{\circ}} \chi_2]\!]\gamma \stackrel{\text{def}}{=} [\![\chi_1]\!]\gamma \mathbin{\hat{\circ}} [\![\chi_2]\!]\gamma$$
$$[\![\varphi \lhd \chi]\!]\gamma \stackrel{\text{def}}{=} [\![\varphi]\!]\gamma \lhd [\![\chi]\!]\gamma$$
$$[\![\chi \rhd \psi]\!]\gamma \stackrel{\text{def}}{=} [\![\chi]\!]\gamma \rhd [\![\psi]\!]\gamma$$
$$[\![\varphi \rightsquigarrow \psi]\!]\gamma \stackrel{\text{def}}{=} [\![\varphi]\!]\gamma \rightsquigarrow [\![\psi]\!]\gamma$$

$$[\![\mathbf{Z_c}]\!]\gamma \stackrel{\text{def}}{=} \mathbf{Z_c} \stackrel{\text{def}}{=} \{(s_0, h, \bot) \in \Delta \mid h|\mathbf{c} = \epsilon\}$$

$$[\![\mathbf{Z}]\!]\gamma \stackrel{\text{def}}{=} \mathbf{Z}$$

$$[\![\mathbf{1_c}]\!]\gamma \stackrel{\text{def}}{=} \{(s_0, h, s) \in \Delta \mid h|\mathbf{c} = \epsilon\}$$

$$[\![\mathbf{1_x}]\!]\gamma \stackrel{\text{def}}{=} \{(s_0, h, \bot), (s_0, h, s) \mid s_0(\bar{x}) = s(\bar{x})\}$$

$$[\![\mathbf{1_{(c,x)}}]\!]\gamma \stackrel{\text{def}}{=} [\![\mathbf{1_c}]\!]\gamma \cap [\![\mathbf{1_x}]\!]\gamma$$

$$[\![\mathit{Pref}(\chi)]\!]\gamma \stackrel{\text{def}}{=} \mathit{Pref}([\![\chi]\!]\gamma)$$

$$[\![\mathit{Kern}(\chi)]\!]\gamma \stackrel{\text{def}}{=} \mathit{Kern}([\![\chi]\!]\gamma)$$

$$[\![\mathit{Close}(\chi)]\!]\gamma \stackrel{\text{def}}{=} \mathbf{Z} \cup \mathit{Pref}([\![\chi]\!]\gamma)$$

□

Lemma 4.31 The constructs defined above can be expressed within our assertion language, as follows:

- $\mathbf{Z_c}$ is expressed by $h|\mathbf{c} = \epsilon \wedge \bot$
  $\mathbf{Z}$ is expressed by $h = \epsilon \wedge \bot$
  $\mathbf{1_c}$ is expressed by $h|\mathbf{c} = \epsilon$
  $\mathbf{1_x}$ is expressed by $\top \rightarrow \bar{x} = \bar{x}^\circ$, where $\{\bar{x}\} = \mathbf{x}$
  $\mathbf{1_{(c,x)}}$ is expressed by $\mathbf{1_c} \wedge \mathbf{1_x} \equiv (h|\mathbf{c} = \epsilon \wedge (\top \rightarrow \bar{x} = \bar{x}^\circ))$

- $\mathit{Pref}(\chi)$ is expressed as follows.

  Let $\bar{x}$ be a list containing all x in $\mathit{var}(\chi)$, and let $\bar{v}$ be a list of fresh logical variables of the same length. Let $\mathbf{c} = \mathit{hchan}(\chi)$, and let t be a fresh logical trace variable. Then $\mathit{Pref}(\chi)$ is expressed by:

  $$\chi \vee \exists t\Big(h|\mathbf{c} \leq t|\mathbf{c} \wedge \bot \wedge (\chi[\bot][t/h] \vee \exists \bar{v}(\chi[\bar{v}/\bar{x},\ t/h]))\Big).$$

- $\mathit{Kern}(\chi)$ is expressed by:

  $$\chi \wedge \forall t(t|\mathbf{c} \leq h|\mathbf{c} \rightarrow \chi[\bot][t/h]),$$

  where $\mathbf{c} = \mathit{hchan}(\chi)$ and where t is a fresh logical trace variable.

- $\mathit{Close}(\chi)$ is expressed by:

  $\mathbf{Z} \vee \mathit{Pref}(\chi)$ and by $\mathit{Pref}(\mathbf{Z} \vee \chi)$

- Let $\bar{x} \in \mathcal{V}ar^*$ be a list containing all $x \in \mathcal{V}ar$ such that $x \in var(\chi_1)$ or $x^\circ \in var^\circ(\chi_2)$, and let $\bar{v} \in \mathcal{L}var^*$ be a list of fresh logical variables of the same length. Let t_1, t_2 be fresh logical trace variables, and let $\mathbf{c}$ denote $hchan(\chi_1, \chi_2)$.

  Then $\chi_1 \hat{\circ} \chi_2$ is expressed by:

  $$\Big[(\chi_1[\bot] \wedge \chi_2[\bot^\circ] \wedge \bot) \quad \vee$$

  $$\top^\circ \wedge \exists t_1 \exists t_2 \exists \bar{v} \Big(h | \mathbf{c} = (t_1 t_2) | \mathbf{c} \wedge \chi_1[\top][t_1/h, \bar{v}/\bar{x}] \wedge \chi_2[\top^\circ][\bar{v}/\bar{x}^\circ, t_2/h]\Big)\Big].$$

- Strongest postcondition.

  Let $\bar{x} \in \mathcal{V}ar^*$ be a list containing all $x \in \mathcal{V}ar$ such that $x \in var(\varphi)$ or $x^\circ \in var^\circ(\chi)$, and let $\bar{v} \in \mathcal{L}var^*$ be a list of fresh logical variables of the same length. Let t_1, t_2 be fresh logical trace variables, and let $\mathbf{c}$ denote $hchan(\varphi, \chi)$.

  Then $\varphi \lhd \chi$ is expressed by:

  $$\Big[(\varphi[\bot] \wedge \chi[\bot^\circ] \wedge \bot) \quad \vee$$

  $$\exists t_1 \exists t_2 \exists \bar{v} \Big(h | \mathbf{c} = (t_1 t_2) | \mathbf{c} \wedge \varphi[\top][t_1/h, \bar{v}/\bar{x}] \wedge \chi[\top^\circ][\bar{v}/\bar{x}^\circ, t_2/h]\Big)\Big].$$

- Weakest precondition.

  $\chi \rhd \psi$ is expressed by:

  $$(\bot \wedge (\chi[\bot^\circ] \rightarrow \psi[\bot])) \quad \vee$$

  $$(\top \wedge (\forall t_1 \forall_\bot x (\chi[\top^\circ][t_1/h] \rightarrow \psi[h t_1/h]))[\bar{x}/\bar{x}^\circ]),$$

  where $\bar{x}$ contains all $x \in \mathcal{V}ar$ such that $x \in var(\chi, \psi)$ or $x^\circ \in var^\circ(\chi)$, and where t_1 is fresh.

- Leads to operator.

  Let $\bar{x}$ contain all x such that $x \in var(\varphi)$ or $x^\circ \in var^\circ(\varphi)$, and let $t_0 \in \mathcal{T}var$ be fresh.
  $\varphi \rightsquigarrow \psi$ is expressed by:

  $$\forall t_0 \Big(\varphi[t_0/h, \bar{x}^\circ/\bar{x}, \top^\circ/\top] \rightarrow \psi[t_0 h/h]\Big).$$

For the Hoare and Invariant systems we need an operation closely related to the kernel operation:

Definition 4.32 (Kernel for $\mathcal{A}ssn(\Sigma)$)

For $\psi \in \mathcal{A}ssn(\Sigma)$ and logical trace variable t_0 we define $\mathcal{K}ern(t_0, \psi)$. Then: Let $\mathbf{c} = hchan(\psi)$.

$Kern(t_0, \psi) \stackrel{\text{def}}{=} \psi \wedge \forall t(t_0|\mathbf{c} \leq t|\mathbf{c} \leq h|\mathbf{c} \rightarrow \psi[\bot][t/h])$.

□

Proof of the lemma.

We consider only a few interesting cases.

• *Kernel*

We prove semantic equality between $Kern(\chi)$ and the representing assertion defined in the lemma.

$[\![Kern(\chi)]\!]\gamma = Kern([\![\chi]\!]\gamma) =$

$\{\delta \in \Delta \mid \forall \delta' \sqsubseteq \delta : \delta' \in [\![\chi]\!]\gamma\} =$

$\{(s_0, h, s) \in [\![\chi]\!]\gamma \mid \forall h'(h' \leq h \rightarrow (s_0, h', \bot) \in [\![\chi]\!]\gamma)\} =$

$\{(s_0, h, s) \in [\![\chi]\!]\gamma \mid \forall h'(h'|\mathbf{c} \leq h|\mathbf{c} \rightarrow (s_0, h', \bot) \in [\![\chi]\!]\gamma)\} =$

$\{(s_0, h, s) \in [\![\chi]\!]\gamma \mid \forall h'(h'|\mathbf{c} \leq h|\mathbf{c} \rightarrow (s_0, h, s) \in [\![\chi[\bot][t/h]]\!](\gamma[h'/t]))\} =$

$\{(s_0, h, s) \in [\![\chi]\!]\gamma \mid \forall h'((s_0, h, s) \in [\![t|\mathbf{c} \leq h|\mathbf{c} \rightarrow \chi[\bot][t/h]]\!](\gamma[h'/t]))\} =$

$[\![\chi \wedge \forall t(t|\mathbf{c} \leq h|\mathbf{c} \rightarrow \chi[\bot][t/h])]\!]\gamma$.

• *Composition*

$[\![\chi_1 \hat{\circ} \chi_2]\!]\gamma =$

$\{(s_0, h, s) \in \Delta \mid \exists h_1, h_2, s_1, \text{ such that } h = h_1 h_2 \wedge$

$(s_0, h_1, s_1) \in [\![\chi_1]\!]\gamma \wedge (s_1, h_2, s) \in [\![\chi_2]\!]\gamma\} =$

$\{(s_0, h, s) \in \Delta \mid s = \bot \wedge (s_0, h, \bot) \in [\![\chi_1]\!]\gamma \wedge (\bot, \varepsilon, \bot) \in [\![\chi_2]\!]\gamma\} \cup$

$\{(s_0, h, s) \in \Delta \mid \exists h_1, h_2, s_1, \text{ such that } h = h_1 h_2 \wedge s_0 \neq \bot \wedge s_1 \neq \bot \wedge$

$(s_0, h_1, s_1) \in [\![\chi_1]\!]\gamma \wedge (s_1, h_2, s) \in [\![\chi_2]\!]\gamma\}$.

Using lemma 4.17 one sees that the first set of this union equals:

$[\![\chi_1[\bot] \wedge \chi_2[\bot^\circ] \wedge \bot]\!]\gamma$.

In the treatment of the second set of the union above, we rely on lemmata 4.11 and 4.17. Assume that (s_0, h, s) is some arbitrary Δ element. The following equivalences hold:

$s_0 \neq \bot \wedge \exists h_1, h_2, s_1 \neq \bot$, such that

$h = h_1 h_2 \wedge (s_0, h_1, s_1) \in [\![\chi_1]\!]\gamma \wedge (s_1, h_2, s) \in [\![\chi_2]\!]\gamma$

iff

$s_0 \neq \bot \wedge \exists h_1, h_2, \bar{w}, s_1 \neq \bot$, such that

$h|\mathbf{c} = (h_1 h_2)|\mathbf{c} \ \wedge s_1(\bar{x}) = \bar{w} \wedge (s_0, h_1, s_1) \in [\![\chi_1]\!]\gamma \wedge (s_1, h_2, s) \in [\![\chi_2]\!]\gamma$

iff

$s_0 \neq \bot \wedge \exists h_1, h_2, \bar{w}$, such that $h|\mathbf{c} = (h_1 h_2)|\mathbf{c} \wedge$

$(s_0, h, s) \in [\![\chi_1[\top][t_1/h, \bar{v}/\bar{x}]]\!]\gamma' \wedge (s_0, h, s) \in [\![\chi_2[\top^\circ][\bar{v}/\bar{x}^\circ, t_2/h]]\!]\gamma'$,

where $\gamma' \stackrel{\text{def}}{=} \gamma[h_1/t_1, h_2/t_2, \bar{w}/\bar{v}]$

iff

$(s_0, h, s) \in$

$[\![\top^\circ \wedge \exists t_1, t_2, \bar{v}\,(h|\mathbf{c} = (t_1 t_2)|\mathbf{c} \wedge \chi_1[\top][t_1/h, \bar{v}/\bar{x}] \wedge \chi_2[\top^\circ][\bar{v}/\bar{x}^\circ, t_2/h])]\!]\gamma$.

Altogether we showed that the the two sets of the union above are expressed by the two disjuncts of the assertion $\chi_1 \hat{\circ} \chi_2$.

• *Strongest postcondition*

Note that the semantic definition of $\varphi \lhd \chi$ is almost the same as for $\varphi \hat{\circ} \chi$, except that the interpretation of the former is by means of a set of trace state pairs rather than as a set of computations. Here we used the fact that φ, although an element of $\mathit{Assn}(\Sigma)$, is *also* an element of Assn, that does not actually refer to the initial state. Thus one sees that:

$(h, s) \in [\![\varphi \lhd \chi]\!]\gamma$ iff $\exists s_0 \neq \bot$ such that $(s_0, h, s) \in [\![\varphi \hat{\circ} \chi]\!]\gamma$.

This explains the formula representing $\varphi \lhd \chi$: It equals $\exists \bar{x}^\circ(\varphi \hat{\circ} \chi)$. Since $\mathit{var}^\circ(\varphi \hat{\circ} \chi) = \emptyset$, this boils down to $(\varphi \hat{\circ} \chi)[\top^\circ]$.

• *Weakest precondition*

$[\![\chi \rhd \psi]\!]\gamma =$

$\{(h, s) \mid \forall h_1, s_1\big((s, h_1, s_1) \in [\![\chi]\!]\gamma \Rightarrow (hh_1, s_1) \in [\![\psi]\!]\gamma\big)\} =$

$\{(h, s) \mid s = \bot \wedge (\bot, \varepsilon, \bot) \in [\![\chi]\!]\gamma \Rightarrow (h, \bot) \in [\![\psi]\!]\gamma\} \ \cup$

$\{(h, s) \mid s \neq \bot \wedge \forall h_1, s_1\big((s, h_1, s_1) \in [\![\chi]\!]\gamma \Rightarrow (hh_1, s_1) \in [\![\psi]\!]\gamma\big)\}$.

Clearly the first set of this union equals:

$[\![\bot \wedge (\chi[\bot^\circ] \rightarrow \psi[\bot])]\!]\gamma$.

We show that the second set equals

$[\![\top \wedge \forall t_1 \forall_\bot x(\chi[\top^\circ][t_1/h] \rightarrow \psi[ht_1/h])[\bar{x}/\bar{x}^\circ]]\!]\gamma$.

Let

$(*) = \forall t_1 \forall_\perp x(\chi[\top^\circ][t_1/h] \rightarrow \psi[ht_1/h])[\bar{x}/\bar{x}^\circ]$

From the definition of strict universal closure it follows that this is the formula:

$\forall t_1(\chi[\top^\circ][\perp][t_1/h][\bar{x}/\bar{x}^\circ] \rightarrow \psi[\perp][ht_1/h]) \wedge$

$\forall t_1 \forall \bar{v}(\chi[\top^\circ][\top][t_1/h][\bar{v}/\bar{x}][\bar{x}/\bar{x}^\circ] \rightarrow \psi[\top][ht_1/h][\bar{v}/\bar{x}])$.

In the following sequence of equivalences we rely on the fact that, although $\chi \in \mathit{Assn}$, it is the case that for instance $\chi[\top^\circ][\top][t_1/h][\bar{v}/\bar{x}][\bar{x}/\bar{x}^\circ]$ is an element of $\mathit{Assn}(\Sigma)$.

$(h, s) \in [\![\top \wedge (*)]\!]\gamma$

iff

$s \neq \perp \wedge$

$\Big[\forall h_1 \Big((h, s) \in [\![\chi[\top^\circ][\perp][t_1/h][\bar{x}/\bar{x}^\circ]]\!](\gamma[h_1/t_1])$

$\Rightarrow\ (h, s) \in [\![\psi[\perp][ht_1/h]]\!](\gamma[h_1/t_1])\Big)\Big] \wedge$

$\Big[\forall h_1 \forall \bar{x} \Big((h, s) \in [\![\chi[\top^\circ][\top][\bar{v}/\bar{x}][t_1/h][\bar{x}/\bar{x}^\circ]]\!](\gamma[h_1/t_1, \bar{w}/\bar{v}])$

$\Rightarrow\ (h, s) \in [\![\psi[\top][\bar{v}/\bar{x}][ht_1/h]]\!](\gamma[h_1/t_1, \bar{w}/\bar{v}])\Big)\Big]$

iff

$s \neq \perp \wedge$

$\forall h_1 \Big((s, h_1, \perp) \in [\![\chi]\!]\gamma \ \Rightarrow\ (hh_1, \perp) \in [\![\psi]\!]\gamma\Big) \ \wedge$

$\forall h_1 \forall s_1 \neq \perp \Big((s, h_1, s_1) \in [\![\chi]\!]\gamma \ \Rightarrow\ (hh_1, s_1) \in [\![\psi]\!]\gamma\Big)$

iff

$s \neq \perp \wedge \forall h_1 \forall s_1 \Big((s, h_1, s_1) \in [\![\chi]\!]\gamma \ \Rightarrow\ (hh_1, s_1) \in [\![\psi]\!]\gamma\Big)$.

This was to be shown.

- *leads to operator*

$[\![\varphi \rightsquigarrow \psi]\!]\gamma \ = \ [\![\varphi]\!]\gamma \rightsquigarrow [\![\psi]\!]\gamma \ =$

$\{(s_0, h, s) \ \mid \ \forall h_0 \Big((h_0, s_0) \in [\![\varphi]\!]\gamma \ \Rightarrow$

$(h_0 h, s) \in [\![\psi]\!]\gamma\Big)\} \ =$

$\{(s_0, h, s) \ \mid \ \forall h_0 \Big((s_0, h, s) \in [\![\varphi[t_0/h, \bar{x}^\circ/\bar{x}, \top^\circ/\top]]\!](\gamma[h_0/t_0]) \ \Rightarrow\ (s_0, h, s) \in$
$[\![\psi[t_0 h/h]]\!](\gamma[h_0/t_0])\Big)\} \ =$

$[\![\forall t_0(\varphi[t_0/h, \bar{x}^\circ/\bar{x}, \top^\circ/\top] \to \psi[t_0 h/h])]\!]\gamma$.

This last set equals the interpretation of the assertion expressing $\varphi \rightsquigarrow \psi$.

□

The operators defined above enjoy a number of simple properties. We list some of these properties.

Lemma 4.33 Properties of semantic predicate transformers

Let $\pi \in \mathcal{P}(\Sigma), \rho \in \mathcal{P}(\Delta), \pi_i \in \mathcal{P}(\Sigma)$ for $i \in I$, and $\rho_i \in \mathcal{P}(\Delta)$ for $i \in I$.

(i) $\rho \rhd \pi$ is antimonotone in ρ, monotone in π.

(ii) $\pi \lhd \rho$ is monotone in both π and ρ.

(iii) $(\cup_i \rho_i) \rhd \pi = \cap_i (\rho_i \rhd \pi)$

(iv) $(\cap_i \rho_i) \rhd \pi \supseteq \cup_i (\rho_i \rhd \pi)$

(v) $\rho \rhd (\cup_i \pi_i) \supseteq \cup_i (\rho \rhd \pi_i)$

(vi) $\rho \rhd (\cap_i \pi_i) = \cap_i (\rho \rhd \pi_i)$

(vii) $\pi \lhd (\cup_i \rho_i) = \cup_i (\pi \lhd \rho_i)$

(viii) $\pi \lhd (\cap_i \rho_i) \subseteq \cap_i (\pi \lhd \rho_i)$

(ix) $(\cup_i \pi_i) \lhd \rho = \cup_i (\pi_i \lhd \rho)$

(x) $(\cap_i \pi_i) \lhd \rho \subseteq \cap_i (\pi_i \lhd \rho)$

(xi) $\pi \subseteq \rho \rhd (\pi \lhd \rho)$

(xii) $(\rho \rhd \pi) \lhd \rho \subseteq \pi$

(xiii) $(\pi \lhd \rho_1) \lhd \rho_2 \subseteq \pi \lhd (\rho_1 \hat{\circ} \rho_2)$

□

The proof of these properties is straightforward from the definitions. To each of the properties above there is a corresponding property for a *syntactic* operator, that follows immediately from the lemma above. We list some of the more useful of those properties.

Lemma 4.34 Properties of syntactic predicate transformers

The following implications and equivalences are (strictly) valid:

(i) $\chi \rhd (\psi_1 \vee \psi_2) \leftarrow (\chi \rhd \psi_1) \vee (\chi \rhd \psi_2)$

(ii) $\chi \rhd (\psi_1 \wedge \psi_2) \leftrightarrow (\chi \rhd \psi_1) \wedge (\chi \rhd \psi_2)$

(iii) $(\varphi_1 \vee \varphi_2) \lhd \chi \leftrightarrow (\varphi_1 \lhd \chi) \vee (\varphi_2 \lhd \chi)$

(iv) $(\varphi_1 \wedge \varphi_2) \lhd \chi \rightarrow (\varphi_1 \lhd \chi) \wedge (\varphi_2 \lhd \chi)$

(v) $\varphi \rightarrow \chi \rhd (\varphi \lhd \chi)$

(vi) $(\chi \rhd \psi) \lhd \chi \rightarrow \psi$

(vii) $(\varphi \lhd \chi_1) \lhd \chi_2 \rightarrow \varphi \lhd (\chi_1 \hat{\circ} \chi_2)$

(viii) $\varphi \lhd (\chi_1 \vee \chi_2) \leftrightarrow (\varphi \lhd \chi_1) \vee (\varphi \lhd \chi_2)$

□

4.10 Natural deduction and correctness formulae

Our goal in the next chapter will be the design of formal proof systems for correctness formulae. The systems are in natural deduction style. Hence, we must define a *satisfaction relation* for formulae, and a notion of *soundness* of proof rules, that suits natural deduction.

Definition 4.35 (Satisfaction relation)

Let $F = \{f_0, \cdots, f_{n-1}\}$ be a finite set of formulae.
Then we define:

- $F \models f$ iff $\forall \eta \in H. \forall \gamma \in \Gamma. \Big(\bigwedge_{f_i \in F} \mathcal{T}[\![f_i]\!]\gamma\eta \Rightarrow \mathcal{T}[\![f]\!]\gamma\eta \Big)$.
- We define *validity* of a formulae f as a special case of this:
  $\models f$ iff $\forall \eta \in H. \forall \gamma \in \Gamma. (\mathcal{T}[\![f]\!]\gamma\eta)$.

□

For a natural deduction system, as opposed to a Hilbert style proof system, *provability* of formulae has the form of a *relation*: $F \vdash f$. (Read: f is provable from the set of hypotheses F)
A formal proof system then, is an inductive definition of the $\vdash$ relation, by means of a number of clauses of the form: "If $F_0 \vdash f_0, \cdots F_{n-1} \vdash f_{n-1}$, then also $F_n \vdash f_n$". This is usually put into the form of a proof rule:

$$\frac{F_0 \vdash f_0, \cdots, F_{n-1} \vdash f_{n-1}}{F_n \vdash f_n}$$

An *axiom* is a rule for which $n = 0$; it is denoted as $F_0 \vdash f_0$.

If we can infer $F \vdash f$ by application of proof rules, we say that $F \vdash f$ is *deducible* or also that f is *provable under the hypotheses* F.

A rule is called *sound* iff

$$\Big(\bigwedge_{i<n} F_i \models f_i\Big) \Rightarrow F_n \models f_n.$$

Clearly, when all (axioms and) rules are sound, the proof system is sound, i.e.:

$$F \vdash f \Rightarrow F \models f$$

Most of our proofrules turn out to be axioms, that is, they are of the form:

$$\{f_0, \ldots, f_{n-1}\} \vdash f_n$$

This is also written as a rule of the form:

$$\frac{f_0, \cdots, f_{n-1}}{f_n}$$

For this latter type of rule, the soundness condition amounts to the following:

$$\forall \eta \in H. \forall \gamma \in \Gamma. \Big(\bigwedge_{i<n} \mathcal{T}[\![f_i]\!]\gamma\eta \Rightarrow \mathcal{T}[\![f_n]\!]\gamma\eta\Big).$$

We will design several proof systems in the next chapter. They differ only as far as the nonlogical rules are concerned. Therefore, we summarize the logical rules here. Essentially, these rules have been taken from [v. Dalen]

4.11 Logical rules

(i) $F \vdash f$ if $f \in F$ (reflexivity)

(ii) $\dfrac{F \vdash f_1, \ldots, F \vdash f_{n-1}\ ,\ \{f_1, \ldots, f_{n-1}\} \vdash f_n}{F \vdash f_n}$ (transitivity)

(iii) $\dfrac{F \vdash f}{F' \vdash f}$ provided $F \subseteq F'$ (weakening)

(iv) $\dfrac{f_1\ ,\ f_2}{f_1 \wedge f_2}$ ($\wedge$-introduction)

(v) $\dfrac{f_1 \wedge f_2}{f_i}$ $(i = 1, 2)$ ($\wedge$-elimination)

(vi) $\dfrac{f_i}{f_1 \vee f_2}$ $(i = 1, 2)$ ($\vee$-introduction)

(vii) $\dfrac{F \cup \{f_1\} \vdash f,\ F \cup \{f_2\} \vdash f,\ F \vdash f_1 \vee f_2}{F \vdash f}$ ($\vee$-elimination)

(viii) $\dfrac{F \cup \{f_1\} \vdash f_2}{F \vdash f_1 \rightarrow f_2}$ ($\rightarrow$-introduction)

(ix) $\dfrac{f_1,\ f_1 \rightarrow f_2}{f_2}$ ($\rightarrow$-elimination)

(x) $\dfrac{F \vdash f}{F \vdash \forall \xi(f)}$ provided $\xi \notin pvar(F)$ ($\forall$-introduction)

$\dfrac{F \vdash f}{F \vdash \forall g(f)}$ provided $g \notin gvar(F)$ ($\forall$-introduction)

(xi) $\dfrac{\forall \xi(f)}{f[m/\xi]}$ ($\forall$-elimination)

(It is understood here that $base(m) \subseteq base(\xi)$, since otherwise the substitution for the process variable ξ is not even defined.)

$\dfrac{\forall g(f)}{f[ge/g]}$ ($\forall$-elimination),

where ge is an expression of the same type as the logical variable g.

□

The soundness of most of these rules is obvious. We give a proof for three cases:

- Rule (viii). This is the analogue of the "deduction theorem" for Hilbert style systems, however, without the restrictions associated with this theorem.

  We must show, for arbitrary $F \stackrel{\text{def}}{=} \{f'_0, \cdots, f'_{n-1}\}$, that:

  $$F \cup \{f_1\} \models f_2 \Rightarrow F \models f_1 \rightarrow f_2,$$

  that is:

  $$\forall \gamma\eta((\bigwedge_{i<n} \mathcal{T}[\![f'_i]\!]\gamma\eta \wedge \mathcal{T}[\![f_1]\!]\gamma\eta) \Rightarrow \mathcal{T}[\![f_2]\!]\gamma\eta) \Rightarrow$$

  $$\forall \gamma\eta(\bigwedge_{i<n} \mathcal{T}[\![f'_i]\!]\gamma\eta \Rightarrow (\mathcal{T}[\![f_1]\!]\gamma\eta \Rightarrow \mathcal{T}[\![f_2]\!]\gamma\eta)).$$

  This is obvious.

- Rule (x) Take some arbitrary $F \stackrel{\text{def}}{=} \{f'_0, \cdots, f'_{n-1}\}$ such that $\xi \notin pvar(f'_i)$ for $i < n$. Assume that $F \models f$, that is:

  $$\forall \gamma\eta(\bigwedge_{i<n} \mathcal{T}[\![f'_i]\!]\gamma\eta \Rightarrow \mathcal{T}[\![f]\!]\gamma\eta).$$

  This implies:

  $$\forall \gamma\eta \forall \rho_\beta \in \mathcal{P}(\Delta_\beta)(\bigwedge_{i<n} \mathcal{T}[\![f'_i]\!]\gamma\eta[\rho_\beta/\xi] \Rightarrow \mathcal{T}[\![f]\!]\gamma\eta[\rho_\beta/\xi]).$$

Since $\xi \notin pvar(f_i')$,

$\mathcal{T}[\![f_i']\!]\gamma\eta[\rho_\beta/\xi] = \mathcal{T}[\![f_i']\!]\gamma\eta, \quad$ for $i < n$.

So we get:

$$\forall\gamma\eta(\bigwedge_{i<n} \mathcal{T}[\![f_i']\!]\gamma\eta \Rightarrow \forall\rho_\beta \in \mathcal{P}(\Delta_\beta) : \mathcal{T}[\![f]\!]\gamma\eta[\rho_\beta/\xi]).$$

That is:

$$\forall\gamma\eta(\bigwedge_{i<n} \mathcal{T}[\![f_i']\!]\gamma\eta \Rightarrow \mathcal{T}[\![\forall\xi(f)]\!]\gamma\eta).$$

So we conclude that $F \models \forall\xi(f)$, as was to be shown.

The corresponding rule for the introduction of universal quantifiers for *logical* variables is proven sound in a completely similar way, except that the role of γ and η is interchanged.

- Rule (xi) For this, substitution lemma 4.28 is needed:

  $\mathcal{T}[\![f[m/\xi]]\!]\gamma\eta = \mathcal{T}[\![f]\!](\gamma)(\eta[Obs[\![m]\!]\gamma\eta/\xi])$.

  We show:

  $\forall\gamma\eta(\mathcal{T}[\![\forall\xi(f)]\!]\gamma\eta \Rightarrow \mathcal{T}[\![f[m/\xi]]\!]\gamma\eta)$.

  So take some arbitrary $\eta \in H$.

  $\mathcal{T}[\![\forall\xi(f)]\!]\gamma\eta$ means: $\forall\rho_\beta \in \mathcal{P}(\Delta_\beta) : \mathcal{T}[\![f]\!]\gamma(\eta[\rho_\beta/\xi])$, where $\beta = base(\xi)$.

  Since $base(m) \subseteq \beta$, we have $Obs[\![m]\!]\gamma\eta \in \mathcal{P}(\Delta_\beta)$. Hence, instantiating ρ_β as $Obs[\![m]\!]\gamma\eta$, we see that $\mathcal{T}[\![f]\!](\gamma)(\eta[Obs[\![m]\!]\gamma\eta/\xi])$ holds. By the lemma, this implies $\mathcal{T}[\![f[m/\xi]]\!]\gamma\eta$.

  Again, the corresponding rule for *logical* variables is proven in a similar way.

□

We give an example, explaining how one can change the name of a bound variable. Let $\forall g(f)$ be some given formula, and let g' be a ghost variable, of the same type as g, that does not occur free in f. We show that $H \vdash \forall g'(f)$ is deducible from $H \vdash \forall g(f)$, even when g' *does* occur free in some of the hypotheses H.

Note that the $\forall$-elimination rule is in fact the following *axiom*:

$\forall g(f) \vdash f[ge/g]$.

If we choose g' for ge we have the following instance:

$$\forall g(f) \vdash f[g'/g].$$

Since g' does not occur free in $\forall g(f)$, we can apply the $\forall$-introduction rule, and obtain:

$$\forall g(f) \vdash \forall g'(f[g'/g]).$$

Now if $H \vdash \forall g(f)$ has been given, then clearly one can combine this with the above, using the transitivity rule. This yields the desired result:

$$H \vdash \forall g'(f[g'/g]).$$

Note that the deduction above is valid even when g' does occur free in H.

4.12 Axioms and rules for (in-) equalities.

$$m \subseteq m \qquad \text{(reflexivity)}$$

$$\frac{m_1 \subseteq m_2 \ , \ m_2 \subseteq m_3}{m_1 \subseteq m_3} \qquad \text{(transitivity)}$$

Remark

$$\frac{m_1 \subseteq m_2 \ , \ m_2 \subseteq m_1}{m_1 = m_2} \qquad \text{(antisymmetry)}$$

is a derived rule, since $m_1 = m_2$ abbreviates $m_1 \subseteq m_2 \wedge m_2 \subseteq m_1$; this latter formula is derivable by means of the $\wedge$-introduction rule.

□

$$\frac{m_1 = m_2 \ , \ f[m_1/P_\beta]}{f[m_2/P_\beta]} \qquad \text{(substitutivity)}$$

(m_1 and m_2 must be substitutable for P_β).

To be able to give corresponding rules for $\subseteq$, we must introduce the notion of positive and negative occurrences of some process variable P_β. To this end, we define, for $f \in \mathit{form}$, the sets f^+ and f^- as follows:

Definition 4.36 (Positive and negative occurrences)

f	f^+	f^-
χ	$\emptyset$	$\emptyset$
$m_1 \subseteq m_2$	$pvar(m_2)$	$pvar(m_1)$
$m\ sat\ \chi$	$\emptyset$	$pvar(m)$
$(\varphi)m(\psi)$	$\emptyset$	$pvar(m)$
$f_1 \wedge f_2$	${f_1}^+ \cup {f_2}^+$	${f_1}^- \cup {f_2}^-$
$f_1 \vee f_2$	${f_1}^+ \cup {f_2}^+$	${f_1}^- \cup {f_2}^-$
$f_1 \rightarrow f_2$	${f_1}^- \cup {f_2}^+$	${f_1}^+ \cup {f_1}^-$
$\forall P_\beta(f)$	$f^+ - \{P_\beta\}$	$f^- - \{P_\beta\}$
$\forall g(f)$	f^+	f^-

We call f monotone in P_β if $P_\beta \notin f^-$, and antimonotone if $P_\beta \notin f^+$. Now we can formulate our rules:

$$\frac{m_1 \subseteq m_2\ ,\ f[m_1/P_\beta]}{f[m_2/P_\beta]}$$ provided f is monotone in P_β.

$$\frac{m_1 \subseteq m_2\ ,\ f[m_2/P_\beta]}{f[m_1/P_\beta]}$$ provided f is antimonotone in P_β.

4.13 Satisfiability

A SAT or Hoare formula $f(X_\beta)$ is an antimonotone formula. We call such a formula *satisfiable* if there exists a **TNP** *process* S such that $f[S/X_\beta]$ is valid. **TNP** processes satisfy the following axiom:

$\mathbf{z} \subseteq S$ (least element).

From this, and the antimonotony rule of the previous section, it follows that $f(X_\beta)$ is satisfiable if and only if $f[\mathbf{z}/X_\beta]$ is valid. A SAT formulae of the form $X_\beta\ sat\ \chi$ is satisfiable iff $\mathbf{z} \subseteq [\![\chi]\!]\gamma$, for all $\gamma \in \Gamma$. This is the case whenever $\chi[\bot][\varepsilon/h]$ is a strictly valid assertion. In this situation, we will also say that χ itself is satisfiable.

□

4.14 The relation between SAT and Hoare formulae

By means of $\rhd, \lhd$ and $\rightsquigarrow$, we can translate SAT formulae into Hoare formulae and vice versa. This is the contents of the following four proofrules

Lemma 4.37 .

(i) $$\frac{m \; sat \; \chi}{(\varphi)\; m\; (\varphi \lhd \chi)} \quad \text{(SP)}$$

(ii) $$\frac{m \; sat \; \chi}{(\chi \rhd \psi)\; m\; (\psi)} \quad \text{(WP)}$$

(iii) $$\frac{(\varphi)\; m\; (\psi)}{m \; sat \; \varphi \rightsquigarrow \psi} \quad \text{(HS)}$$

(iv) $$\frac{m \; sat \; \varphi \rightsquigarrow \psi}{(\varphi)\; m\; (\psi)} \quad \text{(SH)}$$

□

The rules show that $(\varphi)\; m\; (\psi)$ and $m \; sat \; \varphi \rightsquigarrow \psi$ are *equivalent*. In general this is not the case for $m \; sat \; \chi$ and $(\varphi)\; m\; (\varphi \lhd \chi)$ or $(\chi \rhd \psi)\; m\; (\psi)$.

Proof

We prove the soundness of the four rules.

Take some arbitrary $\eta \in H$.

(i) Assume that $Obs[\![m]\!]\gamma\eta \subseteq [\![\chi]\!]\gamma$. To show:

$$M^{\dagger}[\![m]\!]\gamma\eta([\![\varphi]\!]\gamma) \subseteq [\![\varphi \lhd \chi]\!]\gamma.$$

So take some $(h, s) \in M^{\dagger}[\![m]\!]\gamma\eta([\![\varphi]\!]\gamma)$.
Then, there must be some $(t_1, s_1) \in [\![\varphi]\!]\gamma$ and $(s_1, t_2, s) \in Obs[\![m]\!]\gamma\eta$ such that $h = t_1 t_2$.
By the assumption above, $(s_1, t_2, s) \in [\![\chi]\!]\gamma$.
Hence, $(h, s) \in [\![\varphi]\!]\gamma \lhd [\![\chi]\!]\gamma = [\![\varphi \lhd \chi]\!]\gamma$

(ii) Assume again that $Obs[\![m]\!]\gamma\eta \subseteq [\![\chi]\!]\gamma$. To show:

$$M^{\dagger}[\![m]\!]\gamma\eta([\![\chi \rhd \psi]\!]\gamma) \subseteq [\![\psi]\!]\gamma$$

Take some $(h, s) \in M^{\dagger}[\![m]\!]\gamma\eta\big([\![\chi \rhd \psi]\!]\gamma\big)$. Then there are $(t_0, s_0) \in [\![\chi \rhd \psi]\!]\gamma$, $(s_0, t_1, s) \in Obs[\![m]\!]\gamma\eta$ such that $h = t_0 t_1$.
From the premisse of the rule it follows that $(s_0, t_1, s) \in [\![\chi]\!]\gamma$.
But then, by the definition of $\rhd$, $(t_0, t_1, s) \in [\![\psi]\!]\gamma$, that is,
$(h, s) \in [\![\psi]\!]\gamma$.

(iii),(iv) Take some $\eta \in H, \gamma \in \Gamma$. Then we have the following equivalences:

$$\mathcal{M}^{\dagger}[\![m]\!]\gamma\eta([\![\varphi]\!]\gamma) \subseteq [\![\psi]\!]\gamma \text{ iff}$$

$$\forall t_0, s_0, h, s :$$

$$((t_0, s_0) \in [\![\varphi]\!]\gamma \wedge (s_0, h, s) \in \mathcal{O}bs[\![m]\!]\gamma\eta) \Rightarrow (t_0 h, s) \in [\![\psi]\!]\gamma \text{ iff}$$

$$\forall (s_0, h, s) \in \mathcal{O}bs[\![m]\!]\gamma\eta :$$

$$\forall t_0 \Big((t_0, s_0) \in [\![\varphi]\!]\gamma \Rightarrow (t_0 h, s) \in [\![\psi]\!]\gamma\Big) \text{ iff}$$

$$\forall (s_0, h, s) \in \mathcal{O}bs[\![m]\!]\gamma\eta : (s_0, h, s) \in [\![\varphi]\!]\gamma \rightsquigarrow [\![\psi]\!]\gamma \text{ iff}$$

$$\mathcal{O}bs[\![m]\!]\gamma\eta \subseteq [\![\varphi \rightsquigarrow \psi]\!]\gamma.$$

□

4.15 Proper correctness formulae

In [ZRE] an axiomatization is given for specifications that are essentially of the following form:

$I : \{p\}\ S\ \{q\}$,

where S is a process, $I \in \mathcal{A}ssn(\mathcal{T}race)$, $p, q \in \mathcal{A}ssn_p(\Sigma)$.

Within the present context we can define the meaning of this formula as follows:

$$\mathcal{T}[\![I : \{p\}\ S\ \{q\}]\!]\gamma\eta \quad \text{iff}$$

$$\forall t_0 \forall s_0 \neq \perp : \mathcal{T}[\![p]\!](\gamma)(t_0, s_0) \Rightarrow$$

$$\Big(\forall (t_1, s) \in \mathcal{M}[\![S]\!]\eta s_0 : \mathcal{T}[\![I]\!](\gamma)(t_0 \hat{} t_1) \wedge (s \neq \perp \Rightarrow \mathcal{T}[\![q]\!](\gamma)(t_0 \hat{} t_1, s))\Big).$$

This definition can be understand as follows ([ZRE]):

"Let t_0, s_0 be some initial trace and state for which p holds, then:

- I is required to hold for all possible extensions $t_0 \hat{} t_1$ of the initial trace t_0, produced by executing S. Note that, because of prefix closedness of $\mathcal{M}[\![S]\!]\eta s_0$, this amounts to the same as requiring that I holds at all moments during execution of S.
- q is required to hold for the final state and trace, if and when S terminates, since such terminated computations are represented by pairs (t, s) with $s \neq \perp$ in $\mathcal{M}[\![S]\!]\eta s_0$".

We rewrite this definition into a simpler form. Note that:

- $(s_0 \neq \bot \wedge \mathcal{T}[\![p]\!](\gamma)(t_0, s_0)) \Leftrightarrow \mathcal{T}[\![\top \wedge p]\!](\gamma)(t_0, s_0)$,
- $\Big[\mathcal{T}[\![I]\!](\gamma)\, t_0{}^\frown t_1 \wedge \Big(s \neq \bot \Rightarrow \mathcal{T}[\![q]\!](\gamma)(t_0{}^\frown t_1, s)\Big)\Big] \Leftrightarrow$
  $\mathcal{T}[\![I \wedge (\top \rightarrow q)]\!](\gamma)(t_0{}^\frown t_1, s)$,
- $(t_1, s) \in \mathcal{M}[\![S]\!](\eta)(s_0) \Leftrightarrow$
  $(t_1, s) \in \mathcal{M}^\dagger[\![S]\!](\eta)(\{(\varepsilon, s_0)\}) \Leftrightarrow$
  $(t_0{}^\frown t_1, s) \in \mathcal{M}^\dagger[\![S]\!](\eta)(\{(t_0, s_0)\})$.

So, using the complete additivity of $\mathcal{M}^\dagger[\![S]\!]\eta$, we see that:

$\mathcal{T}[\![I : \{p\}\ S\ \{q\}]\!]\gamma\eta$ iff

$\mathcal{M}^\dagger[\![S]\!](\eta)([\![\top \wedge p]\!](\gamma)) \subseteq [\![I \wedge (\top \rightarrow q)]\!](\gamma)$.

But this means that $I : \{p\}\ S\ \{q\}$ is equivalent to:

$(\top \wedge p)\ S\ (I \wedge (\top \rightarrow q))$. $\quad (*)$

That is, the formulae of [ZRE] can be expressed within our formalism!

We shall use $I : \{p\}\ m\ \{q\}$ as an abbreviation of the Hoare formula:

$(\top \wedge p)\ m\ (I \wedge (\top \rightarrow q))$.

We call our new type of correctness formulae *proper formulae*. Note that we, although the motivation for these formulae is based upon a certain intuition about *processes*, we nevertheless allow arbitrary mixed terms in our Invariant formulae.

Although it appears that the expressive power of the formulae of [ZRE] is less than that of our Hoare triples, we now show that this is not the case if we restrict ourselves to **TNP** processes S.

First, we list some proof rules that we will need below. Apart from the bottom closure rule, all of these rules are included in the Hoare system of chapter 5. The bottom closure rule is special, for it is sound only for **TNP** processes and not for arbitrary mixed terms.

$$\frac{(\varphi_1)\ S\ (\psi_1)\ ,\ (\varphi_2)\ S\ (\psi_2)}{(\varphi_1 \wedge \varphi_2)\ S\ (\psi_1 \wedge \psi_2)} \qquad \text{(conjunction)}$$

$$\frac{(\varphi_1)\ S\ (\psi_1)\ ,\ (\varphi_2)\ S\ (\psi_2)}{(\varphi_1 \vee \varphi_2)\ S\ (\psi_1 \vee \psi_2)} \qquad \text{(disjunction)}$$

$$\frac{\forall_\bot(\varphi \rightarrow \varphi'),\ (\varphi')\ S\ (\psi'),\ \forall_\bot(\psi' \rightarrow \psi)}{(\varphi)\ S\ (\psi)} \qquad \text{(consequence)}$$

$(\bot \wedge I)\ S\ (\bot \wedge I)$ where $I \in \mathcal{A}ssn(\mathcal{T}race)$ (strictness)

$$\frac{(\varphi)\ S\ (\bot \rightarrow I)}{(\varphi)\ S\ (I)} \quad \text{where } I \in \mathcal{A}ssn(\mathcal{T}race) \quad \text{(bottom closure)}$$

We shall not prove the soundness of these rules here, since that is the task of chapter 5, except for the bottom closure rule.

Lemma 4.38 (Soundness of the bottom closure rule)

Let $I \in \mathcal{A}ssn(\mathcal{T}race)$, $\eta \in H, \gamma \in \Gamma$. If $\mathcal{T}[\![(\varphi)S(\bot \rightarrow I)]\!]\gamma\eta = \mathbf{true}$, then also $\mathcal{T}[\![(\varphi)S(I)]\!]\gamma\eta = \mathbf{true}$.

□

Proof

$\mathcal{T}[\![(\varphi)S(\bot \rightarrow I)]\!]\gamma\eta = \mathbf{true}$ is interpreted as:

$$\mathcal{M}^{\dagger}[\![S]\!](\eta)([\![\varphi]\!]\gamma) \subseteq [\![\bot \rightarrow I]\!]\gamma.$$

This means that if $(h, s) \in \mathcal{M}^{\dagger}[\![S]\!](\eta)([\![\varphi]\!]\gamma)$ and $s = \bot$, then $h \in [\![I]\!]\gamma$.

We must prove that for (h, s) as indicated except that $s \neq \bot$, it is *also* true that $h \in [\![I]\!]\gamma$.

So assume that $(h, s) \in \mathcal{M}^{\dagger}[\![S]\!](\eta)([\![\varphi]\!]\gamma)$ and that $s \neq \bot$. Then there exists an initial trace state pair (h_0, s_0) such that for some trace h_1, $h = h_0 h_1$ and $(s_0, h_1, s) \in \mathcal{O}bs[\![S]\!]\eta$. Since we assume here that S is a **TNP** process, rather than an arbitrary mixed term, we know from lemma 3.41 that $\mathcal{O}bs[\![S]\!]\eta$ is *prefix closed*. This implies that $(s_0, h_1, \bot) \in \mathcal{O}bs[\![S]\!]\eta$ too,and so, that $(h, \bot) \in \mathcal{M}^{\dagger}[\![S]\!](\eta)([\![\varphi]\!]\gamma)$. But this implies that $h \in [\![I]\!]\gamma$!

We conclude that:

$$\mathcal{M}^{\dagger}[\![S]\!](\eta)([\![\varphi]\!]\gamma) \subseteq [\![I]\!]\gamma,$$

which amounts to the truth of $\mathcal{T}[\![(\varphi)S(I)]\!]\gamma\eta$.

□

Next we show how to represent a formula $(\varphi)\ S\ (\psi)$ in the form of an Invariant formula. From lemma 4.22 it follows that any such formula can be rewritten into:

$$\Big((\bot \wedge I) \vee (\top \wedge p)\Big)\ S\ \Big((\bot \rightarrow J) \wedge (\top \rightarrow q)\Big), \qquad (1)$$

where $I = \varphi[\bot]$, $p = \varphi[\top]$, $J = \psi[\bot]$, $q = \psi[\top]$.

This lemma states that the following equivalences hold:

$\forall_\perp\big(\varphi \longleftrightarrow ((\perp \wedge I) \vee (\top \wedge p))\big)$ and

$\forall_\perp\big(\psi \longleftrightarrow ((\perp \rightarrow J) \wedge (\top \rightarrow q))\big)$.

Hence using the consequence rule and the implication introduction rule twice, one can formally prove: $(\varphi)\ S\ (\psi) \leftrightarrow (1)$. Formula (1) can be factorized into a conjunction of the following four:

$(\perp \wedge I)\ S\ (\perp \rightarrow J)$ (2)

$(\perp \wedge I)\ S\ (\top \rightarrow q)$ (3)

$(\top \wedge p)\ S\ (\perp \rightarrow J)$ (4)

$(\top \wedge p)\ S\ (\top \rightarrow q)$ (5)

Again $(1) \leftrightarrow \big((2) \wedge (3) \wedge (4) \wedge (5)\big)$ can be formally proven, using the conjunction, disjunction and consequence rules.

Now from (4) one derives $(\top \wedge p)\ S\ (J)$ by means of the bottom closure rule. Combining this with (5), using the conjunction and consequence rules, we see that

$(\top \wedge p)\ S\ (J \wedge (\top \rightarrow q))$

is formally derivable from (1). That is, the following is a provable implication:

$(\varphi)\ S\ (\psi) \rightarrow J : \{p\}\ S\ \{q\}$.

If $I \rightarrow J$ is a valid assertion, that is if $\underline{\forall}(I \rightarrow J)$ is true, we can show the reverse too. (Note that since I and J are closed with respect to initial and final states, $\underline{\forall}(I \rightarrow J)$ is equivalent to $\forall_\perp(I \rightarrow J)$.)

