A proof system for the language POOL

A proof system for the language POOL

by

F.S. de Boer

90/14

October , 1990

A proof system for the language Pool

Frank de Boer

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author or the editor.

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

Editors: prof.dr.M.Rem
prof.dr.K.M. van Hee

Contents

1 Introduction

The goal of this paper is to develop a formal system for reasoning about the correctness of a certain class of parallel programs. We shall consider programs written in the language POOL, a parallel object-oriented language [Am]. POOL makes use of the structuring mechanisms of object-oriented programming, integrated with the concepts for expressing concurrency: processes and rendez-vous.

A program of the language POOL describes the behaviour of a whole system in terms of its constituents, *objects*. These objects have the following important properties: First of all, each object has an independent activity of its own: a process that proceeds in parallel with all the other objects in the system. Second, new objects can be created at any point in the program. The identity of such a new object is at first only known to itself and its creator, but from there it can be passed on to other objects in the system. Note that this also means that the number of processes executing in parallel may increase during the evolution of the system.

Objects possess some internal data, which they store in *variables*. The value of a variable is either an element of a predefined data type (Int or Bool), or it is a *reference* to another object. The variables of one object are not accessible to other objects. The objects can interact only by sending *messages*. A message is transferred synchronously from the sender to the receiver, i.e., sending and receiving a message take place at the same time. A message contains a *method* name (procedures are called methods in POOL) and a sequence of actual parameters specified by the sender of the message. While the receiver of the message is executing the method the execution of the sender is suspended. When the result of the execution of the method has been received by the sender of the message it resumes its activity.

Thus we see that a system described by a program in the language P consists of a dynamically evolving collection of objects, which are all executing in parallel, and which know each other by maintaining and passing around references. This means that also the communication structure of the processes is completely dynamic, without any regular structure imposed on it a priori. This is to be contrasted with the static structure (a fixed number of processes, communicating with statically determined partners) in [AFR] and the tree-like structure in [ZREB].

One of the main proof theoretical problems of such an object-oriented language is how to reason about dynamically evolving *pointer structures*. We want to reason about these structures on an abstraction level that is *at least as high as that of the programming language*. In more detail, this means the following:

- The only operations on "pointers" (references to objects) are

  - testing for equality
  - dereferencing (looking at the value of an instance variable of the referenced object)

- In a given state of the system, it is only possible to mention the objects that exist in that state. Objects that do not (yet) exist never play a role.

Strictly speaking, direct dereferencing is not even allowed in the programming language, because each object only has access to its own instance variables. However, for the time being we allow it in the assertion language. Otherwise, even more advanced techniques would be necessary to reason about the correctness of a program.

The above restrictions have quite severe consequences for the proof system. The limited set of operations on pointers implies that first-order logic is too weak to express some interesting properties of pointer structures. Therefore we have to extend our assertion language to make it more expressive. We will do so by allowing the assertion language to reason about *finite sequences* of objects. Furthermore we have to define some special substitution operations to model aliasing and the creation of new objects.

To deal with parallelism, the proof theory we shall develop uses the concepts of *cooperation test*, *global invariant*, *bracketed section* and *auxiliary variables*. These concepts have been developed in the proof theory of CSP [AFR], and have been applied to quite a variety of concurrent programming languages [HR]. In fact our proof method generalizes the application of these concepts to the language ADA [GR]. The main difference between the ADA-rendezvous and the rendezvous mechanism of POOL consists in that in POOL we have *no* static bound to the recursion depth of the rendezvous mechanism whereas in ADA there is. A consequence of this is that the proof method for ADA is *incomplete* when applied to the language POOL. Here completeness means that every true property of a program that can be expressed in the assertion language used can also be proved formally in the proof system. This incompleteness can be resolved by some additional reasoning mechanism which essentially formalizes reasoning about *invariance* properties of a rendezvous.

Described very briefly this proof method applied to our language consists of the following elements:

- A *local* stage. Here we deal with all statements that do not involve message passing or object creation. These statements are proved correct with respect to pre- and postconditions formulated in a *local assertion language*, which only talks about the current object in isolation. At this stage, we use the *assumption/commitment* formalism [HR] to describe the behaviour of the rendezvous and creation statements. The assertions describing the assumptions and commitments are formulated in the local assertion language. The assumptions about these statements then will be verified in the next stage.