From $(\top \wedge p)\ S\ (J \wedge (\top \rightarrow q))$, we get, by using the consequence rule:

$(\top \wedge p)\ S\ ((\perp \rightarrow J) \wedge (\top \rightarrow q))$ (6)

Also, the instance of the strictness axiom $(\perp \wedge I)\ S\ (\perp \wedge I)$ can be weakened by means of the consequence rule into:

$(\perp \wedge I)\ S\ ((\perp \wedge J) \vee (\top \wedge q))$ (7)

Again using lemma 4.22, we obtain the equivalent formula:

$(\perp \wedge I)\ S\ ((\perp \rightarrow J) \wedge (\top \rightarrow q))$ (8)

Then, by applying the disjunction rule to combine (6) and (8), followed by an application of the consequence rule, we derive formula (1). So

$$J \;:\; \{p\}\; S\; \{q\} \;\rightarrow\; (\varphi)\; S\; (\psi)$$

is provable in the case that $\underline{\forall}(I \rightarrow J)$ holds.

Finally we prove that if $\underline{\forall}(I \rightarrow J)$ is not valid then $(\varphi)\; S\; (\psi)$ is *unsatisfiable.* That is, there exists no **TNP** process S such that $(\varphi)\; S\; (\psi)$. For, assume there are some $h \in \mathit{Trace}$ and $\gamma \in \Gamma$, such that $h \in [\![I]\!]\gamma$ but not $h \in [\![J]\!]\gamma$. Then we have:

$$(h, \bot) \in [\![(\bot \wedge I) \vee (\top \wedge p)]\!]\gamma, \text{ and } (h, \bot) \notin [\![(\bot \rightarrow J) \wedge (\top \rightarrow q)]\!]\gamma.$$

Hence, since $M^{\dagger}[\![S]\!](\eta)(\{(h, \bot)\}) = \{(h, \bot)\}$, we see that (1) cannot hold in this case for any η. That is, (1), and so $(\varphi)\; S\; (\psi)$, is unsatisfiable.

All together we proved the following:

Lemma 4.39

Let $\varphi, \psi \in \mathit{Assn}(\Sigma)$, $S \in \mathbf{TNP}$, $p = \varphi[\top]$, $q = \psi[\top]$, $I = \varphi[\bot]$, $J = \psi[\bot]$.

If $\underline{\forall}(I \rightarrow J)$ is true the equivalence:

$$(\varphi)\; S\; (\psi) \;\leftrightarrow\; J \;:\; \{p\}\; S\; \{q\}$$

is valid and can even be proven formally, using the rules above. Else, that is if $\underline{\forall}(I \rightarrow J)$ is not true, $(\varphi)\; S\; (\psi)$ is unsatisfiable.

□

We finish with three proofrules that are used to translate Hoare formulae into Invariant formulae and vice versa. The soundness of the first two rules follows immediate from our definition of Invariant formulae. The soundness of the third rule is part of the lemma above.

(i) $$\frac{I \;:\; \{p\}\; m\; \{q\}}{(\top \wedge p)\; m\; (I \wedge (\top \rightarrow q))} \quad \text{(IH)}$$

(ii) $$\frac{(\top \wedge p)\; m\; (I \wedge (\top \rightarrow q))}{I \;:\; \{p\}\; m\; \{q\}} \quad \text{(HI)}$$

For **TNP** processes S:

(iii) $$\frac{(\varphi)\; S\; (\psi) \quad , \quad \underline{\forall}(\varphi[\bot] \rightarrow \psi[\bot])}{\psi[\bot] \;:\; \{\varphi[\top]\}\; S\; \{\psi[\top]\}} \quad \text{(Prop)}$$

□

Chapter 5

Proof systems for TNP

5.1 Introduction

We introduce three formal proof systems for **TNP** processes, called the SAT system, the Hoare system and the Invariant system. The three systems are axiomatizations of the three types of correctness formulae with corresponding names that we defined in chapter 4. Each system is divided into three groups:

A Axioms for atomic processes.

B Rules corresponding to the **TNP** constructs.

C Adaptation rules.

We introduce the term "adaptation rules" for all those rules that are concerned with the adaptation of some black box specification $spec(\varsigma)$ to another specification $spec'(\varsigma)$ for the same black box. A well known example of such a rule is the "consequence rule" of Hoare's logic. There are important differences with respect to the number and the complexity of adaptation rules for the three systems.

The SAT system is an axiomatization of mixed terms by means of SAT formulae. These formulae are of the form $m\ sat\ \chi$, where m is a mixed term and $\chi(\in \mathcal{A}ssn)$ is an assertion. For the SAT system the adaptation rules are:

- the invariance axiom,
- the conjunction rule, and
- the consequence rule.

The fact that there are only a few of such adaptation rules, and moreover that they are quite elegant, is one of the virtues of the SAT system. The price for this elegance is the added complexity of assertions in comparison with usual Hoare style logic. This is prominently present in the rule for sequential composition. This rule is the following one:

$$\frac{m_1 \textit{ sat } \chi_1, m_2 \textit{ sat } \chi_2}{S_1 ; S_2 \textit{ sat } \chi_1 \hat{\circ} \chi_2}$$

One should keep in mind here that, although the "$\hat{\circ}$" operator allows a compact notation, $\chi_1 \hat{\circ} \chi_2$ is nothing else but an abbreviation of a rather complex assertion. Let us consider the case where one wants to show that a certain process m of the form $m_1; m_2; \cdots; m_k$ satisfies a given assertion χ, where the component m_i is known to satisfy χ_i. Basically, there are two strategies for proving this specification:

(1) First prove that m *sat* $\chi_1 \hat{\circ} \chi_2 \hat{\circ} \cdots \hat{\circ} \chi_k$, by repeated application of the sequential composition rule of the SAT system. (That part is very easy, indeed.) Then, as the second step, prove that m *sat* χ by means of the consequence rule. The latter step includes the validation of the implication: $(\chi_1 \hat{\circ} \chi_2 \hat{\circ} \cdots \hat{\circ} \chi_k) \rightarrow \chi$. The problem with this validation, whether feasible or not, is that it is as complex as the validation of the original specification.

(2) First prove that $m_1; m_2$ *sat* χ_{12}, for some appropriately chosen assertion χ_{12}, by means of the sequential composition rule and the consequence rule. Of course this includes the validation of the assertion $(\chi_1 \hat{\circ} \chi_2) \rightarrow \chi_{12}$. We proceed in this way, inventing "intermediate" assertions for binary sequential compositions that are proven by means of the sequential composition and consequence rules. The validation of one huge "verification condition", that was necessary for the first strategy, has been replaced by the validation of $k-1$ simpler conditions.

Similar problems arise when one tries to formulate a proof rule for the iteration construct. We have not included such a rule in the formal system below since in fact m^* is an abbreviation of $\mu_z \xi \cdot (\textbf{skip or } m ; \xi)$, and the loop rule is a derived rule that follows from the SAT rules for **skip** , the choice and sequential composition, and the recursion construct. Nevertheless, it is interesting to discuss this rule, since it brings out more clearly the differences between the SAT system on the one hand and Hoare style systems on the other hand.

One possibility is to introduce a so called "chop-star" [BKP] operation on assertions that has the same semantics as the Kleene star for processes. That is, we could define χ^* as an assertion that is interpreted as follows:

$\mathcal{T}[\![\chi^*]\!]\gamma(s_0, h, s)$ iff there exist some integer n

and states and traces $s_1, \ldots, s_n$, $h_1, \ldots, h_n$ such that

$s_n = s$, $h = h_1 h_2 \cdots h_n$, and, for $1 \leq i \leq n$: $\mathcal{T}[\![\chi]\!]\gamma(s_{i-1}, h_i, s_i)$

This definition explains the name chop-star: it is a generalization of the "$\hat{o}$" operator that is sometimes called the "chop" operator. There is no simple way to express the chop-star operation in terms of the chop operator in our assertion language, since the chop-star includes an existential quantifier for the *number* n of chops so to say.

The proof rule for the loop construct would then become:

$$\frac{m_1 \; sat \; \chi_1}{m_1^* \; sat \; \chi_1^*} \text{ (chop-star)}$$

The problem with this type of rule is essentially the same as above: *the assertion χ_1^* is as complex as the process itself.*

It will be clear that if some specification $m_1^* \; sat \; \chi$ is to be verified, and m_1 is known to satisfy χ_1, then one must validate the verification condition $\chi_1^* \rightarrow \chi$. This in generally will involve some kind of induction on the number n of chops in the definition of χ_1^*. We regard this as the main difference with the well known proof rule for loops in Hoare's logic: whereas the Hoare style rule explicitly indicates that some induction hypothesis must be formulated (the loop invariant), and also indicates what kind of "induction step" must be proven (the invariance of the loop invariant), the chop-star rule *shifts the problem* to a similar problem within the assertion language.

A more attractive rule is the following one:

$$\frac{m_1 \; sat \; \chi_1 \quad , \quad \textbf{skip} \; sat \; \chi \quad , \quad \forall_\perp((\chi_1 \hat{o} \chi) \rightarrow \chi)}{m_1^* \; sat \; \chi} \text{ (loop)}$$

This time the induction principle *is* captured by the rule. The rule also brings out once more the importance of the axiomatization of sequential composition, here embodied by the chop operator in one of the premisses of the rule.

The problems with the treatment of sequential composition in proof systems in the style of the SAT system formed the incentive for studying a precondition/postcondition style of reasoning about processes. The main idea is that there is not only an initial *state* in which a process starts executing, but that there is also an initial *trace* of communication actions. For a sequential context $m_1\,;\, m_2$ of m_2, the initial trace for m_2 includes the communications performed by m_1. Regarding pairs of states and traces as a generalized state, and using assertions ϕ, ψ that are predicate formulae for generalized states, we were able to formulate the following rule for sequential composition:

$$\frac{(\varphi)\, m_1\, (\rho), (\rho)\, m_2\, (\psi)}{(\varphi) S_1\,;\, S_2 (\psi)}$$

Note that, at least in its outer form, it is exactly the "classical" rule of Hoare's logic for sequential composition. Because of this coincidence, the following rule for the loop construct should not be a surprise:

$$\frac{(\varphi)\, m_1\, (\varphi) \quad , \quad (\varphi)\, \mathbf{skip}\, (\varphi)}{(\varphi)\, m_1^*\, (\varphi)}\,(\text{loop})$$

Apart from the premisse for **skip** this has the same form as the corresponding rule in Hoare's logic. (Again we have not included this rule in the formal system below since it is a derived rule for our approach.)

Comparing these two rules with the corresponding SAT rules, we remark that the Hoare style rules do not introduce complicated operators, like the chop operator.

A more interesting observation is that whereas the SAT rules suggest a *bottom up* approach to program development, the Hoare rules suggest a *top down* development. For instance, the SAT rule for sequential composition constructs a specification from a priori given specifications for the parts, but, on the other hand, the Hoare style rule splits an a priori given specification for the whole into specifications for the parts.

This difference in terms of top down versus bottom up development is present in most of the rules for the two systems.

One notable exception is the SAT rule for channel renaming; as can be seen below, this rule has a top down character. It is not the case that we could not formulate a bottom up style rule for this construct, but the bottom up rule is much less attractive for actual program proving than the top down version. Of course this raises the question whether we have excluded a bottom up

style program development, and if so, what style of development do we propose? To answer the first question: no, we have not excluded bottom up style. This is shown in chapter 6, where we construct, ***bottom up***, a so ***characteristic assertion*** for each mixed term, that is essentially the ***strongest*** assertion that is satisfied by the mixed term. The SAT system allows one to formally prove that a mixed term does satisfy its characteristic assertion. In particular the characteristic assertion for $m_1\langle d/c\rangle$ can be proven from the characteristic assertion for m_1 without undue effort. The reason then why we do not regard the renaming rule as a bottom up style rule is that, given some actual specification for m_1, the rule does not suggest any appropriate specification for $m_1\langle d/c\rangle$.

The section with ***adaptation rules*** for the Hoare system does contain more axioms and rules than the corresponding section for the SAT system. This is a well known phenomenon from Hoare style systems for sequential programs that allow ***procedures***. Of course then the question remains why the SAT system does ***not*** need so many adaptation rules. The reason for this is that many adaptation rules are in fact reformulations of general logic principles. For instance, the $(\exists)$-pre rule of the Hoare system corresponds closely to the following logical principle:

Provided that $g \notin FV(H)$:

$$\frac{H \cup \{f_0\} \vdash f_1}{H \cup \{\exists g(f_0)\} \vdash f_1}\ (\exists \text{ - introduction})$$

Now for the SAT system, the logical principle can be applied via the use of the consequence rule, whereas for the Hoare system one needs a new proof rule. This can be seen more clearly from the soundness proofs that we give for the Hoare system. Instead of proving the soundness of Hoare rules ***directly*** from the semantic definition of **TNP** , we show that the application of a Hoare rule corresponds to a proof in the SAT system preceded and followed by a translation of Hoare formulae into equivalent SAT formulae and vice versa.

One salient point of the Hoare system is the ***prefix invariance axiom***. This axiom is: $(t_0|\mathbf{c} \leq h|\mathbf{c})\ m\ (t_0|\mathbf{c} \leq h|\mathbf{c})$, where t_0 is some logical trace variable, and $\mathbf{c}$ is some arbitrary set of channels.

This axiom has no counterpart in Hoare systems for sequential programs. The soundness of the axiom is clear by the fact that a process can only ***extend*** an existing initial trace by submitting new communications, but it can

neither withdraw nor change the temporal ordering of already performed communications. In our completeness proof for the Hoare system in chapter 7, it is the *preservation of temporal ordering* what we need the prefix invariance axiom for. It has been shown by Josef Hooman [HooZw] that the prefix invariance axiom is *essential* to obtain a *complete* proof system.

Finally we discuss the Invariant system. By inspection of the Hoare system below, one remarks that many axioms are more complicated than one would expect, due to the fact that both terminated and non finished computations must be described by the same assertions. In chapter 4 we already pointed out that for TNP processes an alternative, more attractive, representation of Hoare formulae is possible in the form of Invariant formulae. In fact The Invariant system is, apart from a few exceptions, a reformulation of the Hoare system in terms of Invariant formulae.

5.2 The SAT proof system

5.2.1 Axioms for Atomic Processes

To bring out the structure of the axioms for atomic processes more clearly we first define a so called characteristic assertion $A(\alpha)$ for each atomic process α. In the soundness proofs for these axioms we actually prove that, as far as the free channels and variables of α are concerned, $A(\alpha)$ and α itself do have the *same* semantics. So it is certainly the case that the set of computations denoted by α is *contained* in the semantics of $A(\alpha)$. This inclusion is expressed by the following SAT axioms, where $A(\alpha)$ is as defined below.

Predicative processes of the form $\chi|\beta$, resemble in many aspects atomic processes. We include here the "characteristic assertion", and the axiom for such predicative processes.

- *Atomic processes* α

  α *sat* $A(\alpha)$.

- *Predicative processes*

  $\chi|\beta$ *sat* χ.

□

Definition 5.1 (Characteristic assertions for *Atom* and *Pred*)

$A(\mathbf{abort}) = A(\mathbf{Z}) = \bot$.

$A(\mathbf{skip}) = \mathbf{true}$.

$A(x := e) = \top \rightarrow (x = e[\bar{y}^\circ/\bar{y}] \wedge \bar{w} = \bar{w}^\circ)$,

where $\{\bar{y}\} = var(e)$ and $\{\bar{w}\} = var(e) - \{x\}$.

$A(b) = \top \rightarrow (b \wedge \bar{w} = \bar{w}^\circ)$, where $\{\bar{w}\} = var(b)$.

$A(c.x : b) = (\bot \wedge \mathbf{c} = \epsilon) \vee$

$\exists v[c =< v > \wedge\, b[\bar{w}^\circ/\bar{w}, v/x] \wedge (\top \rightarrow (v = x \wedge \bar{w} = \bar{w}^\circ])$

where $\bar{w} = var(b) - \{x\}$.

$A(\chi|\beta) = \chi$.

□

5.2.2 Rules for the TNP constructs

- *Channel hiding*

$$\frac{m \textit{ sat } \chi}{m\backslash \mathbf{c} \textit{ sat } \chi} \quad \text{provided } hchan(\chi) \cap \mathbf{c} = \emptyset$$

- *Variable hiding*

$$\frac{m \textit{ sat } \chi}{m\backslash \mathbf{x} \textit{ sat } \chi} \quad \text{provided } var(\chi) \cap \mathbf{x} = \emptyset$$

- *Renaming*

$$\frac{m \textit{ sat } \chi[h[d/c]/h]}{m\langle d/c\rangle \textit{ sat } \chi}$$

- *Kernel*

$$\frac{m \textit{ sat } \chi}{Kern(m) \textit{ sat } Kern(\chi)}$$

- *Sequential composition*

$$\frac{m_1 \; sat \; \chi_1, m_2 \; sat \; \chi_2}{m_1; m_2 \; sat \; \chi_1 \hat{\circ} \chi_2}$$

- *Choice*

$$\frac{m_1 \; sat \; \chi_1, m_2 \; sat \; \chi_2}{m_1 \; \mathbf{or} \; m_2 \; sat \; \chi_1 \vee \chi_2}$$

- *Parallel composition*

Provided that for $(i,j) = (1,2)$ and for $(i,j) = (2,1)$:
$abase(\chi_i) \cap \beta_j \subseteq \beta_i$, the following rule is applicable:

$$\frac{m_1 \; sat \; \chi_1, m_2 \; sat \; \chi_2}{m_1 \; \beta_1 \| \beta_2 \; m_2 \; sat \; \chi_1 \wedge \chi_2}$$

- *Recursion*

Provided that $X_\beta \notin pvar(H)$:

$$\frac{H \cup \{X_\beta \; sat \; \chi\} \vdash m \; sat \; \chi}{H \vdash \mu X_\beta.m \; sat \; \chi}$$

- μ_z *Recursion*

Provided that $X_\beta \notin pvar(H)$:

$$\frac{H \vdash \mathbf{Z} \; sat \; \chi \quad , \quad H \cup \{X_\beta \; sat \; \chi\} \vdash m \; sat \; \chi}{H \vdash \mu_z X_\beta.m \; sat \; \chi}$$

- *Process naming*

Provided that $X_\beta \notin pvar(H)$:

$$\frac{H \vdash m_1 \; sat \; \chi_1 \quad , \quad H \cup \{X_\beta \; sat \; \chi_1\} \vdash m_2 \; sat \; \chi_2}{H \vdash X_\beta = m_1 \; \mathbf{in} \; m_2 \; sat \; \chi_2}$$

5.2.3 Adaptation rules for the SAT system

• *Invariance*

Provided that $base(m) \cap (\mathbf{d}, \{\bar{y}\}) = (\emptyset, \emptyset)$:

$$m \; sat \; (h|\mathbf{d} = \varepsilon \wedge (\top \rightarrow \bar{y} = \bar{y}^{\circ}))$$

• *Conjunction*

$$\frac{m \; sat \; \chi_1, m \; sat \; \chi_2}{m \; sat \; (\chi_1 \wedge \chi_2)}$$

• *Consequence*

$$\frac{m \; sat \; \chi_1, \forall_{\perp}(\chi_1 \rightarrow \chi_2)}{m \; sat \; \chi_2}$$

Remark The universal quantor $\forall_{\perp}$, introduced in chapter 4, denotes *strict* universal closure.

5.3 The Hoare system

5.3.1 Axioms for Atomic Processes

In the section for the SAT system we introduced the characteristic assertions $A(\alpha)$ for atomic processes and predicative processes α, that describes the behavior of α as far as the channels and variables in $base(\alpha)$ are concerned.

For bases β such that $base(\alpha) \subseteq \beta$ the assertion $A_{\beta}(\alpha)$, defined as $A(\alpha) \wedge \mathbf{1}_{\beta - base(\alpha)}$, then can be seen to describe this behavior as far as all channels and variables $in\beta$ are concerned.

We have chosen the axiom for atomic process to be of the form:

$$(A_{\beta}(\alpha) \rhd \psi) \; \alpha \; (\psi),$$

where $\beta = base(\alpha) \cup abase(\psi)$. This form shall enable us to deduce, indirectly, the completeness of these axioms from the completeness of the corresponding SAT axioms.

After simplification of the assertions of the form $A_{\beta}(\alpha) \rhd \psi$, one obtains the following set of axioms:

• *Abort,***Z**

$(\psi[\bot])$ **abort** (ψ)

$(\psi[\bot])$ **z** (ψ)

- *Skip*

  $(\psi[\bot] \wedge \psi)$ **skip** (ψ)

- *Guard*

  $(\psi[\bot] \wedge (\top \wedge b) \rightarrow \psi[\top])\ b\ (\psi)$

- *Assign*

  $(\psi[\bot] \wedge (\top \rightarrow \psi[\bot][e/x]))\ x := e\ (\psi).$

- *Com*

  $(\psi[\bot] \wedge$

  $\top \rightarrow \forall v \Big(b[v/x] \rightarrow (\psi[\bot][h < (c,v) > /h] \wedge \psi[\top][h < (c,v) > /h, v/x]) \Big))$

  $c.x!b$

  (ψ)

For predicative processes, we choose the following axiom:

- *Predicative processes*

  $(\varphi)\ (\chi|\beta)\ (\varphi \lhd \chi).$

The axiom has been chosen such as to facilitate the completeness results of chapter 7.

5.4 Rules for the TNP constructs

- *Channel hiding*

$$\frac{(\varphi)\ m\ (\psi)}{(\varphi)\ m\backslash \mathbf{c}\ (\psi)} \quad \text{provided } hchan(\psi) \cap \mathbf{c} = \emptyset$$

- *Variable hiding*

$$\frac{(\varphi)\ m\ (\psi)}{(\varphi)\ m\backslash \mathbf{x}\ (\psi)} \quad \text{provided } var(\psi) \cap \mathbf{x} = \emptyset$$

• *Renaming*

Let c' be some fresh channel name.

$$\frac{(\varphi[\,h[c/c']/h\,] \wedge c = \varepsilon)\; m\; (\psi[\,h[d/c][c/c']/h\,])}{(\varphi)\; m\langle d/c\rangle\; (\psi)}$$

• *Kern*

If $hchan(\varphi, \psi) \subseteq \mathbf{c}$ and t_0 is fresh, then:

$$\frac{(\varphi) m (\psi)}{(\varphi \wedge h|\mathbf{c} = t_0|\mathbf{c})\; Kern(m)\; (Kern(t_0, \psi))}$$

• *Sequential composition*

$$\frac{(\varphi)\; m_1\; (\rho) \quad , \quad (\rho)\; m_2\; (\psi)}{(\varphi)\; m_1; m_2\; (\psi)}$$

• *Choice*

$$\frac{(\varphi)\; m_1\; (\psi) \quad , \quad (\varphi)\; m_2\; (\psi)}{(\varphi)\; m_1 \textbf{ or } m_2\; (\psi)}$$

• *Parallel composition*

Provided that for $(i, j) = (1, 2)$ and for $(i, j) = (2, 1)$:
$abase(\psi_i) \cap \beta_j \subseteq \beta_i$, the following rule is applicable:

$$\frac{(\varphi_1)\; m_1\; (\psi_1) \quad , \quad (\varphi_2)\; m_2\; (\psi_2)}{(\varphi_1 \wedge \varphi_2)\; m_1\; \beta_1 \| \beta_2\; m_2\; (\psi_1 \wedge \psi_2)}$$

• *Recursion*

Let $\bar{g}$ be some arbitrary list of logical variables, possibly the empty list. Provided that $X_\beta \notin pvar(H)$:

$$\frac{H \cup \{\forall \bar{g}[(\varphi)\; X_\beta\; (\psi)]\} \vdash \forall \bar{g}[(\varphi)\; m\; (\psi)]}{H \vdash \forall \bar{g}[(\varphi)\; \mu X_\beta . m\; (\psi)]}$$

- *μ_z Recursion*

Let $\bar{g}$ be some arbitrary list of logical variables, possibly the empty list. Provided that $X_\beta \notin pvar(H)$:

$$\frac{H \vdash \forall \bar{g}[(\varphi)\ \mathbf{z}\ (\psi)] \quad , \quad H \cup \{\forall \bar{g}[(\varphi)\ X_\beta\ (\psi)]\} \vdash \forall \bar{g}[(\varphi)\ m\ (\psi)]}{H \vdash \forall \bar{g}[(\varphi)\ \mu_z X_\beta.m\ (\psi)]}$$

- *Process naming*

Let $\bar{g}$ be some arbitrary list of logical variables, possibly the empty list. Provided that $X_\beta \notin pvar(H)$:

$$\frac{H \vdash \forall \bar{g}[(\varphi_1)\ m_1\ (\psi_1)] \quad , \quad H \cup \{\forall \bar{g}[(\varphi_1) X_\beta (\psi_1)]\} \vdash (\varphi_2)\ m_2\ (\psi_2)}{H \vdash (\varphi_2) X_\beta = m_1 \text{ in } m_2(\psi_2)}$$

5.4.1 Adaptation rules for the Hoare system

- *Invariance*

$$(\psi[\bot] \wedge \psi)\ m\ (\psi)$$

provided that $abase(\psi) \cap base(m) = (\emptyset, \emptyset)$.

- *Prefix invariance*

For arbitrary logical trace variable t_0 and channel set $\mathbf{c}$:

$$(t_0|\mathbf{c} \leq h|\mathbf{c})\ m\ (t_0|\mathbf{c} \leq h|\mathbf{c})$$

- *Strictness*

$$(\bot \wedge \varphi)\ m\ (\bot \wedge \varphi)$$

- *Conjunction*

$$\frac{(\varphi_1)\ m\ (\psi_1) \quad , \quad (\varphi_2)\ m\ (\psi_2)}{(\varphi_1 \wedge \varphi_2)\ m\ (\psi_1 \wedge \psi_2)}$$

- *Disjunction*

$$\frac{(\varphi_1)\, m\, (\psi_1) \quad , \quad (\varphi_2)\, m\, (\psi_2)}{(\varphi_1 \vee \varphi_2)\, m\, (\psi_1 \vee \psi_2)}$$

- *Consequence*

$$\frac{\forall_\perp(\varphi \rightarrow \varphi') \quad , \quad (\varphi')\, m\, (\psi') \quad , \quad \forall_\perp(\psi' \rightarrow \psi)}{(\varphi)\, m\, (\psi)}$$

- $\exists - pre$

$$\frac{\forall \bar{g}\left[(\varphi)\ m\ (\psi)\right]}{(\exists \bar{g}(\varphi))\ m\ (\psi)} \qquad \text{provided } \bar{g} \cap var(\psi) = \emptyset$$

Remark

An alternative formulation is:

Provided $\bar{g} \cap var(\psi) = \emptyset$ and $\{\bar{g}\} \cap gvar(H) = \emptyset$:

$$\frac{H \vdash (\varphi)\ m\ (\psi)}{H \vdash (\exists \bar{g}(\varphi))\ m\ (\psi)}$$

This second version is in fact a derived rule. For if $\{\bar{g}\} \cap gvar(H) = \emptyset$ one can first apply the $\forall$ introduction rule of chapter 4, to obtain $\forall \bar{g}\left[(\varphi)\ m\ (\psi)\right]$ from $(\varphi)\ m\ (\psi)$, and then apply the $\exists - pre$ rule. We often use this derived rule tacitly.

5.4.2 Extra adaptation rules for the Hoare system

- *Strong adaptation*

Provided that $\beta \cap base(m) = (\emptyset, \emptyset)$,

$$\frac{(\varphi_1)\, m\, (\psi_1)}{(\varphi_2)\, m\, (\varphi_2 \lhd (\varphi_1 \rightsquigarrow \psi_1) \wedge \mathbf{1}_\beta)}$$

Although we don't use it, there is also a weakest precondition version of this rule. Under the same conditions for β :

$$\frac{(\varphi_1)\, m\, (\psi_1)}{(((\varphi_1 \rightsquigarrow \psi_1) \wedge \mathbf{1}_\beta) \rhd \psi_2)\, m\, (\psi)} \text{ (Adaptation,WP-version)}$$

- *Initial trace adaptation*

Provided that $\mathbf{c}' \cap hchan(\varphi, \psi) = \emptyset$

$$\frac{(\varphi \wedge h|\mathbf{c}' = \varepsilon)\, m\, (\psi)}{(\varphi)\, m\, (\psi)} \text{ initial trace adaptation}$$

5.5 The Invariant System

5.5.1 Axioms for Atomic Processes

- *Abort,***Z**

  $I : \{I\}$ **abort** $\{$**false**$\}$

  $I : \{I\}$ **Z** $\{$**false**$\}$

- *Skip*

  $I : \{p \wedge I\}$ **skip** $\{p \wedge I\}$

- *Guard*

  $I : \{p \wedge I\}\, b\, \{p \wedge I \wedge b\}$

- *Assign*

  $I : \{p[e/x] \wedge I\}\, x := e\, \{p \wedge I\}$

- *Com*

  $I : \{I \wedge \forall v \Big(b[v/x] \rightarrow (I[h < (c, v) > /h] \wedge p[h < (c, v) > /h][v/x])\Big)\}$

  $c.x : b$

  $\{p \wedge I\}$

- *Predicative processes*

We do not offer an axiom for *arbitrary* predicative processes. We consider only processes denoted by $\langle J, R \rangle$, that are of the following form:

$$\langle J, R \rangle \stackrel{\text{def}}{=} ((\top^\circ \rightarrow J) \wedge (\top \rightarrow R))|base(J, R),$$

where J and R are *proper* assertions, that is, elements of $\mathcal{A}ssn_p$, and where $var(J) = \emptyset$.

So R is proper assertion characterizing the relation between initial state values, communication histories and final state values for *terminated* computations. Similarly, J describes the communication history *during execution* in relation to the initial state values of variables.

For such processes, the following axiom holds:

$$(p \lhd J) : \{p\}\ \langle J, R\rangle\ \{p \lhd (J \wedge R)\}.$$

5.5.2 Rules for the TNP constructs

- *Channel hiding*

$$\frac{I : \{p\}\ m\ \{q\}}{I : \{p\}\ m\backslash \mathbf{c}\ \{q\}} \quad \text{provided } hchan(I, q) \cap \mathbf{c} = \emptyset.$$

- *Variable hiding*

$$\frac{I : \{p\}\ m\ \{q\}}{I : \{p\}\ m\backslash \mathbf{x}\ \{q\}} \quad \text{provided } var(q) \cap \mathbf{x} = \emptyset.$$

- *Renaming*

Let c' be some fresh channel name. Then:

$$\frac{(I[h[d/c][c/c']/h]) : \{p[h[c/c']/h] \wedge \mathbf{c} = \varepsilon\}\ m\ \{q[h[d/c][c/c']/h]\}}{I : \{p\}\ m\langle d/c\rangle\{q\}}$$

- *Kern*

If $hchan(I, p, q) \subseteq \mathbf{c}$ and t_0 is fresh, then:

$$\frac{I : \{p\}\ m\ \{q\}}{(Kern(t_0, I)) : \{p \wedge h|\mathbf{c} = t_0|\mathbf{c}\}\ Kern(m)\ \{q\}}$$

- *Sequential composition*

$$\frac{I : \{p\}\ m_1\ \{r\} \quad , \quad I : \{r\}\ m_2\ \{q\}}{I : \{p\}\ m_1; m_2\ \{q\}}$$

- *Choice*

$$\frac{I : \{p\}\ m_1\ \{q\} \quad , \quad I : \{p\}\ m_2\ \{q\}}{I : \{p\}\ m_1 \textbf{ or } m_2\ \{q\}}$$

- *Parallel composition*

Provided that for $(i,j) = (1,2)$ and for $(i,j) = (2,1)$:
$abase(I_i, q_i) \cap \beta_j \subseteq \beta_i$:

$$\frac{I_1 : \{p_1\}\ m_1\ \{q_1\} \quad , \quad I_2 : \{p_2\}\ m_2\ \{q_2\}}{I_1 \wedge I_2 : \{p_1 \wedge p_2\}\ m_1\ \beta_1 \| \beta_2\ m_2\ \{q_1 \wedge q_2\}}$$

- *Recursion*

Let $\bar{g}$ be some arbitrary list of logical variables, possibly the empty list. Provided that $X_\beta \notin pvar(H)$:

$$\frac{H \cup \{\forall \bar{g}[I : \{p\}\ X_\beta\ \{q\}]\} \vdash \forall \bar{g}[I : \{p\}\ m_1\ \{q\}]}{\forall \bar{g}[I : \{p\}\ \mu X_\beta . m_1\ \{q\}]}$$

- μ_z *Recursion*

Let $\bar{g}$ be some arbitrary list of logical variables, possibly the empty list. Provided that $X_\beta \notin pvar(H)$:

$$\frac{H \cup \{\forall \bar{g}[I : \{p \wedge I\}\ X_\beta\ \{q\}]\} \vdash \forall \bar{g}[I : \{p \wedge I\}\ m_1\ \{q\}]}{\forall \bar{g}[I : \{p \wedge I\}\ \mu_z X_\beta . m_1\ \{q\}]}$$

- *Process naming*

Let $\bar{g}$ be some arbitrary list of logical variables, possibly the empty list. Provided that $X_\beta \notin pvar(H)$:

$$\frac{H \vdash \forall \bar{g}[I_1 : \{p_1\}\ m_1\ \{q_1\}] \quad , \quad H \cup \{\forall \bar{g}[I_1 : \{p_1\}\ m_1\ \{q_1\}]\} \vdash I_2 : \{p_2\}\ m_2\ \{q_2\}}{H \vdash I_2 : \{p_2\}\ X_\beta = m_1 \textbf{ in } m_2\ \{q_2\}}$$

5.5.3 Adaptation rules for the Invariant System

- *Invariance*

$$I : \{p \wedge I\}\ m\ \{p \wedge I\}$$

provided $abase(p, I) \cap base(m) = (\emptyset, \emptyset)$.

- *Prefix invariance*

For arbitrary logical trace variable t_0 and set of channels $\mathbf{c}$:

$$(h|\mathbf{c} \leq t_0|\mathbf{c}) : \{h|\mathbf{c} \leq t_0|\mathbf{c}\}\ m\ \{h|\mathbf{c} \leq t_0|\mathbf{c}\}$$

- *Closure adaptation*

$$\frac{I : \{p\}\ m\ \{q\}}{I : \{p\}\ m\ \{q \wedge I\}}$$

- *Initial trace adaptation*

Provided that $\mathbf{c}' \cap hchan(I, p, q) = \emptyset$

$$\frac{I : \{p \wedge h|\mathbf{c}' = \varepsilon\}\ m\ \{q\}}{I : \{p\}\ m\ \{q\}}$$ initial trace adaptation

- *Conjunction*

$$\frac{I_1 : \{p_1\}\ m\ \{q_1\} \quad , \quad I_2 : \{p_2\}\ m\ \{q_2\}}{I_1 \wedge I_2 : \{p_1 \wedge p_2\}\ m\ \{q_1 \wedge q_2\}}$$

- *Disjunction*

$$\frac{I_1 : \{p_1\}\ m\ \{q_1\} \quad , \quad I_2 : \{p_2\}\ m\ \{q_2\}}{I_1 \vee I_2 : \{p_1 \vee p_2\}\ m\ \{q_1 \vee q_2\}}$$

- *Consequence*

$$I' : \{p'\}\ m\ \{q'\}$$

$$\frac{\underline{\forall}(I' \rightarrow I), \underline{\forall}(p \rightarrow p'), \underline{\forall}(q' \rightarrow q)}{I : \{p\}\ m\ \{q\}}$$

- $\exists - pre$

$$\frac{\forall \bar{g}\left[I : \{p\}\ m\ \{q\}\right]}{I : \{\exists \bar{g}(p)\}\ m\ \{q\}} \quad \text{provided } \bar{g} \cap var(I, q) = \emptyset$$

5.6 Scott's induction rule

The Induction Principle of Scott is well known.[Scott] We rephrase this principle for our particular language of correctness formulae.

Definition 5.2 (Admissible formulae)

A correctness formula f is called *anticontinuous* in a process variable X_β if for any chain $< \rho_i >_{i \in \mathbf{N}}$ in $P(\Delta_\beta)$ the following holds:

$$\mathcal{T}[\![f]\!](\gamma)(\eta[\bigcup_{i \in \mathbf{N}} \rho_i / X_\beta]) \text{ iff}$$
$$\forall i \in \mathbf{N}(\mathcal{T}[\![f]\!](\gamma)(\eta[\rho_i / X_\beta])).$$

If a formulae f is anticontinuous in all its free process variables, we call it *admissible.*

□

Theorem 5.3 Scott's Induction Rule

Let f be a correctness formula that is anticontinuous in X_β. Then the following rule is sound. Provided that $X_\beta \notin pvar(H)$:

$$\frac{H \vdash f[\mathbf{false}/X_\beta]\ ,\ H \cup \{f\} \vdash f[m/X_\beta]}{H \vdash f[\mu X_\beta . m / X_\beta]}$$

□

Proof

The second premisse of the rule can be transformed as in the following derivation. Note that the introduction of the universal quantifier in the second step is allowed since $X_\beta \notin pvar(H)$.

$$\frac{H \cup \{f\} \vdash f[m/X_\beta]}{H \vdash f \to f[m/X_\beta]} \quad (\to \text{ Introduction})$$

$$\frac{H \vdash f \to f[m/X_\beta]}{H \vdash \forall X_\beta(f \to f[m/X_\beta])} \quad (\forall \text{ Introduction})$$

Therefore, what we must prove is that from:

$H \models f[\mathbf{false}/X_\beta]$ and

$H \models \forall X_\beta(f \to f[m/X_\beta])$,

it follows that:

$H \models f[\mu X_\beta.m/X_\beta]$.

Let $H = \{f_0, \ldots, f_n\}$. Take some arbitrary γ, η such that $\bigwedge_{i \in \mathbf{N}} \mathcal{T}[\![f_i]\!]\gamma\eta$ holds.

Then we have the following two facts:

(1)

$\mathcal{T}[\![f[\mathbf{false}/X_\beta]]\!]\gamma\eta$, that is,

$\mathcal{T}[\![f]\!](\gamma)(\eta[Obs[\![\mathbf{false}]\!]\gamma\eta/X_\beta])$, or equivalently,

$\mathcal{T}[\![f]\!](\gamma)(\eta[\emptyset/X_\beta])$.

(2)

$\forall \rho_\beta \in \mathcal{P}(\Delta_\beta)\big(\mathcal{T}[\![f \to f[m/X_\beta]]\!](\gamma)(\eta[\rho_\beta/X_\beta])\big)$, which holds iff

$\forall \rho_\beta \in \mathcal{P}(\Delta_\beta)\big(\mathcal{T}[\![f]\!](\gamma)(\eta[\rho_\beta/X_\beta]) \Rightarrow \mathcal{T}[\![f[m/X_\beta]]\!](\gamma)(\eta[\rho_\beta/X_\beta])\big)$ iff

$\forall \rho_\beta \in \mathcal{P}(\Delta_\beta)\big(\mathcal{T}[\![f]\!](\gamma)(\eta[\rho_\beta/X_\beta]) \Rightarrow \mathcal{T}[\![f]\!](\gamma)(\eta[Obs[\![m]\!](\eta[\rho_\beta/X_\beta])/X_\beta])\big)$.

Define the chain $< \rho^{(i)} >_{i \in \mathbf{N}}$ in $\mathcal{P}(\Delta_\beta)$ as follows:

$$\begin{cases} \rho^{(0)} = \bot = \emptyset \\ \rho^{(i+1)} = Obs[\![m]\!](\gamma)(\eta[\rho^{(i)}/X_\beta]) \quad \text{for } i \geq 0. \end{cases}$$

From point (1) it follows that:

$\mathcal{T}[\![f]\!](\gamma)(\eta[\rho^{(0)}/X_\beta])$,

and from point (2) that, for all $i \in \mathbf{N}$:

$\mathcal{T}[\![f]\!](\gamma)(\eta[\rho^{(i)}/X_\beta]) \Rightarrow \mathcal{T}[\![f]\!](\gamma)(\eta[\rho^{(i+1)}/X_\beta])$.

A simple proof by induction on i yields that:

$$\forall i \in \mathbf{N}\Big(\mathcal{T}[\![f]\!](\gamma)(\eta[\rho^{(i)}/X_\beta])\Big).$$

The anticontinuity of f in X_β then implies that :

$$\mathcal{T}[\![f]\!](\gamma)(\eta[\sqcup_i \rho^{(i)}/X_\beta]).$$

Since $Obs[\![\mu X_\beta.m]\!](\eta) = \sqcup_i \rho^{(i)}$, one sees that:

$\mathcal{T}[\![f]\!](\gamma)(\eta[Obs[\![\mu X_\beta.m]\!](\eta)/X_\beta])$ holds,

which is equivalent to:

$$\mathcal{T}[\![f[\mu X_\beta.m/X_\beta]]\!](\gamma)(\eta),$$

which was to be shown.

□

The Recursion rules for our three proof systems are in fact instances of Scott's Induction rule. This follows from the following lemma.

Lemma 5.4 Admissibility of SAT

Let $m \in \mathit{Mixed}$, $\chi \in \mathit{Assn}$. Then the correctness formula m *sat* χ is admissible.

□

Corollary 5.5

Hoare formulae $\forall \bar{g}[(\varphi)\ m\ (\psi)]$ and Invariant formulae $\forall \bar{g}[I : \{p\}\ m\ \{q\}]$ are admissible.

□

Proof

We must prove the anticontinuity of m *sat* χ in every process variable of $pvar(m \text{ sat } \chi)(= pvar(m))$. So let $X_\beta \in pvar(\chi)$, and let $< \rho_i >_{i\in\mathbf{N}}$ be some chain in $P(\Delta_\beta)$.

The following equivalences hold:

$$\mathcal{T}[\![m \text{ sat } \chi]\!](\gamma)(\eta[\bigcup_{i\in\mathbf{N}} \rho_i/X_\beta]) \text{ iff}$$

$$Obs[\![m]\!](\gamma)(\eta[\bigcup_{i\in\mathbf{N}} \rho_i/X_\beta]) \subseteq [\![\chi]\!]\gamma \text{ iff}$$

$$\Big(\bigcup_{i\in\mathbf{N}} Obs[\![m]\!](\gamma)(\eta[\rho_i/X_\beta])\Big) \subseteq [\![\chi]\!]\gamma \text{ iff}$$

$$\forall i\Big(Obs[\![m]\!](\gamma)(\eta[\rho_i/X_\beta]) \subseteq [\![X]\!]\gamma\Big).$$

This proves the admissibility of SAT formulae.

The admissibility of Hoare formulae follows from the admissibility of SAT formulae and the fact that each Hoare formula $\forall\bar{g}[(\varphi)\ m\ (\psi)]$ is equivalent to the SAT formula $m\ sat\ \forall\bar{g}[\varphi \rightsquigarrow \psi]$. This shows also shows the admissibility of Invariant formulae, as these are (abbreviations of) Hoare formulae.

□

5.7 The soundness of the sat system

We prove the soundness or all SAT axioms and -rules, directly from the definition of the semantics. We recall that the condition for the soundness of the rule:

$$\frac{f_0, \cdots, f_{n-1}}{f_n}$$

is the following semantic condition:

$$\forall\eta \in H \forall\gamma \in \Gamma(\bigwedge_{i<n} \mathcal{T}[\![F_i]\!]\gamma\eta \Rightarrow \mathcal{T}[\![f_n]\!]\gamma\eta).$$

We present the proof as a series of lemmata.

Lemma 5.6

For the assertion $A(\alpha)$, defined for atomic processes and predicative processesα in section 5.2.1, the following equality holds:

$$\alpha = A(\alpha)|base(\alpha).$$

□

Corollary 5.7

The SAT axioms for atomic processes are sound.

□

Proof

Since $A(\alpha)|base(\alpha) \subseteq A(\alpha)$ and $\alpha\ sat\ A(\alpha)$ is just an abbreviation of $\alpha \subseteq A(\alpha)$, the corollary follows directly from the lemma.

The lemma is proven by means of the following semantical calculations.

- Abort, $\mathbf{Z}$

  $Obs[\![\mathbf{abort}]\!]\gamma\eta = \mathbf{Z} = \{(s_0, \varepsilon, \bot) | s_0 \in State_\bot\} =$

  $\{(s_0, h, \bot) | s_0 \in State_\bot\}|(\emptyset, \emptyset) = [\![\bot]\!]\gamma|(\emptyset, \emptyset).$

  The process $\mathbf{Z}$ is treated similarly.

- Skip

  $Obs[\![\mathbf{skip}]\!]\gamma\eta = \mathbf{1} = \{(s_0, \varepsilon, \bot), (s_0, \varepsilon, s_0) | s_0 \in State_\bot\} =$

  $\{(s_0, h, s) | s_0, s \in State_\bot, h \in Trace\}|(\emptyset, \emptyset) = [\![\mathbf{true}]\!]\gamma|(\emptyset, \emptyset).$

- Assign

  Let $\{\bar{y}\} = var(e), \mathbf{z} = \{\bar{y}\} \cup \{x\}, \{\bar{w}\} = \{\bar{y}\} - \{x\}.$

  $Obs[\![x := e]\!]\gamma\eta = Close\Big(\{(s_0, \varepsilon, s_0[\mathcal{E}[\![e]\!]s_0/x]) \mid s_0 \in State\}\Big) =$

  $\mathbf{Z} \cup \{(s_0, \varepsilon, s_0[\mathcal{E}[\![e]\!]s_0/x]) | s_0 \in State\} = \quad (*)$

  $[\![\bot]\!]\gamma|(\emptyset, \mathbf{z}) \cup [\![\top \wedge x = e[\bar{x}^\circ/\bar{x}]]\!]\gamma|(\emptyset, \{x\}) =$

  $[\![\bot]\!]\gamma|(\emptyset, \mathbf{z}) \cup [\![\top \wedge x = e[\bar{x}^\circ/\bar{x}] \wedge \bar{w} = \bar{w}^\circ]\!]\gamma|(\emptyset, \mathbf{z}) =$

  $[\![\bot \vee (\top \wedge x = e[\bar{x}^\circ/\bar{x}]) \wedge \bar{w} = \bar{w}^\circ]\!]\gamma|(\emptyset, \mathbf{z}) =$

  $[\![(\top \rightarrow x = e[\bar{x}^\circ/\bar{x}] \wedge \bar{w} = \bar{w}^\circ)]\!]\gamma|(\emptyset, \mathbf{z}).$

  (As was to be shown). Here the equality $(*)$ follows from the assumptions made on the $\mathcal{E}$ functions in chapter 4. These assumptions imply that if a computation δ is of the form $(s_0, \varepsilon, s_0[\mathcal{E}[\![e]\!]s_0/x])$, then we have the following equalities:

  $\mathcal{E}[\![x]\!]\gamma\delta = (s_0[\mathcal{E}[\![e]\!]s_0/x])(x) = \mathcal{E}[\![e]\!]s_0,$ and

  $\mathcal{E}[\![e[\bar{x}^\circ/\bar{x}]]\!]\gamma\delta = \mathcal{E}[\![e]\!]s_0.$

  So if δ is of the this form then $\delta \in [\![x = e[\bar{x}^\circ/\bar{x}]]\!]\gamma.$

- Guard

  Let $\{\bar{w}\} = var(b).$

  $Obs[\![b]\!]\gamma\eta = Close\Big(\{(s_0, \varepsilon, s_0) | s_0 \in State, \mathcal{B}[\![b]\!]s_0\}\Big)$

  $= \mathbf{Z} \cup \{(s_0, \varepsilon, s_0) | s_0 \in State, \mathcal{B}[\![b]\!]s_0\} =$

  $[\![\bot]\!]\gamma|(\emptyset, \{\bar{w}\}) \cup [\![\top \wedge b]\!]\gamma|(\emptyset, \emptyset) =$

  $[\![\bot]\!]\gamma|(\emptyset, \{\bar{w}\}) \cup [\![\top \wedge b \wedge \bar{w} = \bar{z}^\circ]\!]\gamma|(\emptyset, \{\bar{w}\}) =$

  $[\![\top \rightarrow (b \wedge \bar{w} = \bar{w}^\circ)]\!]\gamma|(\emptyset, \{\bar{w}\}).$

- Com Let $\mathbf{z} = var(b) \cup \{x\}, \{\bar{w}\} = var(b) - \{x\}$.

  $Obs[\![c.x\!:\!b]\!]\gamma\eta =$

  $Close\Big(\{(s_0, < (c, v) >, s_0[v/x])\}|s_0 \in State, \mathcal{B}[\![b]\!](s_0[v/x])\}\Big) =$

  $\mathbf{Z} \cup \{(s_0, < (c, v) >, \perp), (s_0, < (c, v) >, s_0[v/x])|$
  $s_0 \in State, \mathcal{B}[\![b]\!](s_0[v/x])\} =$

  $[\![\perp]\!](\emptyset, \mathbf{z}) \cup$
  $[\![\exists v[c =< v > \wedge b[\bar{w}^\circ/\bar{w}, v/x] \wedge (\top \rightarrow x = v)]]\!]\gamma|(\{c\}, \{x\}) =$

  $[\![\perp \wedge c = \varepsilon]\!]\gamma|(\{c\}, \mathbf{z}) \cup$
  $[\![\exists v[- - -] \wedge \bar{w} = \bar{w}^\circ]\!]\gamma|(\{c\}, \mathbf{z}) =$

  $[\![(\perp \wedge c = \varepsilon) \vee (\exists v[- - -] \wedge \bar{w} = \bar{w}^\circ]\!]\gamma(\{c\}, \mathbf{z}))$.

- Predicative processes

  We must prove that $\chi|\beta = A(\chi|\beta)|base(\chi|\beta)$. This is obvious, since $A(\chi|\beta) = \chi$ and $base(\chi|\beta) = \beta$.

□

Lemma 5.8

The proof rules of the SAT system are sound.

Proof

- *Hiding rules*

For both hiding rules the premisse states that:

$Obs[\![m]\!]\gamma\eta \subseteq [\![\chi]\!]\gamma$

The restrictions for these rules are:

(a) $var(\chi) \subseteq \mathcal{V}ar - \mathbf{x}$, for variable hiding, and

(b) $chan(\chi) \subseteq Chan - \mathbf{c}$, for channel hiding.

From (a) it follows, by corollary 4.13, that

$\rho \subseteq [\![\chi]\!]\gamma$ iff $\rho|(\mathcal{V}ar - \mathbf{x}) \subseteq [\![\chi]\!]\gamma$.

And so, from the premisse, we can infer that:

$$Obs[\![m_1\backslash \mathbf{x}]\!]\gamma\eta = Obs[\![m]\!]\gamma\eta|(\mathcal{V}ar - \mathbf{x}) \subseteq [\![\chi]\!]\gamma$$

By the same corollary, and (b) we see that

$$\rho \subseteq [\![\chi]\!]\gamma \text{ iff } \rho|(Chan - \mathbf{c}) \subseteq [\![\chi]\!]\gamma,$$

Hence:

$$Obs[\![m_1\backslash \mathbf{c}]\!]\gamma\eta = Obs[\![m]\!]\gamma\eta|(Chan - \mathbf{c}) \subseteq [\![\chi]\!]\gamma.$$

• *Renaming*

First note that:

$$\mathcal{TE}[\![h[d/c]]\!]\gamma(s_0, t, s) = (\mathcal{TE}[\![h]\!]\gamma(s_0, t, s))[d/c]$$
$$= t[d/c] = \mathcal{TE}[\![h]\!]\gamma(s_0, t[d/c], s).$$

Since h is the only atomic expression that depends on the trace component of a computation δ, one sees that the following holds:

$$\mathcal{T}[\![\chi[h[d/c]/h]]\!]\gamma\delta \text{ iff } \mathcal{T}[\![\chi]\!]\gamma(\delta[d/c])$$

This implies that

$$\Big([\![\chi[h[d/c]/h]]\!]\gamma\Big)[d/c] = [\![\chi]\!]\gamma.$$

Remark Note that, in general, $[\![\chi[d/c]]\!]\gamma \neq \Big([\![\chi]\!]\gamma\Big)[d/c]$!

Now assume that the premisse of the rule holds, i.e. that:

$$Obs[\![m]\!]\gamma\eta \subseteq [\![\chi[h[d/c]/h]]\!]\gamma.$$

Then from the monotonicity of the renaming operation $[d/c]$ follows:

$$Obs[\![m\langle d/c\rangle]\!]\gamma\eta = \Big(Obs[\![m]\!]\gamma\eta\Big)[d/c] \subseteq$$

$$\Big([\![\chi[h[d/c]/h]]\!]\gamma\Big)[d/c] = [\![\chi]\!]\gamma$$

This proves the soundness of the rule.

• *Kernel*

The premisse of the rule states that:

$$Obs[\![m]\!]\gamma\eta \subseteq [\![\chi]\!]\gamma.$$

From this, and the monotonicity of the *Kern* operation, it follows directly that:

$$Obs[\![Kern(m)]\!]\gamma\eta \subseteq Kern([\![\chi]\!]\gamma) = [\![Kern(\chi)]\!]\gamma$$

• *Sequential composition and nondeterministic choice*

The premisses of both rules state that:

$Obs[\![m_i]\!]\gamma\eta \subseteq [\![\chi_i]\!]\gamma$ for $i = 1, 2$

From this the desired results follows easily:

(*i*) $Obs[\![m_1; m_2]\!]\gamma\eta = Obs[\![m_1]\!]\gamma\eta \hat{\circ} Obs[\![m_2]\!]\gamma\eta \subseteq$ (monotony of $\hat{\circ}$)
$[\![\chi_1]\!]\gamma \hat{\circ} [\![\chi_2]\!]\gamma = [\![\chi_1 \hat{\circ} \chi_2]\!]\gamma$, and

(*ii*) $Obs[\![S_1 \textbf{ or } S_2]\!]\gamma\eta = Obs[\![m_1]\!]\gamma\eta \cup Obs[\![m_2]\!]\gamma\eta \subseteq$ (monotony of $\cup$)
$[\![\chi_1]\!]\gamma \cup [\![\chi_2]\!]\gamma = [\![\chi_1 \vee \chi_2]\!]\gamma$.

• *Parallel composition*

The premisses of the rule state that:

(1) $Obs[\![m_i]\!]\gamma\eta \subseteq [\![\chi_i]\!]\gamma$ for $i = 1, 2$.

We reformulate the restrictions on the free variables of the χ_i:

$abase(\chi_i) \subseteq \beta_i \cup ((Chan, Var) - \beta_j)$,

for $(i, j) = (1, 2)$ and for $(i, j) = (2, 1)$.

From this we must prove that $Obs[\![m_1\ \beta_1 \,\|\, \beta_2\ m_2]\!]\gamma\eta \subseteq [\![\chi_1 \wedge \chi_2]\!]\gamma$, where $Obs[\![m_1\ \beta_1 \,\|\, \beta_2\ m_2]\!]\gamma\eta$ is determined as the largest set $\rho \in P(\Delta)$ such that:

(2) $\rho|\beta_i \subseteq Obs[\![m_i]\!]\gamma\eta$, and

(3) $base(\rho) \subseteq (\beta_1 \cup \beta_2)$

That is, for ρ as above, we must prove that $\rho \subseteq [\![\chi_i]\!]\gamma$.

Now from the restrictions, it follows by corollary 4.13 of chapter 4 that, for (i, j) as above:

(4) $\rho \subseteq [\![\chi_i]\!]\gamma$ iff

$\rho|(\beta_i \cup ((Chan, Var) - \beta_j)) \subseteq [\![\chi_i]\!]\gamma$

Also, property (3) is equivalent to:

(5) $\rho = \rho|(\beta_1 \cup \beta_2)$.

Since $\rho|(\beta_1 \cup \beta_2)|(\beta_i \cup ((Chan, Var) - \beta_j)) = \rho|\beta_i$, we see that from (4) and (5) it follows that:

(6) $\rho \subseteq [\![\chi_i]\!]\gamma$ iff $\rho|\beta_i \subseteq [\![\chi_i]\!]\gamma$, for $i = 1, 2$.

Combining this with (1),(2) yields:

(7) $\rho \subseteq [\![\chi_i]\!]\gamma$ for $i = 1, 2$, that is :

$\rho \subseteq [\![\chi_1 \wedge \chi_2]\!]\gamma$.

(As was to be shown).

• *Recursion*

This case follows from Scott's Induction rule and the fact that SAT formulae are admissible. The first premisse of Scott's rule becomes $H \vdash \mathbf{false}\ sat\ \chi$. Since this is a valid formula, the first premisse can be *omitted* from the rule, and only the second premisse remains. The second premisse is, for the case of SAT formulae:

$H \cup \{X_\beta\ sat\ \chi\} \vdash m\ sat\ \chi$.

This is exactly the premisse of the SAT rule for recursion.

• μ_z *Recursion*

From the premisse $H \vdash \mathbf{Z}\ sat\ \chi$ it follows by means of the weakening rule that

$H \cup \{X_\beta\ sat\ \chi\} \vdash \mathbf{Z}\ sat\ \chi$.

This is one of the premisses of the following derivation:

$$\dfrac{\dfrac{\dfrac{H \cup \{X_\beta\ sat\ \chi\} \vdash \mathbf{Z}\ sat\ \chi\ ,\ H \cup \{X_\beta\ sat\ \chi\} \vdash m\ sat\ \chi}{H \cup \{X_\beta\ sat\ \chi\} \vdash m\ \mathbf{or}\ \mathbf{Z}\ sat\ \chi \vee \chi}\ (\text{Choice})}{H \cup \{X_\beta\ sat\ \chi\} \vdash m\ \mathbf{or}\ \mathbf{Z}\ sat\ \chi}\ (\text{Consequence})}{H \vdash \mu X_\beta.(m\ \mathbf{or}\ \mathbf{Z})\ sat\ \chi}\ (\text{Recursion})$$

The conclusion of this derivation is (abbreviated by) $H \vdash \mu_z X_\beta.m\ sat\ \chi$, which was to be shown.

• *Process naming*

Note that this rule is not of the restricted format, but rather is a natural deduction rule of the general form. We must show the following:

$\left(H \models m_1\ sat\ \chi_1 \ \wedge\ H \cup \{\xi_1\ sat\ \chi_1\} \models m_2\ sat\ \chi_2\right) \Rightarrow$

$H \models \xi_1 = m_1 \mathbf{in}\ m_2\ sat\ \chi_2$

This amounts to the following:

If

$$\forall\eta\forall\gamma\Big(\mathcal{T}[\![H]\!]\gamma\eta \Rightarrow \mathit{Obs}[\![m_1]\!]\gamma\eta \subseteq [\![\chi_1]\!]\gamma\Big) \text{ and}$$

$$\forall\eta\forall\gamma\Big((\mathcal{T}[\![H]\!]\gamma\eta \wedge \eta(\xi_1) \subseteq [\![\chi_1]\!]\gamma) \Rightarrow \mathit{Obs}[\![m_2]\!]\gamma\eta \subseteq [\![\chi_2]\!]\gamma\Big)$$

then:

$$\forall\eta\forall\gamma\Big(\mathcal{T}[\![H]\!]\gamma\eta \Rightarrow \mathit{Obs}[\![\xi_1 = m_1 \mathbf{in}\ m_2]\!]\gamma\eta \subseteq [\![\chi_2]\!]\gamma\Big)$$

So choose some arbitrary η and γ such that $\mathcal{T}[\![H]\!]\gamma\eta$. From the first implication it follows that $\mathit{Obs}[\![m_1]\!]\gamma\eta \subseteq [\![\chi_1]\!]\gamma$. Let $\eta' = \eta[\mathit{Obs}[\![m_1]\!]\gamma\eta/\xi_1]$. From the restriction for the rule it is known that $\xi \notin \mathit{pvar}(H)$. Therefore $\mathcal{T}[\![H]\!]\eta'\gamma$ holds too. Moreover, $\eta'(\xi_1) = \mathit{Obs}[\![m_1]\!]\gamma\eta \subseteq [\![\chi_1]\!]\gamma$. But then we can infer from the second implication that $\mathit{Obs}[\![m_2]\!]\gamma\eta' \subseteq [\![\chi_2]\!]\gamma$. Since $\mathit{Obs}[\![\xi_1 = m_1\mathbf{in}; m_2]\!]\gamma\eta = \mathit{Obs}[\![m_2]\!]\gamma\eta'$, we conclude that

$$\mathit{Obs}[\![\xi_1 = m_1\mathbf{in}; m_2]\!]\gamma\eta \subseteq [\![\chi_2]\!]\gamma,$$

as was to be shown.

- *Invariance*

From the theorem following the semantic definition, we know that:

$$\mathit{Obs}[\![m]\!]\gamma\eta \in \mathcal{P}(\Delta_\beta), \text{ where} \beta = (\mathbf{c}, \mathbf{x}) = \mathit{base}(m).$$

That is, for any $\delta = (s_0, h, s) \in \mathit{Obs}[\![m]\!]\gamma\eta$, we have the inclusion: $\mathit{base}(\delta) \subseteq \mathit{base}(m)$ Therefore, if $\mathbf{d}, \bar{y}$ are such that $(\mathbf{d}, \bar{y}) \cap \mathit{base}(m) = (\emptyset, \emptyset)$, then $h|\mathbf{d} = \varepsilon$ and if $s \neq \perp$ then $s_0(\bar{y}) = s(\bar{y})$ We conclude that $\mathcal{T}[\![h|\mathbf{d} = \varepsilon \wedge (\top \rightarrow \bar{y} = \bar{y}^\circ)]\!]\gamma\delta = \mathbf{true}$ for all $\delta \in \mathit{Obs}[\![m]\!]\gamma\eta$, as was to be shown.

- *Conjunction*

The premisses of the rule state that:

$$\mathit{Obs}[\![m]\!]\gamma\eta \subseteq [\![\chi_i]\!]\gamma \text{ for } i = 1, 2$$

This implies that

$$\mathit{Obs}[\![m]\!]\gamma\eta \subseteq [\![\chi_1]\!]\gamma \cap [\![\chi_2]\!]\gamma = [\![\chi_1 \wedge \chi_2]\!]\gamma.$$

- *Consequence*

From the first premisse we have:

$$\mathit{Obs}[\![m]\!]\gamma\eta \subseteq [\![\chi_1]\!]\gamma \quad (*)$$

The second premisse, $\forall_\perp[\chi_1 \rightarrow \chi_2]$, holds iff $\chi_1 \rightarrow \chi_2$ is a strictly valid assertion. That is, for any $\gamma \in \Gamma$, if $\mathcal{T}[\![\chi_1]\!]\gamma\delta$, then $\mathcal{T}[\![\chi_2]\!]\gamma\delta$. We see that, for any γ, $[\![\chi_1]\!]\gamma \subseteq [\![\chi_2]\!]\gamma$.
Together with $(*)$ we can conclude that:

$$\mathit{Obs}[\![m]\!]\gamma\eta \subseteq [\![\chi_2]\!]\gamma.$$

(As was to be shown.)

5.8 The soundness of the Hoare system

5.8.1 The Hoare SAT transformation

Let $A(\alpha)$ and $A_\beta(\alpha)$ as defined in the section on axioms for atomic processes, where $\beta = base(\alpha) \cup abase(\psi)$. Since for assertion χ, ψ the implication:

$$\chi \rightarrow (\chi \rhd \psi) \rightsquigarrow \psi$$

is valid, we have the following derivation: Let $\beta' \stackrel{\text{def}}{=} abase(\psi) - base(\alpha)$

$$\dfrac{\dfrac{\dfrac{\alpha \; sat \; A(\alpha) \;\; \text{(SAT axiom)} \quad , \quad \alpha \; sat \; \mathbf{1}_{\beta'} \;\; \text{(invariance)}}{\alpha \; sat \; A_\beta(\alpha)} \;\; \text{(Conjunction)}}{\alpha \; sat \; (A_\beta(\alpha) \rhd \psi) \rightsquigarrow \psi} \;\; \text{(Consequence)}}{(A_\beta(\alpha) \rhd \psi)\, \alpha\, (\psi)} \;\; \text{(SH)}$$

Therefore, to prove the soundness of the Hoare axioms for atomic processes, all we need to do is to show that they are actually of the form:

$$(A_\beta(\alpha) \rhd \psi)\, \alpha\, (\psi).$$

This is done as follows, using the representation for $\chi \rhd \psi$ provided in lemma 4.31. In all cases below, let $\beta = (\mathbf{c}, \{\bar{x}\})$.

- Skip

  We must calculate $A_\beta(\mathbf{skip})$. To this end, we first calculate a few sub expressions of this assertion:

  $A(\mathbf{skip}) \equiv \mathbf{true}$, so

  - $A_\beta(\mathbf{skip}) \equiv h|\mathbf{c} = \varepsilon \wedge (\top \rightarrow \bar{x} = \bar{x}^\circ)$
  - $A_\beta(\mathbf{skip})[\bot^\circ]$ iff $\varepsilon|\mathbf{c} = \varepsilon \wedge \mathbf{false} \rightarrow \mathbf{false}$ iff $\mathbf{true}$
  - $A_\beta(\mathbf{skip})[\top^\circ][\bot][t_1/h]$ iff $t_1|\mathbf{c} = \varepsilon$
  - $A_\beta(\mathbf{skip})[\top][t_1/h]$ iff $t_1|\mathbf{c} = \varepsilon \wedge \bar{x} = \bar{x}^\circ$

  Therefore, $A_\beta(\mathbf{skip}) \rhd \psi$ iff

  $(\bot \wedge (\mathbf{true} \rightarrow \psi[\bot])) \vee$

  $(\top \wedge \forall t_1 \forall_\bot \times (A_\beta(\mathbf{skip})[\top^\circ][t_1/h] \rightarrow \psi[h t_1/h])[\bar{x}/\bar{x}^\circ])$ iff

$(\bot \wedge \psi[\bot]) \vee$

$(\top \wedge \forall t_1(t_1|\mathbf{c} = \varepsilon \rightarrow \psi[\bot][ht_1/h]) \wedge \forall t_1 \forall_\bot \times ((t_1|\mathbf{c} = \varepsilon \wedge \bar{x} = \bar{x}^\circ)$

$\rightarrow \psi[\top][ht_1/h])[\bar{x}/\bar{x}^\circ])$ iff

$(\bot \wedge \psi[\bot]) \vee (\top \wedge \psi[\bot] \wedge (\psi[\top][\bar{x}^\circ/\bar{x}]))[\bar{x}/\bar{x}^\circ]$ iff

$\psi[\bot] \wedge (\top \rightarrow \psi[\top])$ iff

$\psi[\bot] \wedge \psi$

This last assertion is indeed the precondition of the **skip** axiom in the Hoare system, as was to be shown.

- Abort, **Z**

  $A(\mathbf{abort}) \equiv \bot$ so $A_\beta(\mathbf{abort}) \equiv (\bot \wedge h|\mathbf{c} = \varepsilon)$

  For the purpose of calculating $A_\beta(\mathbf{abort}) \rhd \psi$ we remark that:

  - $A_\beta(\mathbf{abort})[\bot^\circ]$ iff $\mathbf{true} \wedge \varepsilon|\mathbf{c} = \varepsilon$ iff $\mathbf{true}$
  - $A_\beta(\mathbf{abort})[\top^\circ][\bot][t_1/h]$ iff $\mathbf{true} \wedge t_1|\mathbf{c} = \varepsilon$ iff $t_1|\mathbf{c} = \varepsilon$
  - $A_\beta(\mathbf{abort})[\top][t_1/h]$ iff $\mathbf{false} \wedge t_1|\mathbf{c} = \varepsilon$ iff $\mathbf{false}$

  Therefore we see that:

  $A_\beta(\mathbf{abort}) \rhd \psi$ iff

  $(\bot \wedge (\mathbf{true} \rightarrow \psi[\bot])) \vee$

  $(\top \wedge \forall t_1 \forall_\bot x(A_\beta(\mathbf{abort})[\top^\circ][t_1/h] \rightarrow \psi[ht_1/h])[\bar{x}/\bar{x}^\circ])$ iff

  $(\bot \wedge \psi[\bot]) \vee$

  $(\top \wedge \forall t_1(t_1|\mathbf{c} = \varepsilon \rightarrow \psi[\bot][ht_1/h]) \wedge \forall t_1 \underline{\forall} x(\mathbf{false} \rightarrow \cdots))$ iff

  $(\bot \wedge \psi[\bot]) \vee (\top \wedge \psi[\bot])$ iff

  $\psi[\bot]$

  Here we used the fact that, if $t_1|\mathbf{c} = \varepsilon$ then $\psi[\bot][ht_1/h]$ iff $\psi[\bot]$ since $hchan(\psi) \subseteq \mathbf{c}$. We conclude that $(A_\beta(\mathbf{abort}) \rhd \psi)\mathbf{abort}(\psi)$ is the formula: $(\psi[\bot])\mathbf{abort}(\psi)$, as was to be shown.

  The case **Z** is of course completely similar.

- Guard

  $A(b) \equiv \top\rightarrow(b \wedge \bar{w} = \bar{w}^\circ)$, where $\{\bar{w}\} = var(b)$, so

  – $A_\beta(b) \equiv \top\rightarrow(b \wedge \bar{x} = \bar{x}^\circ) \wedge (h|\mathbf{c} = \varepsilon)$, and

  – $A_\beta(b)[\bot^\circ]$ iff $\mathbf{false}\rightarrow(\mathbf{false} \wedge \varepsilon|\mathbf{c} = \varepsilon)$ iff $\mathbf{true}$

  – $A_\beta(b)[\top^\circ][\bot][t_1/h]$ iff $t_1|\mathbf{c} = \varepsilon$

  – $A_\beta(b)[\top][t_1/h]$ iff $b \wedge \bar{x} = \bar{x}^\circ \wedge t_1|\mathbf{c} = \varepsilon$

  Therefore: $A_\beta(b) \rhd \psi$ iff

  $(\bot \wedge (\mathbf{true}\rightarrow\psi[\bot])) \vee$

  $(\top \wedge \forall t_1 \forall_\bot x(A_\beta(b)[\top^\circ][t_1/h]\rightarrow\psi[ht_1/h])[\bar{x}/\bar{x}^\circ])$ iff

  $(\bot \wedge \psi[\bot]) \vee$

  $(\top \wedge \forall t_1(t_1|\mathbf{c} = \varepsilon\rightarrow\psi[\bot][t_1h/h]) \wedge$

  $\forall t_1 \underline{\forall} x((b \wedge \bar{x} = \bar{x}^\circ \wedge t_1|\mathbf{c} = \varepsilon)\rightarrow\psi[\top][ht_1/h])[\bar{x}/\bar{x}^\circ])$ iff

  $(\bot \wedge \psi[\bot]) \vee (\top \wedge \psi[\bot] \wedge (b\rightarrow\psi[\top][\bar{x}^\circ/\bar{x}])[\bar{x}/\bar{x}^\circ])$ iff

  $\psi[\bot] \wedge (\top \wedge b)\rightarrow\psi[\top]$

  We conclude that $(A_\beta(b) \rhd \psi)b(\psi)$ is the formula:

  $(\psi[\bot] \wedge (\top \wedge b)\rightarrow\psi[\top])b(\psi)$.

- Assign

  $A(x := e) \equiv \top\rightarrow(x = e[\bar{z}^\circ/\bar{z}] \wedge \bar{w} = \bar{w}^\circ)$,

  where $\{\bar{z}\} = var(e), \{\bar{w}\} = var(e) - \{x\}$.

  So:

  – $A_\beta(x := e) \equiv (\top\rightarrow(x = e[\bar{z}^\circ/\bar{z}] \wedge \bar{y} = \bar{y}^\circ)) \wedge (h|\mathbf{c} = \varepsilon)$,

  where $\{\bar{y}\} = \{\bar{x}\} - \{x\}$

  – $A_\beta(x := e)[\bot^\circ]$ iff $(\mathbf{false}\rightarrow\cdots) \wedge \varepsilon|\mathbf{c} = \varepsilon$ iff $\mathbf{true}$

  – $A_\beta(x := e)[\top^\circ][\bot][t_1/h]$ iff $(\mathbf{false}\rightarrow\cdots) \wedge t_1|\mathbf{c} = \varepsilon$ iff $t_1|\mathbf{c} = \varepsilon$

  – $A_\beta(x := e)[\top][t_1/h]$ iff $x = e[\bar{z}^\circ/\bar{z}] \wedge \bar{y} = \bar{y}^\circ \wedge t_1|\mathbf{c} = \varepsilon$

  Therefore, $A_\beta(x := e) \rhd \psi$ iff

  $(\bot \wedge (\mathbf{true}\rightarrow\psi[\bot])) \vee$

$(\top \wedge \forall t_1 \forall_\perp x(A_\beta(x := e)[\top^\circ][t_1/h] \rightarrow \psi[ht_1/h])[\bar{x}/\bar{x}^\circ])$ iff

$(\perp \wedge \psi[\perp]) \vee$

$(\top \wedge \forall t_1(t_1|\mathbf{c} = \varepsilon \rightarrow \psi[\perp][ht_1/h]) \wedge$

$\forall t_1 \underline{\forall} x((x = e[\bar{z}^\circ/\bar{z}] \wedge \bar{y} = \bar{y}^\circ \wedge t_1|\mathbf{c} = \varepsilon) \rightarrow \psi[\top][ht_1/h])[\bar{x}/\bar{x}^\circ])$ iff

$(\perp \wedge \psi[\perp]) \vee$

$(\top \wedge \psi[\perp] \wedge (\psi[\top][e[\bar{z}^\circ/\bar{z}]/x, \bar{y}^\circ/\bar{y}])[\bar{x}/\bar{x}^\circ])$ iff

$\psi[\perp] \wedge (\top \rightarrow \psi[\perp][e/x])$

So we can conclude that $(A_\beta(x := e) \rhd \psi)\ x := e\ (\psi)$ is the assertion:

$(\psi[\perp] \wedge (\top \rightarrow \psi[\perp][e/x]))\ x := e\ (\psi)$.

- Communication

$A(c.x{:}b) \equiv (\perp \wedge c = \varepsilon) \vee$

$\exists v\Big[c =< v > \wedge b[\bar{w}^\circ/\bar{w}][v/x] \wedge (\top \rightarrow (v = x \wedge \bar{w} = \bar{w}^\circ))\Big]$,

where $\bar{w} = var(b) - \{x\}$.

Therefore,

– $A_\beta(c.x{:}b) \equiv (\perp \wedge h|\mathbf{c} = \varepsilon) \vee$

$\Big(h|\mathbf{d} = \varepsilon \wedge \exists v\Big[c =< v > \wedge b[\bar{w}^\circ/\bar{w}][v/x] \wedge (\top \rightarrow (v = x \wedge \bar{y} = \bar{y}^\circ))\Big]\Big)$,

where $\mathbf{d} = \mathbf{c} - \{c\}, \{\bar{y}\} = \{\bar{x}\} - \{x\}$.

– $A_\beta(c.x{:}b)[\perp^\circ]$ iff $(\mathbf{true} \wedge \varepsilon|\mathbf{c} = \varepsilon) \vee \cdots$ iff $\mathbf{true}$

– $A_\beta(c.x{:}b)[\top^\circ][\perp][t_1/h]$ iff $t_1|\mathbf{c} = \varepsilon \vee$

$\Big(t_1|\mathbf{d} = \varepsilon \wedge \exists v[t_1|c =< v > \wedge b[\bar{w}^\circ/\bar{w}][v/x]]\Big)$

– $A_\beta(c.x{:}b)[\top][t_1/h]$ iff

$t_1|\mathbf{d} = \varepsilon \wedge \exists v[t_1|c < v > \wedge b[\bar{w}^\circ/\bar{w}][v/x] \wedge v = x \wedge \bar{y} = \bar{y}^\circ]$

We use this to calculate $A_\beta(c.x{:}b) \rhd \psi$ as follows:

$A_\beta(c.x{:}b) \rhd \psi$

iff

$(\perp \wedge (\mathbf{true} \rightarrow \psi[\perp])) \vee$

$\Big(\top \wedge \forall t_1 \forall_\perp x\Big(A_\beta(c.x{:}b)[\top^\circ][t_1/h] \rightarrow \psi[ht_1/h]\Big)[\bar{x}/\bar{x}^\circ]\Big)$

iff

$(\bot \wedge \psi[\bot]) \vee$

$\Big(\top \wedge \forall t_1((t_1|\mathbf{c} = \varepsilon \vee (t_1|\mathbf{d} = \varepsilon \wedge \exists v[t_1|c =< v > \wedge b[\bar{w}^\circ/\bar{w}][v/x]]))$

$\rightarrow \psi[\bot][ht_1/h])[\bar{x}/\bar{x}^\circ] \wedge \forall t_1 \forall_\bot x((t_1|\mathbf{d} = \varepsilon \wedge \exists v[t_1|c =< v > \wedge b[\bar{w}^\circ/\bar{w}][v/x] \wedge$
$v = x \wedge \bar{y} = \bar{y}^\circ)] \rightarrow \psi[\top][ht_1/h])[\bar{x}/\bar{x}^\circ]\Big)$

iff

$(\bot \wedge \psi[\bot]) \vee (\top \wedge \psi[\bot] \wedge \forall v(b[v/x] \rightarrow \psi[\bot][h < (c, v) > /h])$
$\wedge \forall v(b[v/x] \rightarrow \psi[\top][h < (c, v) > /h][v/x]))$

iff

$\psi[\bot] \wedge \top \rightarrow \forall v(b[v/x] \rightarrow (\psi[\bot][h < (c, v) >] \wedge$
$\psi[\top][h < (c, v) > /h][v/x])).$

This last assertion is the precondition of the communication axiom within the Hoare system, as was to be shown.

- Predicative processes

  The following simple derivation suffices:

$$\frac{(\chi|\beta)\ sat\ \chi \quad \text{(Predicative process)}}{(\varphi)\ (\chi|\beta)\ (\varphi \lhd \chi)}\ \text{(SP)}$$

This ends the soundness proof for the Hoare axioms for atomic processes. The next step is to treat the **TNP** constructs.

- *Hiding rules*

Assume for the channel hiding rule that $hchan(\psi) \cap \mathbf{c} = \emptyset$ and for the variable hiding rule that $var(\psi) \cap \mathbf{x} = \emptyset$.

Since $hchan(\varphi \rightsquigarrow \psi) = hchan(\psi)$ and $var(\varphi \rightsquigarrow \psi) = var(\psi)$, the following derivations are possible:

$$\frac{(\varphi)\ m\ (\psi)}{}\ \text{(HS)}$$

$$\frac{m\ sat\ (\varphi \rightsquigarrow \psi)}{}\ \text{(channel hiding in sat system)}$$

$$\frac{m\backslash\mathbf{c}\ sat\ (\varphi \rightsquigarrow \psi)}{}\ \text{(SH)}$$

$$(\varphi)\ m\backslash\mathbf{c}\ (\psi)$$

$$\frac{(\varphi)\, m\, (\psi)}{m \textit{ sat } (\varphi \rightsquigarrow \psi)}\ \text{(HS)}$$

$$\frac{m \textit{ sat } (\varphi \rightsquigarrow \psi)}{m\backslash \mathbf{x} \textit{ sat } (\varphi \rightsquigarrow \psi)}\ \text{(var hiding in sat system)}$$

$$\frac{m\backslash \mathbf{x} \textit{ sat } (\varphi \rightsquigarrow \psi)}{(\varphi)\, m\backslash \mathbf{x}\, (\psi)}\ \text{(SH)}$$

• *Renaming*

Let c' be fresh, and let $\tilde{\varphi}$ and $\tilde{\psi}$ be as follows:

$$\tilde{\varphi} \stackrel{\text{def}}{=} \varphi[h[c/c']/h] \wedge (c = \varepsilon),$$

$$\tilde{\psi} \stackrel{\text{def}}{=} \psi[h[d/c][c/c']/h].$$

We prove that the following implication is valid:

$$((\tilde{\varphi} \rightsquigarrow \tilde{\psi}) \wedge c' = \varepsilon) \rightarrow ((\varphi \rightsquigarrow \psi)[h[d/c]/h]).$$

Therefore assume:

$$(\tilde{\varphi} \rightsquigarrow \tilde{\psi}) \wedge c' = \varepsilon. \qquad (1)$$

By expanding the definition of the $\rightsquigarrow$ operator one sees that this is equivalent to:

$$\forall t_0 \Big(\tilde{\varphi}[t_0/h, \bar{x}^\circ/\bar{x}, \top^\circ/\top] \rightarrow \tilde{\psi}[t_0 h/h]\Big) \wedge (c' = \varepsilon),$$

that is:

$$\forall t_0 \Big((\varphi[t_0[c/c']/h, \bar{x}^\circ/\bar{x}, \top^\circ/\top] \wedge t_0|c = \varepsilon) \rightarrow$$

$$\psi[(t_0 h)[d/c][c/c']/h]\Big) \wedge (c' = \varepsilon). \qquad (2)$$

From (2) we must prove:

$$\forall t_0' \Big(\varphi[t_0'/h, \bar{x}^\circ/\bar{x}, \top^\circ/\top] \rightarrow \psi[t_0'(h[d/c])/h]\Big). \qquad (3)$$

So take some arbitrary t_0' and assume $\varphi[t_0'/h, \bar{x}^\circ/\bar{x}, \top^\circ/\top]$. We must then prove that $\psi[t_0'(h[d/c])/h]$ holds. Since c' is fresh, so in particular does not occur free within φ or ψ, we may assume, without loss of generality, that $t_0'|c' = \varepsilon$ for this proof.

We now instantiate (2), where we choose t_0 to be $t_0'[c'/c]$.

It is easily checked that:

$t_0[c/c'] = t_0'[c'/c][c/c'] = t_0'$ and that

$t_0|c = t_0'[c'/c]|c = \varepsilon$.

Therefore we can use the implication in (2) to infer $\psi[(t_0 h)[d/c][c/c']/h]$. By the assumption on t_0' and the conjunct $c' = \varepsilon$ in (2), it is seen that

$$(t_0 h)[d/c][c/c'] = (t_0'[c'/c][d/c][c/c'])\hat{}(h[d/c][c/c']) = t_0'(h[d/c]).$$

But then we can conclude that $\psi[t_0'(h[d/c])/h]$ holds, as was to be shown.

We use the implication that we just proved, in the the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\tilde{\varphi})\ m\ (\tilde{\psi})}{m\ sat\ (\tilde{\varphi} \rightsquigarrow \tilde{\psi})}\ \text{(HS)} \quad m\ sat\ (c' = \varepsilon)\ \text{(Invariance)}}{m\ sat\ ((\tilde{\varphi} \rightsquigarrow \tilde{\psi}) \wedge c' = \varepsilon)}\ \text{(Conjunction)}}{m\ sat\ ((\varphi \rightsquigarrow \psi)[h[d/c]/h])}\ \text{(Consequence)}}{m\langle d/c\rangle\ sat\ (\varphi \rightsquigarrow \psi)}\ \text{(Renaming)}}{(\varphi)\ m\langle d/c\rangle\ (\psi)}\ \text{(SH)}$$

•*Kernel*

Let $\mathbf{c} = hchan(\psi)$, and let $\varphi' = \varphi \wedge (h|\mathbf{c} = t_0|\mathbf{c})$.

$$\dfrac{\dfrac{\dfrac{\dfrac{(\varphi)m_1(\psi)}{m_1\ sat\ (\varphi \rightsquigarrow \psi)}\ \text{(HS)}}{Kern(m_1)\ sat\ Kern(\varphi \rightsquigarrow \psi)}\ \text{(Kern)}}{(\varphi')Kern(m_1)(\varphi' \lhd Kern(\varphi \rightsquigarrow \psi))}\ \text{(SP)}}{(\varphi')Kern(m_1)(Kern(t_0, \psi, \mathbf{c}))}\ \text{(Consequence)}$$

The application of the consequence rule in this derivation is justified as follows.

$(h,s) \in [\![\varphi' \lhd \mathit{Kern}(\varphi \rightsquigarrow \psi)]\!]\gamma$

implies

$\exists h_1, h_2, s_1 : h = h_1 h_2 \wedge (h_1, s_1) \in [\![\varphi']\!]\gamma \wedge (s_1, h_2, s) \in [\![\mathit{Kern}(\varphi \rightsquigarrow \psi)]\!]\gamma$

implies

$\exists h_1, h_2, s_1 : h = h_1 h_2 \wedge (h_1, s_1) \in [\![\varphi]\!]\gamma \wedge h_1|\mathbf{c} = \gamma(t_0)|\mathbf{c} \wedge$

$\forall h_1\Big((h_1, s_1) \in [\![\varphi]\!]\gamma \rightarrow ((h_1 h_2), s) \in [\![\psi]\!]\gamma\Big) \wedge$

$\forall h_2' \leq h_2, \forall h_1\Big((h_1, s_1) \in [\![\varphi]\!]\gamma \rightarrow (h_1 h_2', \perp) \in [\![\psi]\!](\gamma)\Big)$

implies

$\exists h_1, h_2, s_1 : h = h_1 h_2 \wedge h_1|\mathbf{c} = \gamma(t_0)|\mathbf{c} \wedge$

$(h_1 h_2, s) \in [\![\psi]\!]\gamma \wedge \forall h_2' \leq h_2\Big((h_1 h_2', \perp) \in [\![\psi]\!]\gamma\Big)$

implies

$\exists h_1, h_2, s_1 : h = h_1 h_2 \wedge$

$(h_1 h_2, s) \in [\![\psi]\!]\gamma \wedge \forall h_2' \leq h_2\Big((\gamma(t_0) h_2', \perp) \in [\![\psi]\!]\gamma\Big)$

implies

$(h,s) \in [\![\psi]\!]\gamma \wedge \forall h'\Big(\gamma(t_0)|\mathbf{c} \leq h'|\mathbf{c} \leq h|\mathbf{c} \rightarrow (h', \perp) \in [\![\psi]\!]\gamma\Big)$

implies

$(h,s) \in [\![\mathit{Kern}(t_0, \psi, \mathbf{c})]\!]\gamma.$

•*Sequential composition*

$$\dfrac{\dfrac{\dfrac{(\varphi)\, m_1\, (\rho)}{m_1\ sat\ (\varphi \rightsquigarrow \rho)}\ (\mathrm{HS}) \quad \dfrac{(\rho)\, m_2\, (\psi)}{m_2\ sat\ (\rho \rightsquigarrow \psi)}\ (\mathrm{HS})}{m_1\,;\, m_2\ sat\ \Big((\varphi \rightsquigarrow \rho)\hat{\circ}(\rho \rightsquigarrow \psi)\Big)}\ (;)}{m_1\,;\, m_2\ sat\ (\varphi \rightsquigarrow \psi)}\ (\mathrm{Consequence})}{(\varphi)\, m_1\,;\, m_2\, (\psi)}\ (\mathrm{SH})$$

• *Choice*

$$\dfrac{\dfrac{(\varphi)\, m_1\, (\psi)}{m_1 \; sat \; (\varphi \rightsquigarrow \psi)}\,(\mathrm{HS}) \qquad \dfrac{(\varphi)\, m_2\, (\psi)}{m_2 \; sat \; (\varphi \rightsquigarrow \psi)}\,(\mathrm{HS})}{m_1 \; \mathbf{or} \; m_2 \; sat \; (\varphi \rightsquigarrow \psi \vee \varphi \rightsquigarrow \psi)}\,(\mathrm{Choice})$$

$$\dfrac{m_1 \; \mathbf{or} \; m_2 \; sat \; (\varphi \rightsquigarrow \psi)}{(\varphi)\; m_1 \; \mathbf{or} \; m_2 \; (\psi)}\,(\mathrm{SH})$$

$$\dfrac{m_1 \; \mathbf{or} \; m_2 \; sat \; (\varphi \rightsquigarrow \psi)}{(\varphi)\; m_1 \; \mathbf{or} \; m_2 \; (\psi)}\,(\mathrm{Consequence})$$

- *Parallel composition*

Assume that $abase(\psi_i) \cap \beta_j \subseteq \beta_i$, for $(i,j) = (1,2)$ and $(i,j) = (2,1)$.

Note that, since $abase(\varphi \rightsquigarrow \psi) = abase(\psi)$ the conditions above still hold if we replace ψ_i by $\varphi_i \rightsquigarrow \psi_i$. This means that the application of the parallel composition rule in the following derivation is justified:

$$\dfrac{\dfrac{(\varphi_1) m_1 (\psi_1)}{m_1 \; sat \; (\varphi_1 \rightsquigarrow \psi_1)}\,(\mathrm{HS}) \qquad \dfrac{(\varphi_2) m_2 (\psi_2)}{m_2 \; sat \; (\varphi_2 \rightsquigarrow \psi_2)}\,(\mathrm{HS})}{m_1 \; \beta_1 \| \beta_2 \; m_2 \; sat \; (\varphi_1 \rightsquigarrow \psi_1 \wedge \varphi_2 \rightsquigarrow \psi_2)}\,(\mathrm{Parallel\ composition})$$

$$\dfrac{m_1 \; \beta_1 \| \beta_2 \; m_2 \; sat \; (\varphi_1 \rightsquigarrow \psi_1 \wedge \varphi_2 \rightsquigarrow \psi_2)}{m_1 \; \beta_1 \| \beta_2 \; m_2 \; sat \; ((\varphi_1 \wedge \varphi_2) \rightsquigarrow (\psi_1 \wedge \psi_2))}\,(\mathrm{Consequence})$$

$$\dfrac{m_1 \; \beta_1 \| \beta_2 \; m_2 \; sat \; ((\varphi_1 \wedge \varphi_2) \rightsquigarrow (\psi_1 \wedge \psi_2))}{(\varphi_1 \wedge \varphi_2) \; m_1 \; \beta_1 \| \beta_2 \; m_2 \; (\psi_1 \wedge \psi_2)}\,(\mathrm{SH})$$

- *Recursion, μ_z Recursion*

The recursion rule is a special case of Scott's Induction rule, since we already showed the admissibility of Hoare formulae. Just as for the case of SAT formulae, the *first* premisse of Scott's rule, which is in this case of the form $H \vdash (\varphi)\ \mathbf{false}\ (\psi)$, can be omitted since it is always satisfied.

The case of μ_z recursion is treated completely similar to the corresponding case for the SAT system.

- *Process naming*

This case is very similar to the corresponding case for the SAT system.

- *Invariance*

Assume that $abase(\psi) = (\mathbf{d}, \{\bar{y}\})$ and that $(\mathbf{d}, \{\bar{y}\}) \cap base(m) = (\emptyset, \emptyset)$.

Then we have the following derivation, starting with an instance of the invariance axiom, for the sat system.

$$\frac{m \; sat \; (h|\mathbf{d} = \varepsilon \wedge (\top \rightarrow \bar{y} = \bar{y}^\circ))}{\left((h|\mathbf{d} = \varepsilon \wedge (\top \rightarrow \bar{y} = \bar{y}^\circ)) \rhd \psi\right) m \left(\psi\right)} \text{(WP)}$$

$$\frac{\left((h|\mathbf{d} = \varepsilon \wedge (\top \rightarrow \bar{y} = \bar{y}^\circ)) \rhd \psi\right) m \left(\psi\right)}{(\psi[\bot] \wedge \psi) \; m \; (\psi)} \text{(Consequence)}$$

(The proof of the property used for the consequence rule in this derivation is very similar to the calculation of the weakest precondition for the **skip** process above.)

• *Prefix Invariance*

Note that:

$$\forall t_0 (t_0|\mathbf{c} \leq h|\mathbf{c} \; \leadsto \; t_0|\mathbf{c} \leq h|\mathbf{c}) \qquad \text{iff}$$

$$\forall t_0' \forall t_0 (t_0|\mathbf{c} \leq t_0'|\mathbf{c} \rightarrow t_0|\mathbf{c} \leq t_0'|\mathbf{c})$$

Clearly the last assertion is universally valid, explaining why we need no prefix invariance axiom for the SAT system!

The soundness of the axiom for the Hoare system follows from:

$$\frac{m \; sat \; \mathbf{true} \quad \text{invariance}}{m \; sat \; \forall t_0 (t_0|\mathbf{c} \leq h|\mathbf{c} \; \leadsto \; t_0|\mathbf{c} \leq h|\mathbf{c})} \text{(Consequence)}$$

$$\frac{m \; sat \; \forall t_0 (t_0|\mathbf{c} \leq h|\mathbf{c} \; \leadsto \; t_0|\mathbf{c} \leq h|\mathbf{c})}{m \; sat \; t_0|\mathbf{c} \leq h|\mathbf{c} \; \leadsto \; t_0|\mathbf{c} \leq h|\mathbf{c}} \text{(Consequence)}$$

$$\frac{m \; sat \; t_0|\mathbf{c} \leq h|\mathbf{c} \; \leadsto \; t_0|\mathbf{c} \leq h|\mathbf{c}}{(t_0|\mathbf{c} \leq h|\mathbf{c}) \; m \; (t_0|\mathbf{c} \leq h|\mathbf{c})} \text{(SH)}$$

• *Strictness*

Note that if $\varphi \in \mathit{Assn}(\Sigma)$, then the *Assn* assertions $\bot^\circ \wedge \varphi[t_0/h, \bar{x}^\circ/\mathit{xl}, \top^\circ\top]$ and $\bot \wedge \varphi[t_0/h]$ do have the same truth value. Also, we recall that $\bot^\circ \rightarrow (h|\mathbf{c} = \varepsilon \wedge \bot)$ is universally valid, by the structure of the domain Δ.

So if we choose $\mathbf{c} = hchan(\varphi)$, then we see that:

$$(\bot \wedge \varphi) \; \leadsto \; (\bot \wedge \varphi),$$

which is equivalent to

$$\forall t_0\Big((\bot^\circ \wedge \varphi[t_0/h, \bar{x}^\circ/xl, \top^\circ\top]) \to (\bot \wedge \varphi[t_0/h])\Big),$$

is universally valid.

This explains why there is no strictness axiom for the SAT system. And similar to above, we can prove the soundness of the strictness axiom for the Hoare system: $\bot^\circ \to (h|\mathbf{c} = \varepsilon \wedge \bot)$

$$\frac{m \; sat \; (\bot \wedge \varphi) \rightsquigarrow (\bot \wedge \varphi)}{(\bot \wedge \varphi) \; m \; (\bot \wedge \varphi)} \; (\mathrm{SH})$$

- *Conjunction*

$$\frac{\dfrac{(\varphi_1)\; m\; (\psi_1)}{m \; sat \; (\varphi_1 \rightsquigarrow \psi_1)}\; (\mathrm{HS}) \qquad \dfrac{(\varphi_2)\; m\; (\psi_2)}{m \; sat \; (\varphi_2 \rightsquigarrow \psi_2)}\; (\mathrm{HS})}{\dfrac{\dfrac{m \; sat \; (\varphi_1 \rightsquigarrow \psi_1 \wedge \varphi_2 \rightsquigarrow \psi_2)}{m \; sat \; ((\varphi_1 \wedge \varphi_2) \rightsquigarrow (\psi_1 \wedge \psi_2))}\; (\mathrm{Consequence})}{(\varphi_1 \wedge \varphi_2)\; m\; (\psi_1 \wedge \psi_2)}\; (\mathrm{SH})}\; (\mathrm{Conjunction})$$

- *Disjunction*

$$\frac{\dfrac{(\varphi_1)\; m\; (\psi_1)}{m \; sat \; (\varphi_1 \rightsquigarrow \psi_1)}\; (\mathrm{HS}) \qquad \dfrac{(\varphi_2)\; m\; (\psi_2)}{m \; sat \; (\varphi_2 \rightsquigarrow \psi_2)}\; (\mathrm{HS})}{\dfrac{\dfrac{m \; sat \; (\varphi_1 \rightsquigarrow \psi_1 \wedge \varphi_2 \rightsquigarrow \psi_2)}{m \; sat \; ((\varphi_1 \vee \varphi_2) \rightsquigarrow (\psi_1 \vee \psi_2))}\; (\mathrm{Consequence})}{(\varphi_1 \vee \varphi_2)\; m\; (\psi_1 \vee \psi_2)}\; (\mathrm{SH})}\; (\mathrm{Disjunction})$$

- *Consequence*

Assume the validity of $\forall_\bot(\varphi \to \varphi')$ and $\forall_\bot(\psi' \to \psi)$. Then is is easily checked that also $(\varphi' \rightsquigarrow \psi') \to (\varphi \rightsquigarrow \psi)$ is valid, and thus we have the following derivation:

$$\dfrac{\dfrac{\dfrac{(\varphi')\ m\ (\psi')}{}\ (\text{HS})}{m\ sat\ (\varphi' \rightsquigarrow \psi')}\ (\text{Consequence})}{m\ sat\ (\varphi \rightsquigarrow \psi)}\ (\text{SH})$$

$$(\varphi)\ m\ (\psi)$$

- $\exists$-*pre*

If $\bar{g}$ are logical variables that do not occur free in ψ, then:

$\forall \bar{g}\big(\varphi \rightsquigarrow \psi\big)$ iff

$\forall \bar{g} \forall t_0\big(\varphi[t_0/h, \bar{x}^\circ/\bar{x}, \top^\circ/\top] \rightarrow \psi[t_0 h/h]\big)$ iff

$\forall t_0\big(\exists \bar{g}(\varphi[t_0/h, \bar{x}^\circ/\bar{x}, \top^\circ/\top]) \rightarrow \psi[t_0 h/h]\big)$ iff

$\exists \bar{g}(\varphi) \rightsquigarrow \psi$.

This is used in the following derivation. Provided that $\bar{g}$ do not occur free in hypotheses of the derivation:

$$\dfrac{\dfrac{\dfrac{(\varphi)\ m\ (\psi)}{}\ (\text{HS})}{m\ sat\ (\varphi \rightsquigarrow \psi)}\ (\forall\text{-Intro})}{m\ sat\ (\forall \bar{g}(\varphi \rightsquigarrow \psi))}\ (\text{Consequence})}{m\ sat\ (\exists \bar{g}(\varphi) \rightsquigarrow \psi)}\ (\text{SH})$$

$$(\exists \bar{g}(\varphi))\ m\ (\psi)$$

- *Strong adaptation*

Under the assumption that $\beta \cap base(m) = (\emptyset, \emptyset)$ we have the following derivation:

$$\dfrac{\dfrac{(\varphi_1)\ m\ (\psi_1)}{}\ (\text{HS}) \qquad m\ sat\ \mathbf{1}_\beta\ \ (\textit{Invariance})}{m\ sat\ (\varphi_1 \rightsquigarrow \psi_1)}\ (\text{SP})$$

$$(\varphi_2)\ m\ (\varphi_2 \lhd (\varphi_1 \rightsquigarrow \psi_1))$$

The weakest precondition version of the rule is derived similarly, using the "WP" rule instead of the "SP" rule.

• *Initial trace adaptation*

The transformation is simple:

$$\dfrac{\dfrac{(\varphi \wedge h|\mathbf{c}' = \varepsilon)\; m\; (\psi)}{m \textit{ sat } (\varphi \wedge h|\mathbf{c}' = \varepsilon) \rightsquigarrow \psi}\ (\text{HS})}{(\varphi)\; m\; (\psi)}\ (\text{Consequence})$$

$$\dfrac{m \textit{ sat } \varphi \rightsquigarrow \psi}{(\varphi)\; m\; (\psi)}\ (\text{SH})$$

The application of the consequence rule is justified by the following calculations. We prove that

$$((\varphi \wedge h|\mathbf{c}' = \varepsilon) \rightsquigarrow \psi) \rightarrow (\varphi \rightsquigarrow \psi)$$

is a valid implication, under the assumption that $\mathbf{c}' \cap \mathit{hchan}(\varphi, \psi) = \emptyset$.