- An *intermediate* stage. In this stage the above assumptions about rendezvous and creation statements are verified. Here a *global assertion language*, which reasons about all the objects in the system, is used. For each creation statement and for each pair of possibly communicating rendezvous statements it is verified that the specification used in the local proof system is consistent with the global behaviour.
- A *global* stage. Here some properties of the system as a whole can be derived from a kind of standard specification that arises from the intermediate stage. Again the global assertion language is used.

We have proved the proof system is sound and complete with respect to a formal semantics. Soundness means that everything that can be proved using the proof system is indeed true in the semantics. Due to the abstraction level of the assertion language we had to modify considerably the standard techniques for proving completeness ([Ap]). In the completeness proof we combine the techniques for proving completeness of the proof system for recursive procedures of sequential languages ([Ap2]) based upon the expressibility of the strongest postcondition with the techniques of [Ap] developed for CSP.

Our paper is organized as follows: In the following section we describe the programming language POOL. In section 3 we define two assertion languages, one to describe the internal data of an object and one to describe a complete system of objects. Also in section 3 we show how to specify the behaviour of an object and a system of objects. In section 4 we describe the proof system. For the formal semantics of the programming language, the assertion languages, and the specification languages we refer to [Bo], where also the soundness and completeness of the proof system is proved.

2 The programming language

In this section we define a abstract version of the programming language POOL of which we shall study the proof theory.

The most important concept is the concept of an *object*. This is an entity containing data and procedures (*methods*) acting on these data. Furthermore, every object has an internal activity of its own. The data are stored in *variables*, which come in two kinds: *instance variables*, whose lifetime is the same as that of the object they belong to, and *temporary variables*, which are local to a method and last as long as the method is active. Variables can contain references to other objects in the system (or even the object under consideration itself). The object a variable refers to (its *value*) can be changed by an *assignment*. The value of a variable can also be nil, which means that it refers to no object at all.

The variables of an object cannot be accessed directly by other objects. The only way for objects to interact is by sending *messages* to each other. If an object sends a message, it specifies the receiver, a method name, and possibly some parameter objects. Then control is transferred from the sender object to the receiver. This receiver then executes the specified method, using the parameters in the message. Note that this method can, of course, access the instance variables of the receiver. The method returns a result, an object, which is sent back to the sender. Then control is transferred back to the sender which resumes its activities, possibly using this result object.

The sender of a message is *blocked* until the result comes back, that is, it cannot answer any message while it still has an outstanding message of its own. Therefore, when an object sends a message to itself (directly or indirectly) this will lead to abnormal termination of the program. This is an important difference with some other object-oriented languages, like Smalltalk-80 [Go].

Objects are grouped into *classes*. Objects in one class (the *instances* of the class) share the same methods and the same statement which specifies their internal activity, so in a certain sense they share the same behaviour. New instances of a given class can be created at any time. There are two standard classes, Int and Bool, of integers and booleans, respectively. They differ from the other classes in that their instances already exist at the beginning of the execution of the program and no new ones can be created. Moreover, some standard operations on these classes are defined.

A program essentially consists of a number of class definitions, together with a statement which specifies the behaviour of the *root-object*, the object which starts the execution. So initially only this object exists: the others still have to be created.

2.1 The syntax

In order to describe the language POOL, which is strongly typed, we use *typed* versions of all variables, expressions, etc. These types are indicated by subscripts or superscripts in this language description. Often, when this typing information is redundant, it is omitted. Of course, for a practical version of the language, a syntactical variant, in which the type of each variable is indicated by a *declaration*, is easier to use.

Assumption 2.1
We assume the following sets to be given:

- A set C of *class names*, with typical element c (this means that metavariables like $c, c', c_1, \ldots$ range over elements of the set C. We assume that $\mathsf{Int}, \mathsf{Bool} \notin C$ and define the set $C^+ = C \cup \{\mathsf{Int}, \mathsf{Bool}\}$ with typical element d.
- For each $c \in C$ and $d \in C^+$ we assume a set $IVar_d^c$ of *instance variables* of type d in class c. By this we mean that such a variable may occur in the definition of class c and that its contents will be an object of type d. The set $IVar_d^c$ will have as a typical element x_d^c. We define $IVar = \bigcup_{c,d} IVar_d^c$ and $IVar^c = \bigcup_d IVar_d^c$.
- For each $d \in C$ we assume a set $TVar_d$ of *temporary variables* of type d, with typical element u_d. We define $TVar = \bigcup_d TVar_d$ and $ITvar = IVar \cup TVar$.
- We shall let the metavariable n range over elements of $\mathbf{Z}$, the set of whole numbers.
- For each $c \in C$ and $d_0, \ldots, d_n \in C^+$ $(n \geq 0)$ we assume a set $MName_{d_0,\ldots,d_n}^c$ of *method names* of class c with result type d_0 and parameter types $d_1, \ldots, d_n$. The set $MName_{d_0,\ldots,d_n}^c$ will have $m_{d_0,\ldots,d_n}^c$ as a typical element.