Assume that:

$$(s_0, h, s) \in [\![(\varphi \wedge h|\mathbf{c}' = \varepsilon) \rightsquigarrow \psi]\!]\gamma.$$

This is equivalent to:

$$\forall h_0 \Big(((h_0, s_0) \in [\![\varphi]\!]\gamma \wedge h_0|\mathbf{c}' = \varepsilon) \;\Rightarrow\; (h_0 h, s) \in [\![\psi]\!]\gamma \Big).$$

We want to show that the conjunct $h_0|\mathbf{c}' = \varepsilon$ can be left out. So assume that we have some arbitrary $(h_0, s_0) \in [\![\varphi]\!]\gamma$. Since $\mathit{hchan}(\varphi) \cap \mathbf{c}' = \emptyset$ this implies that:

$$(h_0 \backslash \mathbf{c}', s_0) \in [\![\varphi]\!]\gamma.$$

It is clear that $(h_0 \backslash \mathbf{c}')|\mathbf{c}' = \varepsilon$, and so by the given implication it follows that:

$$((h_0 \backslash \mathbf{c}')h, s) \in [\![\psi]\!]\gamma.$$

Since also $\mathit{hchan}(\psi) \cap \mathbf{c}' = \emptyset$, one sees that:

$$(h_0 h, s) \in [\![\psi]\!]\gamma,$$

as was to be shown.

□

5.9 Soundness of the Invariant system

• *Abort,Z*

Choosing $\psi = (I \wedge (\top \rightarrow \textbf{false}))$, we have:

$$\dfrac{\dfrac{(\psi[\bot])\ \textbf{abort}\ (\psi)\quad \text{(Abort)}}{(I)\ \textbf{abort}\ (I \wedge (\top \rightarrow \textbf{false}))}\ \text{(Consequence)}}{(\top \wedge I)\ \textbf{abort}\ (I \wedge (\top \rightarrow \textbf{false}))}\ \text{(Consequence)}$$

$$\dfrac{(\top \wedge I)\ \textbf{abort}\ (I \wedge (\top \rightarrow \textbf{false}))}{I : \{I\}\ \textbf{abort}\ \{\textbf{false}\}}\ \text{(HI)}$$

The process **Z** is treated completely similar.

• *Skip*

Choosing $\psi = (I \wedge (\top \rightarrow \text{p}\ ))$, we have:

$$\dfrac{\dfrac{(\psi[\bot] \wedge \psi)\ \textbf{skip}\ (\psi)\quad \text{(Skip)}}{(I \wedge (I \wedge \top \rightarrow p))\ \textbf{skip}\ (I \wedge (\top \rightarrow p))}\ \text{(Consequence)}}{(\top \wedge p \wedge I)\ \textbf{skip}\ (I \wedge (\top \rightarrow (p \wedge I)))}\ \text{(Consequence)}$$

$$\dfrac{(\top \wedge p \wedge I)\ \textbf{skip}\ (I \wedge (\top \rightarrow (p \wedge I)))}{I : \{p \wedge I\}\ \textbf{skip}\ \{p \wedge I\}}\ \text{(HI)}$$

• *Assign*

Choosing ψ as $I \wedge (\top \rightarrow (p \wedge I))$ we have that:

$\psi[\bot] \Leftrightarrow I$ and

$\psi[\top][e/x] \Leftrightarrow (p \wedge I)[e/x] \Leftrightarrow (p[e/x] \wedge I)$.

Hence:

$$\dfrac{\dfrac{(\psi[\bot] \wedge \psi[\top][e/x])\ x := e\ (\psi)\quad \text{(Assign)}}{(\top \wedge p[e/x] \wedge I)\ x := e\ (I \wedge (\top \rightarrow (p \wedge I)))}\ \text{(Consequence)}}{I : \{p[e/x] \wedge I\}\ x := e\ \{p \wedge I\}}\ \text{(HI)}$$

• *Guard*

Choosing ψ as $I \wedge (\top \rightarrow (p \wedge I \wedge b))$ we have that:

$\psi[\bot] \Leftrightarrow I$ and

$\psi[\top] \Leftrightarrow (p \wedge I \wedge b)$.

Hence:

$$\dfrac{\dfrac{(\psi[\bot] \wedge (\top \wedge b) \rightarrow \psi[\top])\ b\ (\psi) \quad (Guard)}{(I \wedge (\top \wedge b) \rightarrow (p \wedge I \wedge b))\ b\ (I \wedge (\top \rightarrow (p \wedge I \wedge b)))}\ \text{(Consequence)}}{(\top \wedge p \wedge I)\ b\ (I \wedge (\top \rightarrow b))}\ \text{(Consequence)}$$

$$I : \{p \wedge I\}\ b\ \{p \wedge I \wedge b\}\ \text{(HI)}$$

• *Com*

Let $\psi = I \wedge (\top \rightarrow (p \wedge I))$. Then:

$\psi[\bot] \Leftrightarrow I$ and

$\psi[\top] \Leftrightarrow p \wedge I$.

The communication axiom in the Hoare system for this ψ is:

$(I \wedge$

$\top \rightarrow \forall v \Big(b[v/x] \rightarrow (I[h < (c, v) > /h] \wedge (p \wedge I)[h < (c, v) > /h, v/x]) \Big))$

$c.x : b$

$(I \wedge (\top \rightarrow (p \wedge I)))$.

By means of the the Consequence rule one can derive from this the following formula:

$(\top \wedge I \wedge$

$\forall v \Big(b[v/x] \rightarrow (I[h < (c, v) > /h] \wedge p[h < (c, v) > /h, v/x]) \Big))$

$c.x : b$

$(I \wedge (\top \rightarrow (p \wedge I)))$.

By definition, formalized in the "(HI)" rule, the last formula is:

$$I : \{I \wedge \forall v \Big(b[v/x] \rightarrow (I[h < (c, v) > /h] \wedge p[h < (c, v) > /h][v/x]) \Big) \}$$

$c.x : b$

$\{p \wedge I\}$.

• *Predicative processes*

Let $\langle J, R\rangle$ be a predicative process as indicated for the axiom. We calculate the following strongest postconditions:

$(\top \wedge p) \lhd ((\top^\circ \to J) \wedge (\top \to R))$

iff

$\exists t_1 \exists t_2 \exists \bar{v}\Big(h|\mathbf{c} = (t_1 t_2)|\mathbf{c} \wedge p[t_1/h, \bar{v}/\bar{x}] \wedge J[\bar{v}/\bar{x}^\circ, t_2/h] \wedge (\top \to R[\bar{v}/\bar{x}^\circ, t_2/h])\Big)$.

$p \lhd J$ iff

$(p[\bot] \wedge \mathbf{true} \wedge \bot) \quad \vee)$

$\exists t_1 \exists t_2 \exists \bar{v}\Big(h|\mathbf{c} = (t_1 t_2)|\mathbf{c} \wedge p[t_1/h, \bar{v}/\bar{x}] \wedge J[\bar{v}/\bar{x}^\circ, t_2/h]\Big)$.

$\top \to (p \lhd (J \wedge R))$ iff

$\top \to \Big[\exists t_1 \exists t_2 \exists \bar{v}\Big(h|\mathbf{c} = (t_1 t_2)|\mathbf{c} \wedge p[t_1/h, \bar{v}/\bar{x}] \wedge J[\bar{v}/\bar{x}^\circ, t_2/h] \wedge R[\bar{v}/\bar{x}^\circ, t_2/h]\Big)\Big]$.

From these calculations it is clear that the following is a strictly valid assertion:

$\Big[(\top \wedge p) \lhd ((\top^\circ \to J) \wedge (\top \to R))\Big] \to \Big[(p \lhd J) \wedge (\top \to (p \lhd (J \wedge R)))\Big]$.

Therefore, the following derivation is admitted.

$$\dfrac{\dfrac{(\top \wedge p)\ \langle J, R\rangle\ ((\top \wedge p) \lhd ((\top^\circ \to J) \wedge (\top \to R)) \quad \text{(Pred.)}}{(\top \wedge p)\ \langle J, R\rangle\ (p \lhd J \wedge (\top \to (p \lhd (J \wedge R))))}\ \text{(Consequence)}}{p \lhd J\ :\ \{p\}\ \langle J, R\rangle\ \{p \lhd (J \wedge R)\}}\ \text{(HI)}$$

• *Hiding rules*

If $\mathbf{c} \cap hchan(q, I) = \emptyset$, then also $\mathbf{c} \cap hchan(I \wedge (\top \to q)) = \emptyset$. Hence the following derivation is allowed:

$$\dfrac{\dfrac{\dfrac{I : \{p\}\ m_1\ \{q\}}{(\top \wedge p)\ m_1\ (I \wedge (\top \rightarrow q))}\ \text{(IH)}}{(\top \wedge p)\ m_1 \backslash \mathbf{c}\ (I \wedge (\top \rightarrow q))}\ \text{(Hiding)}}{I : \{p\}\ m_1 \backslash \mathbf{c}\ \{q\}}\ \text{(HI)}$$

Variable hiding is very similar since $\mathbf{x} \cap var(q) = \mathbf{x} \cap var(I \wedge (\top \rightarrow q))$.

- *Renaming*

We use the following two abbreviations for substitution:

$[1] \stackrel{\text{def}}{=} [h[c/c']/h]$ and

$[2] \stackrel{\text{def}}{=} [h[d/c][c/c']/h]$.

Then we have the following derivation:

$$\dfrac{\dfrac{I : \{p\}\ m_1\langle d/c\rangle\{q\}}{I[2] : \{p[1] \wedge c = \varepsilon\}\ m_1\ \{q[2]\}}\ \text{(Consequence)}}{(\top \wedge p[1] \wedge c = \varepsilon)\ m_1\ (I[2] \wedge (\top \rightarrow q[2]))}\ \text{(IH)}$$

- *Kernel*

Let $\mathbf{c} \subseteq hchan(I, q)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{I : \{p\}\ m_1\ \{q\}}{(\top \wedge p)\ m_1\ (I \wedge (\top \rightarrow q))}\ \text{(IH)}}{(\top \wedge p \wedge h|\mathbf{c} = t_0|\mathbf{c})\ Kern(m_1)\ (Kern(t_0, I \wedge (\top \rightarrow q)))}\ \text{(Kern)}}{(\top \wedge p \wedge h|\mathbf{c} = t_0|\mathbf{c})\ Kern(m_1)\ (Kern(t_0, I) \wedge (\top \rightarrow q))}\ \text{(Consequence)}}{Kern(t_0, I) : \{p \wedge h|\mathbf{c} = t_0|\mathbf{c}\}\ Kern(m_1)\ \{q\}}\ \text{(HI)}$$

- *Sequential composition*

We have the following derivations:

(A)

$$\dfrac{\dfrac{I : \{p\}\ m_1\ \{r\}}{(\top \wedge p)\ m_1\ (I \wedge (\top \rightarrow r))}\ \text{(IH)}}{(\top \wedge p)\ m_1\ ((\bot \wedge I) \vee (\top \wedge r))}\ \text{(Consequence)}$$

(B)

$$\dfrac{\dfrac{\dfrac{I : \{r\}\ m_2\ \{q\}}{(\top \wedge r)\ m_2\ (I \wedge (\top \rightarrow q))}\ \text{(IH)} \quad (\bot \wedge I)\ m_2\ (\bot \wedge I)\ \text{(Strictness)}}{((\bot \wedge I) \vee (\top \wedge r))\ m_2\ ((\bot \wedge I) \vee (I \wedge (\top \rightarrow q)))}\ \text{(Disj.)}}{((\bot \wedge I) \vee (\top \wedge r))\ m_2\ (I \wedge (\top \rightarrow q))}\ \text{(Consequence)}$$

(C)

$$\dfrac{\dfrac{(\top \wedge p)\ m_1\ ((\bot \wedge I) \vee (\top \wedge r)) \qquad ((\bot \wedge I) \vee (\top \wedge r))\ m_2\ (I \wedge (\top \rightarrow q))}{(\top \wedge p)\ m_1; m_2\ (I \wedge (\top \rightarrow q))}\ (;)}{I : \{p\}\ m_1; m_2\ \{q\}}\ \text{(HI)}$$

Clearly the soundness of the rule follows by combining these three derivations.

- *Choice*

$$\dfrac{\dfrac{\dfrac{I : \{p\}\ m_1\ \{q\}}{(\top \wedge p)\ m_1\ (I \wedge (\top \rightarrow q))}\ \text{(IH)} \qquad \dfrac{I : \{p\}\ m_2\ \{q\}}{(\top \wedge p)\ m_2\ (I \wedge (\top \rightarrow q))}\ \text{(IH)}}{((\top \wedge p) \wedge (\top \wedge p))\ m_1\ \mathbf{or}\ m_2\ ((I \wedge (\top \rightarrow q)) \vee (I \wedge (\top \rightarrow q)))}\ \text{(Choice)}}{(\top \wedge p)\ m_1\ \mathbf{or}\ m_2\ (I \wedge (\top \rightarrow q))}\ \text{(Cons.)}}{I : \{p\}\ m_1\ \mathbf{or}\ m_2\ \{q\}}\ \text{(HI)}$$

- *Parallel composition*

Assume that for $i = 1, 2$: $abase(q_i, I_i) \cap \beta_j \subseteq \beta_i$. Then we have the following derivation:

$$\dfrac{\dfrac{\dfrac{I_1 : \{p_1\}\ m_1\ \{q_1\}}{(\top \wedge p_1)\ m_1\ (I_1 \wedge (\top \rightarrow q_1))}\text{(IH)} \quad \dfrac{I_2 : \{p_2\}\ m_2\ \{q_2\}}{(\top \wedge p_2)\ m_2\ (I_2 \wedge (\top \rightarrow q_2))}\text{(IH)}}{((\top \wedge p_1) \wedge (\top \wedge p_2))\ m_1\ \beta_1 \| \beta_2\ m_2\ ((I_1 \wedge (\top \rightarrow q_1)) \wedge (I_2 \wedge (\top \rightarrow q_2)))}\text{(P.C.)}}{\dfrac{(\top \wedge (p_1 \wedge p_2))\ m_1\ \beta_1 \| \beta_2\ m_2\ ((I_1 \wedge I_2) \wedge \top \rightarrow (q_1 \wedge q_2))}{I_1 \wedge I_2 : \{p_1 \wedge p_2\}\ m_1\ \beta_1 \| \beta_2\ m_2\ \{q_1 \wedge q_2\}}\text{(HI)}}$$

The application of the parallel composition rule for Hoare formulae is justified by the fact that:

$$abase(I_i \wedge (\top \rightarrow q_i)) \cap \beta_j = abase(I_i, q_i) \cap \beta_j \subseteq \beta_i.$$

- *Recursion, μ_z Recursion*

Since $I : \{p\}\ m_1\ \{q\}$ formulae are (abbreviations of) Hoare formulae, the corresponding rules for this latter type of formulae are applicable. For the case of μ_z recursion one could have expected the following rule:

$$\dfrac{H \vdash \forall \bar{g}[I : \{p\}\ \mathbf{Z}\ \{q\}]\ ,\ H \cup \{\forall \bar{g}[I : \{p\}\ X_\beta\ \{q\}]\} \vdash \forall \bar{g}[I : \{p\}\ m_1\ \{q\}]}{\forall \bar{g}[I : \{p\}\ \mu_z X_\beta . m_1\ \{q\}]}$$

The premisse of the form $H \vdash \forall \bar{g}[I : \{p\}\ \mathbf{Z}\ \{q\}]$ however boils down to $H \vdash \underline{\forall}(p \rightarrow I)$. The premisse could be omitted by including the invariant I as a conjunct to the preconditions of the relevant formulae, as we have done for the rule for μ_z recursion in the Invariant system. The following simple derivation shows that in this case the premisse for $\mathbf{Z}$ in the rule above is always valid.

$$\dfrac{\left(\dfrac{I : \{I\}\ \mathbf{Z}\ \{\mathbf{false}\}}{I : \{p \wedge I\}\ \mathbf{Z}\ \{q\}}\text{(Consequence)}\right.}{\forall \bar{g}[I : \{p \wedge I\}\ \mathbf{Z}\ \{q\}]}(\forall-\text{ introduction})$$

- *Process naming.*

The rule for the invariance system is just a special case of the corresponding rule for the Hoare system.

- *Invariance.*

Let $\psi = I \wedge (\top \rightarrow (p \wedge I))$.

The assumption for the axiom is that $abase(p, I) \cap base(m) = (\emptyset, \emptyset)$. This implies that $abase(\psi) \cap base(m) = (\emptyset, \emptyset)$.

Since $\psi[\bot] \Leftrightarrow I$ it is clear that: $(\top \wedge p \wedge I) \Rightarrow (\psi[\bot] \wedge \psi)$.

Hence the following derivation is possible:

$$\dfrac{\dfrac{(\psi[\bot] \wedge \psi)\ m_1\ (\psi) \quad \text{(invariance)}}{(\top \wedge p \wedge I)\ m_1\ (I \wedge (\top \rightarrow (p \wedge I)))}\ \text{(Consequence)}}{I : \{p \wedge I\}\ m_1\ \{p \wedge I\}}\ \text{(HI)}$$

- *Prefix invariance*

$$\dfrac{\dfrac{h|\mathbf{c} \leq t_0|\mathbf{c})\ m\ (h|\mathbf{c} \leq t_0|\mathbf{c}) \quad \text{(prefix invariance)}}{\top \wedge h|\mathbf{c} \leq t_0|\mathbf{c})\ m\ (h|\mathbf{c} \leq t_0|\mathbf{c} \wedge (\top \rightarrow h|\mathbf{c} \leq t_0|\mathbf{c}))}\ \text{(Consequence)}}{(h|\mathbf{c} \leq t_0|\mathbf{c}) : \{h|\mathbf{c} \leq t_0|\mathbf{c}\}\ m\ \{h|\mathbf{c} \leq t_0|\mathbf{c}\}}\ \text{(HI)}$$

- *Closure adaptation*

$$\dfrac{\dfrac{\dfrac{I : \{p\}\ m\ \{q\}}{(\top \wedge p)\ m\ (I \wedge (\top \rightarrow q))}\ \text{(IH)}}{(\top \wedge p)\ m\ (I \wedge (\top \rightarrow (q \wedge I)))}\ \text{(Consequence)}}{I : \{p\}\ m\ \{q \wedge I\}}\ \text{(HI)}$$

- *Initial trace adaptation*

$$\dfrac{\dfrac{\dfrac{I : \{p \wedge h|\mathbf{c} = \varepsilon\}\ m\ \{q\}}{(\top \wedge p \wedge h|\mathbf{c} = \varepsilon)\ m\ (I \wedge (\top \rightarrow q))}\text{(IH)}}{(\top \wedge p)\ m\ (I \wedge (\top \rightarrow q))}\text{(Initial trace adaptation)}}{I : \{p\}\ m\ \{q\}}\text{(HI)}$$

- *Conjunction*

$$\dfrac{\dfrac{\dfrac{I_1 : \{p_1\}\ m_1\ \{q_1\}}{(\top \wedge p_1)\ m_1\ (I_1 \wedge (\top \rightarrow q_1))}\text{(IH)} \quad \dfrac{I_2 : \{p_2\}\ m_2\ \{q_2\}}{(\top \wedge p_2)\ m_2\ (I_2 \wedge (\top \rightarrow q_2))}\text{(IH)}}{((\top \wedge p_1) \wedge (\top \wedge p_2))\ m\ ((I_1 \wedge (\top \rightarrow q_1)) \wedge (I_2 \wedge (\top \rightarrow q_2)))}\text{(Conj.)}}{(\top \wedge (p_1 \wedge p_2))\ m\ ((I_1 \wedge I_2) \wedge (\top \rightarrow (q_1 \wedge q_2)))}\text{(Cons.)}}{(I_1 \wedge I_2) : \{p_1 \wedge p_2\}\ m\ \{q_1 \wedge q_2\}}\text{(HI)}$$

- *Disjunction*

$$\dfrac{\dfrac{\dfrac{I_1 : \{p_1\}\ m_1\ \{q_1\}}{(\top \wedge p_1)\ m_1\ (I_1 \wedge (\top \rightarrow q_1))}\text{(IH)} \quad \dfrac{I_2 : \{p_2\}\ m_2\ \{q_2\}}{(\top \wedge p_2)\ m_2\ (I_2 \wedge (\top \rightarrow q_2))}\text{(IH)}}{((\top \wedge p_1) \vee (\top \wedge p_2))\ m\ ((I_1 \wedge (\top \rightarrow q_1)) \vee (I_2 \wedge (\top \rightarrow q_2)))}\text{(Disj.)}}{(\top \wedge (p_1 \vee p_2))\ m\ ((I_1 \vee I_2) \wedge (\top \rightarrow (q_1 \vee q_2)))}\text{(Cons.)}}{(I_1 \vee I_2) : \{p_1 \vee p_2\}\ m\ \{q_1 \vee q_2\}}\text{(HI)}$$

- *Consequence*

From the premisse:

$$\underline{\forall}(I' \rightarrow I), \underline{\forall}(p \rightarrow p'), \underline{\forall}(q' \rightarrow q),$$

it follows that:

$$\forall_\perp((\top \wedge p) \rightarrow (\top \wedge p'))\ ,\ \forall_\perp((I' \wedge (\top \rightarrow q')) \rightarrow (I \wedge (\top \rightarrow q))).$$

Therefore the following derivation is possible:

$$\dfrac{\dfrac{\dfrac{I' : \{p'\}\ m\ \{q'\}}{(\top \wedge p')\ m\ (I' \wedge (\top \rightarrow q'))}\ (\text{IH}) \quad \begin{array}{l}\forall_\perp((\top \wedge p) \rightarrow (\top \wedge p')),\\ \forall_\perp((I' \wedge (\top \rightarrow q')) \rightarrow (I \wedge (\top \rightarrow q)))\end{array}}{(\top \wedge p)\ m\ (I \wedge (\top \rightarrow q))}\ (\text{Cons.})}{I : \{p\}\ m\ \{q\}}\ (\text{HI})$$

- $\exists - pre$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall \bar{g}\left[I : \{p\}\ m\ \{q\}\right]}{I : \{p\}\ m\ \{q\}}\ (\forall - \text{ elimination})}{(\top \wedge p)\ m\ (I \wedge (\top \rightarrow q))}\ (\text{IH})}{\forall \bar{g}\left[(\top \wedge p)\ m\ (I \wedge (\top \rightarrow q))\right]}\ (\forall - \text{ introduction})}{(\exists \bar{g}(\top \wedge p))\ m\ (I \wedge (\top \rightarrow q))}\ (\exists - \text{ pre})}{(\top \wedge \exists \bar{g}(p))\ m\ (I \wedge (\top \rightarrow q))}\ (\text{Consequence})}{I : \{\exists \bar{g}(p)\}\ m\ \{q\}}\ (\text{HI})$$

Chapter 6

Completeness

6.1 Introduction

Now that we have formulated several proof systems, we consider the deductive strength of these systems. In relation to this we examine the expressive power of specifications.

First of all we formulate some rather general yardsticks that can be applied to arbitrary proposed specification methods for **TNP** processes. Then we apply these yardsticks to our own method, as proposed in chapter 4. This results in the definition of so called characteristic specifications, that will play a major role in the completeness proof for the SAT system.

6.2 The expressive power of specifications

Consider some arbitrary proposed specification method for **TNP** processes. So for the moment the term "specification" refers to a certain type of formulae from this method, not necessarily to the specifications of chapter 4. A reasonable assumption that we will make is that processes with equal semantics and with the same bases satisfy the same specifications. That is, if $S_1 = S_2$ and $base(S_1) = base(S_2)$, and "spec" is some specification, then either *both* S_1 and S_2 satisfy spec or *none* of the two does. Let $S_1 \sqsubseteq S_2$ denote that $S_1 \subseteq S_2$ and $base(S_1) \subseteq base(S_2)$. Also let $S_1 \cong S_2$ denote that $S_1 = S_2$ and $base(S_1) = base(S_2)$. Four criteria about the expressiveness of the method, that might be satisfied or not, are:

1. If S_1 and S_2 satisfy the same specifications, then $S_1 \cong S_2$.

(Together with the assumption made above this means that $S_1 \cong S_2$ iff S_1 and S_2 satisfy the same specifications.)

2. For each process S there is a weak characteristic specification $Wch(S)$ in the following sense: if $S_1 \ncong S_2$, then S_1 does not satisfy $Wch(S_2)$ or S_2 does not satisfy $Wch(S_1)$. (It is understood here that S itself does satisfy $Wch(S)$.)

3. For each process S there is a characteristic specification $Ch_{\sqsubseteq}(S)$ with the property that any S' satisfies $Ch_{\sqsubseteq}(S)$ iff $S' \sqsubseteq S$.

4. Similar to 3, there is a specification $Ch_{\cong}(S)$, such that S' satisfies $Ch_{\cong}(S)$ iff $S' \cong S$.

Criterion 1 is the weakest of these four.
An alternative formulation is: If $S_1 \ncong S_2$, then there exists some separating specification $Sep(S_1, S_2)$ that is satisfied by one of the two processes but not by the other.

Criterion 2 is stronger: it requires that specifications $Wch(S)$ are determined on forehand such that $Sep(S_1, S_2)$ can be chosen as either $Wch(S_1)$ or $Wch(S_2)$.

Criterion 2 is implied by criterion 3, since if we have available characteristic specifications $Ch_{\sqsubseteq}(S)$, we can choose $Wch(S)$ to be $Ch_{\sqsubseteq}(S)$. Then, if $S_1 \ncong S_2$, we must have $S_1 \not\sqsubseteq S_2$, or $S_2 \not\sqsubseteq S_1$, and therefore that S_1 does not satisfy $Ch_{\sqsubseteq}(S_2)$ or S_2 does not satisfy $Ch_{\sqsubseteq}(S_1)$.

Clearly criterion 4 is still stronger than criterion 3. It implies a stronger version of criterion 2, in that the separating specification $Sep(S_1, S_2)$ can be chosen as $Ch_{\cong}(S_1)$ and also as $Ch_{\cong}(S_2)$.

Criterion 3 is the best of these four that one can hope for in the case of a system for *safety* properties. The reason for this is that if spec is such a specification that is satisfied by S, it is also satisfied by all S' such that $S' \sqsubseteq S$. This follows from the characterization of safety properties of chapter 3. This shows that criterion 4 cannot be met by specifications methods for safety properties, including our own methods from chapter 4.

An example of a specification method for **TNP** not meeting criterion 1 would be the class of Hoare formulae $\{pre\}P_\beta\{post\}$, with the usual interpretation: If P_β starts in an initial trace and state satisfying "*pre*", then if and when P_β terminates, "*post*" will hold for the resulting final trace and state. This is essentially our class of invariant formulae $I : \{pre\}P_\beta\{post\}$ with I identical to **true**. For this method, the processes S_1 and S_2, defined as:

(Together with the assumption made above this means that $S_1 \cong S_2$ iff S_1 and S_2 satisfy the same specifications.)

2. For each process S there is a weak characteristic specification $Wch(S)$ in the following sense: if $S_1 \not\cong S_2$, then S_1 does not satisfy $Wch(S_2)$ or S_2 does not satisfy $Wch(S_1)$. (It is understood here that S itself does satisfy $Wch(S)$.)
3. For each process S there is a characteristic specification $Ch_{\sqsubseteq}(S)$ with the property that any S' satisfies $Ch_{\sqsubseteq}(S)$ iff $S' \sqsubseteq S$.
4. Similar to 3, there is a specification $Ch_{\cong}(S)$, such that S' satisfies $Ch_{\cong}(S)$ iff $S' \cong S$.

Criterion 1 is the weakest of these four.
An alternative formulation is: If $S_1 \not\cong S_2$, then there exists some separating specification $Sep(S_1, S_2)$ that is satisfied by one of the two processes but not by the other.

Criterion 2 is stronger: it requires that specifications $Wch(S)$ are determined on forehand such that $Sep(S_1, S_2)$ can be chosen as either $Wch(S_1)$ or $Wch(S_2)$.

Criterion 2 is implied by criterion 3, since if we have available characteristic specifications $Ch_{\sqsubseteq}(S)$, we can choose $Wch(S)$ to be $Ch_{\sqsubseteq}(S)$. Then, if $S_1 \not\cong S_2$, we must have $S_1 \not\sqsubseteq S_2$, or $S_2 \not\sqsubseteq S_1$, and therefore that S_1 does not satisfy $Ch_{\sqsubseteq}(S_2)$ or S_2 does not satisfy $Ch_{\sqsubseteq}(S_1)$.

Clearly criterion 4 is still stronger than criterion 3. It implies a stronger version of criterion 2, in that the separating specification $Sep(S_1, S_2)$ can be chosen as $Ch_{\cong}(S_1)$ and also as $Ch_{\cong}(S_2)$.

Criterion 3 is the best of these four that one can hope for in the case of a system for *safety* properties. The reason for this is that if spec is such a specification that is satisfied by S, it is also satisfied by all S' such that $S' \sqsubseteq S$. This follows from the characterization of safety properties of chapter 3. This shows that criterion 4 cannot be met by specifications methods for safety properties, including our own methods from chapter 4.

An example of a specification method for **TNP** not meeting criterion 1 would be the class of Hoare formulae $\{pre\}P_\beta\{post\}$, with the usual interpretation: If P_β starts in an initial trace and state satisfying "*pre*", then if and when P_β terminates, "*post*" will hold for the resulting final trace and state. This is essentially our class of invariant formulae $I : \{pre\}P_\beta\{post\}$ with I identical to **true**. For this method, the processes S_1 and S_2, defined as:

$S_1 \stackrel{\text{def}}{=} \mu_z P_\beta.P_\beta$ and $S_2 \stackrel{\text{def}}{=} \mu_z P_\beta.c!0; P_\beta$

would satisfy the same specifications, since both have base β and neither of them terminates. However, $S_1 \neq S_2$ and so $S_1 \not\cong S_2$.

It is noteworthy that the system consisting of "classical" Hoare style specifications for, *sequential* programs does satisfy criterion 1 and 2, but that it does not meet criterion 3. For these systems a specification consists of a pair (pre, post) of assertions in a first order predicate language. A program S satisfies a specification if $\{pre\}S\{post\}$ is a valid formula. To prove that criterion 3 cannot be met, let us assume on the contrary that $(pre_{skip}, post_{skip})$ is a characteristic specification for the program **skip**. That is, we assume that:

$\{pre_{skip}\}S'\{post_{skip}\}$ iff $S' \sqsubseteq \mathbf{skip}$

Now since the set of free variables occurring in pre_{skip} or $post_{skip}$ is a finite one, there must be some variable, say x, not occurring in it. But then it is easily seen that:

$\{pre_{skip}\}\, x := 1\, \{post_{skip}\}$

is a valid formula, despite the fact that $x := 1 \not\sqsubseteq \mathbf{skip}$.

The simple counterexample as above is not possible for the Hoare specifications of chapter 4, who have the form $(pre)P_\beta(post)$. Here the base β attached to P expresses the invariance of the infinitely many variables and channels not present in β. For instance, a characteristic specification for **skip** is:

$(\mathbf{true})\, P_{(\emptyset,\emptyset)}\, (\mathbf{true})$

The process $x := 1$ does not satisfy this specification because it is not even substitutable for $P_{(\emptyset,\emptyset)}$. We shall prove that for closed processes there are characteristic specifications conform criterion 3 in the form of Hoare specifications as well as in the form of SAT formulae. To deal with non closed processes we must enlarge these specification classes if we want criterion 3 still to be satisfied. We prove that in this case specifications of the following form suffice:

$$f(\xi) \stackrel{\text{def}}{=} \forall \varsigma_1 \ldots \forall \varsigma_n \Big(\big(\bigwedge_{i=1,n} spec_i(\varsigma_i) \big) \rightarrow spec(\xi) \Big) \quad ,$$

where each $spec_i(\varsigma_i)$ is a SAT formula or Hoare formula. As mentioned before in chapter 4, the formula $f(\xi)$ above specifies a process S such that S satisfies *spec* provided the "modules" $\varsigma_1 \ldots \varsigma_n$ satisfy $spec_1 \ldots spec_n$.

The proof of these claims introduces the so called *characteristic assertions* $A(m)$ for mixed terms m. We shall define $A(m)$ as an assertion that satisfies the equality: $A(m)|base(m) = m$. If m does contain free process variables, then $A(m)$ is a *parametrized* assertion. These characteristic assertions play a major role in the completeness proof for the SAT system.

Before we proceed to the definition of $A(m)$ we would like to compare our treatment of bases with that of others. In [*Olderog2*], specifications are identified with assertions χ that are closed with respect to ghost variables. Satisfaction of such a specification by a (closed) process S simply means that $S \subseteq \chi$ is valid, and, in particular, no explicit condition on bases is included. This means that such specifications correspond to formulae of the form $P \subseteq \chi$ in our system. Here we adopt the convention that P without base subscript abbreviates $P_{(\mathcal{C}han,\mathcal{V}ar)}$.

The problem with these formulae is that the assertion χ must somehow express the invariance of (infinitely many) channels and variables. Now, similar as in [*Olderog2*], we can include a conjunct $h = h|\mathbf{c}$ in χ to enforce that no communications outside $\mathbf{c}$ can occur. A problem with such conjuncts is that they are not preserved by the parallel composition operator. For instance, the following inference is not sound:

$$\frac{S_1 \subseteq h = h|\mathbf{c}_1 \quad , \quad S_2 \subseteq h = h|\mathbf{c}_2}{S_1 \parallel S_2 \subseteq (h = h|\mathbf{c}_1) \wedge (h = h|\mathbf{c}_2)}$$

The conclusion of this inference can be rewritten into:

$$S_1 \parallel S_2 \subseteq (h = h|(\mathbf{c}_1 \cap \mathbf{c}_2)).$$

This is incorrect however; in fact we may conclude only that:

$$S_1 \parallel S_2 \subseteq (h = h|(\mathbf{c}_1 \cup \mathbf{c}_2)).$$

Indeed the restrictions associated with our parallel composition rule in general forbid the inference above. The reason is that $hchan(h = h|\mathbf{c}_i) = \mathcal{C}han$, and so the condition that $hchan(h = h|\mathbf{c}_i) \cap chan(S_j) \subseteq chan(S_i)$ for $(i,j) = (1,2)$ and for $(i,j) = (2,1)$, implies that $chan(S_1) = chan(S_2)$, which is usually not the case.

An even more serious problem is caused by the fact that, within our assertion language, there exists no assertion at all that expresses the invariance of infinitely many *assignable variables*. This could be regarded as a weakness of this language that should be corrected by adding for instance special

assertions of the form $mod(\mathbf{x})$. The interpretation of these new assertions then must be as follows:

$$[\![mod(\mathbf{x})]\!](\gamma) = \{(s_0, h, s) \mid s = \perp \text{ or } s(y) = s_0(y) \text{ for all } y \in \mathcal{V}ar - \mathbf{x}\}.$$

Another alternative would have been to allow for *infinite* conjunctions, such as $\bigwedge_{y \in \mathcal{V}ar - \mathbf{x}} (y = y^\circ)$. Both approaches occur in the literature; see for example [Lam2], [Jonkers].

Of course, if we express the invariance of variables within assertions, we will have the same type of problems with parallel composition as was the case with the invariance of channels. For example, the following inference, analogous to the one above, is not sound either:

$$\frac{S_1 \subseteq mod(\mathbf{x}_1) \quad , \quad S_2 \subseteq mod(\mathbf{x}_2)}{S_1 \parallel S_2 \subseteq mod(\mathbf{x}_1) \wedge mod(\mathbf{x}_2)}$$

The foregoing explains why we decided not to rely on the assertion language to express the base of processes, but rather include bases as a separate part of specifications.

A similar approach, for a language dealing with traces only, can be found in [Snep]. There the base of a process is called its alphabet.

6.3 Characteristic specifications

In chapter 5 we introduced characteristic assertions $A(\alpha)$ for atomic processes α. We proved that not only α *sat* $A(\alpha)$, but that even $\alpha = A(\alpha)|base(\alpha)$ is the case. This means that, of all assertions χ such that $abase(\chi) \subseteq base(\alpha)$, $A(\alpha)$ is the *strongest* assertion that is still satisfied by α.

The next step is now to extend A by assigning such a characteristic assertion $A(m)$ to just *any* mixed term m.

For m with free process variables contained in $\xi_1, \cdots, \xi_n$, $A(m)$ is *parameterized* by assertions $\chi_1, \cdots, \chi_n$. We indicate this by $A(m)(\chi_1, \cdots, \chi_n)$, where the matching between the parameters χ_i and process variables ξ_i is to be understood from the context. Sometimes we abbreviate $A(m)(\bar{\chi})$ as $A(m)(\chi_1, \cdots, \chi_n)$ or even omit the list $\bar{\chi}$ completely. Similarly we use $m(\bar{\chi}|\bar{\beta})$ as an abbreviation for $m[(\chi_1|\beta_1)/\xi_1, \cdots, (\chi_n|\beta_n)/\xi_n]$.

We shall prove the following equality:

$$A(m)(\bar{\chi})|base(m) = m(\bar{\chi}|\bar{\beta})$$

In particular for *closed* m this implies that:

$$A(m)|base(m) = m$$

The free channels and variables in $A(m)$ are chosen such that:

$$abase(A(m)) \subseteq base(m)$$

It is seen that for predicates χ such that $abase(\chi) \subseteq \beta$ the following equation holds for $\beta' \supseteq \beta$:

$$(\chi \wedge \mathbf{1}_{\beta'-\beta})|\beta' = \chi|\beta$$

As a consequence of this, if we define for $\beta \supseteq base(m)$ the assertion $A_\beta(m)$ as

$$A_\beta(m) = A(m) \wedge \mathbf{1}_{\beta-base(m)}$$

then:

$$A_\beta(m)|\beta = A(m)|base(m) = m.$$

We use this abbreviation $A_\beta(m)$ already in the definition of $A(m)$ below. First we repeat definition 5.1, which defines $A(\alpha)$ for atomic processes α.

Definition 6.1 (Characteristic assertions for $\mathcal{A}tom$)

$$A(\mathbf{skip}) = \mathbf{true}$$

$$A(\mathbf{abort}) = \bot$$

$$A(x := e) = \top \rightarrow (x = e[\bar{y}^\circ/\bar{y}] \wedge \bar{w} = \bar{w}^\circ),$$

where $\{\bar{y}\} = var(e)$ and $\{\bar{w}\} = var(e) - \{x\}$

$$A(b) = \top \rightarrow (b \wedge \bar{w} = \bar{w}^\circ), \text{ where } \{\bar{w}\} = var(b)$$

$$A(c.x : b) = (\bot \wedge c = \varepsilon) \vee$$

$$\exists v[c = < v > \wedge b[\bar{w}^\circ/\bar{w}, v/x] \wedge (\top \rightarrow (v = x \wedge \bar{w} = \bar{w}^\circ])]$$

where $\bar{w} = var(b) - \{x\}$

□

Apart from the recursion construct we give an explicit definition of $A(m)$. For the recursion construct we can prove that there *exists* such an assertion, without being able to give such an explicit definition. In section 6.5 we point out that in an assertion language that includes *infinite* disjunctions of formulae, one *can* give an explicit definition of characteristic assertions for the recursion construct.

Definition 6.2 (Characteristic assertion for mixed terms)

For m *not* a μ-construct we define:

$A(\chi|\beta)(\bar{X}) = \chi$

$A(\mathbf{Z})(\bar{X}) = \bot$

$A(\mathbf{1})(\bar{X}) = \mathbf{true}$

$A(X^i_{\beta_i})(\bar{X}) = \chi_i$, where $\bar{X} = \chi_1, \cdots, \chi_n$.

$A(m_1\backslash\mathbf{c})(\bar{X}) = \exists t\Big(A(m_1)(\bar{X})[t/h] \wedge h|\mathbf{c}' = t|\mathbf{c}'\Big)$,

where $\mathbf{c}' = \mathit{chan}(m_1\backslash\mathbf{c}) = \mathit{chan}(m_1) - \mathbf{c}$.

$A(m_1\backslash\mathbf{x})(\bar{X}) = \exists\bar{x}\Big(A(m_1)(\bar{X})\Big)$, where $\{\bar{x}\} = \mathbf{x}$.

$A(m_1\langle d/c\rangle)(\bar{X}) = \exists t\Big(A(m_1)(\bar{X})[t/h] \wedge h|\mathbf{c} = (t|\mathbf{c}_1)[d/c]\Big)$

where $\mathbf{c} = \mathit{chan}(m_1[d/c]) = (\mathit{chan}(m_1) - \{c\}) \cup \{d\}$

and $\mathbf{c}_1 = \mathit{chan}(m_1)$.

$A(\mathit{Kern}(m_1))(\bar{X}) = \mathit{Kern}(A(m_1)(\bar{X}))$

$A(m_1; m_2)(\bar{X}) = A_\beta(m_1)(\bar{X}) \,\hat{\circ}\, A_\beta(m_2)(\bar{X})$, where $\beta = \mathit{base}(m_1; m_2)$.

$A(m_1 \cup m_2)(\bar{X}) = A_\beta(m_1)(\bar{X}) \vee A_\beta(m_2)(\bar{X})$, where $\beta = \mathit{base}(m_1 \cup m_2)$.

$A(m_1 \;{}_{\beta_1}\|_{\beta_2}\; m_2)(\bar{X}) = A_{\beta_1}(m_1)(\bar{X}) \wedge A_{\beta_2}(m_2)(\bar{X})$

$A(X_{\beta_0} = m_0 \textbf{ in } m_1)(\bar{X}) = A(m_1)(\chi_0, \bar{X})$, where $\chi_0 = A_{\beta_0}(m_0)(\bar{X})$,

and where we assumed that $m_1 = m_1(X^0_{\beta_0}, X^1_{\beta_1}, \cdots X^n_{\beta_n})$.

□

To prove the existence of an assertion $A(m)$ for the case that m contains the μ construct, we first show that the set of (codes of) m computations is recursively enumerable (r.e.). From this it then easily follows that $A(m)$ is arithmetical, that is, representable by some formula from first order arithmetic. A slight complication will be that m possibly contains subterms of the form $(\chi|\beta)$ or X_β. Such terms do not necessarily correspond to some r.e. set. Now since we aim at a representing assertion for $m(\bar{\chi}|\bar{\beta})$, we may assume without loss of generality that m itself does not contain subterms of the form $(\chi|\beta)$. A second, more technical, problem is that computations cannot be enumerated by a Turing machine (T.M.) since they are infinite objects, due to the presence of states within them. However, it suffices to enumerate integer *codes* for computations. The coding relies on the finite base of m.

Since our assertion language "includes", but not equals the language of arithmetic, we first show that arithmetical sets are representable in our assertion language. So assume that $base(m) = \beta = (\mathbf{c}, \mathbf{x}) = (\mathbf{c}, \{x_1, \cdots, x_n\})$. Choose some injection

$$f : \mathcal{V}al^n \times \mathcal{T}race \times \mathcal{V}al^n \rightarrow \mathbf{N}$$

that is representable within arithmetic, and define for $\delta \in \Delta, \rho \in \mathcal{P}(\Delta)$:

$$code_\beta(\delta) = code_\beta((s_0, h, s)) \stackrel{\text{def}}{=} f(s_0(\bar{x}), h|\mathbf{c}, s(\bar{x})),$$

$$code_\beta(\rho) = \{code_\beta(\delta) \mid \delta \in \rho\}.$$

Since our assertion language does contain the expressions $x_1{}^\circ, x_2{}^\circ, \cdots x_n^\circ$, $h|\mathbf{c}, x_1, \cdots x_n$, and also includes arithmetic, it is clear that there is some assertion χ_{code} say, with $hchan(\chi_{code}) = \mathbf{c}, var^\circ(\chi_{code}) = \mathbf{x}, lvar(\chi_{code}) = \{n\}$ that satisfies:

$$\delta \in [\![\chi_{code}]\!]\gamma \text{ iff } \gamma(n) = code_\beta(\delta).$$

Abbreviate $m(\bar{\chi}|\bar{\beta})$ by $m[\bullet/\bullet]$. From the assumption that $base(m) = \beta$, it follows that $base(Obs[\![m[\bullet/\bullet]]\!]\gamma\eta) \subseteq \beta$. From this and the fact that $code_\beta(\delta) = code_\beta(\delta')$ iff δ and δ' agree with respect to the channels and variables in β it follows that:

$$\delta|\beta \in Obs[\![m[\bullet/\bullet]]\!]\gamma\eta \text{ iff } code_\beta(\delta) \in code_\beta(Obs[\![m[\bullet/\bullet]]\!]\gamma\eta).$$

Below we argue that the set $code_\beta(Obs[\![m[\bullet/\bullet]]\!]\gamma\eta)$ is arithmetical. Then there is some assertion, χ_m say, such that: $hchan(\chi_m) = var^{(\circ)}(\chi_m) = \emptyset, lvar(\chi_m) = lvar(m[\bullet/\bullet]) \cup \{n\}$ and

$$Tr[\![\chi_m]\!]\delta\gamma \text{ iff } \gamma(n) \in code_\beta(Obs[\![m[\bullet/\bullet]]\!]\gamma\eta).$$

(The δ argument for $Tr[\![\chi_m]\!]$ is arbitrary here).

Define $Arr(m)(\bar{\chi})$ to be:

$$\exists n(\chi_m \wedge \chi_{code}).$$

Then we have that:

$$\delta \in [\![Arr(m)(\bar{\chi})]\!]\gamma$$

iff for some integer n such that $n = code_\beta(\delta)$:

$$n \in code_\beta(Obs[\![m[\bullet/\bullet]]\!]\gamma\eta)$$

iff $\delta|\beta \in Obs[\![m[\bullet/\bullet]]\!]\gamma\eta$.

That is, $[\![Arr(m)(\bar{\chi})]\!]\gamma|\beta = Obs[\![m[\bullet/\bullet]]\!]\gamma\eta$,

• We define $A(\mu X_\beta.m_0)(\bar{\chi})$ as $Arr(\mu X_\beta.m_0)(\bar{\chi})$.

It remains to be shown that $code_\beta(Obs[\![m[\bullet/\bullet]]\!]\gamma\eta)$ is arithmetical. We give an argument based on recursive enumerability. A set ρ is called recursively enumerable with respect to sets $\rho_1, \cdots, \rho_n$ if ρ can be enumerated by a Turing machine TM that is allowed to consult so called oracles to decide whether some object is in ρ_i or not. ([Hop].) Call a mixed term $m(X^1_{\beta_1}, \cdots X^n_{\beta_n})$ r.e. with respect to $X^1_{\beta_1}, \cdots X^n_{\beta_n}$ if $code_\beta(Obs[\![m]\!]\gamma\eta)$ is r.e. with respect to $code_{\beta_1}(\eta(X^1_{\beta_1})), \cdots, code_{\beta_n}(\eta(X^n_{\beta_n}))$.

Theorem 6.3

Every mixed term $m(X^1_{\beta_1}, \cdots X^n_{\beta_n})$ not containing subterms of the form $(\chi|\beta)$ is recursively enumerable with respect to $X^1_{\beta_1}, \cdots X^n_{\beta_n}$.

□

Theorem 6.4 If m is r.e. with respect to $X^1_{\beta_1}, \cdots X^n_{\beta_n}$, then $m(\bar{\chi}|\bar{\beta})$ is arithmetical.

□

Proof of theorem 6.3

We heavily rely on general techniques as occurring in, for instance, [Hop].

What must be seen is the following:

There exists a TM M that given as input a mixed term m and the codes for oracle TM's for the parameters $X^1_{\beta_1}, \cdots, X^n_{\beta_n}$ produces the code for a TM O(m) that enumerates the (codes of the) computations of m.

The reason that it does not suffice to show mere *existence* of O is that then our induction argument below gets stuck for the case of (nested) recursion.

The proof is by means of induction on the number of recursion constructs appearing within m. (That is, there is a different M for each number of recursion constructs.) The TM M for some *given* number of recursion constructs in m, constructs O by means of a syntax directed translation. That is, for a composed m, M first creates TM's O(m_i) for the parts m_i, and then uses these for the construction of O(m). This means a case distinction, according to the various **TNP** constructs. We treat a few interesting cases. We show what the structure of O(m) must be. That a TM M exists that *constructs* O(m) will be reasonably clear in the cases below.

- Constructs such as hiding.

  Let m be of the form $m_1\backslash\mathbf{c}$. Our TM M must construct $O(m)$ from $O(m_1)$. Now $O(m)$ is as follows: It simulates $O(m_1)$, and for each (code of a) computation generated during this simulation, it removes all $\mathbf{c}$ communications. The result is then output generated by our machine $O(m)$.

- Kernel operation

  Assume m is of the form $\mathit{Kern}(m_1)$. This case is different from operations such as hiding since Kern is not the pointwise extension of some operation on single computations. However, if we can enumerate the codes of computations of m_1, then we can also enumerate finite sets of such codes. For each of such sets, our TM $O(m)$ can check whether it consists of the code of some computation δ together with the codes of all δ' such that $\delta' \sqsubseteq \delta$. If this is the case, then our TM outputs the code of δ, otherwise it proceeds immediately with generating the next finite set. This is seen to generate exactly the codes for $\mathit{Kern}(m_1)$.

- Process variables.

  Obvious, since we have available oracle TM's for such variables.

- Recursion.

  So assume that m is of the form $\mu X_\beta.m_1$. The syntactic approximations $m_1^{[i]}$, as defined in chapter 3, do not have a simpler syntactic structure than m, but the number of recursion constructs within them is one less than for m. So we may assume by induction that there is some TM M' that constructs $O(m_i)$ from input m_i and the oracle TM's. (Note that we do not need an extra oracle for X_β because this variable does not occur free in any of the $m^{[i]}$.) The TM M, that *we* are to construct, must construct $O(m)$. The latter TM operates as follows: By means of "dovetailing" it enumerates the syntactic approximations $m^{[i]}$, for each $m^{[i]}$ it first constructs $O(m^{[i]})$ using the TM M', and finally it simulates $O(m^{[i]})$, where the codes generated during this simulation are included in the enumeration of our TM $O(m)$. This generates exactly all codes for the (infinite) union of the semantic denotations for the $m^{[i]}$, and from chapter 3 it follows that this equals the set of codes for the semantics of the recursion construct $\mu X_\beta.m_1$.

□

Proof of theorem 6.4

The proof that r.e. sets are arithmetical is completely standard, and can be found in [Schoen]. Now we have sets that are r.e. *relative* to given oracles. But the sets enumerated by these oracles are certainly arithmetical since they equal $code_{\beta_i}([\![\chi_i|\beta_i]\!]\gamma$, where χ_i is by definition within our assertion language.

□

6.4 Expressiveness of characteristic assertions

We prove a theorem that is a generalization of lemma 5.2 to the case of *arbitrary* mixed terms, rather than atomic processes only. It is shown that a mixed term m and its characteristic assertion $A_\beta(m)(\bar{\chi})$ have the same semantics as far as the channels and variables in β are concerned, and provided that m is placed in a context where its free process variables have the same semantics as the parameters $\bar{\chi}$, again as far as the bases of these variables are concerned.

Theorem 6.5 Let $base(m) \subseteq \beta$ and $pvar(m) \subseteq \{X^1_{\beta_1}, \ldots, X^n_{\beta_n}\}$. For $A(m)(\bar{\chi})$ defined as above:

$$(\bigwedge_{i=1..n} X^i_{\beta_i} = \chi_i|\beta_i) \rightarrow (A_\beta(m)(\bar{\chi}))|\beta = m$$

Corollary 6.6

$$(A_\beta(m)(\bar{\chi}))|\beta = m(\bar{\chi}|\bar{\beta})$$

□

Proof

Clearly the corollary follows from the theorem since

$$(\bigwedge_i X^i_{\beta_i} = \chi_i|\beta_i) \rightarrow m = m(\bar{\chi}|\bar{\beta})$$

is a valid formula.

The proof of the theorem is by induction on the structure or m. To avoid tedious notation we omit in most cases the $\bar{\chi}$ parameter list and also implicitly assume the premisse $\bigwedge_{i=1..n} X^i_{\beta_i} = \chi_i|\beta_i$

- Atomic processes

  These have already been taken care of in the proof of lemma 5.2

- Specification

$$Obs[\![\chi|\beta]\!]\gamma\eta = [\![\chi]\!]\gamma|\beta = [\![A(\chi|\beta)(\bar{\chi})]\!]\gamma|\beta$$

- cases $m \equiv \mathbf{Z}$ and $m \equiv \mathbf{1}$. These are completely similar to **abort** and **skip**.
- Process variable.

  That is, assume m is one of the free process variables $X^i_{\bar{\beta}_i}$. Then:

  $$Obs[\![m(\bar{\chi}|\bar{\beta})]\!]\gamma\eta = Obs[\![\chi_i|\beta_i]\!]\eta\gamma.$$
  $$= [\![\chi_i]\!]\gamma|\beta_i = [\![A(X^i_{\bar{\beta}_i})(\bar{\chi})]\!]\gamma|\beta_i$$
  $$= [\![A(m)(\bar{\chi})]\!]\gamma|base(m)$$

- Channel hiding and renaming

  First a simple lemma. Assume that f is some operation on traces. Recall that this induces an operation f on Δ as well as on $P(\Delta)$. Moreover, assume that f is representable in our assertion language.

  Lemma 6.7

  $$(f([\![\chi]\!]\gamma))|\mathbf{c} = ([\![\exists t[\chi[t/h] \wedge h|\mathbf{c} = f(t)|\mathbf{c}]]\!]\gamma)|\mathbf{c}.$$

  Proof

  $$([\![\exists t[\chi[t/h] \wedge h|\mathbf{c} = f(t)|\mathbf{c}]]\!]\gamma)|\mathbf{c} =$$
  $$\{(s_0, h, s) \mid \exists h', h'' \in \mathcal{T}race \text{ such that } h = h'|\mathbf{c},$$
  $$(s_0, h', s) \in [\![\chi[t/h] \wedge h|\mathbf{c} = f(t)|\mathbf{c}]\!]\gamma[h''/t]\}$$
  $$= \{(s_0, h, s) \mid \exists h', h'' \in \mathcal{T}race \text{ such that } h = h'|\mathbf{c},$$
  $$(s_0, h'', s) \in [\![\chi]\!]\gamma \text{ and } h'|\mathbf{c} = f(h'')|\mathbf{c}\}$$
  $$= \{(s_0, h, s) \mid \exists h'' \in \mathcal{T}race \text{ such that}$$
  $$(s_0, h'', s) \in [\![\chi]\!]\gamma \text{ and } h = f(h'')|\mathbf{c}\}$$
  $$= (f([\![\chi]\!]\gamma))|\mathbf{c}.$$

  □

  Using this lemma we handle channel hiding as follows: Let $\mathbf{c}_1 = chan(m_1), \mathbf{x}_1 = var^{(\circ)}(m_1)$.

  $$Obs[\![m_1\backslash\mathbf{c}]\!]\gamma\eta = (Obs[\![m_1]\!]\gamma\eta)\backslash\mathbf{c} =$$
  $$(Obs[\![m_1]\!]\gamma\eta)|(Chan - \mathbf{c}) =$$
  $$([\![A(m_1)]\!]\gamma)|\mathbf{c}_1|\mathbf{x}_1|(Chan - \mathbf{c}) =$$

Here the last but one equality follows from the fact that:

$$(s_0, h, s'[\bar{w}/\bar{x}])|(\mathbf{x}_1 - \mathbf{x}) = (s_0, h, s')|(\mathbf{x}_1 - \mathbf{x}).$$

□

Let $(\mathbf{c}_1, \mathbf{x}_1) = base(m_1)$ (So $(\mathbf{c}_1, \mathbf{x}_1 - \mathbf{x}) = base(m_1\backslash\mathbf{x})$).

Using the lemma, one sees that:

$$Obs[\![m_1\backslash\mathbf{x}]\!]\gamma\eta = (Obs[\![m_1]\!]\gamma\eta)\backslash\mathbf{x} =$$
$$(Obs[\![m_1]\!]\gamma\eta)|(\mathcal{V}ar - \mathbf{x}) =$$
$$(([\![A(m_1)]\!]\gamma)|\mathbf{c}_1|\mathbf{x}_1)|(\mathcal{V}ar - \mathbf{x}) =$$
$$([\![A(m_1)]\!]\gamma)|(\mathbf{x}_1 - \mathbf{x})|\mathbf{c}_1 =$$
$$([\![\exists\bar{v}(A(m_1)[\bar{v}/\bar{x}])]\!]\gamma)|(\mathbf{x}_1 - \mathbf{x})|\mathbf{c}_1 =$$
$$([\![A(m_1)\backslash\mathbf{x}]\!]\gamma)|\mathbf{c}_1|(\mathbf{x}_1 - \mathbf{x}).$$

- Kern

Lemma 6.9

If $\rho \in \mathcal{P}(\Delta)$ is saturated w.r.t. $(\mathcal{C}han, \mathcal{V}ar) - \beta$ in the sense that $abase(\rho) \subseteq \beta$ then

$$Kern(\rho|\beta) = Kern(\rho)|\beta.$$

Proof: Let ρ be as indicated. Then $\rho\uparrow\beta = \rho$. This has the following consequences:

$\delta \in \rho$ implies $\delta|\beta \in \rho|\beta \subseteq \rho\uparrow\beta = \rho$, implies $\delta \in \rho$.

$\delta|\beta \in \rho$ implies $\delta|\beta|\beta \in \rho|\beta$, implies $\delta|\beta \in \rho|\beta$.

From this we conclude that if $\rho' \in \mathcal{P}(\Delta)$, then $\rho' \subseteq \rho$ iff $\rho'|\beta \subseteq \rho$. In the proof below we apply this to the set $down(\delta')$ defined as:

$$down(\delta') = \{\delta \in \Delta \mid \delta \sqsubseteq \delta'\}.$$

We also use, in the one but last step, the fact that $\rho|\beta \subseteq \rho\uparrow\beta = \rho$.

Take some arbitrary $\delta \in \Delta$. Then:

$\delta \in Kern(\rho)|\beta$ iff

$\exists\delta'(\delta = \delta'|\beta$ and $\delta' \in Kern(\rho))$ iff

$\exists\delta'(\delta = \delta'|\beta$ and $down(\delta') \subseteq \rho)$ iff

$\exists\delta'(\delta = \delta'|\beta$ and $down(\delta')|\beta \subseteq \rho)$ iff

$([\![A(m_1)]\!]\gamma)|(\mathbf{c}_1 - \mathbf{c})|\mathbf{x}_1 =$ (By the lemma, choosing f to be identity)

$([\![\exists t\Big(A(m_1)[t/h] \wedge h|(\mathbf{c}_1 - \mathbf{c}) = t|(\mathbf{c}_1 - \mathbf{c})\Big)]\!]\gamma)|(\mathbf{c}_1 - \mathbf{c})|\mathbf{x}_1$

$([\![A(m_1)\backslash \mathbf{c}]\!]\gamma)|(\mathbf{c}_1 - \mathbf{c}, \mathbf{x}_1).$

Next we handle channel renaming by choosing $f(h)$ to be $(h|\mathbf{c}_1)[d/c]$. Let $(\mathbf{c}_1, \mathbf{x}_1) = base(m_1\langle d/c\rangle)$. From rules (vii),(viii) and (ix) in chapter 4, it follows that:

$(h|\mathbf{c}_1)[d/c] = (h|\mathbf{c}_1|\mathbf{c}_1 \cup \{c,d\})[d/c] = ((h|\mathbf{c}_1)[d/c])|\mathbf{c}_1 \cup \{c,d\}$

$= ((h|\mathbf{c}_1)[d/c])|((\mathbf{c}_1 - \{c\}) \cup \{d\}) = ((h|\mathbf{c}_1)[d/c])|\mathbf{c}.$

This equality, and the fact that the $|\mathbf{x}_1$ and $[d/c]$ operations commute from the basis for the following derivation:

$Obs[\![m_1\langle d/c\rangle]\!]\gamma\eta = (Obs[\![m_1]\!]\gamma\eta)[d/c] =$

$([\![A(m_1)]\!]\gamma|\mathbf{c}_1|\mathbf{x}_1)[d/c] =$

$(([\![A(m_1)]\!]\gamma|\mathbf{c}_1)[d/c])|\mathbf{c}|\mathbf{x}_1 =$ (By the lemma)

$([\![\exists t[A(m_1)[t/h] \wedge h|\mathbf{c} = (t|\mathbf{c}_1)[d/c]|\mathbf{c}]]\!]\gamma)|\mathbf{c}|\mathbf{x}_1 =$

$([\![\exists t[A(m_1)[t/h] \wedge h|\mathbf{c} = (t|\mathbf{c}_1)[d/c]]]\!]\gamma)|\mathbf{c}|\mathbf{x}_1 =$

$([\![A(m_1\langle d/c\rangle)]]\!]\gamma)|\mathbf{c}_1|\mathbf{x}_1.$

- Variable hiding

First a lemma similar to that used for channel hiding.

Lemma 6.8

$[\![\chi]\!]\gamma|(\mathbf{x}_1 - \mathbf{x}) = ([\![\exists \bar{v}(\chi[\bar{v}/\bar{x}])]\!]\gamma)|(\mathbf{x}_1 - \mathbf{x}).$

Proof

$([\![\exists \bar{v}(\chi[\bar{v}/\bar{x}])]]\!]\gamma)|(\mathbf{x}_1 - \mathbf{x}) =$

$\{(s_0, h, s)|\exists s' \in State_\perp$ such that

$(s_0, h, s') \in [\![\exists \bar{v}(\chi[\bar{v}/\bar{x}])]\!]\gamma$ and $(s_0, h, s) = (s_0, h, s')|(\mathbf{x}_1 - \mathbf{x})\} =$

$\{(s_0, h, s)|\exists s' \in State_\perp, \exists \bar{w} \in Val^*$ such that

$(s_0, h, s'[\bar{w}/\bar{x}]) \in [\![\chi]\!]\gamma$ and $(s_0, h, s) = (s_0, h, s')|(\mathbf{x}_1 - \mathbf{x})\} =$

$\{(s_0, h, s)|\exists s'' \in State_\perp$ such that (s_0, h, s'')

$\in [\![\chi]\!]\gamma$ and $(s_0, h, s) = (s_0, h, s'')|(\mathbf{x}_1 - \mathbf{x})\} =$

$([\![\chi]\!]\gamma)|(\mathbf{x}_1 - \mathbf{x}).$

$\exists \delta'(\delta = \delta'|\beta$ and $down(\delta'|\beta) \subseteq \rho)$ iff

$\delta = \delta|\beta$ and $down(\delta) \subseteq \rho$ iff

$down(\delta) \subseteq \rho|\beta$ iff

$\delta \in Kern(\rho|\beta)$

□

We use the lemma to show that the given assertion for $A(Kern(m_1))$ is the appropriate one: Let $(\mathbf{c}, \mathbf{x}) = base(m_1)(= base(Kern(m_1)))$.

$Obs[\![Kern(m_1)]\!]\gamma\eta = Kern(Obs[\![m_1]\!]\gamma\eta) =$

$Kern([\![A(m_1)]\!]\gamma|\mathbf{c}|\mathbf{x}) =$ (By the lemma)

$(Kern([\![A(m_1)]\!]\gamma))|\mathbf{c}|\mathbf{x}.$

- Sequential composition

Let $\beta = base(m_1; m_2) = base(m_1) \cup base(m_2)$. Then:

$Obs[\![m_1; m_2]\!]\gamma\eta = Obs[\![m_1]\!]\gamma\eta \,\hat{\circ}\, Obs[\![m_2]\!]\gamma\eta =$

$([\![A_\beta(m_1)]\!]\gamma)|\beta \,\hat{\circ}\, ([\![A_\beta(m_2)]\!]\gamma)|\beta =$ (By lemma shown below)

$\left([\![A_\beta(m_1)]\!]\gamma \,\hat{\circ}\, [\![A_\beta(m_2)]\!]\gamma\right)|\beta =$

$([\![A_\beta(m_1) \,\hat{\circ}\, A_\beta(m_2)]\!]\gamma)|\beta =$

$([\![A(m_1; m_2)]\!]\gamma)|\beta.$

In this proof we relied on the fact that, in the above case, the projection $|\beta$ distributes over the composition operator $\hat{\circ}$. In general it is not true that for $\rho_1, \rho_2 \in \mathcal{P}(\Delta)$ we have $(\rho_1 \,\hat{\circ}\, \rho_2)|\beta = \rho_1|\beta \,\hat{\circ}\, \rho_2|\beta$! However, if there are assertions χ_1, χ_2 say, such that $\rho_i = [\![\chi_i]\!]\gamma$ and moreover, $(hchan(\chi_i), var^{(\circ)}(\chi_i)) \subseteq \beta$ for $i = 1, 2$, then we do have this distributive property.

In fact this is a consequence of lemma 4.12. The following proof relies heavily on this lemma. Let $\beta = (\mathbf{c}, \mathbf{x}), \{\bar{x}\} = \mathbf{x}$, and let ρ_1, ρ_2 be as indicated. We show that $(\rho_1 \,\hat{\circ}\, \rho_2)|\beta = \rho_1|\beta \,\hat{\circ}\, \rho_2|\beta$ as follows:

For arbitrary $\delta \in \Delta$, we have that

$\delta \in (\rho_1 \,\hat{\circ}\, \rho_2)|\beta$

iff

$\exists s_0, s_1, s_2, h_1, h_2$ such that $\delta = (s_0(h_1, h_2)|\mathbf{c}, s_0|\bar{x} : s_2(\bar{x}))$

and $(s_0, h_1, s_1) \in \rho_1, (s_1, h_2, s_2) \in \rho_2$

iff

$\exists s_0, s_1, s_2, h_1, h_2$ such that $\delta = (s_0, h_1|\mathbf{c}\hat{}h_2|\mathbf{c}, s_0|\bar{x} : s_2(\bar{x}))$

and $(s_0, h_1, s_1) \in \rho_1, (s_0|\bar{x} : s_1(\bar{x}), h_2, s_2) \in \rho_2$

iff

$\exists s_0, s_1', s_2', h_1', h_2'$ such that $\delta = (s_0, h_1'|\mathbf{c}\hat{}h_2'|\mathbf{c}, s_0|\bar{x} : s_2'(\bar{x}))$ and

$(s_0, h_1'|\mathbf{c}, s_0|\bar{x} : s_1'(\bar{x})) \in \rho_1|\beta$, $(s_0|\bar{x} : s_1'(\bar{x}), h_2', s_0|\bar{x} : s_2'(\bar{x})) \in \rho_2|\beta$

iff

$\exists s_0, s_1'', s_2'', h_1'', h_2''$ such that $\delta = (s_0, h_1''h_2'', s_2'')$

and $(s_0, h_1'', s_1'') \in \rho_1|\beta, (s_1'', h_2'', s_2'') \in \rho_2|\beta$

iff

$\delta \in \rho_1|\beta \,\hat{\circ}\, \rho_2|\beta$.

- Choice

  Let $\beta = base(m_1 \textbf{ or } m_2) = base(m_1) \cup base(m_2)$. Then:

  $Obs[\![m_1 \textbf{ or } m_2]\!]\gamma\eta = Obs[\![m_1]\!]\gamma\eta \cup Obs[\![m_2]\!]\gamma\eta =$

  $([\![A_\beta(m_1)]\!]\gamma)|\beta \cup ([\![A_\beta(m_2)]\!]\gamma)|\beta =$

  $\Big([\![A_\beta(m_1)]\!]\gamma \cup [\![A_\beta(m_2)]\!]\gamma\Big)|\beta =$

  $([\![A_\beta(m_1) \vee A_\beta(m_2)]\!]\gamma)|\beta =$

  $([\![A(m_1 \textbf{ or } m_2)]\!]\gamma)|\beta$

  Remark

  This resembles the case of sequential composition, except that distributivity of the projection over set union is clear. In fact,

  $$(\rho_1 \cup \rho_2)|\beta = \rho_1|\beta \cup \rho_2|\beta$$

  follows from the complete additivity of the projection operator, as shown in chapter 3.

- Parallel composition

  In the next derivation we use properties of the projection and chaotic closure operations, the fact that $abase(A_\beta(m)) \subseteq \beta$, and the characterization of the parallel composition operator in lemma 3.35.

Let $\beta = \beta_1 \cup \beta_2 = base(m_1\ \beta_1 \| \beta_2\ m_2)$. Then:

$Obs[\![m_1\ \beta_1 \| \beta_2\ m_2]\!]\gamma\eta = Obs[\![m_1]\!]\gamma\eta\ \beta_1 \| \beta_2\ Obs[\![m_2]\!]\gamma\eta =$

$([\![A_{\beta_1}(m_1)]\!]\gamma)|\beta_1\ \beta_1 \| \beta_2\ ([\![A_{\beta_2}(m_2)]\!]\gamma)|\beta_2 =$

$\Big(([\![A_{\beta_1}(m_1)]\!]\gamma|\beta_1)\uparrow\beta_1 cap([\![A_{\beta_2}(m_2)]\!]\gamma|\beta_2)\uparrow\beta_2\Big)|\beta =$

$\Big(([\![A_{\beta_1}(m_1)]\!]\gamma)\uparrow\beta \cap ([\![A_{\beta_2}(m_2)]\!]\gamma)\uparrow\beta\Big)|\beta =$

$\Big([\![A_{\beta_1}(m_1)]\!]\gamma \cap [\![A_{\beta_2}(m_2)]\!]\gamma\Big)|\beta =$

$\Big([\![A_{\beta_1}(m_1) \wedge A_{\beta_2}(m_2)]\!]\gamma\Big)|\beta =$

$([\![A(m_1\ \beta_1 \| \beta_2\ m_2)]\!]\gamma)|\beta.$

- Process naming

Since $A_\beta(X^0_{\beta_0} = m_0 \textbf{ in } m_1)(\chi_1, \cdots \chi_n) = A_\beta(m_1)(\chi_0, \chi_1, \cdots \chi_n)$,

where $\chi_0 \equiv A_{\beta_0}(m_0)(\chi_1, \cdots \chi_n)$, we must prove:

$$\Big(\bigwedge_{1\le i\le n} X^i_{\beta_i} = \chi_i|\beta_i\Big) \to (X^0_{\beta_0} = m_0 \textbf{in} m_1) = A_\beta(m_1)(\chi_0, \chi_1, \cdots \chi_n)|\beta \quad (1)$$

By induction we may assume:

$$\Big(\bigwedge_{1\le i\le n} X^i_{\beta_i} = \chi_i|\beta_i\Big) \to m_0 = A_{\beta_0}(m_0)(\chi_1, \cdots \chi_n)|\beta_0, \qquad (2)$$

$$\Big(\bigwedge_{0\le i\le n} X^i_{\beta_i} = \chi_i|\beta_i\Big) \to m_1 = A_\beta(m_1)(\chi_0, \chi_1, \cdots \chi_n)|\beta. \qquad (3)$$

Now take some γ, η, such that

$$Tr[\![\bigwedge_{1\le i\le n} X^i_{\beta_i} = \chi_i|\beta_i]\!]\gamma\eta.$$

Then:

$$Obs[\![X^0_{\beta_0} = m_0 \textbf{ in } m_1]\!]\gamma\eta = Obs[\![m_1]\!]\gamma\tilde{\eta}, \qquad (4)$$

where $\tilde{\eta} = \eta[(Obs[\![m_0]\!]\gamma\eta)/X^0_{\beta_0}]$.

Using (2) we see that actually:

$$\tilde{\eta} = \eta[([\![A_{\beta_0}(m_0)(\chi_1, \cdots \chi_n)]\!]\gamma|\beta_0)/X^0_{\beta_0}].$$

By the definition of χ_0 we then see that:

$$Tr[\![\bigwedge_{0\le i\le n} X^i_{\beta_i} = \chi_i|\beta_i]\!]\gamma\tilde{\eta}.$$

So we can use (3) to infer that:

$$\mathcal{T}r[\![m_1 = A_\beta(m_1)(\chi_0, \chi_1, \cdots \chi_n)|\beta]\!]\gamma\tilde{\eta},$$

that is:

$$Obs[\![m_1]\!]\gamma\tilde{\eta} = [\![A_\beta(m_1)(\chi_0, \chi_1, \cdots \chi_n)]\!]\gamma|\beta$$

Combining this with (4) we see that (1) is satisfied, as was to be shown.

- Recursion

  We defined $A(\mu X_\beta.m_0)(\bar{\chi}) = Arr(\mu X_\beta.m_0)(\bar{\chi})$, and we know that

  $$(\mu X_\beta.m_0)(\bar{\chi}|\bar{\beta}) = Arr(\mu X_\beta.m_0)(\bar{\chi})|\beta.$$

  So $(\mu X_\beta.m_0)(\bar{\chi}|\bar{\beta}) = A(\mu X_\beta.m_0)(\bar{\chi})|\bar{\beta}$ follows immediately from the definition of characteristic assertions for μ constructs. This last equality is equivalent to:

  $$(\bigwedge_{1\leq i\leq n} X^i_{\beta_i} = \chi_i|\beta_i) \rightarrow \mu X_\beta.m_0 = A(\mu X_\beta.m_0)(\bar{\chi})|\beta.$$

□

6.5 Characteristic assertions and recursion -again

Although we succeeded in defining $A(\mu X_\beta.m_0)(\bar{\chi})$ as an assertion, we need a more *tractable* representation for the completeness proof to be given below.

For $m = \mu X_\beta.m_0$ define the *syntactic approximations* $\chi^{[i]}$ for $i \geq 0$:

$$\begin{cases} \chi^0 = \textbf{false} \\ \chi^{[i+1]} = A_\beta(m_0)(\chi^{[i]}, \bar{\chi}) \end{cases}$$

(Here we assume that $m_0 = m_0(X_\beta, X^1_{\beta_1}, \cdots X^n_{\beta_n})$).

The characteristic assertion for m could be written by means of an *infinite* disjunction:

$$"A(\mu X_\beta.m_0)(\bar{\chi}) = \bigvee_{i\geq 0} \chi^{[i]}". \qquad (*)$$

Of course such an infinite formula is not an assertion itself. However, we can treat the formula

$$"m = \bigvee_{i\geq 0} \chi^{[i]}"$$

as an abbreviation for the following *semantic* equality:

$$Obs[\![m]\!]\gamma\eta = \bigcup_{i\geq 0} [\![\chi^{[i]}]\!]\gamma.$$

We now prove that in this sense the equality (*) above is valid. So what we want to prove is:

$$(\bigwedge_{1\leq i\leq n} X^i_{\beta_i} = \chi_i|\beta_i) \rightarrow \mu X_\beta.m_0 = \bigvee_{j\geq 0} \chi^{[j]},$$

that is, for arbitrary η, γ :

$$(\bigwedge_{1\leq i\leq n} \eta(\chi^i_{\beta_i}) = [\![\chi_i]\!]\gamma|\beta_i) \Rightarrow$$

$$Obs[\![\mu X_\beta.m_0]\!]\gamma\eta = (\bigcup_{j\geq 0} [\![\chi^{[j]}]\!]\gamma)|\beta.$$

Now from the definition of Obs for recursion it follows that:

$$Obs[\![\mu X_\beta.m_0]\!]\gamma\eta = \bigcup_{j\geq 0} \rho^{[j]}$$

where

$$\begin{cases} \rho^{[0]} = \emptyset \\ \rho^{[j+1]} = Obs[\![m_0]\!](\gamma)(\eta[\rho^{[j]}/\chi_\beta]) \end{cases}$$

Therefore, using the complete additivity of the projection operator $|\beta$, we see that it suffices to show that for all $j \geq 0$:

$$\Big(\bigwedge_{1\leq i\leq n} \eta(X^i_{\beta_i}) = [\![\chi_i]\!]\gamma|\beta_i\Big) \Rightarrow \rho^{[j]} = [\![\chi^{[j]}]\!]\gamma|\beta$$

This last equality then, can be shown to hold by induction on j. So assume that η and γ are such that:

$$\bigwedge_{1\leq i\leq n} \eta(X^i_{\beta_i}) = [\![\chi_i]\!]\gamma|\beta_i$$

- case $j = 0$:

  $$\rho^{[0]} = \emptyset = [\![\mathbf{false}]\!]\gamma|\beta = [\![\chi^{[0]}]\!]\gamma|\beta.$$

- Induction step: Assume $\rho^{[j]} = [\![\chi^{[j]}]\!]\gamma|\beta$. Then:

  $$\rho^{[j+1]} = Obs[\![m_0]\!](\gamma)(\eta[\rho^{[j]}/X_\beta]) = Obs[\![m_0]\!]\gamma\tilde{\eta}, \qquad (1)$$

  where $\tilde{\eta} = \eta[([\![\chi^{[j]}]\!]\gamma|\beta)/X_\beta]$.

  Using also the assumption for η we see that:

  $$\mathcal{T}r[\![(\bigwedge_{1\leq i\leq n} X^i_{\beta_i} = \chi_i|\beta_i) \wedge (X_\beta = \chi^{[j]}|\beta)]\!]\gamma\tilde{\eta}.$$

But this means that theorem 6.5 is applicable, and thus we infer that:

$$Obs[\![m_0]\!]\tilde{\eta}\gamma = [\![A_\beta(m_0)(\chi^{[j]}, \chi_1, \cdots \chi_n)]\!]\gamma|\beta = [\![\chi^{[j+1]}]\!]\gamma|\beta$$

Together with (1) this shows that

$$\rho^{[j+1]} = [\![\chi^{[j+1]}]\!]\gamma|\beta$$

□

Define the following abbreviations:

$$\chi^{[\omega]} \stackrel{\text{def}}{=} (\mu X_\beta.m_0)(\bar{\chi}),$$

$$\chi^{[\omega+1]} \stackrel{\text{def}}{=} A(m_0)(\chi^{[\omega]}, \bar{\chi}).$$

Lemma 6.10

$\chi^{[\omega]}$ and $\chi^{[\omega+1]}$ are equivalent assertions.

Proof

Since the free channels and variables of both assertions are contained within β, it suffices to show that $\chi^{[\omega]}|\beta = \chi^{[\omega+1]}|\beta$. For the proof of this equality we use the fact that (the semantics of) $\mu X_\beta.m_0$ is given as a (least) fixed point of (the semantics of) m_0. Therefore we know that:

$$m_0[(\mu X_\beta.m_0)/X_\beta] = \mu X_\beta.m_0.$$

Consequently:

$$\chi^{[\omega+1]}|\beta = A(m_0)(\chi^{[\omega]}, \bar{\chi})|\beta = m_0[(\chi^{[\omega]}|\beta)/X_\beta, - - -] =$$

$$m_0[(\mu X_\beta.m_0)/X_\beta, - - -] = \mu X_\beta.m_0[- - -] = A(\mu X_\beta.m_0)(\bar{\chi})|\beta = \chi^{[\omega]}|\beta.$$

□

6.6 Compositional completeness of the SAT system

We are in the position to give the proof of one of the main results of this thesis: that of the *compositional completeness* of the SAT system.

First we show in lemma 6.12 that the characteristic assertion for some mixed term m is *satisfied* by this mixed term, and moreover that it is the *strongest* assertion with this property.

Then, in lemma 6.13, essentially the completeness proof is given for all language constructs except recursion and process naming.

Finally, in theorem 6.15, we prove the compositional completeness of the whole SAT system.

Lemma 6.11

If $pvar(m) \subseteq \{\varsigma_i, \cdots \varsigma_n\}$, $base(m) \subseteq \beta$ and H is some set of hypotheses, of the form $\{\varsigma_1\ sat\ \chi_1, \ldots, \varsigma_n\ sat\ \chi_n\}$, then

$$H \models m\ sat\ A_\beta(m)(\chi_1, \cdots \chi_n). \qquad (1)$$

Moreover if $abase(\chi) \subseteq \beta$, and

$$H \models m\ sat\ \chi, \qquad (2)$$

then the following assertion is valid:

$$A_\beta(m)(\chi_1, \cdots \chi_n) \rightarrow \chi \qquad (3)$$

□

Proof

$\varsigma_i\ sat\ \chi_i$ implies $\varsigma_i \subseteq \chi_i|\beta_i$ where $\beta_i = base(\varsigma_i)$. So, using the monotonicity of $m(\varsigma_i, \cdots \varsigma_n)$, we see that :

$$\{\varsigma_1\ sat\ \chi_1, \ldots, \varsigma_n\ sat\ \chi_n\} \models m \subseteq m(\bar{\chi}|\bar{\beta}),$$

where $m(\bar{\chi}|\bar{\beta})$ abbreviates $m[(\chi_1|\beta_1)/\varsigma_1, \cdots, (\chi_n|\beta_n)/\varsigma_n]$.

On the other hand we know from corollary 6.6 that:

$$m(\bar{\chi}|\bar{\beta}) = A(m)(\chi_1, \cdots \chi_n)|base(m).$$

Therefore we conclude that:

$$\{\varsigma_1\ sat\ \chi_1, \ldots, \varsigma_n\ sat\ \chi_n\} \models m \subseteq A(m)(\chi_1, \cdots \chi_n)|base(m)$$

is valid, that is, (1) is valid, indeed.

Now assume that (2) is valid. Note that (2) implies that $m(\bar{\chi}|\bar{\beta})\ sat\ \chi$, and so, $m(\bar{\chi}|\bar{\beta}) \subseteq \chi$. Since, by corollary 6.6, $m(\bar{\chi}|\bar{\beta}) = A_\beta(m)(\chi_1, \cdots \chi_n)|\beta$ we see that:

$$A_\beta(m)(\chi_1, \cdots \chi_n)|\beta \subseteq \chi. \qquad (4)$$

It is easily proven that for arbitrary χ_a, χ_b with $abase(\chi_a) = \beta_a \supseteq abase(\chi_b)$ that if $\chi_a|\beta_a \subseteq \chi_b$, then $\chi_a \subseteq \chi_b$, that is, $\chi_a \rightarrow \chi_b$ is valid in this case. To see this, let $\rho_a = [\![\chi_a]\!]\gamma, \rho_b = [\![\chi_b]\!]\gamma$, and assume that $\rho_a|\beta_a \subseteq \rho_b$. Then we have the following (in)equalities:

$$\rho_a = \rho_a \uparrow \beta_a = \rho_a|\beta_a \uparrow \beta_a \subseteq \rho_b \uparrow \beta_a = \rho_b.$$

Applying this to the situation above one sees that from (4) it follows that:

$$A_\beta(m)(\chi_1, \cdots \chi_n) \rightarrow \chi$$

is valid, since $abase(A_\beta(m)(\cdots)) = \beta \supseteq abase(\chi)$ by the assumption made for χ.

□

The main step is to show that a characteristic assertion for some mixed term is not only *satisfied* by this term, but that it possible to *formally deduce* that fact. We first prove that this is the case for **TNP** constructs that do not introduce bound process variables. By inspection of the SAT system, one sees that it contains only *compositional proofrules*. Therefore, what we prove in lemma 6.13 is essentially *elementary compositional completeness* for the SAT system.

Theorem 6.12 (Completeness w.r.t. characteristic assertions)

Let *"op"* be one of the mixed term operators, where we assume that it is *not* the recursion construct or the process naming construct. Let:

- $m \equiv op(m_1, \cdots, m_k)$,
- $pvar(m) \subseteq \{\varsigma_1, \cdots \varsigma_n\}$
- $\chi^{(m)} \equiv A(m)(\chi_1, \cdots \chi_n)$
- $\chi^{(m_i)} \equiv A(m_i)(\chi_1, \cdots \chi_n)$ for $i = 1 \cdots k$

Then the following is deducible within the SAT system:

$$m_1 \; sat \; \chi^{(m_1)}, \cdots, m_k \; sat \; \chi^{(m_k)} \vdash op(m_1, \cdots, m_k) \; sat \; \chi^{(m)}$$

□

Proof

We check simply that the claim holds for each of the operators in turn.

- *op* is a nullary operator.

  That is, $k = 0$ and so $m \equiv op$. Then m is either one of the atomic processes α, or it is a predicative term $\chi|\beta$.

  Hence, we must show the following for atomic processes:

  $$\vdash \alpha \; sat \; A(\alpha).$$

This is the case, indeed, since by inspection of the system one sees that for each of the atomic processes, α *sat* $A(\alpha)$ is one of the SAT axioms for $\alpha \in Atom$.

For predicative processes $m \equiv \chi|\beta$, we must show that

$$\vdash \chi|\beta \; sat \; \chi,$$

which follows immediately from the presence of the axiom for predicative processes.

- $m \equiv m_1\backslash \mathbf{c}$

  $\chi^{(m_1)} = A(m_1)(\bar{\chi})$

  $\chi^{(m)} = \exists t(\chi^{(m_1)}[t/h] \wedge h|\mathbf{c}' = t|\mathbf{c}')$

  where $\mathbf{c}' = chan(m_1\backslash \mathbf{c}) = chan(m_1) - \mathbf{c}$

  Note that $\chi^{(m_1)} \rightarrow \chi^{(m)}$ is valid, and that $hchan(\chi^{(m)}) \cap \mathbf{c} = \emptyset$. This justifies the following derivation:

$$\frac{m_1 \; sat \; \chi^{(m_1)}}{m_1 \; sat \; \chi^{(m)}} \text{ (Consequence)}$$

$$\frac{m_1 \; sat \; \chi^{(m)}}{m_1\backslash \mathbf{c} \; sat \; \chi^{(m)}} \text{ (Channel hiding)}$$

- $m \equiv m_1\backslash \mathbf{x}$

  $\chi^{(m_1)} = A(m_1)(\bar{\chi})$

  $\chi^{(m)} = \exists \bar{x}(\chi^{(m_1)})$, where $\{\bar{x}\} = \mathbf{x}$

  Similar to the case above, we have that $\chi^{(m_1)} \rightarrow \chi^{(m)}$ is valid and that $var(\chi^{(m)}) \cap \mathbf{x} = \emptyset$. So the obvious derivation is:

$$\frac{m_1 \; sat \; \chi^{(m_1)}}{m_1 \; sat \; \chi^{(m)}} \text{ (Consequence)}$$

$$\frac{m_1 \; sat \; \chi^{(m)}}{m_1\backslash \mathbf{x} \; sat \; \chi^{(m)}} \text{ (Variable hiding)}$$

- $m \equiv m_1\langle d/c\rangle$

  $\chi^{(m_1)} = A(m_1)(\bar{\chi})$

$$\chi^{(m)} = \exists t(\chi^{(m_1)}[t/h] \wedge h|\mathbf{c} = (t|\mathbf{c}_1)[d/c])$$

where $\mathbf{c} = chan(m_1\langle d/c\rangle) = (chan(m_1) - \{c\}) \cup \{d\}$

and $\mathbf{c}_1 = chan(m_1)$.

Let $\tilde{\mathbf{c}} = \{c, d\} - \mathbf{c}_1$. First we show that the following assertion is valid:

$$(\chi^{(m_1)} \wedge h|\tilde{\mathbf{c}} = \varepsilon) \rightarrow (\chi^{(m)}[h[d/c]/h]). \qquad (*)$$

So assume $\chi^{(m_1)}$ and $h|\tilde{\mathbf{c}} = \varepsilon$. We must prove the existence of a trace t such that both

$$\chi^{(m_1)}[t/h] \text{ and } h[d/c]||\mathbf{c} = (t|\mathbf{c}_1)[d/c]$$

are true. Now choose $t = h$. Then clearly the first of these latter two is satisfied. Note that $h|\tilde{\mathbf{c}} = \varepsilon$ implies that $h|\mathbf{c}_1 = h|(\mathbf{c}_1 \cup \tilde{\mathbf{c}}) = h|(\mathbf{c}_1 \cup \{c, d\})$. Using this equality and the algebraic laws from chapter 4, one sees that:

$$h[d/c]||\mathbf{c} = h[d/c]||(\mathbf{c} \cup \{c\}) = h[d/c]||(\mathbf{c}_1 \cup \{c, d\})$$
$$= (h|(\mathbf{c}_1 \cup \{c, d\}))[d/c] = (h|\mathbf{c}_1)[d/c].$$

So if we choose $t = h$, then the second of the two assertions above is satisfied too.

The assertion $(*)$ forms the basis for the following derivation:

$$\dfrac{\dfrac{\dfrac{m_1 \; sat \; \chi^{(m_1)} \quad , \quad m_1 \; sat \; h|\tilde{\mathbf{c}} = \varepsilon \; \text{(Invariance)}}{m_1 \; sat \; (\chi^{(m_1)} \wedge h|\tilde{\mathbf{c}} = \varepsilon)} \; \text{(Conjunction)}}{m_1 \; sat \; \chi^{(m)}[h[d/c]/h]} \; \text{(Consequence,(*))}}{m_1\langle d/c\rangle \; sat \; \chi^{(m)}} \; \text{(Renaming)}$$

The conclusion is that $m_1 \; sat \; \chi^{(m_1)} \vdash m_1\langle d/c\rangle \; sat \; \chi^{(m)}$, as was to be shown.

- Kern

Let $m = Kern(m_1)$.

If $\chi^{(m_1)} = A(m_1)(\bar{X})$, then $A(Kern(m_1))(\bar{X}) = Kern(\chi^{(m_1)})$.

So the following simple derivation suffices:

$$\frac{m_1 \; sat \; \chi^{(m_1)}}{Kern(m_1) \; sat \; Kern(\chi^{(m_1)})} \text{ (Kern)}$$

- Sequential composition

  Let $m = m_1; m_2, \beta = base(m) = \beta_1 \cup \beta_2$, where $\beta_i = base(m_i)$ for $i = 1, 2$. Recall that $A_\beta(m_i)(\bar{X}) = A(m_i)(\bar{X}) \wedge \mathbf{1}_{\beta - \beta_i}$.

  For $i = 1, 2$ we have the following derivation:

$$\frac{m_i \; sat \; A(m_i)(\bar{X}), \quad m_i \; sat \; \mathbf{1}_{\beta-\beta_i} \text{ (Invariance)}}{m_i \; sat \; A_\beta(m_i)(\bar{X})} \text{ (Conjunction)}$$

  The conclusions of these two derivations are combined by the sequential composition rule:

$$\frac{m_1 \; sat \; A_\beta(m_1)(\bar{X}) \; , \; m_2 \; sat \; A_\beta(m_2)(\bar{X})}{m_1; m_2 \; sat \; A_\beta(m_1)(\bar{X}) \, \hat{\circ} \, A_\beta(m_2)(\bar{X})} \text{ (Sequential composition)}$$

  This is the desired result since $A(m_1; m_2)(\bar{X}) = A_\beta(m_1)(\bar{X}) \hat{\circ} A_\beta(m_2)(\bar{X})$.

  So we conclude that:

$$m_1 \; sat \; A(m_1)(\bar{X}) \, , \; m_2 \; sat \; A(m_2)(\bar{X}) \vdash m_1; m_2 \; sat \; A(m_1; m_2)(\bar{X}).$$

- Choice

  Let $m = m_1$ **or** m_2, and

  let $\beta = base(m) = \beta_1 \cup \beta_2$, where $\beta_i = base(m_i)$ for $i = 1, 2$.

  Similar to the case for sequential composition one can derive $m_i \; sat \; A_\beta(m_i)(\bar{X})$ from $m_i \; sat \; A(m_i)(\bar{X})$, using the invariance axiom. Then the choice rule combines those two formulae:

$$\frac{m_1 \; sat \; A_\beta(m_1)(\bar{X}) \; , \; m_2 \; sat \; A_\beta(m_2)(\bar{X})}{m_1 \textbf{ or } m_2 \; sat \; A_\beta(m_1)(\bar{X}) \vee A_\beta(m_2)(\bar{X})} \text{ (Choice)}$$

  This is what was to be shown since:

$$A(m_1 \textbf{ or } m_2)(\bar{X}) = A_\beta(m_1)(\bar{X}) \vee A_\beta(m_2)(\bar{X}).$$

- Parallel composition

  Let $m = m_1\ \beta_1 \| \beta_2\ m_2$, $\beta = \beta_1 \cup \beta_2$. The syntactic restrictions on mixed terms require that $base(m_i) \subseteq \beta_i$ for $i = 1, 2$.

  As before, $m_i\ sat\ A_{\beta_i}(m_i)(\bar{X})$ is derivable from $m_i\ sat\ A(m_i)(\bar{X})$, using the invariance axiom. Now $abase(A_{\beta_i}(m_i)(\bar{X})) \subseteq \beta_i$, and so the following application of the rule for parallel composition is justified:

$$\frac{m_1\ sat\ A_{\beta_1}(m_1)(\bar{X})\ ,\ m_2\ sat\ A_{\beta_2}(m_2)(\bar{X})}{m_1\ \beta_1 \| \beta_2\ m_2\ sat\ A_{\beta_1}(m_1)(\bar{X}) \wedge A_{\beta_2}(m_2)(X)}\ \text{(Parallel composition)}$$

  This was to be shown since the conclusion of this rule is the required assertion $A(m_1\ \beta_1 \| \beta_2\ m_2)(\bar{X})$.

□

Finally we arrive at the main theorem:

Theorem 6.13

The SAT system is compositionally complete.

□

According to the definition of compositional completeness this means the following. Let $mspec(m)$ denote a modular specification for the SAT system. We shall use the "natural deduction form" for such specifications in this chapter.

That is, for a modular specification

$$mspec(m) \stackrel{\text{def}}{=} \Big(\bigwedge_{i=1..n} \varsigma_i\ sat\ \chi_i \Big) \rightarrow m\ sat\ \chi$$

we shall freely use the equivalent formula:

$$H \vdash m\ sat\ \chi,$$

where H is the set of hypotheses $\varsigma_1\ sat\ \chi_1, \ldots, \varsigma_n\ sat\ \chi_n$. As argued in chapter 1 we may assume that the process variables ς_i are all distinct.

Compositional completeness the amounts to the following:

If m is of the form $C(m_1, \ldots, m_k)$, where C is one of the mixed term constructs, and $mspec(m)$ is valid, that is,

$$H \models m\ sat\ \chi,$$

then there exist $mspec_1, \ldots, mspec_k$ such that:

(a) $mspec_i(m_i)$ is valid for i=1..k.

(b) $mspec(C(m_1, \ldots, m_k))$ is deducible from the hypotheses $mspec_i(m_i)$.

Note that C can be one of the mixed term operators, but that it can also be the process naming construct or the recursion construct.

Proof

We may assume that H contains only specifications for process variables that actually *do* occur free in m. For if this is not the case we first show the deducibility of $(H \cap pvar(m)) \vdash m\ sat\ \chi$ and then apply the weakening rule of chapter 4 to deduce $H \vdash m\ sat\ \chi$.

The definition of compositional completeness allows a completely free choice of modular specifications $mspec_i$ for the parts m_i as long as (a) and (b) are satisfied. But actually we can limit our choice as follows. If $pvar(m_i) = pvar(m)$, that is, C does not bind a process variable or else the bound variable does not occur free in m_i, then we shall choose the hypotheses part of $mspec_i$ *equal* to the corresponding part $(\bigwedge_{i \leq n} \varsigma_i\ sat\ \chi_i)$ of $mspec(m)$. So in this case $mspec_i(m_i)$ is of the form:

$$H \vdash spec^{(m_i)}(m_i),$$

where $spec^{(m_i)}(m_i)$ is a SAT specification of the form $m_i\ sat\ \chi_i$.

And if C *does* bind the process variable θ_1 and this variable occurs free in m_i, then we shall choose $mspec_i$ of the form:

$$H \cup \{\theta_1\ sat\ \chi^{(\theta_1)}\} \vdash spec^{(m_i)}(m_i),$$

where $spec^{(m_i)}(m_i)$ is a SAT formula as above.

So let m be some mixed term with $pvar(m) \subseteq \{\varsigma_1 \cdots \varsigma_n\}$ and let $\chi, \chi_1, \cdots \chi_n$ be assertions such that, for H as above,

$$H \models m\ sat\ \chi \qquad (*).$$

Either m is of the form $op(m_1, \cdots, m_k)$ where op is one of the mixed term operators, or m is a process naming or recursion construct.

Case 1. m of the form $op(m_1, \cdots, m_k)$ for some $k \leq 0$.

This is a case where no process variables are bound, and consequently:

$$pvar(m_i) \subseteq pvar(m) \subseteq \{\varsigma_1 \cdots \varsigma_n\} \text{ for } i \in 1 \cdots k.$$

We must show the existence of specifications $spec^{(m_i)}$ such that clause (a) and (b) from the definition of compositionality are satisfied. We make the following choice:

$$spec^{(m_i)}(m_i) \equiv m_i \; sat \; A(m_i)(\chi_1, \cdots \chi_n).$$

We must prove, for i=1..k,

(a) $H \models m_i \; sat \; A(m_i)(\chi_1, \cdots \chi_n)$, and

(b) $H \vdash op(m_1, \cdots m_k) \; sat \; \chi$ is deducible from the following set of specifications:

$H \vdash m_i \; sat \; A(m_i)(\chi_1, \cdots \chi_n)$, where $i \in 1 \cdots k$.

Now (a) follows easily from lemma 6.12 above. To prove (b) it *suffices* to show that $op(m_1, \cdots m_k) \; sat \; \chi$ is derivable from hypotheses $m_i \; sat \; A(m_i)(\bar{\chi})$. The transitivity rule of chapter 4 then ensures that (b) is satisfied.

By theorem 6.13, $op(m_1, \cdots, m_k) \; sat \; A(m)(\bar{\chi})$ is derivable from the given hypotheses.

Let $\tilde{\beta} = base(m) \cup (hchan(\chi), var^{(\circ)}(\bar{\chi}))$. By lemma 6.12, $A_{\tilde{\beta}}(m)(\bar{\chi}) \rightarrow \chi$ is a valid assertion. This justifies the application of the consequence rule in the following derivation.

$$\dfrac{\dfrac{op(m_1, \cdots m_k) \; sat \; A(m)(\bar{\chi}) \;,\; op(m_1, \cdots m_k) \; sat \; \mathbf{1}_{\tilde{\beta}-\beta} \; (Invariance)}{op(m_1, \cdots m_k) \; sat \; A_{\tilde{\beta}}(m)(\bar{\chi})} \; (\text{Conj.})}{op(m_1, \cdots m_k) \; sat \; \chi} \; (\text{Consequence})$$

The conclusion of this derivation is the desired specification.

Case 2. m is of the form $\theta_1 = m_1 \; \mathbf{in} \; m_2$.

Here we have the following situation with respect to free process variables:

$pvar(m_1) \subseteq pvar(m)$,

$pvar(m_2) \subseteq \{\theta_1\} \cup pvar(m)$.

Let

$\chi^{(m_1)} \equiv A(m_1)(\chi_1, \cdots, \chi_n)$, and

$\chi^{(m_2)} \equiv A(m_2)(\chi^{(m_1)}, \chi_1, \cdots, \chi_n)$.

Choose $spec^{(m_i)}(m_i) \equiv m_i \; sat \; \chi^{(m_i)}$ for $i = 1, 2$, and take $spec^{(\theta_1)} \equiv spec^{(m_1)}$.

It suffices to show the following:

(a') $H \models m_1 \; sat \; \chi^{(m_1)}$ and $H \cup \{\theta_1 \; sat \; \chi^{(m_1)}\} \models m_2 \; sat \; \chi^{(m_2)}$.