Now we can specify the syntax of our language. We start with the expressions:

Definition 2.2
For any $c \in C$ and $d \in C^+$ we define the set Exp_d^c of *expressions* of type d in class c, with typical element e_d^c, as follows:

$$
\begin{array}{lcll}
e_d^c & ::= & x_d^c & \\
 & | & u_d & \\
 & | & \mathsf{nil}_d & \\
 & | & \mathsf{self} & \text{if } c = d \\
 & | & \mathsf{true} \mid \mathsf{false} & \text{if } d = \mathsf{Bool} \\
 & | & n & \text{if } d = \mathsf{Int} \\
 & | & e_{1d'}^c \doteq e_{2d'}^c & \text{if } d = \mathsf{Bool} \\
 & | & {e_1}^c + {e_2}^c & \text{if } d = \mathsf{Int} \\
 & & \vdots & \\
 & | & {e_1}^c < {e_2}^c & \text{if } d = \mathsf{Bool} \\
 & & \vdots &
\end{array}
$$

The intuitive meaning of these expressions will probably be clear. Note that in the language we put a dot over the equal sign ($\doteq$) to distinguish it from the equality sign we use in the meta-language.

Definition 2.3
The set $SExp_d^c$ of expressions with possible *side effect* of type d in class c, with typical element s_d^c, is defined as follows:

$$
\begin{array}{lcll}
s_d^c & ::= & e_d^c & \\
 & | & \mathsf{new}_d & \text{if } d \in C \ (d \neq \mathsf{Int}, \mathsf{Bool}) \\
 & | & e_{0c_0}^c \,!\, m^{c_0}_{d,d_1,\ldots,d_n}(e_{1d_1}^c, \ldots, e_{nd_n}^c) & (n \geq 1)\ e_1 = \mathsf{self}
\end{array}
$$

The first kind of side effect expression is a normal expression, which has no actual side effect, of course. The second kind is the creation of a new object. This new object will also be the value of the side effect expression. The third kind of side effect expression specifies that a message is to be sent to the object that results from e_0, with method name m and with arguments (the objects resulting from) $e_1, \ldots, e_n$. Note that we require the first argument to be the sender itself (so we have that $d_1 = c$). This requirement is not present in the language POOL. It is introduced for proof theoretical reasons only, however every POOL program can be transformed into an equivalent one satisfying this requirement.

Definition 2.4
The set $Stat^c$ of *statements* in class c, with typical element S^c, is defined by:

$$
\begin{array}{rcl}
S^c & ::= & x_d^c \leftarrow s_d^c \\
& | & u_d \leftarrow s_d^c \\
& | & \textsf{answer}(m_1, \ldots, m_n) \\
& | & S_1^c ; S_2^c \\
& | & \textsf{if } e^c \textsf{ then } S_1^c \textsf{ else } S_2^c \textsf{ fi} \\
& | & \textsf{while } e^c \textsf{ do } S^c \textsf{ od}
\end{array}
$$

Again, the intuitive meaning of these statements will probably be clear.

Definition 2.5
The set $MethDef^c_{d_0,\ldots,d_n}$ of *method definitions* of class c with result type d_0 and parameter types $d_1, \ldots, d_n$ (with typical element $\mu^c_{d_0,\ldots,d_n}$) is defined by:

$$\mu^c_{d_0,\ldots,d_n} ::= (u_{1d_1}, \ldots, u_{nd_n}) : S^c \uparrow e^c_{d_0}$$

Here we require that the u_{id_i} are all different and that none of them occurs at the left hand side of an assignment in S^c (and that $n \geq 1$).

When an object is sent a message, the method named in the message is invoked as follows: The variables $u_1, \ldots, u_n$ (the parameters of the methods) are given the values specified in the message, all other temporary variables are initialized to nil, and then the statement S is executed. After that the expression e is evaluated and its value, the result of the method, is sent back to the sender of the message, where it will be the value of the send-expression that sent the message.