(b') $H \vdash \theta_1 = m_1 \; \mathbf{in} \; m_2 \; sat \; \chi$ is deducible from:

$H \vdash m_1 \; sat \; \chi^{(m_1)}$, together with: $H \cup \{\theta_1 \; sat \; \chi^{(m_1)}\} \vdash m_2 \; sat \; \chi^{(m_2)}$.

As before, (a') follows from lemma 6.12.

Note that θ_1 does not occur in any of the hypotheses H, by the assumption made above, and the fact that θ_1 does not occur free in m. Therefore, one application of the process naming rule suffices to deduce $H \vdash \theta_1 = m_1 \; \mathbf{in} \; m_2 \; sat \; \chi^{(m_2)}$ from the two hypotheses. Since actually $\chi^{(m_2)} = A(m)(\bar{\chi})$, we can use the same reasoning as for (b) above to deduce $H \vdash \theta_1 = m_1 \; \mathbf{in} \; m_2 \; sat \; \chi$.

Case 3. m is of the form $\mu\theta_1.m_1$

Here, $pvar(m_1) \subseteq \{\theta\} \cup pvar(m)$

Let $\chi^{[w]} = A(\mu\theta_1.m_1)(\chi_1, \cdots, \chi_n)$, and choose $spec^{(m_1)}(m_1) = m_1 \; sat \; \chi^{[w]}$. Take $spec^{(\theta_1)} = spec^{(m_1)}$.

We must prove:

(a") $H \cup \{\theta_1 \; sat \; \chi^{[w]}\} \models m_1 \; sat \; \chi^{[w]}$, and

(b") $H \vdash \mu\theta_1.m_1 \; sat \; \chi$ is deducible from: $H \cup \{\theta_1 \; sat \; \chi\} \vdash m_1 \; sat \; \chi^{[w]}$.

If $\chi^{[w]} \equiv A(m_1)(\chi^{[w]}, \chi_1, \cdots, \chi_n)$, then from lemma 6.12 above it follows that:

$$H \cup \{\theta_1 \; sat \; \chi^{[w]}\} \models m_1 \; sat \; \chi^{[w+1]}$$

is valid. Lemma 6.11 guarantees that $\chi^{[w]} \leftrightarrow \chi^{[w+1]}$ is valid too. From these two facts it directly follows that (a") is satisfied.

The proof of (b")is fairly similar to that of (b').

Again we may assume that the variable θ_1 does not occur free in the hypotheses H. Using the recursion rule one first deduces $H \vdash \mu\theta_1.m_1 \; sat \; \chi^{[w]}$, and then, in the way described under (b), deduces $H \vdash \mu\theta_1.m_1 \; sat \; \chi^{[w]}$,.

This ends case 3, and therefore the proof of the theorem.

□

6.7 Modular Completeness

Our aim is to show that the SAT system is modular complete. Since we already know that the system is compositionally complete, it remains to see that it is *adaptation complete*. For then theorem 1.9 of chapter 1 guarantees that the system is modular complete too.

Theorem 6.14

The SAT system is adaptation complete.

□

Proof

Assume that for certain process variable ς and assertions χ, χ' the following implication is valid:

$$\varsigma \; sat \; \chi \rightarrow \varsigma \; sat \; \chi'. \qquad (1)$$

We must show that $\varsigma \; sat \; \chi'$ is derivable from $\varsigma \; sat \; \chi$. To this end we prove first a simple lemma.

Lemma 6.15

Let $base(\varsigma) = \beta$ and $abase(\chi, \chi') = \beta'$, and let (1) be valid.

Then the assertion $(\chi \wedge \mathbf{1}_{\beta'-\beta}) \rightarrow \chi'$ is (strictly) valid.

□

To prove this we must show for arbitrary $\gamma \in \Gamma$ that

$$[\![\chi \wedge \mathbf{1}_{\beta'-\beta}]\!]\gamma \subseteq [\![\chi']\!]\gamma \qquad (2).$$

By interpreting (1) we see that for all $\gamma \in \Gamma$ and all $\rho \in \mathcal{P}(\Delta_\beta)$ the following holds:

$$\rho \subseteq [\![\chi]\!]\gamma \rightarrow \rho \subseteq [\![\chi']\!]\gamma.$$

We apply this to the set ρ defined by:

$$\rho \stackrel{\text{def}}{=} ([\![\chi \wedge \mathbf{1}_{\beta'-\beta}]\!]\gamma)|\beta.$$

It is obvious that $\rho \in \mathcal{P}(\Delta_\beta)$. We prove that it is also contained in $[\![\chi]\!]\gamma$. We use the fact that $abase(\chi \wedge \mathbf{1}_{\beta'-\beta}) \subseteq \beta'$.

$$([\![\chi \wedge \mathbf{1}_{\beta'-\beta}]\!]\gamma)|\beta = ([\![\chi \wedge \mathbf{1}_{\beta'-\beta}]\!]\gamma)|(\beta \cup \beta') \subseteq$$

$$([\![\chi \wedge \mathbf{1}_{\beta'-\beta}]\!]\gamma)\uparrow(\beta \cup \beta') = [\![\chi \wedge \mathbf{1}_{\beta'-\beta}]\!]\gamma \subseteq [\![\chi]\!]\gamma.$$

We may conclude from this that:

$$([\![\chi \wedge \mathbf{1}_{\beta'-\beta}]\!]\gamma)|\beta \subseteq [\![\chi']\!]\gamma.$$

But then we see that:

$$[\![\chi \wedge \mathbf{1}_{\beta'-\beta}]\!]\gamma = ([\![\chi \wedge \mathbf{1}_{\beta'-\beta}]\!]\gamma)\uparrow(\beta \cup \beta') =$$
$$([\![\chi \wedge \mathbf{1}_{\beta'-\beta}]\!]\gamma)|(\beta \cup \beta')\uparrow(\beta \cup \beta') =$$
$$([\![\chi \wedge \mathbf{1}_{\beta'-\beta}]\!]\gamma)|\beta\uparrow(\beta \cup \beta') \subseteq ([\![\chi']\!]\gamma)\uparrow(\beta \cup \beta') =$$
$$[\![\chi']\!]\gamma.$$

This proves the lemma.

□

The lemma justifies the use of the consequence rule in the following derivation:

$$\dfrac{\dfrac{\varsigma \; sat \; \chi, \quad \varsigma \; sat \; \mathbf{1}_{\beta'-\beta} \text{ (Invariance)}}{\varsigma \; sat \; \chi \wedge \mathbf{1}_{\beta'-\beta}} \text{ (Conjunction)}}{\varsigma \; sat \; \chi'} \text{ (Consequence)}$$

As was to be shown.

□

Theorem 6.16

The SAT system is modular complete

□

Proof: Immediate from theorems 1.9, 6.13 and 6.14.

Chapter 7

The Hoare and Invariant systems

7.1 The SAT-Hoare transformation

In this chapter we consider the completeness question for the Hoare system and the Invariant system. Rather than proving the completeness of these systems directly from the semantic definition for mixed terms we point out how proofs given within the SAT system can be *transformed* into proofs within the Hoare system. Moreover, proofs within the Hoare system can be transformed into proofs within the Invariant system. In this way we are able to prove the compositional completeness of these two systems.

In chapter 4 we explained that any Hoare formula $(\varphi)\ m\ (\psi)$ can be represented by the SAT formula $m\ sat\ \varphi \rightsquigarrow \psi$. The idea is to transform a proof of the latter formula into a proof of the former one. Below we formulate a theorem that essentially states that if $m\ sat\ \chi$ is provable, in the SAT system that is, then $(\varphi)m(\varphi \lhd \chi)$ is provable in the Hoare system, where φ is arbitrary. Then one can obtain a proof of $(\varphi)\ m\ (\psi)$ by first transforming a proof of $m\ sat\ \varphi \rightsquigarrow \psi$ into a proof of $(\varphi)\ m\ (\varphi \lhd (\varphi \rightsquigarrow \psi))$. One more application of the consequence rule then suffices to prove $(\varphi)\ m\ (\psi)$ since $(\varphi \lhd (\varphi \rightsquigarrow \psi)) \rightarrow \psi$ is a valid implication. A slight complication arises for the transformation of proofs that rely on the introduction and discharge of *hypotheses* concerning free process variables. One problem is that the formula $(\varphi)\ m\ (\varphi \lhd \chi)$ is in general *weaker* than $m\ sat\ \chi$, and for hypotheses this is undesirable. Another problem is that hypotheses specify the behavior of a *black box* that can occur several times within the program text, and so one must be able to *adapt* such specifications. Therefore, we prove that a deduction of $H \vdash m\ sat\ \chi$, where H is a set of SAT formulae $\varsigma_1\ sat\ \chi_1, \ldots, \varsigma_l\ sat\ \chi_l$, can be transformed into a deduction of $\tilde{H} \vdash (\varphi)\ m\ (\varphi \lhd \chi)$, where $\tilde{H}$ is a set of Hoare formulae, $\forall \bar{g}_1[(\varphi_1)\varsigma_1(\psi_1)], \ldots, \forall \bar{g}_l[(\varphi_l)\varsigma_l(\psi_l)]$ that are all *adaptable*

and moreover are *equivalent* to the corresponding SAT formulae. We consider both the Hoare system with and without the "extra adaptation rules" of section 5.4.2. In the latter case not *all* formulae are adaptable. Nevertheless, we prove that also in this case it is always possible to choose an adaptable and equivalent Hoare formulae for some given SAT specification ς *sat* χ.

Theorem 7.1 Transformation of SAT proofs to Hoare proofs

Let $m(m_1, \ldots, m_n)$ be some mixed term with occurrences of the meta variables $m_1, \ldots, m_n$, let χ some satisfiable assertion and let some proof scheme be given that shows how

$$H \vdash m(m_1, \ldots, m_n) \; sat \; \chi$$

is deduced from

$$H_1 \vdash m_1 \; sat \; \chi_1,$$

$$\vdots$$

$$H_n \vdash m_n \; sat \; \chi_n.$$

Let $\tilde{H}$ be a given set of adaptable Hoare formulae equivalent to the formulae in H. Let $\varphi \in \mathcal{A}ssn(\Sigma)$ be arbitrary.

Then, for $i = 1..n$, there are sets $\tilde{H}_i$ of adaptable Hoare formulae equivalent to the SAT formulae in H_i and assertions φ_i, such that

$$\tilde{H} \vdash (\varphi) \; m(m_1, \ldots, m_n) \; (\varphi \lhd \chi)$$

is deducible from:

$$\tilde{H}_1 \vdash (\varphi_1) \; m_1 \; (\varphi_1 \lhd \chi_1),$$

$$\vdots$$

$$\tilde{H}_n \vdash (\varphi_n) \; m_n \; (\varphi_n \lhd \chi_n).$$

□

Lemma 7.2

For each mixed term m and satisfiable assertion χ there exist a list of (fresh) logical variables $\bar{g}$ and an assertion φ such that $\forall \bar{g}[(\varphi) \; m \; (\varphi \lhd \chi)]$ is an adaptable Hoare formula, and moreover the following equivalence is valid:

$$m \; sat \; \chi \leftrightarrow \forall \bar{g}[(\varphi) \; m \; (\varphi \lhd \chi)].$$

□

We start with the proof of the theorem, where we already use the lemma. The proof of the lemma is given thereafter, but of course does not depend on the theorem.

Proof of theorem 7.1

The proof is by induction on the length of the given deduction in the SAT system. We make a case distinction, according to the proof rule used in the last step of this deduction. For most cases, the sets H_i are the same as H, and it will be clear that we then chose the sets $\tilde{H}_i$ equal to $\tilde{H}$. For such cases we avoid mentioning the sets of hypotheses, to avoid lengthy notation.

Some cases below need a separate treatment of bottom - and non bottom *initial* states. It is profitable to treat bottom initial states once and for all. This is done as follows.

Assume a given deduction ends as:

$$\frac{\vdots}{m \; sat \; \chi}$$

Let some $\varphi \in \mathcal{A}ssn(\Sigma)$ be given. Define:

$\varphi^{\perp} \equiv \perp \wedge \varphi[\perp]$,

$\varphi^{\top} \equiv \top \wedge \varphi[\top]$.

In each case *below* we shall prove that

$(\varphi^{\top})\ m\ (\varphi^{\top} \lhd \chi)$ (1)

is derivable. *Here* we treat $(\varphi^{\perp})\ m\ (\varphi^{\perp})$, and how to combine the formulae for the bottom and top case.

We made the assumption that χ is a *satisfiable* predicate, implying that $\chi[\perp^{\circ}]$ is valid. We use this in a (partial) expansion of $\varphi^{\perp} \lhd \chi$:

$\varphi^{\perp} \lhd \chi$ iff

$(\varphi^{\perp}[\perp] \wedge \chi[\perp^{\circ}] \wedge \perp) \vee - - - - -$ iff

$(\mathbf{true} \wedge \varphi[\perp] \wedge \mathbf{true} \wedge \perp) \vee - - - - -$ iff

$(\perp \wedge \varphi[\perp]) \vee - - - - -$

This already suffices to conclude that

$\varphi^{\perp} \rightarrow (\varphi^{\perp} \lhd \chi)$

is a valid implication. We use this in the following derivation, showing the derivability of $\varphi \lhd \chi$:

$$\dfrac{\dfrac{(\varphi^{\perp})\ m\ (\varphi^{\perp})(\text{strictness})}{(\varphi^{\perp})\ m\ (\varphi^{\perp} \lhd \chi)}(\text{Consequence}) \qquad \dfrac{\vdots}{(\varphi^{\top})\ m\ (\varphi^{\top} \lhd \chi)}}{(\varphi^{\perp} \vee \varphi^{\top})\ m\ ((\varphi^{\perp} \lhd \chi) \vee (\varphi^{\top} \lhd \chi))}(\text{Disjunction})$$

$$\dfrac{(\varphi^{\perp} \vee \varphi^{\top})\ m\ ((\varphi^{\perp} \lhd \chi) \vee (\varphi^{\top} \lhd \chi))}{(\varphi)\ m\ (\varphi \lhd \chi)}(\text{Consequence})$$

Here the second application of the consequence rule is justified by the following laws:

- φ iff $\varphi^{\perp} \vee \varphi^{\top}$, (lemma 4.22)
- $(\varphi_1 \lhd \chi) \vee (\varphi_2 \lhd \chi)$ iff $((\varphi_1 \vee \varphi_2) \lhd \chi)$ (lemma 4.34).

Convention. In the rest of the proof we show the derivability of $(\varphi)\ m\ (\varphi \lhd \chi)$ where φ can be assumed to be of the form $\top \wedge \varphi'[\top]$. Of course the induction hypothesis can be applied for ***arbitrary*** preconditions, not only for preconditions of the form above.

We now make our case distinction with respect to the last step of the deduction.

- Atomic processes

  Assume that the last step of the deduction is an instance of one of the SAT axioms for an atomic process. In chapter 6 we showed that these axioms are of the form:

  $\alpha\ sat\ A(\alpha)$ (1)

  In section 5.8.1 we showed that the Hoare system axioms for atomic processes are all of the form:

  $(A_{\beta}(\alpha) \rhd \psi)\ \alpha\ (\psi)$, (2)

  where $\beta = base(\alpha) \cup abase(\psi)$.

  Now let φ be given, and let $\beta = base(\alpha) \cup abase(\varphi)$. Then one can construct the following derivation:

$$\dfrac{(A_{\beta}(\alpha) \rhd (\varphi \lhd A_{\beta}(\alpha)))\ \alpha\ (\varphi \lhd A_{\beta}(\alpha)) \quad (\alpha\text{-axiom})}{(\varphi)\ \alpha\ (\varphi \lhd A(\alpha))}(\text{Consequence})$$

The application of the consequence rule is justified by lemma 4.34 and the fact that $A_\beta(\alpha) \rightarrow A(\alpha)$ is a valid assertion.

- Predicative processes

  Assume that the last step of the deduction is an instance of the SAT axioms for a predicative process. This axiom has the form:

  $(\chi|\beta)$ *sat* χ.

  The corresponding Hoare axioms are, for arbitrary precondition φ :

  $(\varphi)\ (\chi|\beta)\ (\varphi \lhd \chi)$.

  The provability of these last formulae was to be shown.

- Channel hiding

  Assume that the given deduction ends as:

  $$\frac{m_1 \textit{ sat } \chi}{m_1\backslash\tilde{\mathbf{c}} \textit{ sat } \chi} \text{ (Hiding)}$$

  The restriction associated with the rule enforces that $hchan(\chi) \cap \tilde{\mathbf{c}} = \emptyset$ here. If φ happens to be such that $hchan(\varphi) \cap \tilde{\mathbf{c}} = \emptyset$ too, then also $hchan(\varphi \lhd \chi) \cap \tilde{\mathbf{c}} = \emptyset$, and so the following derivation is possible:

  $$\frac{(\varphi)\ m_1\ (\varphi \lhd \chi)}{(\varphi)\ m_1\backslash\tilde{\mathbf{c}}\ (\varphi \lhd \chi)} \text{ (Hiding)}$$

  Since by induction $(\varphi)\ m_1\ (\varphi \lhd \chi)$ is derivable, we have shown that $(\varphi)\ m_1\backslash\tilde{\mathbf{c}}\ (\varphi \lhd \chi)$ is derivable for φ with $hchan(\varphi) \cap \tilde{\mathbf{c}} = \emptyset$. We use this fact to handle the general case, i.e. without any restriction on φ.

  Let $\mathbf{c} = hchan(\varphi, \chi)$ and let $\mathbf{c}' = \mathbf{c} - \tilde{\mathbf{c}}$. Define φ' as follows: Take some fresh logical trace variable t_0 and let:

  $$\varphi' \equiv \varphi[t_0/h] \wedge (t_0|\mathbf{c}' = h|\mathbf{c}').$$

  Note that $hchan(\varphi') \cap \tilde{\mathbf{c}} = \mathbf{c}' \cap \tilde{\mathbf{c}} = \emptyset$. Therefore, from above we already know that the following formula is derivable:

  $$(\varphi')\ m_1\backslash\tilde{\mathbf{c}}\ ((\varphi' \lhd \chi)). \qquad (1)$$

  Apart from this derivation we have the following instance of the prefix invariance axiom:

$$(t_0|\mathbf{c} \leq h|\mathbf{c})\ m_1\backslash\tilde{\mathbf{c}}\ (t_0|\mathbf{c} \leq h|\mathbf{c}).$$

An application of the conjunction rule to combine this with (1) yields:

$$(\varphi' \wedge (t_0|\mathbf{c} \leq h|\mathbf{c}))\ m_1\backslash\tilde{\mathbf{c}}\ ((\varphi' \lhd \chi) \wedge (t_0|\mathbf{c} \leq h|\mathbf{c})). \qquad (2)$$

The precondition of (2) is:

$$\varphi[t_0/h] \wedge (t_0|\mathbf{c}' = h|\mathbf{c}') \wedge (t_0|\mathbf{c} \leq h|\mathbf{c}).$$

Since $\mathbf{c}' \subseteq \mathbf{c}$ this is clearly implied by:

$$\varphi[t_0/h] \wedge (t_0|\mathbf{c} = h|\mathbf{c}).$$

Since $hchan(\varphi) \subseteq \mathbf{c}$, this last assertion is equivalent to:

$$\varphi \wedge (t_0|\mathbf{c} = h|\mathbf{c}).$$

Moreover, we prove below that:

$$\Big((\varphi' \lhd \chi) \wedge (t_0|\mathbf{c} \leq h|\mathbf{c})\Big) \rightarrow (\varphi \lhd \chi) \qquad (3)$$

is valid.

Therefore we can apply the consequence rule to (2) as is done in the following derivation:

$$\cfrac{\cfrac{\cfrac{(2)}{(\varphi \wedge (t_0|\mathbf{c} = h|\mathbf{c}))\ m_1\backslash\tilde{\mathbf{c}}\ (\varphi \lhd \chi)}\ \text{(Consequence)}}{(\exists t_0[\varphi \wedge (t_0|\mathbf{c} = h|\mathbf{c})])\ m_1\backslash\tilde{\mathbf{c}}\ (\varphi \lhd \chi)}\ (\exists\text{-pre})}{(\varphi)\ m_1\backslash\tilde{\mathbf{c}}\ (\varphi \lhd \chi)}\ \text{(Consequence)}$$

The conclusion of this derivation is the formula which was to be shown provable.

We are left with the proof of (3). Take some arbitrary $\gamma \in \Gamma, (h, s) \in \Sigma$ and assume:

- $(h, s) \in [\![(\varphi[t_0/h] \wedge (t_0|\mathbf{c}' = h|\mathbf{c}')) \lhd \chi]\!]\gamma$,
  that is:
  $\exists h_0' \exists h_1' \exists s_0' \Big(h = h_0' h_1' \wedge (\gamma(t_0), s_0') \in [\![\varphi]\!]\gamma \wedge$
  $(\gamma(t_0)|\mathbf{c}' = h_0'|\mathbf{c}') \wedge (s_0', h_1', s) \in [\![\chi]\!]\gamma\Big)$ (4)

- $(h, s) \in [\![t_0|\mathbf{c} \leq h|\mathbf{c}]\!]\gamma$, that is
  $\gamma(t_0)|\mathbf{c} \leq h|\mathbf{c}$ (5)
  From (4) and (5) we must show that :
- $(h, s) \in [\![(\varphi \lhd \chi)]\!]\gamma$, that is :
  $\exists h_0 \exists h_1 \exists s_0 \big(h = h_0 h_1 \wedge (h_0, s_0) \in [\![\varphi]\!]\gamma \wedge (s_0, h_1, s) \in [\![\chi]\!]\gamma \big)$ (6)

Now take some h_0', h_1', s_0' as indicated by (4). From (5) it follows that there exist traces h_0 such that $h_0 \leq h$ and $\gamma(t_0)|\mathbf{c} = h_0|\mathbf{c}$. Take one of these and take h_1 such that $h = h_0 h_1$. Choose $s_0 = s_0'$. We must prove for this choice of h_0, h_1 and s_0 that $(h_0, s_0) \in [\![\varphi]\!]\gamma$ and $(s_0, h_1, s) \in [\![\chi]\!]\gamma$. To this end we first show that actually $h_0|\mathbf{c}' = h_0'|\mathbf{c}'$ and $h_1|\mathbf{c}' = h_1'|\mathbf{c}'$.

First of all:

$$h_0|\mathbf{c}' = h_0|\mathbf{c}|\mathbf{c}' = \gamma(t_0)|\mathbf{c}|\mathbf{c}' = \gamma(t_0)|\mathbf{c}', \text{ and}$$

by (4) we know that $\gamma(t_0)|\mathbf{c}' = h_0'|\mathbf{c}'$. This shows that $h_0|\mathbf{c}' = h_0'|\mathbf{c}'$. Secondly, since $h_0 h_1 = h = h_0' h_1'$ we have that

$$(h_0|\mathbf{c}')\hat{}(h_1|\mathbf{c}') = (h_0'|\mathbf{c}')\hat{}(h_1'|\mathbf{c}')$$

So we conclude that also $h_1'|\mathbf{c}' = h_1'|\mathbf{c}'$.

As to the proof of $(h_0, s_0) \in [\![\varphi]\!]\gamma$ we remark that, since $\gamma(t_0)|\mathbf{c} = h_0|\mathbf{c}$ and $s_0 = s_0'$, the pairs (h_0, s_0) and $(\gamma(t_0), s_0')$ agree on $abase(\varphi)$. But we know already that $(\gamma(t_0), s_0') \in [\![\varphi]\!]\gamma$, and so we can infer that also $(h_0, s_0) \in [\![\varphi]\!]\gamma$.

The restriction on $hchan(\chi)$ implies that $hchan(\chi) \subseteq \mathbf{c}'$. This means that (s_0, h_1, s) and (s_0', h_1', s) agree on $abase(\chi)$. And since we know that $(s_0', h_1', s) \in [\![\chi]\!]\gamma$ we can conclude that also $(s_0', h_1', s) \in [\![\chi]\!]\gamma$. (As was to be shown).

□

- Variable hiding

Assume the given deduction ends as:

$$\frac{m_1 \; sat \; \chi}{m_1 \backslash \mathbf{x} \; sat \; \chi} \text{ (Hiding)}$$

The restriction for this rule then requires that $var(\chi) \cap \mathbf{x} = \emptyset$. (Note that nothing is required with respect to $var^\circ(\chi)$). Now $var(\varphi \lhd \chi) =$

$var(\chi)$, and so $var(\varphi \lhd \chi) \cap \mathbf{x} = \emptyset$ too, for arbitrary φ. So the following derivation is allowed:

$$\frac{(\varphi)\ m_1\ (\varphi \lhd \chi)}{(\varphi)\ m_1 \backslash \mathbf{x}\ (\varphi \lhd \chi)}\ \text{(Hiding)}$$

(The restriction for this rule requires that $var(\varphi \lhd \chi) \cap \mathbf{x} = \emptyset$, but nothing is required with respect to $var(\varphi)$.)

Since $(\varphi)\ m_1\ (\varphi \lhd \chi)$ can by induction assumed to be derivable, we see that $(\varphi)\ m_1 \backslash \mathbf{x}\ (\varphi \lhd \chi)$ is derivable, as was to be shown.

- Channel renaming

Assume that the given deduction ends as follows:

$$\frac{m_1\ sat\ \chi[h[d/c]/h]}{m_1\langle d/c\rangle\ sat\ \chi}\ \text{(Renaming)}$$

Let $\varphi' \equiv \varphi[h[c/c']/h] \wedge (c = \varepsilon) \wedge (c' = t')$, where c' is some fresh channel name and t' is some fresh logical trace variable. By induction the following is derivable:

$$(\varphi')\ m_1\ (\varphi' \lhd (\chi[h[d/c]/h])). \qquad (1)$$

The invariance rule can be used to derive:

$$(c' = t')\ m_1\ (c' = t'). \qquad (2)$$

Combining (1) and (2) using the conjunction rule yields:

$$(\varphi')\ m_1\ ((\varphi' \lhd (\chi[h[d/c]/h])) \wedge (c' = t')). \qquad (3)$$

Below we prove the validity of the following assertion:

$$\Big(\varphi' \lhd (\chi[h[d/c]/h]) \wedge (c' = t')\Big) \rightarrow \Big((\varphi \lhd \chi)[h[d/c][c/c']/h]\Big) \qquad (4)$$

We use this in the following deduction that shows the derivability of $(\varphi)\ m_1\langle d/c\rangle\ (\varphi \lhd \chi)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{(\varphi')\ m_1\ ((\varphi' \lhd (\chi[h[d/c]/h])) \wedge (c' = t'))}{(\varphi')\ m_1\ ((\varphi \lhd \chi)[h[d/c][c/c']/h])}\ (\text{Consequence})}{(\exists t'[\varphi'])\ m_1\ ((\varphi \lhd \chi)[h[d/c][c/c']/h])}\ (\exists\text{-pre})}{(\varphi[h[c/c']/h] \wedge (c = \varepsilon))\ m_1\ ((\varphi \lhd \chi)[h[d/c][c/c']/h])}\ (\text{Consequence})}{(\varphi)\ m_1\langle d/c\rangle\ (\varphi \lhd \chi)}\ (\text{Renaming})$$

Proof of (4).

Take some arbitrary $\gamma \in \Gamma, (h, s) \in \Sigma$ and assume:

- $(h, s) \in [\![(\varphi[h[c/c']/h] \wedge (c = \varepsilon) \wedge (c' = t')) \lhd (\chi[h[d/c]/h])]\!]\gamma$.

that is:

$$\exists h_0' \exists h_1' \exists {s_0}' \Big(h = h_0' h_1' \wedge (h_0'[c/c'], {s_0}') \in [\![\varphi]\!]\gamma \wedge$$

$$(h_0'|\{c\} = \varepsilon) \wedge (h_0'|\{c'\} = \gamma(t')) \wedge ({s_0}', h_1'[d/c], s) \in [\![\chi]\!]\gamma \Big).$$

- $(h, s) \in [\![c' = t']\!]\gamma$, that is: $h|\{c'\} = \gamma(t')$.

We must prove from these assumptions that

- $(h, s) \in [\![(\varphi \lhd \chi)[h[d/c][c/c']/h]]\!]\gamma$, that is:

$$\exists h_0 \exists h_1 \exists s_0 \Big(h[d/c][c/c'] = h_0 h_1 \wedge (h_0, s_0) \in [\![\varphi]\!]\gamma \wedge (s_0, h_1, s) \in [\![\chi]\!]\gamma \Big).$$

To see that this is the case, take $h_0', h_1', {s_0}'$ as indicted, and choose:

- $h_0 = h_0'[c/c']$,
- $h_1 = h_1'[d/c]$,
- $s_0 = {s_0}'$.

Clearly the requirements $(h_0, s_0) \in [\![\varphi]\!]\gamma$ and $(s_0, h_1, s) \in [\![\chi]\!]\gamma$ follow directly from the assumptions. There remains to prove:

$$h[d/c][c/c'] = h_0 h_1 = (h_0'[c/c'])\hat{}(h_1'[d/c]).$$

Since $h = h_0' h_1'$ it suffices to prove:

(i) $h_0'[d/c][c/c'] = h_0'[c/c']$, and
(ii) $h_1'[d/c][c/c'] = h_1'[d/c]$.

Now (i) follows immediately from the assumption $h'_0|\{c\} = \varepsilon$ implying $h'_0[d/c] = h'_0$. Equality (ii) will follow similarly if we can show that $(h'_1[d/c])|\{c'\} = \varepsilon$, or, equivalently, that $h'_1|\{c'\} = \varepsilon$. This last equality follows from:

$$h'_0|\{c'\} = \gamma(t') = h|\{c'\} = (h'_0 h'_1)|\{c'\} = (h'_0|\{c'\})\hat{}(h'_1|\{c'\}).$$

□

- Kernel

Assume the given deduction ends as:

$$\frac{m_1 \; sat \; \chi}{Kern(m_1) \; sat \; Kern(\chi)} \text{ (Kernel)}$$

Let some precondition φ be given. Let $(\mathbf{c}, \{\bar{x}\}) = abase(\varphi, \chi)$ and let $\bar{v}$ be a list of fresh logical variables of some length as $\bar{x}$. Take some fresh logical trace variable t_0. Define:

$$\varphi' \equiv \varphi \wedge (h|\mathbf{c} = t_0|\mathbf{c}) \wedge (\bar{x} = \bar{v})$$

We have the following derivation, where the premisse itself is derivable by the induction hypotheses:

$$\dfrac{\dfrac{\dfrac{(\varphi') \; m_1 \; (\varphi' \lhd \chi)}{(\varphi' \wedge h|\mathbf{c} = t_0|\mathbf{c}) \; Kern(m_1) \; (Kern(t_0, \varphi' \lhd \chi))} \text{ (Kernel)}}{(\varphi') \; Kern(m_1) \; (\varphi \lhd Kern(\chi))} \text{ (Consequence)}}{(\exists t_0 \exists \bar{v}[\varphi']) \; Kern(m_1) \; (\varphi \lhd Kern(\chi))} \exists\text{-pre}$$

$$\frac{(\exists t_0 \exists \bar{v}[\varphi']) \; Kern(m_1) \; (\varphi \lhd Kern(\chi))}{(\varphi) \; Kern(m_1) \; (\varphi \lhd Kern(\chi))} \text{ (Consequence)}$$

Here the application of the consequence rule relies on the fact, to be proven below, that

$$Kern(t_0, (\varphi' \lhd \chi)) \rightarrow \varphi \lhd Kern(\chi) \quad (*)$$

is valid.

Proof of $(*)$.

Let $(h, s) \in [\![Kern(t_0, \varphi' \lhd \chi)]\!]\gamma$.

By expanding the definition of $Kern(t_0, \varphi' \lhd \chi)$ we see that this means:

$(h, s) \in [\![\varphi' \lhd \chi]\!]\gamma$, and

$(h, s) \in [\![\forall t(t_0|\mathbf{c} \leq t|\mathbf{c} \leq h|\mathbf{c} \rightarrow (\varphi' \lhd \chi)[\bot][t/h])]\!]\gamma$.

We interpret these two formulae, and also use the fact that φ can be assumed to be of the form $\top \wedge - - -$. (So $(h_0, s_0) \in [\![\varphi]\!]\gamma$ implies $s_0 \neq \bot$.) This interpretation amounts to the following:

(a) $\exists h_0, h_1, s_0\Big(h = h_0 h_1 \wedge (h_0, s_0) \in [\![\varphi]\!]\gamma \wedge (h_0|\mathbf{c} = \gamma(t_0)|\mathbf{c})$

$\wedge s_0 \neq \bot \wedge (s_0(\bar{x}) = \gamma(\bar{v})) \wedge (s_0, h_1, s) \in [\![\chi]\!]\gamma\Big)$.

(b) If, for some h', $\gamma(t_0)|\mathbf{c} \leq h'|\mathbf{c} \leq h|\mathbf{c}$ then:

$\exists h_0', h_1', s_0'\Big(h' = h_0' h_1' \wedge (h_0', s_0') \in [\![\varphi]\!]\gamma \wedge (h_0'|\mathbf{c} = \gamma(t_0)|\mathbf{c})$

$\wedge\ s_0' \neq \bot \wedge (s_0'(\bar{x}) = \gamma(\bar{v})) \wedge (s_0', h_1', \bot) \in [\![\chi]\!]\gamma\Big)$.

From (a) and (b) we must prove:

$(h, s) \in [\![\varphi \lhd \mathit{Kern}(\chi)]\!]\gamma$, that is:

$\exists h_0, h_1, s_0\Big(h = h_0 h_1 \wedge (h_0, s_0) \in [\![\varphi]\!]\gamma \wedge (s_0, h_1, s) \in [\![\mathit{Kern}(\chi)]\!]\gamma\Big)$.

To prove this, take h_0, h_1, and s_0 as indicated by (a). The requirement $(s_0, h_1, s) \in [\![\mathit{Kern}(\chi)]\!]\gamma$ splits into:

(i) $(s_0, h_1, s) \in [\![\chi]\!]\gamma$ and

(ii) If $h_1'' \leq h_1$ then $(s_0, h_1'', \bot) \in [\![\chi]\!]\gamma$.

Clearly (i) follows immediately from (a). Now apply (b) and choose $h' = h_0 h_1''$. Since, by (a), $\gamma(t_0)|\mathbf{c} = h_0|\mathbf{c} \leq (h_0 h_1'')|\mathbf{c} \leq (h_0 h_1)|\mathbf{c} = h|\mathbf{c}$, we see that there exist h_0', h_1', s_0' with the properties as indicated under (b). From $h_0|\mathbf{c} = \gamma(t_0)|\mathbf{c} = h_0'|\mathbf{c}$ and $(h_0 h_1'')|\mathbf{c} = h'|\mathbf{c} = (h_0' h_1')|\mathbf{c}$ it follows that $h_1''|\mathbf{c} = h_1'|\mathbf{c}$. Similarly we see: $s_0(\bar{x}) = \gamma(\bar{v}) = s_0'(\bar{x})$. Therefore, $(s_0', h_1', \bot)$ and $(s_0, h_1'', \bot)$ agree on $\mathit{abase}(\chi)$ and since we know already that $(s_0', h_1', \bot) \in [\![\chi]\!]\gamma$ by (b), we see that also $(s_0, h_1'', \bot) \in [\![\chi]\!]\gamma$, as was to be shown.

- Sequential composition

Assume the given deduction ends with:

$$\frac{m_1\ \mathit{sat}\ \chi_1 \quad , \quad m_2\ \mathit{sat}\ \chi_2}{m_1; m_2\ \mathit{sat}\ \chi_1 \hat{\circ} \chi_2} \quad \text{(Sequential composition)}$$

By induction, $(\varphi)\ m_1\ (\varphi \lhd \chi_1)$ and $(\varphi \lhd \chi_1)\ m_2\ ((\varphi \lhd \chi_1) \lhd \chi_2)$ are derivable. Since by lemma 4.34 $(\varphi \lhd \chi_1) \lhd \chi_2$ implies $\varphi \lhd (\chi_1 \hat{\circ} \chi_2)$, we have the following derivation:

$$\dfrac{\dfrac{(\varphi)\ m_1\ (\varphi \lhd \chi_1) \quad , \quad (\varphi \lhd \chi_1)\ m_2\ ((\varphi \lhd \chi_1) \lhd \chi_2)}{(\varphi)\ m_1; m_2\ ((\varphi \lhd \chi_1) \lhd \chi_2)}\ \text{(Seq. comp.)}}{(\varphi)\ m_1; m_2\ (\varphi \lhd (\chi_1 \hat{\circ} \chi_2))}\ \text{(Consequence)}$$

- Choice

  Assume the given deduction ends with:

$$\dfrac{m_1\ sat\ \chi_1 \quad , \quad m_2\ sat\ \chi_2}{m_1\ \mathbf{or}\ m_2\ sat\ \chi_1 \vee \chi_2}$$

  By induction, $(\varphi)\ m_1\ (\varphi \lhd \chi_1)$ and $(\varphi)\ m_2\ (\varphi \lhd \chi_2)$ are derivable. By lemma 4.33, point (ii), $(\varphi \lhd \chi_i)$ implies $(\varphi \lhd (\chi_1 \vee \chi_2))$, for $i = 1, 2$. So we have:

$$\dfrac{\dfrac{(\varphi)\ m_1\ (\varphi \lhd \chi_1)}{(\varphi)\ m_1\ (\varphi \lhd \chi_1 \vee \chi_2)}\ \text{(Cons.)} \qquad (\varphi)\ m_2\ (\varphi \lhd \chi_1 \vee \chi_2)}{(\varphi)\ m_1\ \mathbf{or}\ m_2\ (\varphi \lhd (\chi_1 \vee \chi_2))}\ \text{(Choice)}$$

- Parallel composition

  Assume that the given deduction ends as follows:

$$\dfrac{m_1\ sat\ \chi_1 \quad , \quad m_2\ sat\ \chi_2}{m_1\ \tilde{\beta}_1 \| \tilde{\beta}_2\ m_2\ sat\ \chi_1 \wedge \chi_2}\ \text{(Parallel composition)}$$

  Let in the following "j" stand for $3 - i$. From the restrictions associated with the rule it the follows that $abase(\chi_i) \cap \tilde{\beta}_j \subseteq \tilde{\beta}_i$ for $i \in 1, 2$.

  We want to show the derivability of:

  $(\varphi)\ m_1\ \tilde{\beta}_1 \| \tilde{\beta}_2\ m_2\ (\varphi \lhd (\chi_1 \wedge \chi_2))$,

  where we may assume, by induction, the derivability of:

  $(\varphi_i) m_i (\varphi_i \lhd \chi_i)$,

for $i = 1, 2$, for arbitrary φ_i.

The first idea is to choose $\varphi_1 \equiv \varphi_2 \equiv \varphi$. Unfortunately this will not do, for the resulting specifications for the parts in general cannot be combined by the parallel composition rule from the Hoare system. The reason is that in general $(abase(\varphi) \cap \tilde{\beta}_j) \not\subseteq \tilde{\beta}_i$.

Therefore we choose φ_i as in the following definition. We rely on the fact that we can assume that φ is of the form $\top \wedge \varphi'$.

Define for $i \in 1, 2$:

- $\beta \stackrel{\text{def}}{=} \tilde{\beta}_1 \cup \tilde{\beta}_2 \cup abase(\varphi, \chi_1, \chi_2) =$ (say) $(\mathbf{c}, \{\bar{x}\})$,
- $\beta_i \stackrel{\text{def}}{=} \tilde{\beta}_i \cup (abase(\varphi, \chi_1, \chi_2) - \tilde{\beta}_j) =$ (say) $(\mathbf{c}_i, \{\bar{x}_i\})$,
- Let $\bar{v}$ be a list, of the same length as $\bar{x}$, of distinct fresh logical variables. Let $\bar{v}_i$ be the sublist of $\bar{v}$ corresponding to $\bar{x}_i$. Also, let t_0 be a fresh logical trace variable,
- $\varphi_i \stackrel{\text{def}}{=} \varphi[\bar{v}/\mathbf{x}][t_0/h] \wedge (\bar{v}_i = \bar{x}_i) \wedge (t_0|\mathbf{c}_i = h|\mathbf{c}_i)$.

Informally speaking, we have divided up β into β_1 and β_2, that is $\beta_1 = \beta_1 \cup \beta_2$, in such a way that $\beta_i \cap \tilde{\beta}_j \subseteq \tilde{\beta}_i$. The assertion $\exists \bar{v} \exists t_0(\varphi_i)$ is the strongest one possible on the channels and variables in β_i given that φ holds.

In fact we have that $abase(\varphi_i) \cap \tilde{\beta}_j = \beta_i \cap \tilde{\beta}_j \subseteq \tilde{\beta}_i$. Moreover, from the restriction on $abase(\chi_i)$ stated above, it follows that :

$$abase(\varphi_i \lhd \chi_i) \cap \tilde{\beta}_j = (abase(\varphi_i) \cup abase(\chi_i)) \cap \tilde{\beta}_j \subseteq \tilde{\beta}_i$$

From these calculations we conclude that the restrictions associated with the following application of the parallel composition rule are met:

$$\frac{(\varphi_1)\ m_1\ (\varphi_1 \lhd \chi_1) \quad , \quad (\varphi_2)\ m_2\ (\varphi_2 \lhd \chi_2)}{(\varphi_1 \wedge \varphi_2)\ m_1\ \tilde{\beta}_1 \| \tilde{\beta}_2\ m_2\ ((\varphi_1 \lhd \chi_1) \wedge (\varphi_2 \lhd \chi_2))}\ \text{(parallel comp.)}$$

Let φ_p denote $(t_0|\mathbf{c} \leq h|\mathbf{c})$ and let $m \equiv m_1\ \tilde{\beta}_1 \parallel \tilde{\beta}_2\ m_2$. Clearly, $(\varphi_p)\ m\ (\varphi_p)$ is an instance of the prefix invariance axiom. Using the conjunction rule to combine this axiom with the conclusion of the rule above, we obtain:

$$(\varphi_1 \wedge \varphi_2 \wedge \varphi_p)\ m\ ((\varphi_1 \lhd \chi_1) \wedge (\varphi_2 \lhd \chi_2) \wedge \varphi_p). \quad (1)$$

Below we give the proof of the following:

Lemma 7.3

The following assertions are valid:

(i) $(\varphi_1 \wedge \varphi_2 \wedge \varphi_p) \leftrightarrow (\varphi \wedge (\bar{v} = \bar{x}) \wedge (t_0|\mathbf{c} = h|\mathbf{c}))$.

(ii) $((\varphi_1 \lhd \chi_1) \wedge (\varphi_2 \lhd \chi_2) \wedge \varphi_p) \rightarrow (\varphi \lhd (\chi_1 \wedge \chi_2))$.

□

Using the lemma we obtain the following derivation:

Let ψ denote $\varphi \wedge (\bar{v} = \bar{x}) \wedge (t_0|\mathbf{c} = h|\mathbf{c})$.

$$\dfrac{\dfrac{\dfrac{(\varphi_1 \wedge \varphi_2 \wedge \varphi_p)\ m\ ((\varphi_1 \lhd \chi_1) \wedge (\varphi_2 \lhd \chi_2) \wedge \varphi_p)}{(\psi)\ m\ (\varphi \lhd (\chi_1 \wedge \chi_2))}\ \text{(Consequence)}}{(\exists \bar{v} \exists t_0[\psi])\ m\ (\varphi \lhd (\chi_1 \wedge \chi_2))}\ (\exists\text{-pre})}{(\varphi)\ m\ (\varphi \lhd (\chi_1 \wedge \chi_2))}\ \text{(Consequence)}$$

The last formula was the one to be shown to be provable.

Proof of the lemma.

The important observation to be made here is that although

$$(t_0|\mathbf{c} = h|\mathbf{c}) \not\leftrightarrow ((t_0|\mathbf{c}_1 = h|\mathbf{c}_1) \wedge (t_0|\mathbf{c}_2 = h|\mathbf{c}_2)),$$

it *is* true that

$$(t_0|\mathbf{c} = h|\mathbf{c}) \leftrightarrow ((t_0|\mathbf{c}_1 = h|\mathbf{c}_1) \wedge (t_0|\mathbf{c}_2 = h|\mathbf{c}_2) \wedge (t_0|\mathbf{c} \leq h|\mathbf{c})).$$

The implication from left to right is easy: if $t_0|\mathbf{c} = h|\mathbf{c}$ then also $t_0|\mathbf{c}_i = t_0|\mathbf{c}|\mathbf{c}_i = h|\mathbf{c}|\mathbf{c}_i = h|\mathbf{c}_i$. Reverse, assume $t_0|\mathbf{c}_i = h|\mathbf{c}_i$ for $i = 1, 2$ and $t_0|\mathbf{c} \leq h|\mathbf{c}$. Let us assume to the contrary that $t_0|\mathbf{c} \neq h|\mathbf{c}$. Since $t_0|\mathbf{c} \leq h|\mathbf{c}$ it follows that $|t_0|\mathbf{c}| \neq |h|\mathbf{c}|$. This excludes the case that $t_0|\mathbf{c}$ and $h|\mathbf{c}$ differ only in that the *order* of the communications is different. Rather there must be some channel $d \in \mathbf{c}$ such that $|t_0|\{d\}| \neq |h|\{d\}|$. However, $d \in \mathbf{c}_1$ or $d \in \mathbf{c}_2$ and so $t_0|\mathbf{c}_1 = h|\mathbf{c}_1$ is violated or $t_0|\mathbf{c}_2 = h|\mathbf{c}_2$ is violated. By reduction ad absurdum, $t_0|\mathbf{c} = h|\mathbf{c}$.

Using the equivalence above we now prove part(i) of the lemma:

$\varphi_1 \wedge \varphi_2 \wedge \varphi_p$ iff

$\varphi[\bar{v}/\bar{x}][t_0/h] \wedge (\bar{v}_1 = \bar{x}_1) \wedge (\bar{v}_2 = \bar{x}_2) \wedge (t_0|\mathbf{c}_1 = h|\mathbf{c}_1) \wedge$
$(t_0|\mathbf{c}_2 = h|\mathbf{c}_2) \wedge (t_0|\mathbf{c} \leq h|\mathbf{c})$ iff
$\varphi[\bar{v}/\bar{x}][t_0/h] \wedge (\bar{v} = \bar{x}) \wedge (t_0|\mathbf{c} = h|\mathbf{c})$ iff
$\varphi \wedge (\bar{v} = \bar{x}) \wedge (t_0|\mathbf{c} = h|\mathbf{c})$ (since $abase(\varphi) \subseteq (\mathbf{c}, \mathbf{x})$).

□

Next we prove part (ii) of the lemma. We give a semantic argument. That is, we prove that for arbitrary $\gamma \in \Gamma$, $(h, s) \in \Sigma$,

if $(h, s) \in [\![(\varphi_1 \lhd \chi_1) \wedge (\varphi_2 \lhd \chi_2) \wedge \varphi_p]\!]\gamma$
then $(h, s) \in [\![\varphi \lhd (\chi_1 \wedge \chi_2)]\!]\gamma$.

Let us first rewrite this, by expanding the (semantic) $\lhd$ operators.

We may assume:

- $(h, s) \in [\![\varphi_i \lhd \chi_i]\!]\gamma$, that is, for $i \in 1, 2$:
  $\exists h_0^i \exists h_1^i \exists s_0^i \Big(h = h_0^i h_1^i \wedge (h_0^i, s_0^i) \in [\![\varphi_i]\!]\gamma \wedge (s_0^i, h_1^i, s) \in [\![\chi_i]\!]\gamma\Big)$. (1)
- $(h, s) \in [\![t_0|\mathbf{c} \leq h|\mathbf{c}]\!]\gamma$, that is: $\gamma(t_0)|\mathbf{c} \leq h|\mathbf{c}$. (2)

  From this it must be shown that:
- $(h, s) \in [\![\varphi \lhd (\chi_1 \wedge \chi_2)]\!]\gamma$, that is:
  $\exists h_0 \exists h_1 \exists s_0 \Big(h = h_0 h_1 \wedge (h_0, s_0) \in [\![\varphi]\!]\gamma \wedge$
  $(s_0, h_1, s) \in [\![\chi_1]\!]\gamma \wedge (s_0, h_1, s) \in [\![\chi_2]\!]\gamma\Big)$. (3)

So assume (1) and (2). Take some h_0^i, h_1^i and s_0^i as indicated by (1). From (2) it follows that there exists traces h_0' such that $h_0' \leq h$ and $\gamma(t_0)|\mathbf{c} = h_0'|\mathbf{c}$. Choose h_0 as the longest of these h_0' traces. Take h_1 such that $h_0 h_1 = h$. (We postpone the choice of some s_0.)

First we show that actually $h_0|\mathbf{c}_i = h_0^i|\mathbf{c}_i$ and $h_1|\mathbf{c}_i = h_1^i|\mathbf{c}_i$. For we have that:

$h_0|\mathbf{c}_i = h_0|\mathbf{c}|\mathbf{c}_i = \gamma(t_0)|\mathbf{c}|\mathbf{c}_i = \gamma(t_0)|\mathbf{c}_i$.

And from $(h_0^i, s_0^i) \in [\![\varphi_i]\!]\gamma$ follows that:

$\gamma(t_0)|\mathbf{c}_i = h_0^i|\mathbf{c}_i$.

Therefore $h_0|\mathbf{c}_i = h_0^i|\mathbf{c}_i$. Moreover,

$(h_0|\mathbf{c}_i)\hat{}(h_1|\mathbf{c}_i) = (h_0 h_1)|\mathbf{c}_i = h|\mathbf{c}_i = (h_0^i h_1^i)|\mathbf{c}_i = (h_0^i|\mathbf{c}_i)\hat{}(h_1^i|\mathbf{c}_i)$.

We conclude that also $h_1|\mathbf{c}_i = h_1^i|\mathbf{c}_i$.