Definition 2.6
The set $ClassDef^c_{m_1,\ldots,m_n}$ of *class definitions* of class c defining methods $m_1, \ldots, m_n$, with typical element $D^c_{m_1,\ldots,m_n}$, is defined by:

$$D^c_{m_1,\ldots,m_n} ::= \langle m_1{}^c_{\bar{d}_1} \Leftarrow \mu_1{}^c_{\bar{d}_1}, \ldots, m_n{}^c_{\bar{d}_n} \Leftarrow \mu_n{}^c_{\bar{d}_n} \rangle : S^c$$

where we require that all the method names are different (and $n \geq 0$) and $TVar(S^c) = \emptyset$. (Here $TVar(S^c)$ denotes the set of temporary variables occurring in S^c. Furthermore, $\bar{d}_i$, denotes a sequence of types.)

Definition 2.7
The set $Unit^{c_1,\ldots,c_n}_{m_1,\ldots,m_k}$ of *units* with classes $c_1, \ldots, c_n$ defining methods $m_1, \ldots, m_k$, with typical element $U^{c_1,\ldots,c_n}_{m_1,\ldots,m_k}$, is defined by:

$$U^{c_1,\ldots,c_n}_{m_1,\ldots,m_k} ::= D_{1\bar{m}_1}{}^{c_1}, \ldots, D_{n\bar{m}_n}{}^{c_n}$$

where $m_1, \ldots, m_k = \bar{m}_1, \ldots, \bar{m}_n$, that is, $m_1, \ldots, m_k$ results from concatenating the sequences of method names $\bar{m}_i$. We require that all the class names are different.

Definition 2.8
Finally, the set $Prog^c$ of *programs* in class c, with typical element ρ^c, is defined by:

$$\rho^c ::= \langle U^{c_1,\ldots,c_n}_{m_1,\ldots,m_k} | c : S^c \rangle$$

Here we require that c does not occur in $c_1, \ldots, c_n$ and that no assignment $x \leftarrow \mathsf{new}$, x of type c, and $u \leftarrow \mathsf{new}$, u of type c, occurs in $U^{c_1,\ldots,c_n}_{m_1,\ldots,m_k}$ and S^c. Finally, we require that $TVar(S^c) = \emptyset$. (The symbol '|' is part of the syntax, not of the meta-syntax.)

We call a program $\rho = \langle U^{c_1,\ldots,c_n}_{m_1,\ldots,m_k} | c : S^c \rangle$ *closed* iff every method name occurring in it is defined by U, and only variables x_d, u_d, with $d \in \{c_1, \ldots, c_n, \mathsf{Int}, \mathsf{Bool}\}$, occur in ρ.

The interpretation of such a program is that the statement S is executed by some object of class c (the root object) in the context of the declarations contained in the unit U. We assume that at the beginning of the execution this root object is the only existing non-standard object. The additional requirement ensures that throughout the execution the root-object will be the only existing object of its own class.

3 The assertion language

In this section we define two different assertion languages, i.e., sets of assertions. An *assertion* is used to describe the state of (a part of) the system at one specific point during its execution. The first assertion language describes the *internal state* of a single object. This language will be called the *local* assertion language. The other one is to be used to describe a whole system of objects. The latter language will be called the *global* assertion language.

3.1 The local assertion language

First we introduce a new kind of variables: For $d = \mathsf{Int}, \mathsf{Bool}$, let $LVar_d$ be an infinite set of *logical variables* of type d, with typical element z_d. We assume that these sets are disjoint from the other sets of syntactic entities. Logical variables do not occur in a program, but only in assertions.

Definition 3.1
The set $LExp_d^c$ of *local expressions* of type d in class c, with typical element l_d^c, is defined as follows:

$$
\begin{array}{rcll}
l_d^c & ::= & z_d & \\
 & | & x_d^c & \\
 & | & u_d & \\
 & | & \mathsf{self} & \text{if } d = c \\
 & | & \mathsf{nil} & \\
 & | & n & \text{if } d = \mathsf{Int} \\
 & | & \mathsf{true} \mid \mathsf{false} & \text{if } d = \mathsf{Bool} \\
 & | & {l_1}^c + {l_2}^c & \text{if } d = \mathsf{Int} \\
 & & \vdots & \\
 & | & l_{1d}^c \doteq l_{2d}^c & \text{if } d = \mathsf{Bool}
\end{array}
$$