Next we must choose some s_0 and then prove that $(h_0, s_0) \in [\![\varphi]\!]\gamma$ and $(s_0, h_1, s) \in [\![\chi_i]\!]\gamma$. Because of the assumption that φ is of the form $\top \wedge \cdots$, the same holds for the φ_i assertions. So from the assumption that $(h_0^i, s_0^i) \in [\![\varphi_i]\!]\gamma$ it follows that $s_0^i \neq \bot$, for both $i = 1$ and $i = 2$. Therefore we can, for instance, choose $s_0 = s_0^1|\bar{x} : \gamma(\bar{v})$. From the lemma, part(i), follows that instead of proving $(h_0, s_0) \in [\![\varphi]\!]\gamma$ it *suffices* to show that $(h_0, s_0) \in [\![\varphi_i]\!]\gamma$ and $(h_0, s_0) \in [\![\varphi_p]\!]\gamma$. This is done as follows:

From $(h_0^i, s_0^i) \in [\![\varphi_i]\!]\gamma$ follows:

(a) $h_0^i|\mathbf{c}_i = h_0|\mathbf{c}_i$ (This was shown above.)

(b) $s_0^i(\bar{x}_i) = \gamma(\bar{v}_i) = (s|\bar{x} : \gamma(\bar{v}))(\bar{x}_i) = s_0(\bar{x}_i)$

So (h_0, s_0) and (h_0^i, s_0^i) agree on $abase(\varphi_i)$, and therefore $(h_0^i, s_0^i) \in [\![\varphi_i]\!]\gamma$ implies that $(h_0, s_0) \in [\![\varphi_i]\!]\gamma$.

The proof that $(h_0, s_0) \in [\![\varphi_p]\!]\gamma$ is straightforward: $\gamma(t_0)|\mathbf{c} = h_0|\mathbf{c}$ by the choice of h_0.

We are left with the proof of : $(s_0, h_1, s) \in [\![\chi_i]\!]\gamma$. Now this is clear, as (s_0, h_1, s) and (s_0^i, h_1^i, s) agree on $abase(\chi_i)$ and $(s_0^i, h_1^i, s) \in [\![\chi_i]\!]\gamma$.

This ends the proof of the lemma, and so of the parallel composition case.

□

- Invariance

Assume we have given an instance of the invariance axiom:

$$m \; sat \; (h|\mathbf{d} = \varepsilon) \wedge (\top \rightarrow \bar{y} = \bar{y}^\circ).$$

From this axiom it then follows that $(\mathbf{d}, \{\bar{y}\}) \cap abase(m) = (\emptyset, \emptyset)$.

Let some φ be given. We want to prove:

$$(\varphi) \; m \; (\varphi \lhd ((h|\mathbf{d} = \varepsilon) \wedge (\top \rightarrow \bar{y} = \bar{y}^\circ))).$$

- Let $abase(\varphi) \cup (\mathbf{d}, \{\bar{y}\}) = (\mathbf{c}, \{\bar{x}\})$.
- Let $\bar{v}$ be a list, of the same length as $\bar{x}$ of fresh logical variables. Let $\bar{w}$ be the sublist corresponding to $\bar{y}$. Let t_0 be some fresh logical trace variable.
- Let $\varphi' \equiv (\varphi[\top][\bar{v}/\bar{x}][t_0/h])$.

- Let $\varphi_f \equiv (t_0|\mathbf{d} = h|\mathbf{d}) \wedge (\top \rightarrow \bar{w} = \bar{y})$.
- Let $\varphi_p \equiv (t_0|\mathbf{c} \leq h|\mathbf{c})$.

Note that we have the following instances of the invariance axiom:

$$((\varphi' \wedge \varphi_f) \wedge (\varphi' \wedge \varphi_f)[\bot])\ m\ (\varphi' \wedge \varphi_f).$$

Since the precondition of this formula is equivalent to $\varphi' \wedge \varphi_f$, we can use the consequence rule to derive:

$$(\varphi' \wedge \varphi_f)\ m\ (\varphi' \wedge \varphi_f).$$

We use the conjunction rule to combine this with the following instance of the prefix invariance axiom of the Hoare system:

$$(\varphi_p)\ m\ (\varphi_p).$$

This yields the formula:

$$(\varphi' \wedge \varphi_f \wedge \varphi_p)\ m\ (\varphi' \wedge \varphi_f \wedge \varphi_p).$$

Below we prove the validity of:

$$(\varphi' \wedge \varphi_f \wedge \varphi_p) \rightarrow (\varphi \lhd ((h|\mathbf{d} = \varepsilon) \wedge (\top \rightarrow \bar{y} = \bar{y}^\circ))). \quad (*)$$

Therefore the following derivation is possible:

$$\dfrac{\dfrac{\dfrac{(\varphi' \wedge \varphi_f \wedge \varphi_p)\ m\ (\varphi' \wedge \varphi_f \wedge \varphi_p)}{(\varphi' \wedge \varphi_f \wedge \varphi_p)\ m\ (\varphi \lhd ((h|\mathbf{d} = \varepsilon) \wedge (\top \rightarrow \bar{y} = \bar{y}^\circ)))}\ (\text{Consequence})}{(\exists t_0 \exists \bar{v}[\varphi' \wedge \varphi_f \wedge \varphi_p])\ m\ (\varphi \lhd ((h|\mathbf{d} = \varepsilon) \wedge (\top \rightarrow \bar{y} = \bar{y}^\circ)))}\ (\exists\text{-pre})}{(\varphi)\ m\ (\varphi \lhd ((h|\mathbf{d} = \varepsilon) \wedge (\top \rightarrow \bar{y} = \bar{y}^\circ)))}\ (\text{Consequence})$$

(As was to be shown.)

Proof of $(*)$:

Assume $(h, s) \in [\![\varphi' \wedge \varphi_f \wedge \varphi_p]\!]\gamma$.
We must show: $(h, s) \in [\![\varphi \lhd (h|\mathbf{d} = \varepsilon \wedge (\top \rightarrow \bar{y} = \bar{y}^\circ))]\!]\gamma$.

Take some arbitrary state $s' \neq \bot$. We have have the following situation:

(a) $s' \neq \bot$,
(b) $(h, s) \in [\![\varphi[\top][\bar{v}/\bar{x}][t_0/h]]\!]\gamma$, that is: $(\gamma(t_0), s'|\bar{x} : \gamma(\bar{v})) \in [\![\varphi]\!]\gamma$,

(c) $\gamma(t_0)|\mathbf{d} = h|\mathbf{d}$,

(d) $\gamma(t_0)|\mathbf{c} \leq h|\mathbf{c}$,

(e) if $s \neq \bot$ then $\gamma(\bar{w}) = s(\bar{y})$.

From (a) - (e) must be shown:

$$\exists h_0, h_1 \exists s_0 \Big((s_0, h_1, s) \in \Delta \wedge h = h_0 h_1 \wedge (h_0, s_0) \in [\![\varphi]\!]\gamma \wedge$$

$$(h_1|\mathbf{d} = \varepsilon) \wedge (s \neq \bot \rightarrow s(\bar{y}) = s_0(\bar{y})) \Big).$$

[**Remark** The clause "$(s_0, h_1, s) \in \Delta$" might appear superfluous, for by our naming conventions we already know that $s_0 \in \mathit{State}_\bot, h_1 \in \mathit{Trace}$ and $s \in \mathit{State}_\bot$. However, this is not sufficient since we have no guarantee that if $s_0 = \bot$ then $h_1 = \varepsilon$ and $s = \bot$, as is the case for Δ elements. In all cases before, there was some clause of the form "$(s_0, h_1, s) \in [\![\chi]\!]\gamma$" present, and since $[\![\chi]\!]\gamma \subseteq \Delta$, the condition above was enforced automatically. Not so in this case, explaining the conjunct $(s_0, h_1, s) \in \Delta$.]

Now choose h_0 such that $h_0 \leq h$ and $h_0|\mathbf{c} = \gamma(t_0)$. (There exists such a h_0 by (d)). Take h_1 such that $h_0 h_1 = h$. Choose $s_0 = s'|\bar{x} : \gamma(\bar{v})$.

Because $s' \neq \bot$, we have that $s_0 \neq \bot$ and this guarantees that $(s_0, h_1, s) \in \Delta$. Note that (h_0, s_0) and $(\gamma(t_0), s'|\bar{x} : \gamma(\bar{v}))$ agree on $(\mathbf{c}, \{\bar{x}\})$, and so agree on $\mathit{abase}(\varphi)$. From this and (b) it follows that $(h_0, s_0) \in [\![\varphi]\!]\gamma$.

We chose h_0 such that $h_0|\mathbf{c} = \gamma(t_0)|\mathbf{c}$, implying that $h_0|\mathbf{d} = \gamma(t_0)|\mathbf{d}$, for we have $\mathbf{d} \subseteq \mathbf{c}$. Then, using equality (c), we see that

$$(h_0 h_1)|\mathbf{d} = h|\mathbf{d} = \gamma(t_0)|\mathbf{d} = h_0|\mathbf{d}.$$

Conclusion: $h_1|\mathbf{d} = \varepsilon$.

Similarly, because $\bar{w}$ is a sublist of $\bar{v}$, we have that if $s \neq \bot$:

$$s(\bar{y}) = \gamma(\bar{w}) = s|\bar{x} : \gamma(\bar{v})(\bar{y}) = s_0(\bar{y}).$$

This ends the proof of (*), and so of the "invariance" case.

- Conjunction

Let the given deduction end with

$$\frac{m \mathit{\ sat\ } \chi_1 \quad , \quad m \mathit{\ sat\ } \chi_2}{m \mathit{\ sat\ } (\chi_1 \wedge \chi_2)} \text{ (Conjunction)}$$

We want to deduce $(\varphi)\ m\ (\varphi \lhd (\chi_1 \wedge \chi_2))$. Surprisingly the last formula cannot be proven from $(\varphi)\ m\ (\varphi \lhd \chi_1)$ and $(\varphi)\ m\ (\varphi \lhd \chi_2)$ by applying the conjunction- and consequence rule. The reason for this is that:

$$((\varphi \lhd \chi_1) \wedge (\varphi \lhd \chi_2)) \rightarrow (\varphi \lhd (\chi_1 \wedge \chi_2))$$

is *not* valid in general. Rather it follows from lemma 4.33, point (viii), that the *reverse* implication holds. However define:

$$\varphi' \equiv \varphi \wedge (h|\mathbf{c} = t_0|\mathbf{c}) \wedge (\bar{x} = \bar{v}),$$

where $(\mathbf{c}, \{\bar{x}\}) = abase(\varphi, \chi_1, \chi_2)$ and where t_0 and $\bar{v}$ are fresh logical variables. Below we prove that:

$$((\varphi' \lhd \chi_1) \wedge (\varphi' \lhd \chi_2)) \rightarrow (\varphi \lhd \chi_1 \wedge \chi_2). \qquad (*)$$

is a valid assertion. Moreover, by induction we may assume provability of $(\varphi')\ m\ (\varphi' \lhd \chi_i)$ for $i = 1, 2$. This leads to the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{(\varphi')\ m\ (\varphi' \lhd \chi_1) \quad , \quad (\varphi')\ m\ (\varphi' \lhd \chi_2)}{(\varphi' \wedge \varphi')\ m\ ((\varphi' \lhd \chi_1) \wedge (\varphi' \lhd \chi_2))}\ \text{(Conjunction)}}{(\varphi')\ m\ (\varphi \lhd (\chi_1 \wedge \chi_2))}\ \text{(Consequence)}}{(\exists t_0 \exists \bar{v}[\varphi'])\ m\ (\varphi \lhd (\chi_1 \wedge \chi_2))}\ (\exists\text{-pre})}{(\varphi)\ m\ (\varphi \lhd (\chi_1 \wedge \chi_2))}\ \text{(Consequence)}$$

There remains the proof of $(*)$

We may assume that φ is of the form $\top \wedge \tilde{\varphi}$. Therefore $(h, s) \in \varphi'$ implies that $s \neq \bot$.

Now assume $(h, s) \in [\![(\varphi' \lhd \chi_1) \wedge (\varphi' \lhd \chi_2)]\!]\gamma$, that is:

For $i = 1$ and $i = 2$:

$$\exists h_0^i, \exists h_1^i, \exists {s_0}^i \Big(h = h_0^i h_1^i \wedge {s_0}^i \neq \bot \wedge (h_0^i, {s_0}^i) \in [\![\varphi]\!]\gamma \wedge$$

$$(h_0^i|\mathbf{c} = \gamma(t_0)|\mathbf{c}) \wedge ({s_0}^i(\bar{x}) = \gamma(\bar{v})) \wedge ({s_0}^i, h_1^i, s) \in [\![\chi_i]\!]\gamma \Big).$$

To prove: $(h, s) \in [\![\varphi \lhd (\chi_1 \wedge \chi_2)]\!]\gamma$, that is:

$$\exists h_0 \exists h_1 \exists s_0 \Big(h = h_0 h_1 \wedge (h_0, s_0) \in [\![\varphi]\!]\gamma \wedge$$

$$(s_0, h_1, s) \in [\![\chi_1]\!]\gamma \wedge (s_0, h_1, s) \in [\![\chi_2]\!]\gamma\Big).$$

Now take $h_0^i, h_1^i, {s_0}^i$ as indicated. We then have that:

(a) $h_0^1|\mathbf{c} = \gamma(t_0)|\mathbf{c} = h_0^2|\mathbf{c}$,

(b) Since $h_0^1 h_1^1 = h = h_0^2 h_1^2$, from (a) it follows that also $h_1^1|\mathbf{c} = h_1^2|\mathbf{c}$.

(c) ${s_0}^1(\bar{x}) = \gamma(\bar{v}) = {s_0}^2(\bar{x})$.

We see that $({s_0}^1, h_1^1, s)$ and $({s_0}^2, h_1^2, s)$ agree on $abase(\chi_1, \chi_2)$. Therefore we can choose h_0, h_1 and s_0 as follows:

$$h_0 = h_0^1, \quad h_1 = h_1^1, \quad s_0 = {s_0}^1$$

It is immediately clear that $h = h_0 h_1$, that $(h_0, s_0) \in [\![\varphi]\!]\gamma$ and that $(s_0, h_1, s) \in [\![\chi_1]\!]\gamma$. Since, by the above, (s_0, h_1, s) and $({s_0}^2, h_1^2, s)$ agree on $abase(\chi_2)$, we see that also $(s_0, h_1, s) \in [\![\chi_2]\!]\gamma$. As was to be shown.

- Consequence

The given deduction,

$$\frac{m \; sat \; \chi, \forall_\perp(\chi \rightarrow \chi')}{m \; sat \; \chi'} \quad \text{(Consequence)}$$

is easily transformed into:

$$\frac{(\varphi)\; m\; (\varphi \lhd \chi) \quad , \quad \forall_\perp((\varphi \lhd \chi) \rightarrow (\varphi \lhd \chi'))}{(\varphi)\; m\; (\varphi \lhd \chi')} \quad \text{(Consequence)}$$

- Process naming

Assume the given deduction ends with an application of the process naming rule:

$$\frac{H \vdash m_1 \; sat \; \chi_1 \quad , \quad H \cup \{\xi_1 \; sat \; \chi_1\} \vdash m_2 \; sat \; \chi_2}{H \vdash \xi_1 = m_1 \mathbf{in}\; m_2 \; sat \; \chi_2}$$

From the restriction for this rule it follows that $\xi \notin H$. We recall that $\tilde{H}$ is some given set of adaptable Hoare formulae that are equivalent to the formulae in H. It is clear that $\xi \notin \tilde{H}$ too.

By lemma 7.2 we can take some adaptable specification of the form:

$$\forall \bar{g}\Big[(\varphi_F)\xi_1(\varphi_F \lhd \chi_1)\Big]., \qquad (1)$$

such that (1) is equivalent to ξ_1 *sat* χ_1. We may assume that the logical variables $\bar{g}$ have been chosen fresh, so they do not occur free in the set $\tilde{H}$.

By induction we may assume, for arbitrary φ, the deducibility of:

$\tilde{H} \vdash (\varphi_F)\, m_1\, (\varphi_F \lhd \chi_1)$, and

$\tilde{H} \cup \{(1)\} \vdash (\varphi)\, m_2\, (\varphi \lhd \chi_2)$.

Using the $\forall-$ introduction rule, one derives from the first of these two formulae:

$$\tilde{H} \vdash \forall \bar{g}\Big[(\varphi_F)\, m_1\, (\varphi_F \lhd \chi_1)\Big].$$

From this, the desired formula is deduced in one step, using the process naming rule of the Hoare system:

$$\frac{\tilde{H} \vdash (1) \quad , \quad \tilde{H} \cup \{(1)\} \vdash (\varphi)\, m_2\, (\varphi \lhd \chi_2)}{\tilde{H} \vdash (\varphi)\ \xi_1 = m_1 \text{ in } m_2\ (\varphi \lhd \chi_2)} \text{ (Process naming)}$$

- (μ_z) Recursion

Assume that the given deduction ends with an application of the recursion rule as follows:

$$\frac{H \vdash \mathbf{Z}\ \textit{sat}\ \chi\ ,\ H \cup \{\xi\ \textit{sat}\ \chi \vdash m_1\}\ \textit{sat}\ \chi}{H \vdash \mu_Z \xi.m_1\ \textit{sat}\ \chi} \ (\mu_Z \text{ recursion})$$

From the restriction for this rule it follows that $\xi \notin H$, and so also $\xi \notin \tilde{H}$, where $\tilde{H}$ is the corresponding set of Hoare formulae. We must show the deducibility of $\tilde{H} \vdash (\varphi)\mu_Z \xi.m_1(\varphi \lhd \chi)$ for arbitrary φ.

By lemma 7.2 we can take some Hoare specification of the form:

$$\forall \bar{g}\Big[(\varphi_F)\xi(\varphi_F \lhd \chi)\Big], \qquad (1)$$

such that it is equivalent to ξ *sat* χ and is also adaptable.

By induction we may assume the deducibility of:

$\tilde{H} \vdash (\varphi_F)\mathbf{Z}(\varphi_F \lhd \chi)$ and

$\tilde{H} \cup \{\forall \bar{g}\Big[(\varphi_F)\xi(\varphi_F \lhd \chi)\Big]\} \vdash (\varphi_F)\, m_1\, (\varphi_F \lhd \chi)$.

Since we may assume that $\bar{g}$ has been chosen such that $\bar{g} \notin lvar(\tilde{H})$, the $\forall-$ introduction rule can be used to infer:

$$\tilde{H} \vdash \forall \bar{g}\big[(\varphi_F)\mathbf{Z}(\varphi_F \lhd \chi)\big] \text{ and}$$

$$\tilde{H}, \forall \bar{g}\big[(\varphi_F)\xi(\varphi_F \lhd \chi)\big] \vdash \forall \bar{g}\big[(\varphi_F)\, m_1\, (\varphi_F \lhd \chi)\big].$$

Clearly we can apply the recursion rule from the Hoare system, and obtain thus:

$$\tilde{H} \vdash \forall \bar{g}\big[(\varphi_F)\mu_Z \xi.m_1(\varphi_F \lhd \chi)\big].$$

From the equivalence between (1) and ξ *sat* χ and the validity of the implication

$$(\xi \textit{ sat } \chi) \rightarrow (\varphi)\ \xi\ (\varphi \lhd \chi),$$

it follows that the implication

$$\forall \bar{g}\big[(\varphi_F)\ \xi\ (\varphi_F \lhd \chi)\big] \rightarrow (\varphi)\ \xi\ (\varphi \lhd \chi)$$

is valid too. Because of the adaptability of (1) this implication is ***provable*** or, equivalently:

$$\tilde{H} \vdash (\varphi)\ \xi\ (\varphi \lhd \chi)$$

can be deduced from:

$$\tilde{H} \vdash \forall \bar{g}\big[(\varphi_F)\ \xi\ (\varphi_F \lhd \chi)\big].$$

By replacing ξ by $\mu_Z \xi.m_1$ in this proof we see that:

$$\tilde{H} \vdash (\varphi)\mu_Z \xi.m_1(\varphi \lhd \chi)$$

is deducible from:

$$\tilde{H} \vdash \forall \bar{g}[(\varphi_F)\mu_Z \xi.m_1(\varphi_F \lhd \chi)].$$

This shows that the latter formula is deducible, since we already proved above that the former formula is deducible.

Finally we treat (non μ_Z) recursion. This case is identical to the case of μ_Z recursion except that the corresponding proof rules of the SAT system and the Hoare system do not have the premisse of the form $\mathbf{Z}$ *sat* χ.

This ends the proof of theorem 7.1.

□

Theorem 7.4 Compositional Completeness for the Hoare system

The Hoare system is compositionally complete for the class of Hoare formulae $\tilde{H} \vdash \forall \bar{g}[(\varphi)\ m\ (\psi)]$ where $\tilde{H}$ consists of adaptable Hoare formulae, and where $\forall \bar{g}[(\varphi)\ \xi\ (\psi)]$ is a *satisfiable* specification.

□

Proof

Assume that $\tilde{H}$ is the following given set of adaptable Hoare specifications:

$$\tilde{H} \stackrel{\text{def}}{=} \{\forall \bar{g}_1[(\tilde{\varphi_1})\ \varsigma_1\ (\tilde{\psi_1})], \ldots, \forall \bar{g}_n[(\tilde{\varphi_n})\ \varsigma_n\ (\tilde{\psi_n})]\}.$$

Assume that $\forall \bar{g}[(\varphi)\ \xi\ (\psi)]$ is satisfiable, and that:

$$\tilde{H} \models \forall \bar{g}[(\varphi)\ m(m_1, \ldots, m_k)\ (\psi)]. \qquad (1)$$

To prove the theorem we must show the existence of sets of adaptable Hoare specifications $\tilde{H}_i$, and assertions φ_i, ψ_i such that:

(a) $\tilde{H}_i \models (\varphi_i)\ m_i\ (\psi_i)$, for $i = 1..k$, and

(b) $\tilde{H} \vdash \forall \bar{g}[(\varphi)\ m(m_1, \ldots, m_k)\ (\psi)]$ is deducible from:

$\tilde{H}_1 \vdash (\varphi_1)\ m_1\ (\psi_1), \ldots, \tilde{H}_k \vdash (\varphi_k)\ m_k\ (\psi_k)$.

Let $H \stackrel{\text{def}}{=} \{\varsigma_1\ sat\ \forall \bar{g}_1(\tilde{\varphi_1} \rightsquigarrow \tilde{\psi_1}), \ldots, \varsigma_n\ sat\ \forall \bar{g}_n(\tilde{\varphi_n} \rightsquigarrow \tilde{\psi_n})\}$.

From (1) it follows that:

$$\tilde{H} \models (\varphi)\ m(m_1, \ldots, m_k)\ (\psi),$$

and so:

$$H \models m(m_1, \ldots, m_k)\ sat\ (\varphi \rightsquigarrow \psi).$$

By the compositional completeness of the SAT system this implies the existence of sets of SAT specifications H_i and assertions χ_i such that:

(a') $H_i \models m_i\ sat\ \chi_i$, for $i = 1..k$, and

(b') $H \vdash m(m_1, \ldots, m_k)\ sat\ (\varphi \rightsquigarrow \psi)$ is deducible from:

$H_1 \vdash m_1\ sat\ \chi_1, \ldots, H_k \vdash m_k\ sat\ \chi_k$,

Note that $\tilde{H}$ as defined above is a set of adaptable Hoare specifications that are equivalent to the SAT specifications H. But then, by theorem 7.1, there do exist sets of Hoare specifications $\tilde{H}_i$ and assertions φ_i such that:

(a") $\tilde{H}_i \models (\varphi_i)\ m_i\ (\varphi_i \lhd \chi_i)$ for $i = 1..k$, and

(b") $\tilde{H} \vdash (\varphi)\ m(m_1, \ldots, m_k)\ (\varphi \lhd (\varphi \rightsquigarrow \psi))$ is deducible from:

$$\tilde{H}_1 \vdash (\varphi_1)\ m_1\ (\varphi_1 \lhd \chi_1), \ldots, \tilde{H}_k \vdash (\varphi_k)\ m_k\ (\varphi_k \lhd \chi_k).$$

Therefore, if we choose $\psi_i \stackrel{\text{def}}{=} \varphi_i \lhd \chi_i$, then point (a) is satisfied. Moreover, $(\varphi \lhd (\varphi \rightsquigarrow \psi)) \rightarrow \psi$ is a valid implication, and so, one can apply the consequence rule to deduce:

$$\tilde{H} \vdash (\varphi)\ m(m_1, \ldots, m_k)\ (\psi)$$

from:

$$\tilde{H} \vdash (\varphi)\ m(m_1, \ldots, m_k)\ (\varphi \lhd (\varphi \rightsquigarrow \psi)).$$

If $\bar{g} \cap gvar(\tilde{H}) = \emptyset$, then we can apply the $\forall$- introduction rule to obtain:

$$\tilde{H} \vdash \forall \bar{g}[(\varphi)\ m(m_1, \ldots, m_k)\ (\psi)],$$

and we are finished.

If $\bar{g} \cap gvar(\tilde{H}) \neq \emptyset$, then we first choose some list of fresh ghost variables $\bar{g}'$, and deduce as above:

$$\tilde{H} \vdash \forall \bar{g}'[(\varphi[\bar{g}'/\bar{g}])\ m(m_1, \ldots, m_k)\ (\psi[\bar{g}'/\bar{g}])],$$

In section 4.11 we already showed how to change the name of bound variables, and in this way one can deduce:

$$\tilde{H} \vdash \forall \bar{g}[(\varphi)\ m(m_1, \ldots, m_k)\ (\psi)].$$

This was to be shown.

□

7.2 Freeze predicates

We are left with the proof of lemma 7.2. We define a special class of assertions that we shall use to prove this lemma. The assertions of this class are called *freeze predicates* since they are used to "freeze" the values of the channels and assignable variables in some particular state in the sense that logical variables are introduced that are set equal to these values. If such a freeze predicate is used as precondition to freeze the initial state values and initial trace values of variables, then one can (indirectly) refer to these values in the postcondition via the logical variables introduced by the freeze predicate.

Definition 7.5 (Freeze predicates)

Let $\beta = (\mathbf{c}, \{\bar{x}\})$ and let $\bar{v}_0$ be a list of the same length as $\bar{x}$ of logical variables. Let t_0 be a logical trace variable. We define the *freeze predicate* $\phi_\beta(t_0, \bar{v}_0)$ as follows:

$$\phi_\beta(t_0, \bar{v}_0) \stackrel{\text{def}}{=} \top \wedge (h|\mathbf{c} = t_0|\mathbf{c}) \wedge (\bar{x} = \bar{v}_0).$$

□

Next we prove the following lemma, which clearly implies lemma 7.2.

Lemma 7.6

Let χ be a satisfiable predicate such that $abase(\chi) \subseteq \beta$ and $gvar(\chi) \cap \{t_0, \bar{v}_0\} = \emptyset$. Then the following holds:

(i) For any mixed term m not containing t_0 or $\bar{v}_0$ free:

m *sat* χ iff $\forall t_0 \forall \bar{v}_0[(\phi_\beta(t_0, \bar{v}_0))m(\phi_\beta(t_0, \bar{v}_0) \lhd \chi)]$.

(ii) $\forall t_0 \forall \bar{v}_0[(\phi_\beta(t_0, \bar{v}_0)) \; \xi \; (\phi_\beta(t_0, \bar{v}_0) \lhd \chi)]$ is an adaptable specification for the Hoare system even when it does *not* include the "extra" adaptation rules of section 5.4.2.

□

Proof

Abbreviate $\phi_\beta(t_0, \bar{v}_0)$ as ϕ_β.

• If m *sat* χ then $\forall t_0 \forall \bar{v}_0[(\phi_\beta)m(\phi_\beta \lhd \chi)]$ as is shown by the following derivation:

$$\dfrac{\dfrac{m \textit{ sat } \chi}{(\phi_\beta)m(\phi_\beta \lhd \chi)}\ (\text{SP})}{\forall t_0 \forall \bar{v}_0[(\phi_\beta)m(\phi_\beta \lhd \chi)]}\ (\forall\text{–introduction})$$

To prove the reverse we start with a derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall t_0 \forall \bar{v}_0[(\phi_\beta)m(\phi_\beta \lhd \chi)]}{(\phi_\beta)m(\phi_\beta \lhd \chi)}\ (\forall\text{–elimination})}{m \textit{ sat } \phi_\beta \rightsquigarrow (\phi_\beta \lhd \chi)}\ (\text{HS})}{m \textit{ sat } \forall t_0 \forall \bar{v}_0[\phi_\beta \rightsquigarrow (\phi_\beta \lhd \chi)]}\ (\forall\text{–introduction})}{m \textit{ sat } \chi}\ (\text{Consequence})$$

The application of the consequence rule must be justified by proving the validity of:

$$\forall t_0 \forall \bar{v}_0 \Big[\phi_\beta \rightsquigarrow (\phi_\beta \lhd \chi)\Big] \rightarrow \chi. \qquad (1)$$

The implication (1) is not valid for arbitrary assertions ϕ_β; rather it is a typical property of our freeze predicates. To prove (1) we first expand the definition of $\phi_\beta \lhd \chi$.

$\phi_\beta \lhd \chi$ iff

$(\phi_\beta[\bot] \wedge \chi[\bot^\circ] \wedge \bot) \vee$

$\Big(\exists t_1, t_2 \exists \bar{v}(h|\mathbf{c} = (t_1 t_2)|\mathbf{c} \wedge \phi_\beta[\top][t_1/h][\bar{v}/\bar{x}] \wedge \chi[\top^\circ][\bar{v}/\bar{x}^\circ][t_2/h])\Big)$

iff

$(\mathbf{false} \wedge - - -) \vee \exists\Big(h|\mathbf{c} = (t_0 t)|\mathbf{c} \wedge \chi[\top^\circ][\bar{v}_0/\bar{x}^\circ][t/h]\Big).$

This expansion is used in the following expansion of $\forall t_0 \forall \bar{v}_0[\phi_\beta \rightsquigarrow (\phi_\beta \lhd \chi)]$:

$\forall t_0 \forall \bar{v}_0 \Big[\phi_\beta \rightsquigarrow (\phi_\beta \lhd \chi)\Big]$ iff

$\forall t_0 \forall \bar{v}_0 \forall t_0' \Big(\phi_\beta[t_0'/h, \bar{x}^\circ/\bar{x}, \top^\circ/\top] \rightarrow (\phi_\beta \lhd \chi)[t_0' h/h]\Big)$ iff

$\forall t_0 \forall \bar{v}_0 \forall t_0' \Big((\top^\circ \wedge t_0'|\mathbf{c} = t_0|\mathbf{c} \wedge \bar{x}^\circ = \bar{v}_0) \rightarrow$

$\exists t((t_0' h)|\mathbf{c} = (t_0 t)|\mathbf{c} \wedge \chi[\top^\circ][\bar{v}_0/\bar{x}^\circ][t/h])\Big).$ (2)

Choosing $t_0' = t_0 = \varepsilon$, $\bar{v}_0 = \bar{x}^\circ$, we see that (2) implies:

$$\top^\circ \rightarrow \exists t\Big(h|\mathbf{c} = t|\mathbf{c} \wedge \chi[\top^\circ][t/h]\Big). \qquad (3)$$

And from (3) it follows that:

$$\top^\circ \rightarrow \chi[\top^\circ]. \qquad (4)$$

Since χ is satisfiable, $\chi[\bot^\circ]$is valid. And therefore we see that:

$$\chi \text{ iff } \Big(\top^\circ \rightarrow \chi[\top^\circ]) \wedge (\bot^\circ \rightarrow \chi[\bot^\circ]) \text{ iff } (\top^\circ \rightarrow \chi[\top^\circ]\Big).$$

So (4) actually is equivalent to χ.

This ends the proof of implication (1), and so of the proof of part (i) of the lemma.

Next we prove part (ii).

First we show how to adapt the formula

$$\forall t_0 \forall \bar{v}_0 \Big[(\phi_\beta)\ \xi\ (\phi_\beta \lhd \chi) \Big] \qquad (1)$$

to a similar formula that expresses extra invariance properties.

Let $\beta' = (\mathbf{d}, \{\bar{y}\})$ such that $\beta' \cap base(\xi) = (\emptyset, \emptyset)$.

Let $\bar{w}_0$ be a list of logical variables of the same length as $\bar{y}$. If the i-th element of $\bar{y}$ actually is the same variable as the j-th element of $\bar{x}$, then take for the i-th element of $\bar{w}_0$ variables the j-th element of $\bar{v}_0$. Choose *fresh* $\bar{w}_0$ variables for those $\bar{y}$ elements not already occurring in $\bar{x}$. In short: although the $\bar{x}$ and $\bar{y}$ list need not be disjunct we have chosen the $\bar{w}_0$ consistent with the already given $\bar{v}_0$ list. Now our claim is that if:

$$\phi_{\tilde{\beta}} \stackrel{\text{def}}{=} \top \wedge (t_0|(\mathbf{c} \cup \mathbf{d}) = h|(\mathbf{c} \cup \mathbf{d})) \wedge \bar{x} = \bar{v}_0 \wedge \bar{y} = \bar{w}_0,$$

then we can derive from (1) the formula:

$$\forall t_0 \forall \bar{v}_0, \bar{w}_0 \Big[(\phi_{\tilde{\beta}})\ \xi\ (\phi_{\tilde{\beta}} \lhd (\chi \wedge \mathbf{1}_{\beta'})) \Big]. \qquad (2)$$

Let:

$$\varphi_F \stackrel{\text{def}}{=} (t_0|\mathbf{d} = h|\mathbf{d} \wedge (\top \rightarrow \bar{y} = \bar{w}_0)),$$

$$\varphi_P \stackrel{\text{def}}{=} (t_0|(\mathbf{c} \cup \mathbf{d}) \leq h|(\mathbf{c} \cup \mathbf{d})).$$

We use instances of the invariance- and prefix invariance axioms:

$$(\varphi_F[\bot] \wedge \varphi_F)\ \xi\ (\varphi_F), \qquad (3)$$

$$(\varphi_P)\ \xi\ (\varphi_P). \qquad (4)$$

By application of the $\forall - E$ rule, followed by the conjunction rule we derive from (1), (3) and (4):

$$(\phi_\beta \wedge \varphi_F[\bot] \wedge \varphi_F \wedge \varphi_P) \xi ((\phi_\beta \lhd \chi) \wedge \varphi_F \wedge \varphi_P). \quad (5)$$

Clearly it is the case that

$$\phi_{\tilde{\beta}} \rightarrow (\phi_\beta \wedge \varphi_F[\bot] \wedge \varphi_F \wedge \varphi_P)$$

is valid. If we can prove that

$$((\phi_\beta \lhd \chi) \wedge \varphi_F \wedge \varphi_P) \rightarrow (\phi_{\tilde{\beta}} \lhd (\chi \wedge \mathbf{1}_{\beta'})) \quad (6)$$

is valid too, then the consequence rule is applicable to (5) and yields:

$$(\phi_{\tilde{\beta}})\ \xi\ (\phi_{\tilde{\beta}} \lhd (\chi \wedge \mathbf{1}_{\beta'})). \qquad (7)$$

Since neither t_0 nor any of the $\bar{v}_0$ or $\bar{w}_0$ occur free in hypotheses of this derivation of (7) we can use the $\forall$–introduction rule to obtain the desired formula (2) from (7).

We are left with the proof of (6).

Assume, for arbitrary chosen $(h, s) \in \Sigma, \gamma \in \Gamma$, that:

$(h, s) \in [\![(\phi_\beta \lhd \chi) \wedge \varphi_F \wedge \varphi_P]\!]\gamma$,

that is:

(a) $\exists h_0', h_1' \exists s_0' \big(h = h_0' h_1' \wedge s_0' \neq \perp \wedge (h_0'|\mathbf{c} = \gamma(t_0)|\mathbf{c}) \wedge$

$s_0'(\bar{x}) = \gamma(\bar{v}_0) \wedge (s_0', h_1', s) \in [\![\chi]\!]\gamma\big)$, and

(b) $h|\mathbf{d} = \gamma(t_0)|\mathbf{d} \wedge (s \neq \perp \rightarrow s(\bar{y}) = \gamma(\bar{w}_0))$, and

(c) $\gamma(t_0)|(\mathbf{c} \cup \mathbf{d}) \leq h|(\mathbf{c} \cup \mathbf{d})$.

From the above it must be shown that:

$(h, s) \in [\![\phi_{\bar{\beta}} \lhd (\chi \wedge \mathbf{1}_{\beta'})]\!]\gamma$,

that is:

$\exists h_0, h_1 \exists s_0 \big(h = h_0 h_1 \wedge s_0 \neq \perp \wedge (h_0|(\mathbf{c} \cup \mathbf{d}) = \gamma(t_0)|(\mathbf{c} \cup \mathbf{d}))$

$\wedge s_0(\bar{x}) = \gamma(\bar{v}_0) \wedge s_0(\bar{y}) = \gamma(\bar{w}_0) \wedge (s_0, h_1, s) \in [\![\chi]\!]\gamma$

$\wedge h_1|\mathbf{d} = \varepsilon \wedge (s \neq \perp \rightarrow s(\bar{y}) = s_0(\bar{y}))\big)$.

Let h_0', h_1', s_0' be as indicated under (a). By (c) we can choose some h_0 such that $h_0 \leq h$ and $\gamma(t_0)|(\mathbf{c} \cup \mathbf{d}) = h_0|(\mathbf{c} \cup \mathbf{d})$. Take for this choice of h_0 a trace h_1 such that $h_0 h_1 = h$. Choose $s_0 = s_0'[\gamma(\bar{w}_0)/\bar{y}]$. Since $s_0' \neq \perp$ also $s_0 \neq \perp$, and

$$s_0(\bar{y}) = s_0'[\gamma(\bar{w}_0)/\bar{y}] = \gamma(\bar{w}_0).$$

For those x from $\bar{x}$ not already occurring in $\bar{y}$:

$$s_0(x) = s_0'[\gamma(\bar{w}_0)/\bar{y}] = s_0'(x) = \gamma(v_0),$$

where v_0 is the $\bar{v}_0$ element corresponding to x. Since the list $\bar{w}_0$ was chosen in a way consistent with $\bar{v}_0$ and $s_0(\bar{y}) = \gamma(\bar{w}_0)$ we see that $s_0(\bar{x}) = \gamma(\bar{v}_0)$ is the case.

To see that $(s_0, h_1, s) \in [\![\chi]\!]\gamma$ it will suffice to show that (s_0, h_1, s) and (s_0', h_1', s) agree on $abase(\chi)(\subseteq (\mathbf{c}, \{\bar{x}\}))$.
For we know already that $(s_0', h_1', s) \in [\![\chi]\!]\gamma$. From (a) follows $s_0'(\bar{x}) = \gamma(\bar{v}_0)$,

and we just showed $s_0(\bar{x}) = \gamma(\bar{v}_0)$, so s_0 and $s_0{}'$ do agree on $\bar{x}$. From the definition of h_0 follows directly that $\gamma(t_0)|\mathbf{c} = h_0|\mathbf{c}$ and $\gamma(t_0)|\mathbf{d} = h_0|\mathbf{d}$. Combining with equalities from (a) and (b) we see that:

$h_0|\mathbf{c} = \gamma(t_0)|\mathbf{c} = h_0'|\mathbf{c}$, and

$h_0|\mathbf{d} = \gamma(t_0)|\mathbf{d} = h|\mathbf{d}$.

Since $h_0 h_1 = h = h_0' h_1'$, this implies:

$h_1|\mathbf{c} = h_1'|\mathbf{c}$ and

$h_1|\mathbf{d} = \varepsilon$.

We have shown that (s_0, h_1, s) and $(s_0{}', h_1', s)$ agree on $abase(\chi)$ and so that $(s_0, h_1, s) \in [\![\chi]\!]\gamma$. Also $h_1|\mathbf{d} = \varepsilon$ has been shown, so there remains to see that if $s \neq \perp$ then $s(\bar{y}) = s_0(\bar{y})$. But this is clear since both sides equal $\gamma(\bar{w}_0)$ in this case, as follows from the definition of s_0 and (b).

At last we can continue the proof of part (ii).

Assume that the following specification implication holds:

$$\forall\xi\Big[\forall t_0\forall\bar{v}_0((\phi_\beta)\ \xi\ (\phi_\beta \lhd \chi)) \rightarrow \forall\bar{g}[(\varphi)\ \xi\ (\psi)]\Big]. \quad (8)$$

We must show that

$$\forall\bar{g}[(\varphi)\ \xi\ (\psi)]$$

can de derived from:

$$\forall t_0\forall\bar{v}_0((\phi_\beta)\ \xi\ (\phi_\beta \lhd \chi)).$$

We have already shown how to change the names of bound logical variables, and so we may assume here that the logical variables $\{\bar{g}\}$ are disjunct from the "freeze" variables $\{t_0, \bar{v}_0\}$ and from any logical variables that occur in hypotheses. Under these conditions, (8) implies the following formula:

$$\forall\xi\Big[\forall t_0\forall\bar{v}_0((\phi_\beta)\ \xi\ (\phi_\beta \lhd \chi)) \rightarrow (\varphi)\ \xi\ (\psi)\Big].$$

By part (i) of the lemma, this is equivalent to:

$$\forall\xi\Big[\xi\ sat\ \chi \rightarrow (\varphi)\ \xi\ (\psi)\Big], \quad (9)$$

and this is equivalent to:

$$\forall\xi\Big[\xi\ sat\ \chi \rightarrow \xi\ sat\ (\varphi \rightsquigarrow \psi)\Big]. \quad (10)$$

Let $\beta' = abase(\chi, \varphi \rightsquigarrow \psi) - base(\xi)$. Then lemma 6.15 states that (10) implies the validity of:

$$(\chi \wedge \mathbf{1}_{\beta'}) \rightarrow (\varphi \rightsquigarrow \psi). \quad (11)$$

We have shown already how to adapt the formula

$$\forall t_0 \forall \bar{v}_0((\phi_\beta)\ \xi\ (\phi_\beta \lhd \chi))$$

to:

$$\forall t_0 \forall \bar{v}_0, \bar{w}_0\Big((\phi_{\bar{\beta}})\ \xi\ (\phi_{\bar{\beta}} \lhd (\chi \wedge \mathbf{1}_{\beta'}))\Big). \quad (12)$$

(Notation as above, except that we have now fixed our choice for β' as $abase(\varphi' \rightsquigarrow \psi') - base(\xi)$). If we remove the quantifiers by means of the $\forall$–elimination rule and then apply the consequence rule we obtain:

$$(\phi_{\bar{\beta}})\ \xi\ (\phi_{\bar{\beta}} \lhd (\varphi \rightsquigarrow \psi)). \quad (13)$$

Let $\varphi' \equiv \varphi[\top][t_0/h][\bar{v}_0/\bar{x}, \bar{w}_0/\bar{y}]$.

Then $(\varphi')\ \xi\ (\varphi')$ follows directly from an instance of the invariance axiom. Combining this formula with (13) via the conjunction rule yields:

$$(\phi_{\bar{\beta}} \wedge \varphi')\ \xi\ ((\phi_{\bar{\beta}} \lhd (\varphi \rightsquigarrow \psi)) \wedge \varphi'). \quad (14)$$

We prove below, in lemma 7.7, the validity of:

$$\Big((\phi_{\bar{\beta}} \lhd (\varphi \rightsquigarrow \psi)) \wedge \varphi'\Big) \rightarrow \psi. \quad (15)$$

Therefore we can proceed the derivation as follows:

$$\dfrac{\dfrac{\dfrac{(14)}{(\phi_{\bar{\beta}} \wedge \varphi')\ \xi\ (\psi)}\ \text{(consequence)}}{(\exists t_0 \exists \bar{v}_0 \exists \bar{w}_0 [\phi_{\bar{\beta}} \wedge \varphi'])\ \xi\ (\psi)}\ (\exists - pre)}{(\top \wedge \varphi)\ \xi\ (\psi)}\ \text{(consequence)}$$

By the assumption that χ is a satisfiable assertion, together with (9) it follows that $(\varphi)\ \xi\ (\psi)$ is a satisfiable formula. As before, this implies that $(\bot \wedge \varphi) \rightarrow \psi$ is valid and so that:

$$\dfrac{(\bot \wedge \varphi)\ \xi\ (\bot \wedge \varphi)\ \text{strictness}}{(\bot \wedge \varphi)\ \xi\ (\psi)}\ \text{(consequence)}$$

is a legitimate derivation. Using the disjunction rule and consequence rule again we obtain the formula:

$(\varphi)\ \xi\ (\psi)$.

Finally, by means of the $\forall$−introduction rule we obtain the desired formula:

$\forall \bar{g}[(\varphi)\ \xi\ (\psi)]$.

□

Lemma 7.7

Let $\varphi, \psi \in \mathit{Assn}(\Sigma)$ with $\mathit{abase}(\varphi, \psi) \subset \beta = (\mathbf{c}, \{\bar{x}\})$, and $t_0, \bar{v}_0 \notin \mathit{gvar}(\varphi, \psi)$. If $\phi_\beta(t_0, \bar{v}_0)$ is the freeze predicate $\top \wedge t_0|\mathbf{c} = h|\mathbf{c} \wedge \bar{x} = \bar{v}_0$, and $\varphi' \equiv \varphi[\top][t_0/h][\bar{v}_0/\bar{x}]$ then the following assertion is valid:

$$\Big((\phi_\beta(t_0, \bar{v}_0) \lhd (\varphi \rightsquigarrow \psi)) \wedge \varphi'\Big) \rightarrow \psi.$$

□

Proof

Take some arbitrary $(h, s) \in \Sigma, \gamma \in \Gamma$. If $(h, s) \in [\![\phi_\beta(t_0, \bar{v}_0) \lhd (\varphi \rightsquigarrow \psi)]\!]\gamma$ then:

$$\exists h_0 \exists h_1 \exists s_0 \Big(h = h_0 h_1 \wedge (h_0, s_0) \in [\![\phi_\beta(t_0, \bar{v}_0)]\!]\gamma \wedge$$

$$(s_0, h_1, s) \in [\![\varphi \rightsquigarrow \psi]\!]\gamma\Big), \text{ that is:}$$

$$\exists h_0 \exists h_1 \exists s_0 \Big(h = h_0 h_1 \wedge s_0 \neq \bot \wedge \gamma(t_0)|\mathbf{c} = h_0|\mathbf{c} \wedge \gamma(\bar{v}_0) = s_0(\bar{x})$$

$$\wedge \forall h_0((h_0, s_0) \in [\![\varphi]\!]\gamma \Rightarrow (h_0 h_1, s) \in [\![\psi]\!]\gamma)\Big). \quad (1)$$

If also $(h, s) \in [\![\varphi']\!]\gamma$ then for any $\tilde{s} \in \mathit{State}$, i.e. $\tilde{s} \neq \bot$,

$$((\gamma(t_0)), \tilde{s}|\bar{x} : \gamma(\bar{v}_0)) \in [\![\varphi]\!]\gamma. \quad (2)$$

We must prove from (1) and (2) that $(h, s) \in [\![\psi]\!]\gamma$. This is straightforward: Take h_0, h_1 and s_0 as indicted by (1). Then from (1) it follows that (h_0, s_0) and $(\gamma(t_0), \tilde{s}|\bar{x} : \gamma(\bar{v}_0))$ agree on $\mathit{abase}(\varphi)$. By (2) the latter pair is member of $[\![\varphi]\!]\gamma$ and therefore also $(h_0, s_0) \in [\![\varphi]\!]\gamma$. But then the implication in (1) guarantees that $(h_0 h, s) \in [\![\psi]\!]\gamma$, i.e, that $(h, s) \in [\![\psi]\!]\gamma$.

□

7.3 Adaptation completeness for the Hoare system

We prove that the Hoare system is adaptation complete if it includes the "strong adaptation rule". That is if

$$\forall \bar{g}_1\Big[(\varphi_1)\ X_\beta\ (\psi_1)\Big] \to \forall \bar{g}_2\Big[(\varphi_2)\ X_\beta\ (\psi_2)\Big]$$

is a valid formula, then $\forall \bar{g}_2\Big[(\varphi_2)\ X_\beta\ (\psi_2)\Big]$ is provable from the hypothesis $\forall \bar{g}_1\Big[(\varphi_1)\ X_\beta\ (\psi_1)\Big]$.

In chapter 4 we showed that our formal system allows one to change the names of bound variables. That is, if g' is a fresh variable, then the following is deducible:

$$\forall g\Big[(\varphi)\ X_\beta\ (\psi)\Big] \vdash \forall g'\Big[(\varphi[g'/g])\ X_\beta\ (\psi[g'/g])\Big].$$

Therefore, we may assume without loss of generality that, for the formulae as above, the following conditions are satisfied:

$\{\bar{g}_1\} \cap gvar(\varphi_2, \psi_2) = \emptyset$, and

$\{\bar{g}_2\} \cap gvar(\varphi_1, \psi_1) = \emptyset$.

Under these conditions, it is the case that

$\models \forall \bar{g}_1\Big[(\varphi_1)\ X_\beta\ (\psi_1)\Big] \to \forall \bar{g}_2\Big[(\varphi_2)\ X_\beta\ (\psi_2)\Big]$ iff

$\models \forall \bar{g}_1 \forall \bar{g}_2\Big[(\varphi_1)\ X_\beta\ (\psi_1) \to (\varphi_2)\ X_\beta\ (\psi_2)\Big]$ iff

$\models (\varphi_1)\ X_\beta\ (\psi_1) \to (\varphi_2)\ X_\beta\ (\psi_2)$ iff

$\models X_\beta\ sat\ (\varphi_1 \rightsquigarrow \psi_1) \to X_\beta\ sat\ (\varphi_2 \rightsquigarrow \psi_2)$.