Definition 3.2
The set $LAss^c$ of *local assertions* in class c, with typical element p^c, is defined as

follows:

$$\begin{array}{rcl} p^c & ::= & l^c \\ & | & \neg p^c \\ & | & p_1^c \wedge p_2^c \\ & & \vdots \\ & | & \exists z_d\, p^c \end{array}$$

Local expressions l_d^c and local assertions p^c are evaluated with respect to the local state of an object of class c (plus a logical environment that assigns values to the logical variables). They talk about this single object in isolation. It is important to note that we allow only logical variables ranging over integers and booleans to occur in local expressions. The intuition behind this is that the internal data of an object consists of the objects stored in its instance variables, so only these objects and the standard ones are known by this object. A logical variable of a type $c' \in C$ would provide a "window" to the external world. Furthermore, as we will explain below, we will define the range of quantification over a class c' to be the set of *existing* objects of this class. But the set of existing objects of the class c' is a global aspect of the system, what is known locally is in general a subset of this set. So quantification over objects of some class c' can not be evaluated by looking only at the local state of some object.

3.2 The global assertion language

Next we define the global assertion language. As we want to quantify in the global assertion language also over objects of some arbitrary class c we now need for every $c \in C$ a new set $LVar_c$ of logical variables of type c, with typical element z_c. To be able to describe interesting properties of pointer structures we also introduce logical variables ranging over *finite sequences* of objects. To do so we first introduce for every $d \in C^+$ the type d^* of finite sequences of objects of type d. We define $C^* = \{d^* : d \in C^+\}$ and take $C^\dagger = C^+ \cup C^*$, with typical element a. Now we assume in addition for every $d \in C^+$ the set $LVar_{d^*}$ of logical variables of type d^*, which range over finite sequences of elements of type d. Therefore in total we now have a set $LVar_a$ of logical variables of type a for every $a \in C^\dagger$.

Definition 3.3
The set $GExp_a$ of *global expressions* of type a, with typical element g_a, is defined as

follows:

$$
\begin{array}{lcll}
g_a & ::= & z_a & \\
 & | & \mathsf{nil} & \\
 & | & n & \text{if } a = \mathsf{Int} \\
 & | & \mathsf{true} \mid \mathsf{false} & \text{if } a = \mathsf{Bool} \\
 & | & g_c.x_d^c & \text{if } a = d \\
 & | & g_{d^*} : g & \text{if } a = d \\
 & | & |g_{d^*}| & \text{if } a = \mathsf{Int} \\
 & | & g_1 + g_2 & \text{if } a = \mathsf{Int} \\
 & & \vdots & \\
 & | & \mathsf{if}\ g\ \mathsf{then}\ g_{1_a}\ \mathsf{else}\ g_{2_a}\ \mathsf{fi} & \\
 & | & g_{1_d} \doteq g_{2_d} & \text{if } a = \mathsf{Bool}
\end{array}
$$

A global expression is evaluated with respect to a complete system of objects plus a logical environment. A complete system of objects consists of a set of existing objects together with their local states. The expression $g.x$ denotes the value of the variable x of the object denoted by g. Note that in this global assertion language we must explicitly specify the object of which we want to access the internal data. The expression $g_1 : g_2$ denotes the n^{th} element of the sequence denoted by g_1, where n is the value of g_2. (If the value of g_2 is less than 1 or greater than the length of the sequence denoted by g_1 we define the value of $g_1 : g_2$ to be undefined, i.e., equivalent to nil.) The expression $|g|$ denotes the length of the sequence denoted by g. For sequence types the expression nil denotes the empty sequence. The conditional expression if-then-else-fi is introduced to facilitate the handling of aliasing . If the condition is undefined, i.e., equals nil, then the result of the conditional expression is undefined, too. Finally, note that we do not have temporary variables in the global assertion language.

Definition 3.4
The set *GAss* of *global assertions*, with typical element P, is defined as follows:

$$
\begin{array}{lcl}
P & ::= & g \\
 & | & \neg P \mid P_1 \wedge P_2 \\
 & & \vdots \\
 & | & \exists z_a\, p
\end{array}
$$

Quantification over (sequences of) integers and booleans is interpreted as usual. However, quantification over (sequences of) objects of some class c is interpreted as ranging

only over the *existing* objects of that class, i.e., the objects that have been created up to the current point in the execution of the program. For example, the assertion $\exists z_c\,\mathbf{true}$ is false in some state iff there are no objects of class c in this state.