From the results on modular completeness for the SAT system, we know that the last formula implies the validity of the following assertion:

$$\Big((\varphi_1 \rightsquigarrow \psi_1) \wedge \mathbf{1}_{(\mathbf{d},\{\bar{y}\})}\Big) \to (\varphi_2 \rightsquigarrow \psi_2),$$

where $(\mathbf{d}, \{\bar{y}\}) \stackrel{\text{def}}{=} abase(\varphi_1 \rightsquigarrow \psi_1, \varphi_2 \rightsquigarrow \psi_2) - \beta$.

In chapter 4 we also proved the validity of

$$\Big(\varphi \lhd (\varphi \rightsquigarrow \psi)\Big) \to \psi.$$

Altogether we now see that the following derivation is possible within the Hoare system:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\forall \bar{g}_1[(\varphi_1)\ X_\beta\ (\psi_1)]}{(\varphi_1)\ X_\beta\ (\psi_1)}\ (\forall\text{-elimination})}{(\varphi_2)\ X_\beta\ (\varphi_2 \lhd ((\varphi_1 \rightsquigarrow \psi_1) \wedge \mathbf{1}_{(\mathbf{d},\{\bar{y}\})}))}\ (\text{Strong adaptation})}{(\varphi_2)\ X_\beta\ (\varphi_2 \lhd ((\varphi_2 \rightsquigarrow \psi_2))}\ (\text{Consequence})}{(\varphi_2)\ X_\beta\ (\psi_2)}\ (\text{Consequence})}{\forall \bar{g}_2[(\varphi_2)\ X_\beta\ (\psi_2)]}\ \forall\text{-introduction})$$

This proves the adaptation completeness of the Hoare system.

Theorem 7.8

The Hoare system is modular complete.

□

Proof

Since we already showed the compositional completeness of this system, it follows from the results of chapter 1 that the Hoare system is modular complete.

7.4 The Invariant system

We prove the compositional completeness of the Invariant system. Since the formulae and proof rules of the Invariant system are so closely related to the formulae and rules of the Hoare system, we shall base our proof upon a transformation of Hoare style proofs in Invariant style proofs.

For a given Hoare formula f of the form $(\varphi)\ m\ (\psi)$ we denote by $f^\dagger$ the corresponding Invariant formula $\psi[\bot] : \{\varphi[\top]\}\ m\ \{\psi[\top]\}$. In chapter 4 we proved the equivalence of f and $f^\dagger$, for **TNP** *processes* S. For a set of Hoare formulae H we use $H^\dagger$ for the set of Invariant formulae obtained from H by applying the $\dagger$ operation to each formula in H.

Theorem 7.9 Transformation of Hoare proofs to Invariant proofs

Let $m(m_1, \ldots, m_k)$ be some mixed term with occurrences of the meta variables $m_1, \ldots, m_k$. Assume that all predicative processes used within m are of the restricted form, $\langle J, R \rangle$, as defined with the Invariant rule for predicative processes, and that they are all satisfiable.

Let some Hoare style proof scheme be given that shows how

$$H \vdash (\varphi)\ m(m_1, \ldots, m_k)(\psi)$$

is deduced from

$$H_1 \vdash (\varphi_1)\ m_1\ (\psi_1),$$

$$\vdots$$

$$H_k \vdash (\varphi_k)\ m_k\ (\psi_k),$$

and that does *not* use the "extra adaptation rules" of section 5.4.2. Assume also that all Hoare specifications used in this scheme are *satisfiable*.

Then there is a proof scheme for the Invariant system for which:

$$H^\dagger \vdash \psi[\bot] : \{\varphi[\top]\}\ m(m_1, \ldots, m_k)\{\psi[\top]\}$$

is deduced from:

$$H_1^\dagger \vdash \psi_1[\bot] : \{\varphi_1[\top]\}\ m_1\ \{\psi_1[\top]\},$$

$$\vdots$$

$$H_k^\dagger \vdash \psi_k[\bot] : \{\varphi_k[\top]\}\ m_k\ \{\psi_k[\top]\}.$$

□

Proof

The proof is with induction on the length of the given Hoare style deduction. We make a case distinction, according to the proof rule used in the last step of this deduction.

- For a number of axioms and rules it is the case that if

$$\frac{(\varphi_1)m_1(\psi_1), \cdots, (\varphi_k)m_k(\psi_k)}{(\varphi)m(\psi)}$$

is an instance of the (axiom or) rule in the Hoare system, then

$$\frac{\psi_1[\bot] : \{\varphi_1[\top]\}m_1\{\psi_1[\top]\}, \cdots, \psi_k[\bot] : \{\varphi_k[\top]\}m_k\{\psi_k[\top]\}}{\psi[\bot] : \{\varphi[\top]\}m\{\psi[\top]\}}$$

is an instance of the corresponding (axiom or) rule in the Invariant system. In these cases the induction step is immediately clear. By inspection of the rules one sees that this is the case for the following axioms and rules:

- channel hiding
- variable hiding
- channel renaming
- parallel composition
- choice
- conjunction
- disjunction
- prefix invariance
- $\exists$-pre

We check the other possible axioms and rules.

- Abort,**Z**

  The given instance of the abort axiom:

  $(\psi[\bot])$ **abort** (ψ)

  transforms into:

  $$\frac{\psi[\bot] : \{\psi[\bot]\}\ \textbf{abort}\ \{\textbf{false}\}\ \text{(abort)}}{\psi[\bot] : \{\psi[\bot]\}\ \textbf{abort}\ \{\psi[\top]\}}\ \text{(Consequence)}$$

  The process **Z** is treated completely similar.

- Skip

  The given instance of the skip axiom:

  $(\psi[\bot] \wedge \psi)$ **skip** (ψ)

  transforms into:

  $$\frac{\psi[\bot] : \{\psi[\top] \wedge \psi[\bot]\}\ \textbf{skip}\ \{\psi[\top] \wedge \psi[\bot]\}\ \text{(skip)}}{\psi[\bot] : \{\psi[\bot] \wedge \psi[\top]\}\ \textbf{skip}\ \{\psi[\top]\}}\ \text{(Consequence)}$$

- Guard

  The given instance:

  $$(\psi[\bot] \wedge (b \rightarrow \psi[\top]))\ b\ (\psi)$$

  transforms into:

  $$\frac{\psi[\bot] : \{\psi[\bot] \wedge (b \rightarrow \psi[\top])\}\ b\ \{\psi[\bot] \wedge (b \rightarrow \psi[\top]) \wedge b\}}{\psi[\bot] : \{\psi[\bot] \wedge (b \rightarrow \psi[\top])\}\ b\ \{\psi[\top]\}}\ \text{(Consequence)}$$

  The derived conclusion is the desired formula since

  $$\Big(\psi[\bot] \wedge (b \rightarrow \psi[\top])\Big)[\top] \equiv$$
  $$\psi[\bot][\top] \wedge (b[\top] \rightarrow \psi[\top][\top]) \equiv$$
  $$\psi[\bot] \wedge (b \rightarrow \psi[\top]).$$

- Assign

  The given instance:

  $$(\psi[\bot] \wedge (\top \rightarrow \psi[\top][e/x]))\ x := e\ (\psi)$$

  transforms into:

  $$\frac{\psi[\bot] : \{\psi[\bot] \wedge \psi[\top][e/x]\}\ x := e\ \{\psi[\bot] \wedge \psi[\top]\}\ \text{(assign)}}{\psi[\bot] : \{\psi[\bot] \wedge \psi[\top][e/x]\}\ x := e\ \{\psi[\top]\}}\ \text{(Cons.)}$$

  The derived conclusion is the desired formula since

  $$\Big(\psi[\bot] \wedge (\top \rightarrow \psi[\top])\Big)[\top] \equiv$$
  $$\psi[\bot][\top] \wedge (\top[\top] \rightarrow \psi[\top][\top]) \equiv$$
  $$\psi[\bot] \wedge \psi[\top]$$

- Communication

  The given instance is:

  $$(\psi[\bot] \wedge$$
  $$\top \rightarrow \forall v \Big(b[v/x] \rightarrow (\psi[\bot][h < (c, v) > /h] \wedge \psi[\top][h < (c, v) > /h, v/x])\Big))$$
  $$c.x : b$$
  $$(\psi)$$

The corresponding derivation in the invariant system is:

$\psi[\bot]$:

$\{\psi[\bot] \wedge$
$\forall v \big(b[v/x] \rightarrow (\psi[\bot][h < (c,v) > /h] \wedge \psi[\top][h < (c,v) > /h][v/x]) \big) \}$

$c.x : b$

$\{\psi[\top] \wedge \psi[\bot]\}$,

followed by an application of the consequence rule to remove the conjunct $\psi[\bot]$ from the postcondition.

- Predicative processes

We only consider the transformation for predicative processes of the form $\langle J, R \rangle$, as defined with the Invariant system. The term $\langle J, R \rangle$ abbreviates the process:

$$\langle J, R \rangle \stackrel{\text{def}}{=} ((\top^\circ \rightarrow J) \wedge (\top \rightarrow R)) | base(J, R).$$

Now assume that we have an instance of the axiom for predicative processes of the following form:

$$(\varphi)\ \langle J, R \rangle\ (\varphi \lhd ((\top^\circ \rightarrow J) \wedge (\top \rightarrow R))).$$

By expansion of the definition of the $\lhd$ operator, one sees that the following implications are valid:

$(\varphi[\top] \lhd J) \rightarrow (\varphi \lhd ((\top^\circ \rightarrow J) \wedge (\top \rightarrow R))[\bot])$,

$(\varphi[\top] \lhd (J \wedge R)) \rightarrow (\varphi \lhd ((\top^\circ \rightarrow J) \wedge (\top \rightarrow R))[\top])$.

These facts form the basis for the following derivation.
Let $\chi_{JR} \stackrel{\text{def}}{=} \varphi \lhd ((\top^\circ \rightarrow J) \wedge (\top \rightarrow R))$.

$$\frac{\varphi[\top] \lhd J : \{\varphi[\top]\}\ \langle J, R \rangle\ \{\varphi[\top] \lhd (J \wedge R)\}}{(\varphi \lhd \chi_{J,R})[\bot] : \{\varphi[\top]\}\ \langle J, R \rangle\ \{(\varphi \lhd \chi_{JR})[\top]\}}\ \text{(Consequence)}$$

- Parallel composition

Although parallel composition is one of the "trivial" cases from the list above, we show what the transformation is, as an example:

$$\frac{(\varphi_1)\ m_1\ (\psi_1) \quad , \quad (\varphi_2)\ m_2\ (\psi_2)}{(\varphi_1 \wedge \varphi_2)\ m_1\ \beta_1 \| \beta_2\ m_2\ (\psi_1 \wedge \psi_2)}$$

transforms into:

$$\frac{\psi_1[\bot] : \{\varphi_1[\top]\}\ m_1\ \{\psi_1[\top]\}, \psi_2[\bot] : \{\varphi_2[\top]\}\ m_2\ \{\psi_2[\top]\}}{(\varphi_1 \wedge \varphi_2)[\bot] : \{(\varphi_1 \wedge \varphi_2)[\top]\}\ m_1\ \beta_1 \| \beta_2\ m_2\ \{(\psi_1 \wedge \psi_2)[\top]\}}$$

Here we used the fact that $(\varphi_1 \wedge \varphi_2)[\top] \equiv \varphi_1[\top] \wedge \varphi_2[\top]$ etc. It is easily checked that the restrictions for the rule are equivalent to those for the rule in the given proof step above.

- Sequential composition

  Let the given derivation end as:

$$\frac{(\varphi)\ m_1\ (\rho) \quad , \quad (\rho)\ m_2\ (\psi)}{(\varphi)\ m_1; m_2\ (\psi)}$$

  We first show that $\rho[\bot] \rightarrow \psi[\bot]$ is a valid assertion.

  By the assumption that all predicative processes in the given scheme are satisfiable, we know that $\mathbf{Z} \subseteq m_2$. So we conclude that $(\rho)\ \mathbf{Z}\ (\psi)$ is valid.

  Since $\mathcal{T}r[\![\mathbf{Z}]\!]\gamma\eta([\![\bot \wedge \rho[\bot]]\!]\gamma) = [\![\bot \wedge \rho[\bot]]\!]\gamma$ we see that $[\![\bot \wedge \rho[\bot]]\!]\gamma \subseteq [\![\psi]\!]\gamma$ must hold. Now if $(h, s) \in [\![\rho[\bot]]\!]\gamma$ then $(h, \bot) \in [\![\bot \wedge \rho[\bot]]\!]\gamma$ so $(h, \bot) \in [\![\bot \wedge \psi[\bot]]\!]\gamma$ and so $(h, s) \in [\![\psi[\bot]]\!]\gamma$. This proves the validity of the implication $\rho[\bot] \rightarrow \psi[\bot]$.

  Therefore, the proof step above can be transformed into:

$$\dfrac{\dfrac{\rho[\bot] : \{\rho[\top]\} m_1 \{\rho[\top]\}}{\psi[\bot] : \{\varphi[\top]\} m_1 \{\rho[\top]\}}\text{(Cons.)} \quad \psi[\bot] : \{\rho[\top]\} m_2 \{\psi[\top]\}}{\psi[\bot] : \{\varphi[\top]\} m_1; m_2 \{\psi[\top]\}}\text{(Seq. comp.)}$$

- (μ_z) Recursion

  Let the given deduction end as follows:

$$\frac{H \vdash \forall \bar{g}[(\varphi)\ \mathbf{Z}\ (\psi)] \quad , \quad H \cup \{\forall \bar{g}[(\varphi)\ X_\beta\ (\psi)]\} \vdash \forall \bar{g}[(\varphi)\ m\ (\psi)]}{H \vdash \forall \bar{g}[(\varphi)\ \mu_z X_\beta . m\ (\psi)]}$$

  The restriction for this rule is that $X_\beta \notin pvar(H)$.

  From the validity of the premisse $H \vdash (\varphi)\ \mathbf{Z}\ (\psi)$ follows that:

$$\mathcal{T}r(Obs[\![\mathbf{Z}]\!]\gamma\eta)([\![\varphi]\!]\gamma) \subseteq [\![\psi]\!]\gamma.$$

($\mathcal{T}r$ was defined in section 3.9).

That is:

$$\{(h, \bot) \mid (h, s) \in [\![\varphi]\!]\gamma\} \subseteq [\![\psi]\!]\gamma.$$

Since for $s \neq \bot$ it is the case that $(h, s) \in [\![\varphi[\top]]\!]\gamma$ iff $(h, s) \in [\![\varphi]\!]\gamma$ and $(h, s) \in [\![\psi[\bot]]\!]\gamma$ iff $(h, \bot) \in [\![\psi]\!]\gamma$, one sees that:

$$\varphi[\top] \rightarrow \psi[\bot]$$

is proper valid, and so $\varphi[\top] \leftrightarrow (\varphi[\top] \wedge \psi[\bot])$ is a proper equivalence. Therefore the proof step above is transformed into:

$$\frac{\begin{array}{l} H \cup \{\forall \bar{g}[\psi[\bot] : \{\varphi[\top] \wedge \psi[\bot]\}\ X_\beta\ \{\psi[\top]\}]\} \vdash \\ \forall \bar{g}[\psi[\bot] : \{\varphi[\top] \wedge \psi[\bot]\}\ m_1\ \{\psi[\top]\}] \end{array}}{\dfrac{\forall \bar{g}[\psi[\bot] : \{\varphi[\top] \wedge \psi[\bot]\}\ \mu_z X_\beta.m_1\ \{\psi[\top]\}]}{\forall \bar{g}[\psi[\bot] : \{\varphi[\top]\}\ \mu_z X_\beta.m_1\ \{\psi[\top]\}]}\ \text{(Cons.)}}$$

- Invariance

$$(\psi[\bot] \wedge \psi)m(\psi)$$

with the restriction $Abase(\psi) \cap Base(m) = (\emptyset, \emptyset)$, transforms into:

$$\frac{\psi[\bot] : \{\psi[\bot] \wedge \psi[\top]\}m\{\psi[\bot] \wedge \psi[\top]\} \quad \text{(Invariance)}}{\psi[\bot] : \{\psi[\bot] \wedge \psi[\top]\}m\{\psi[\top]\}}\ \text{(Consequence)}$$

The conclusion of this derivation is the one that was to be shown provable, since

$$(\psi[\bot] \wedge \psi)[\top] \equiv \psi[\bot][\top] \wedge \psi[\top] \equiv \psi[\bot] \wedge \psi[\top].$$

($\psi[\bot]$ does not contain $\bot$or $\top$ symbols, so $\psi[\bot][\top] \equiv \psi[\bot]$).

- Consequence

$$\frac{\begin{array}{l} \forall_\bot(\varphi \rightarrow \varphi'), \forall_\bot(\psi' \rightarrow \psi), \\ (\varphi')m(\psi') \end{array}}{(\varphi)m(\psi)}\ \text{(Consequence)}$$

transforms into:

$$\frac{\underline{\forall}(\psi'[\bot]\rightarrow\psi[\bot]),\underline{\forall}(\varphi[\top]\rightarrow\varphi'[\top]),\underline{\forall}(\psi'[\top]\rightarrow\psi[\top])\quad \psi'[\bot]:\{\varphi'[\top]\}m\{\psi'[\top]\}}{\psi[\bot]:\{\varphi[\top]\}m\{\psi[\top]\}}$$

- Strictness

  If f is an instance of the strictness axiom of the form:

  $$(\bot\wedge\varphi)\ S\ (\bot\wedge\varphi),$$

  then $f^\dagger$ is the formula $\varphi[\bot]$: {**false**} S {**false**}.
  Now **false** : {**false**} S {**false**} is an instance of the invariance axiom of the Invariant system, and using the consequence rule one derives easily the desired formula $f^\dagger$ from this instance.

- Process naming

  Assume the Hoare deduction ends as:

  $$\frac{H\vdash\forall\bar{g}[(\varphi_1)\ m_1\ (\psi_1)]\quad,\quad H\cup\{\forall\bar{g}[(\varphi_1)X_\beta(\psi_1)]\}\vdash(\varphi_2)\ m_2\ (\psi_2)}{H\vdash(\varphi_2)X_\beta=m_1\ \mathbf{in}\ m_2(\psi_2)}$$

  This is transformed in a straightforward way:

  $$\frac{H\vdash\forall\bar{g}[\psi[\bot]_1:\{\varphi[\top]_1\}\ m_1\ \{\psi[\top]_1\}]\quad H\cup\{\forall\bar{g}[\psi[\bot]_1:\{\varphi[\top]_1\}\ m_1\ \{\psi[\top]_1\}]\}\vdash\psi[\bot]_2:\{\varphi[\top]_2\}\ m_2\ \{\psi[\top]_2\}}{H\vdash\psi[\bot]_2:\{\varphi[\top]_2\}\ X_\beta=m_1\ \mathbf{in}\ m_2\ \{\psi[\top]_2\}}$$

  We have proven the compositional completeness of the Invariant system.

□

[AFR] K.R. Apt, N. Francez, W.P. de Roever,
A proofsystem for Communicating Sequential Processes.
TOPLAS 2 (3) 1980.

[Apt] K.R. Apt,
Ten years of Hoare's logic.
TOPLAS 3 (4), 1981.

[Arbib] M.A. Arbib, E.G. Manes,
Arrows, Structures, and Functors. The Categorical Imperative. Academic Press, Inc. N.Y. 1975.

[Bakker] J. de Bakker,
Mathematical theory of program correctness.
Prentice-Hall, 1980.

[BKP] H. Barringer, R. Kuiper and A. Pnueli,
Now you may compose temporal logic specifications.
Proc. of the 16th ACM Symposium on Theory of Computing, Washington, 1984.

[BHR] S.D. Brookes, C.A.R. Hoare, A.W. Roscoe,
A theory of Communicating Sequential Processes.
JACM 31(7), 1984.

[BR] S.D. Brookes, A.W. Roscoe,
An improved failures model for Communicating Sequential Processes.
Proc. of NSF-SERC Seminar on Concurrency, LNCS 85, Springer Verlag.

[BrBö] A. de Bruin, A.P.W. Böhm,
The denotational semantics of dynamic networks of processes.
report RUU-CS-82-13, University of Utrecht.

[Bruin] A. de Bruin,
Experiments with continuation semantics: jumps, backtracking, dynamic networks.
Ph.D. Thesis, Vrije Universiteit Amsterdam, 1986.

[Cla] E.M. Clarke,
Programming constructs for which it is impossible to obtain good Hoare like axiom systems.
JACM 26 (1) jan 1979.

[Cla2] E.M. Clarke,
The characterization problem for Hoare logics.

in: Mathematical logic and programming languages, Prentice-Hall, 1985.

[Dijk] E.W. Dijkstra,
Programming considered as a human activity.
Proc. of the 1965 IFIP Congress, Amsterdam, North-Holland 1965.

[Dijk2] E.W. Dijkstra,
A short introduction to the art of programming.
EWD 316, aug 1972.

[FHLR] N. Francez, C.A.R. Hoare, D. Lehman, W.P. de Roever,
Semantics of nondeterminism, concurrency and communication,
JCSS 19, 1979.

[Floyd] R.W. Floyd,
Assigning meaning to programs.
in Mathematical Aspects of Computer Science, AMS, 1967.

[FLP] N. Francez, D.Lehman, and A. Pnueli,
A linear History Semantics for Distributed languages.
Proc. 21st IEEE Symposium on Foundations of Computer Science,
Syracuse, N.Y. 1980.

[GerRo] R. Gerth and W.P. de Roever,
Proving monitors revisited:
A first step towards verifying object oriented systems.
Fundamenta Informaticae IX, Noth-Holland, 1986.

[Gerth] R, Gerth,
Transition logic: how to reason about temporal properties in a compositional way.
Proc. STOC, 1984.

[Gödel] K. Gödel,
Uber Formal Unentscheidbare Satze der Principia Mathematica und Verwandter Systeme, I.
Monatshefte fur Mathematik und Physik, 38, 1931.

[Gorelick] G.A. Gorelick,
A complete axiomatic system for proving assertions about recursive and non-recursive programs.
Technical Report 75, University of Toronto, 1975.

[Harel] D. Harel,
First-order dynamic logic.
LNCS 68, Springer, 1979.

[Hoare] C.A.R. Hoare,
An axiomatic basis for computer programming.
CACM 12, 1969.

[Hoare2] C.A.R. Hoare,
Communicating Sequential Processes.
CACM, 21(8), 1978.

[Hoare3] C.A.R. Hoare,
Communicating Sequential Processes.
Prentice-Hall, 1985.

[Hoare4] C.A.R. Hoare,
Procedures and parameters: An axiomatic approach.
Symp. on Semantics of Algorithmic Languages, LNM 188, 1971.

[Hailpern] B.T. Hailpern,
Specifying and verifying protocols represented as abstract programs.
IBM Research Report RC 8674, 1981.

[HailOw] B. Hailpern, S. Owicki,
Modular verification of computer communication protocols.
IBM Research Report RC8726, 1981.

[HeHo] E.C.R. Hehner, C.A.R. Hoare,
A more complete model of Communicating Processes.
TCS 26, 1983.

[HooRo] J. Hooman, W.P. de Roever,
The Quest goes on: A survey of proof systems for partial correctness of CSP.
LNCS 227,

[HooZw] J. Hooman, J. Zwiers,
Combining sequential and parallel composition: unexpected implications for compositional proof systems.
Unpublished manuscript, 1987.

[Hop] J.E. Hopcroft and J.D. Ullman,
Introduction to automata theory, languages and computation.
Addison-Wesley, 1979.

[JanEmBo] T.M.V. Janssen, P. van Emde Boas,
Some observations on compositional semantics.
Proc. of workshop on Logics of Programs, LNCS 131, 1981.

[Jones] C.B. Jones,
Software development, A rigorous approach.
Prentice-Hall, 1980.

[Jonkers] H.B.M. Jonkers,
Informal Description of the Design Language COLD-K,
Technical Report,ESPRIT project 432, Doc.Nr. METEOR/t7/PRLE/2 (1986).

[Jon] B. Jonsson,
A model and proof system for asynchroneous networks.
Proc. of the 4th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, 1985.

[JoPa] M. Joseph, P.Pandya,
Specification and verification of total correctness of distributed programs.
Research Report RR96, University of Warwick, 1987.

[KaQu] G. Kahn, D. McQueen,
Coroutines and networks of parallel processes.
Proc. of IFIP conference 1977.

[Kleene] S.C. Kleene,
Introduction to Meta-mathematics,
D. Van Nostrand, Inc. Princeton, N.j.

[Lam1] L. Lamport,
The Hoare Logic of concurrent programs,
Acta Informatica 14 1980.

[Lam2] L. Lamport,
Specifying Concurrent Program Modules.
ACM TOPLAS 6 (2) 1983.

[MC] J. Misra, and M. Chandy,
Proofs of Networks of Processes.
IEEE SE 7 (4) 1981.

[Mil] R. Milner,
A calculus of Communicating Sytems.
Springer LNCS 92 1980.

[Naur] P. Naur,
Proofs of algorithms by general snapshots.
BIT 6, 1966.

[Nguyen] Van Nguyen,
The incompleteness of Misra and Chandy's proof systems for networks of processes.
IPL 21, 1985.

[NDO] Van Nguyen, A. Demers, S. Owicki,
A model and temporal proof system for networks of processes.
Distributed Computing 1(1) 1986.

[OG] S. Owicki and D. Gries,
An aximatic proof technique for parallel programs.
Acta Informatica 6 1976.

[OH] E.R. Olderog and C.A.R. Hoare,
Specification oriented semantics for communicating processes.
Proc. of the 10th ICALP conference 1983, LNCS 154.

[Olderog1] E.R. Olderog,
Specification oriented programming in TCSP.
INRIA Advanced Nato Study Institute on Logics and Models for Verification and Specification of Concurrent Sytems 1984.

[Olderog2] E.R. Olderog,
Process Theory: Semantics, Specification and Verification.
ESPRIT/LPC Advanced School on Current Trnds in Concurrency 1985.

[Olderog3] E.R. Olderog,
On the notion of expressiveness and the rule of adaptation.
TCS 24, 1983.

[Pnu] A. Pnueli,
Compositional verification of concurrent programs using temporal logic.

INRIA Advanced Nato Study Institute on Logics and Models for Verification and Specification of Concurrent Sytems 1984.

[Pnu2] A. Pnueli,
Specification and development of reactive systems.
Proc. of IFIP conference, North-Holland, 1986.

[Pnu3] A. Pnueli,
In transition from global to modular reasoning about programs.
Proc. of conf. on Logics and Models of Concurrent Systems, ed. K.R. Apt, Springer-Verlag.

[Pratt] V.R. Pratt,
A definition of "process".
Stanford University report, 1981.

[Rem] M. Rem,
The VLSI chalenge: complexity bridling.
VLSI '81 (Very Large Scale Integration), ed. J.B. Gray, Academic Press, 1981.

[Roever] W.P. de Roever,
The quest for compositionality - A survey of proof systems for concurrency, Part I.
Proc. of IFIP working group "The role of abstract models in Computer Science", ed. E.J. Neuhold, North-Holland, 1985.

[Sanchis] L.E. Sanchis,
Reflexive Domains.
in: To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, ed. by J.P. Seldin & J.R. Hindley. Academic Press 1980.

[Scott] D. Scott,
Outline of a Mathematical Theory of Computation.
4th Annual Princeton Conf. Information Sciences & Systems.

[ScoBa] D.S. Scott, J.W. de Bakker,
A theory of programs.
unpublished seminar notes, IBM, Vienna, 1969.

[Snep] J.L.A. van de Snepscheut,
Trace Theory and VLSI Disign.
LNCS 200, 1985.

[SouDa] N. Soundararajan, O.J. Dahl,
Partial correctness semantics for Communicating Sequential Processes.
Research Report 66, Institute for Informatics, University of Oslo, 1982.

[Soun] N.Soundararajan,
Axiomatic semantics of Communicating Sequential Processes.
ACM TOPLAS 6 (4), 1984.

[Turing] A. Turing,
On checking a large routine.
Report of a conference on high-speed automatic calculating machines, University Mathematical Laboratory, Cambridge, 1949.

[WGS] J. Widom, D. Gries, F.B. Schneider,
Completeness and incompleteness of trace-based network proof systems.
Proc. of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 1987.

[Widom] J. Widom,
Trace-based network proof systems: Expressiveness and Completeness.
Ph.D. Thesis 87-833, Cornell University, Ithaca, NewYork, 1987.

[Wijn] A. van Wijngaarden,
Generalized ALGOL.
Annual review in automatic programming, vol 3. 1963.

[Yon] A. Yonezawa,
Specification and verification techniques for parallel programs based on message passing semantics.
Ph.D. Thesis MIT/LCS/TR-191, Massachusetts Institute of Technology, 1977.

[ZBR] J. Zwiers, A. de Bruin, W.P. de Roever,
A proof system for partial correctness of Dynamic Networks of Processes.
Proc. of the Conference on Logics of Programs 1983, LNCS 164, 1984.

[ZH] Zhou Chao-Chen, C.A.R. Hoare,
Partial correctness of CSP.
IEEE Int. Conf. on Distributed Computer Sytems 1981.

[ZRE] J. Zwiers, W.P. de Roever and P. van Emde Boas,
Compositionality and Concurrent Networks: Soundness and Completeness of a Proofsystem.
Proc. of ICALP '85, LNCS 194, 1985.

[ZRE2] J. Zwiers, W.P. de Roever and P. van Emde Boas,
Compositionality and Concurrent Networks: Soundness and Completeness of a Proofsystem.
University of Nijmegen, report no. 57, dec 1984.

SAMENVATTING

Het construeren van correct werkende programmatuur is een van de belangrijkste, slechts gedeeltelijk opgeloste, problemen in de informatica. Sinds de constructie van de eerste electronische rekenapparatuur in de veertiger jaren van deze eeuw is de capaciteit van deze apparatuur even dramatisch toegenomen als de prijs daarvoor is afgenomen. Helaas heeft de ontwikkeling van programmatuur daar geen gelijke tred mee weten te houden. Sinds 1968 is men zelfs gaan spreken van de "software crisis". Een van de oorzaken daarvoor moet gezocht worden in de (fundamentele) onmogelijkheid om programmatuur afdoende uit te testen. Immers, het aantal mogelijke inputs met de daarop volgende responses van een programma is dermate groot dat dit als uitgesloten wordt beschouwd. We citeren E. Dijkstra: "Het testen van programma's kan gebruikt worden om fouten aan te tonen, maar niet om de afwezigheid daarvan aan te tonen". Als alternatief voor het testen is het verifiëren van de correctheid van programma's met behulp van wiskundige middelen ingevoerd. Het principe bestaat daaruit dat eerst in een zgn. specificatietaal een (wiskundige) beschrijving wordt gegeven van het gewenste gedrag van een te construeren programma, waarna tijdens de constructie, weer met wiskundige middelen, geverifiëerd wordt dat het programma ook daadwerkelijk voldoet aan deze beschrijving. Dit proefschrift beschrijft en analyseert een drietal van dergelijke specificatietalen en de daarbij behorende verificatiemethoden. De aandacht gaat daarbij uit naar de specificatie, constructie en verificatie van parallelle, d.w.z. gelijktijdig opererende, processen. Processen kunnen elkaar daarbij beïnvloeden door onderlinge synchronisatie en door communicatie van gegevens. Bovendien kunnen er tijdens het executeren van een programma nieuwe processen bijkomen en weer verdwijnen.

Een belangrijk middel om niet ten onder te gaan in de complexiteit van omvangrijke programmatuur is het "verdeel en heers" principe. Daartoe wordt een programma onderverdeeld in een aantal onafhankelijk van elkaar te construeren modulen. Een specificatie van het gedrag van zo'n module dient als een soort contract tussen de gebruiker en de implementator van de module. De gebruiker construeert zijn programma met behulp van deze modulen en verifiëert de programmaspecificatie op basis van de specificaties van de modulen. De interne constructie van de modulen hoeft en wenst hij niet te kennen. Een module moet daarbij opgevat worden als een programma dat zelf weer verder onderverdeeld kan worden in nog eenvoudiger modulen. De term compositioneel verwijst in deze context naar de mogelijkheid om de correctheid van een uit modulen samengesteld programma te verifiëren op grond van specificaties van deze modulen, maar zonder kennis van de

inwendige constructie daarvan. De specificatie- en verificatiemethoden die in dit proefschrift bestudeerd worden, onderscheiden zich doordat zij ontworpen zijn om een dergelijke compositionele specificatie en verificatie mogelijk te maken. De kwaliteiten van de diverse specificatie en verificatiemethoden worden in dit licht met elkaar vergeleken.

Hoofdstuk 1 bevat een overzicht, en een informele inleiding in de verificatie van **TNP** programma's. Verder worden de begrippen "compositionele volledigheid", "adaptatie volledigheid" en "modulaire volledigheid" ingevoerd.

Hoofdstuk 2 bevat een definitie van de grammatica van de talen **DNP** ("Dynamic Networks of Processes"), en **TNP** ("Theoretical Networks of Processes").

Hoofdstuk 3 behandelt de betekenis van **TNP**, ook wel (denotationele) semantiek genoemd.

Hoofdstuk 4 voert de verschillende talen voor het specificeren van **TNP** programma's in. Dit zijn de assertietaal, de klassen van SAT, Hoare en Invarianten specificaties, en tenslotte een mengvorm tussen **TNP** en de assertietaal, genaamd "Mixed terms".

Hoofdstuk 5 introduceert drie bewijssystemen die gebruikt kunnen worden voor de verificatie van **TNP** programma's, te weten het SAT systeem, het Hoare systeem en het Invarianten systeem. De soundness, d.w.z. geldigheid, van deze systemen wordt bewezen.

Hoofdstuk 6 en 7 beschouwen de volledigheid aan van de drie bewijssytemen. Het SAT systeem en het Hoare systeem worden beiden zowel compositioneel, adaptatie als modulair volledig bewezen. Voor het Invarianten systeem wordt compositionele volledigheid aangetoond.

CURRICULUM VITAE

Naam:	Job Zwiers.
Geboren:	31 mei 1956 te Leiden.
1974:	Examen Atheneum B, Rijksscholengemeenschap Schoonoord, Zeist.
1978:	Kandidaatsexamen Wis- en Natuurkunde, Rijksuniversiteit Utrecht.
1984:	Doctoraalexamen Wiskunde, Rijksuniversiteit Utrecht.
1983-1986:	Wetenschappelijk medewerker, Katholieke Universiteit Nijmegen.
1986-heden:	Wetenschappelijk medewerker Philips Research, Waalre.

CURRENT ADDRESS:

Philips Research Laboratories,
P.O. Box 80.000,
5600 JA Eindhoven,
The Netherlands

Stellingen

behorend bij het proefschrift
Compositionality, Concurrency and Partial correctness
van
Job Zwiers

Stelling 1

Het in dit proefschrift ingevoerde "generalized state" begrip maakt het mogelijk de specificatietaal COLD uit te breiden naar parallel communicerende systemen, waarbij het bestaande COLD (voor sequentiële systemen) behouden blijft.

L.M.G. Feijs, H.B.M. Jonkers, C.P.J. Koymans, G.R. Renardel de Lavalette,
"Formal Definition of the Design Language COLD-K", Philips Research Technical Note 234/87, ook als ESPRIT rapport METEOR/t7/PRLE/7.

Stelling 2

In "Trace-Based Network Proof Systems: Expressiveness and Completeness" door J. Widom wordt de specificatietaal STL ingevoerd. Een STL specificatie wordt "precise" genoemd voor een gegeven process wanneer, informeel gesproken, ieder door de specificatie toegelaten communicatie gedrag ook wordt toegelaten door het process en vice versa. Het begrip speelt een belangrijke rol in het volledigheids resultaat van Widom, waarin wordt aangetoond dat een bepaalde parallelle compositie regel, verwant aan de overeenkomstige regel van het SAT systeem van dit proefschrift, preciseness behoudt. Er zijn echter processen waarvoor geen "precise" STL specificatie bestaat. Dit volgt uit het feit dat er in STL geen onderscheid gemmakt kan worden tussen de volgende twee processes:

$(c; d \text{ or } d)$ en $(c; d \text{ or } d; c \text{ or } d)$.

J. Widom, "Trace-based network proof systems: Expressiveness and Completeness",
Ph.D. Thesis 87-833, Cornell University, Ithaca, NewYork, 1987.

Stelling 3

In hoofdstuk 5 van "Mathematical Theory of Program Correctness" van J. de Bakker wordt een Hoare logica voor recursieve sequentiële programma's zonder parameter mechanisme ingevoerd. Om een relatief volledig bewijssyteem te verkrijgen blijkt een relatief groot aantal minder aantrekkelijke adaptatieregels te moeten worden toegevoegd, zoals bijvoorbeeld de verschillende substitutie regels.

Een aanmerkelijk eenvoudiger en eleganter relatief volledig systeem kan echter worden verkregen door niet uit te gaan van Hoare logica maar van een formalisme analoog aan het SAT systeem van dit proefschrift. Er is dan natuurlijk geen specificatie van communicatie gedrag meer nodig, en daarmee vervalt eveneens de noodzaak om het gedrag van nog niet getermineerde berekeningen vast te leggen. Slechts een drietal eenvoudige adaptatieregels is dan nodig, te weten een invariantie axioma, een conjunctie- en een consequence regel. Wanneer slechts gestreefd wordt naar relatieve volledigheid, en afgezien wordt van modulaire volledigheid zoals gedefiniëerd in hoofdstuk 1 van dit proefschrift, dan kan wat de adaptatie regels betreft zelfs volstaan worden met uitsluitend een consequence regel.

J. de Bakker, "Mathematical Theory of Program Correctness", Prentice-Hall.

Stelling 4

In K. Apt's "Ten years of Hoare's Logic" wordt een bewijsregel gegeven voor ALGOL type declaratie van locale variabelen die *niet* compositioneel is, omdat in de premisse van deze regel een substitutie wordt uitgevoerd in de syntactische component S van de constructie **begin** *new* x; S **end**. Apt stelt daarbij dat, wanneer recursieve procedures zijn toegelaten, een regel, geïntroduceerd door Lauer,

waarbij niet in de programmatekst maar in de pre- en postcondities van Hoare formules gesubstitueerd wordt correspondeert met een *dynamic scope* semantiek, in plaats van de gebruikelijke semantiek met de *static scope* aanname.

Echter, in een opzet die uitgaat van *natuurlijke deductie* kan op eenvoudige wijze een compositionele regel gegeven worden, waarbij de static scope aanname behouden blijft. Deze regel heeft de volgende vorm:

$$\frac{\tilde{H} \vdash \{p[y/x]\}\ S\ \{q[y/x]\}}{H \vdash \{p\}\ \textbf{begin}\ \textit{new}\ x\ ;\ S\ \textbf{end}\ \{q\}}$$

Hierbij is H een verzameling hypothesen in de vorm van Hoare formules $\{pre_i\}\ P_i\ \{post_i\}$, waarbij P_i een procedure naam is. $\tilde{H}$ is uit H verkregen door in alle daarin voorkomende pre- en postcondities de variabele y voor x te substitueren. De variabele y moet noch vrij voorkomen in de gebruikte pre- en postcondities pre_i en $post_i$, noch in S.

K.R. Apt, "Ten years of Hoare's logic", TOPLAS 3, (4), 1981.
P.E. Lauer, "Consistent formal theories of the semantics of programming languages",
Technical Report TR.25.121, IBM Laboratory Vienna, 1971.

Stelling 5

In "Ten years of Hoare's logic" (ref. zie boven) bespreekt Apt verschillende mechanismen voor parameter overdracht voor procedures, waaronder het call-by-value/result mechanisme. Apt merkt op dat corresponderende bewijsregel het nadeel heeft dat voor iedere procedure call een afzonderlijk bewijs nodig is voor de premisse die betrekking heeft op de body van de desbetreffende procedure. Daarom wordt voorgesteld om een parametersubstitutie regel van Cook, oorspronkelijk bedoeld voor het call-by-name mechanisme, te gebruiken waarbij dan wel de nodige restricties die inherent zijn aan deze regel moeten worden geaccepteerd. Deze restricties zijn de volgende. Als procedure $P(\bar{x},\bar{v})$ met formele value/result parameters $\bar{x}$ en formele value parameters $\bar{v}$ gedeclareerd is met procedure body S_0, dan moet voor iedere call van de vorm $P(\bar{u},\bar{e})$ voldaan zijn aan de volgende voorwaarden: $\bar{u}$ is een lijst van *verschillende* variabelen, $\bar{e}$ is een lijst van expressies die geen van de $\bar{u}$ variabelen bevatten, en geen variabele in $\bar{u}$ of $\bar{e}$ anders dan een formele parameter mag voorkomen in de body S_0. Bovendien mag in S_0 niet geassigneerd worden aan de formele value parameters $\bar{v}$. De regel wordt dan de volgende. Mits $var(post) \cap \{\bar{x},\bar{v}\} \subseteq \{\bar{u}\}$:

$$\frac{\{pre_0\}\ P(\bar{x},\bar{v})\ \{post_0\}}{\{pre_0[\bar{u}/\bar{x},\bar{e}/\bar{v}]\}\ P(\bar{u},\bar{e})\ \{post[\bar{u}/\bar{x},\bar{e}/\bar{v}]\}}$$

Dit resultaat kan als volgt worden verbeterd. Voor het call-by-value/result mechanisme kunnen al de genoemde restricties achterwege blijven wanneer de volgende regel wordt gebruikt:

Als $var(post) \cap \{\bar{x},\bar{v}\} \subseteq \{\bar{u}\}$ dan geldt:

$$\frac{\{pre_0\}\ P(\bar{x},\bar{v})\ \{post_0\} \quad , \quad pre \rightarrow pre_0[\bar{u}/\bar{x},\bar{e}/\bar{v}]\ ,\ post_0 \rightarrow post[\bar{x}/\bar{u}]}{\{pre\}\ P(\bar{u},\bar{e})\ \{post\}}$$

Wanneer *toch* aan alle restricties is voldaan kan Cook's regel afgeleid worden uit de hier voorgestelde regel.

S.A. Cook, "Soundness and completeness of an axiom system for program verification",
SIAM Journal on Computing, vol. 7, no 1, 1978.

Stelling 6

De correctheid van programmatransformaties gebaseerd op het bestaan van een simulatierelatie α tussen een "abstract" programma S_A en een implementerend "concreet" programma S_C komt neer op het aantonen van een inclusierelatie van de vorm $\alpha^{-1}; S_C; \alpha \subseteq S_A$. Dergelijke transformaties vormen de basis van onder meer de methoden voor datastructuren implementatie van [Hoare], [Hoare&Jifeng] en [Reynolds], en van de reïficatie techniek zoals gebruikt voor de VDM methode [Jones]. Als het abstracte programma gespecificeerd is door middel van een Hoare specificatie $\{pre\}\ S_A\ \{post\}$ dan is aan bovenstaande inclusie voldaan wanneer het implementerende programma voldoet aan $\{\langle\alpha\rangle pre\}\ S_C\ \{[\alpha]post\}$.

Een Hoare formule $\{pre\}\ S\ \{post\}$ wordt vaak geïnterpreteerd als $\forall\bar{v}[\{pre\}\ S\ \{post\}]$ waarbij de lijst $\bar{v}$ bestaat uit alle vrije logische variabelen van de formule.

Echter, het gebruik van deze impliciete universele quantificatie is ongewenst omdat in dat geval bovenstaande verificatiemethode voor programmatransformatie ***onvolledig*** blijkt te zijn. Wanneer geen impliciete quantificatie wordt aangenomen, dan kan volledigheid worden aangetoond.

[Hoare] C.A.R. Hoare, "Proof of Correctness of Data Representations",
Acta Informatica, vol 1, 1972.
[Hoare&Jifeng] C.A.R. Hoare, Jifeng He, J.W. Sanders, "Data Refinement Refined",
Proc. 1st ESOP, LNCS 213, 1986.
[Jones] C.B. Jones, "Systematic software development using VDM", Prentice-Hall.
[Reynolds] J. Reynolds, "The craft of programming", Prentice-Hall.

Stelling 7

Programmaspecificaties gebaseerd op assumption-commitment paren zijn geïntroduceerd door Misra en Chandy. Het "generalized state" concept zoals geïntroduceerd in dit proefschrift kan in de Misra/Chandy aanpak geïncorporeerd worden, en leidt dan tot formules van de volgende vorm:

$$(A, C)\ :\ \{pre\}\ S\ \{post\}.$$

Hierin zijn A en C een assumption en commitment betreffende het communicatie gedrag en zijn *pre* en *post* een pre- en postconditie betreffende zowel het communicatie gedrag als het state-transformer gedrag. De interpretatie van deze formule is:

Als voor de initiële communicatiegeschiedenis en programmatoestand waarin process S start voldaan is aan de preconditie *pre*, dan is de commitment C gegarandeerd op ieder moment tijdens executie mits op alle voorafgaande momenten de assumptie A niet geschonden is, en als het process termineert en de assumptie A is nooit geschonden tijdens de executie, dan voldoen de finale communicatie geschiedenis en toestand aan de postconditie *post*.

Het is echter niet noodzakelijk om Misra/Chandy type formules als een nieuwe vorm van specificaties te beschouwen, omdat dergelijke formules al gerepresenteerd kunnen worden door "Invariant" formules zoals ingevoerd in hoofdstuk 4 van dit proefschrift. De representerende formule is:

$$\forall t_0[\mathit{Past}(\mathbf{c}, t_0, A) \rightarrow C\ :\ \{pre \wedge t_0|\mathbf{c} = h|\mathbf{c}\}\ S\ \{\mathit{Kern}(\mathbf{c}, t_0, A) \rightarrow post\}],$$

waarbij $\mathit{Past}(\mathbf{c}, t_0, A)$ staat voor de assertie $\forall t(t_0|\mathbf{c} \leq t|\mathbf{c} < h|\mathbf{c} \rightarrow A[t/h])$, en $\mathit{Kern}(\mathbf{c}, t_0, A)$ voor $A \wedge \mathit{Past}(\mathbf{c}, t_0, A)$.

J. Misra, M. Chandy, "Proofs of Networks of Processes", IEEE SE 7 (4), 1981.

Stelling 8

De aantrekkelijke kant van Misra/Chandy type specificaties wordt vooral gevormd door onderstaande bewijsregel voor parallelle compositie van processen. In [ZBR] is beargumenteerd dat daarmee bepaalde bewijzen met inductie naar de lengte van de communicatie geschiedenis vereenvoudigd worden op een manier die vergelijkbaar is met de vereenvoudiging die de welbekende "while rule" voor iteratie betekent ten op zichte van een expliciet inductie bewijs naar de lengte van de berekening van een programma. Om deze reden is onderstaande regel superieur aan bewijsregels

voor parallelle compositie die in essentie gebaseerd zijn op het nemen van een ***conjunctie*** van de specificaties voor de delen S_1 en S_2 als specificatie voor de parallelle compositie $S_1 \parallel S_2$, zoals ingevoerd door Zhou Chao-Chen en Hoare [ZH], en zoals bestudeerd in dit proefschrift. De regel is echter een ***afgeleide*** regel voor het Invariant system uit hoofdstuk 5, zie [ZRE]. De regel is geldig voor **TNP** processes, maar niet voor willekeurige "mixed terms".

Mits $abase(A_i, C_i, p_i, q_i) \cap base(S_{3-i}) \subseteq base(S_i)$, voor $i = 1, 2$:

$$\frac{(A_1, C_1) : \{p_1\}\ S_1\ \{q_1\}\ ,\ (A_2, C_2) : \{p_2\}\ S_2\ \{q_2\} \qquad \underline{\forall}(A \wedge C_1 \rightarrow A_2)\ ,\ \underline{\forall}(A \wedge C_2 \rightarrow A_1)}{(A, C_1 \wedge C_2) : \{p_1 \wedge p_2\}\ S_1 \parallel S_2\ \{q_1 \wedge q_2\}}\ \textbf{(MC)}$$

[ZBR] J. Zwiers, A. de Bruin, W.P. de Roever, "A proof system for partial correctness of Dynamic Networks of Processes", Proc. of the Conference on Logics of Programs 1983, Springer Lecture Notes in Computer Science 164, 1984.

[ZH] Zhou Chao-Chen, C.A.R. Hoare, "Partial correctness of CSP", IEEE Int. Conf. on Distributed Computer Sytems 1981.

[ZRE] J. Zwiers, W.P. de Roever en P. van Emde Boas, "Compositionality and Concurrent Networks: Soundness and Completeness of a Proofsystem", in de versie van het rapport No.57, sectie Informatica Katholieke Universiteit Nijmegen.

Stelling 9

Van Nguyen merkt in [Nguyen] op dat het bewijssysteem zoals gegeven door Misra en Chandy onvolledig is. Door het invoeren van onderstaand axioma kan deze onvolledigheid worden opgeheven.

$$(A, Past(\mathbf{c}, t_0, A)) : \{t_0|\mathbf{c} = h|\mathbf{c}\}\ S\ \{\mathbf{true}\}.$$

Hierin is $\mathbf{c}$ de verzameling channels vrij voorkomend in A, en is t_0 een niet in A vrij voorkomende trace variabele.

Een relatief volledig bewijssyteem kan worden verkregen door modificatie van het Invariant system op de wijze zoals aangegeven in [ZRE].

[Nguyen] Van Nguyen, "The incompleteness of Misra and Chandy's proof systems for networks of processes", Information Processing Letters 21, 1985.

[ZRE] Zie boven.

Stelling 10

Uit de handleiding van het LaTeX "Document Preparation System", gebruikt voor o.m. het zetten van deze stelling, blijkt niet dat de auteur een expert op het terrein van programma specificatie is.

Leslie Lamport, "LaTeX User's Guide & Reference Manual", Addison-Wesley, 1986.