Next we define a transformation of a local expression or assertion to a global one. This transformation will be used to verify the assumptions made in the local proof system about the send, answer, and new-statements. These assumptions are formulated in the local language. As the reasoning in the cooperation test uses the global assertion language we have to transform these assumptions from the local language to the global one.

Definition 3.5
Given a local expression l_a^c we define $l_a^c[g_c, \bar{g}/\mathsf{self}, \bar{u}]$, with $\bar{g}$ a sequence of global expressions of the same length as $\bar{u}$ such that the type of g_i equals that of u_i, by induction on the complexity of the local expression l_a^c:

$$\begin{aligned}
z_d[g_c, \bar{g}/\mathsf{self}, \bar{u}] &= z_d \\
x_d^c[g_c, \bar{g}/\mathsf{self}, \bar{u}] &= g_c.x_d^c \\
u_i[g_c, \bar{g}/\mathsf{self}, \bar{u}] &= g_i \\
\mathsf{self}[g_c, \bar{g}/\mathsf{self}, \bar{u}] &= g_c \\
&\vdots
\end{aligned}$$

The omitted cases follow directly from the transformation of the subexpressions. For a local assertion p^c we define $p^c[g_c, \bar{g}/\mathsf{self}, \bar{u}]$ as follows:

$$\begin{aligned}
l^c[g_c, \bar{g}/\mathsf{self}, \bar{u}] &\quad \text{as above} \\
(\neg p^c)[g_c, \bar{g}/\mathsf{self}, \bar{u}] &= (\neg p^c[g_c, \bar{g}/\mathsf{self}, \bar{u}]) \\
&\vdots \\
(\exists z_d p^c)[g_c, \bar{g}/\mathsf{self}, \bar{u}] &= \exists z_d (p^c[g_c, \bar{g}/\mathsf{self}, \bar{u}])
\end{aligned}$$

It will probably be clear that the object of which the local state is described by the assertion p^c is denoted by g_c in the resulting assertion $p^c[g_c, \bar{g}/\mathsf{self}, \bar{u}]$.

3.3 Correctness formulas

In this section we define how we specify an object and a complete system of objects. For the specification of an object we use the assumption/commitment formalism ([HR]). First we introduce two sets of labels Lab_A and Lab_C such that

$Lab_A \cap Lab_C = \emptyset$. Elements of $Lab_A \cup Lab_C$ we denote by $\mathsf{l}, \ldots$. We extend the class of statements by the rule

$$S \ ::= \ \mathsf{l}$$

The execution of a label is equivalent to a skip statement. A label is used to mark a control point. We use labeled local assertions, notation: $\mathsf{l}.p$, to characterize the state during the execution of the corresponding label. The notion of an assumption will be formalized as a labeled local assertion, $\mathsf{l}.p$, with $\mathsf{l} \in Lab_A$. A *commitment* is defined as a local assertion labeled with an element of Lab_C. We can now give the definition of a specification of an object.

Definition 3.6
We define a *local correctness formula* to be of the following form:

$$(A, C : \{p^c\} S^c \{q^c\})$$

where

- $A \subseteq \{\mathsf{l}.p'^c : \ l \in Lab_A\}$
- $C \subseteq \{\mathsf{l}.p'^c : \ l \in Lab_C\}$.

The meaning of such a correctness formula is described informally as follows:

> For an arbitrary prefix of a computation of S^c by an object of class c the following holds:
> *If*
> p^c holds initially and in an arbitrary state of this sequence whenever the object is executing a label $\mathsf{l} \in Lab_A$ the corresponding assumption holds
> *then*
> if the object is about to execute a label $\mathsf{l} \in Lab_C$ then the corresponding commitment holds and if the execution is terminated then q^c characterizes its final state.

As said before, reasoning about the local correctness of an object will be done relative to assumptions concerning those parts of its local process that depend on the environment. These parts are called *bracketed sections:*

Definition 3.7
Let R_1 and R_2 be statements in which there occur no send, answer, and new-statements. A bracketed section is a construct of one of the following forms:

- $\mathsf{l}_1; R_1; \mathsf{l}; x \leftarrow e_0!m(e_1, \ldots, e_n); R_2; \mathsf{l}_2$,
  where $Ass(R_1) \cap Var(e_0, e_1, \ldots, e_n) = \emptyset$.
- $\mathsf{l}_1; R_1; x \leftarrow \mathsf{new}; R_2; \mathsf{l}_2$,
  where $x \notin Ass(R_2)$.
- $\mathsf{l}_1; \mathsf{answer}(m_1, \ldots, m_n); \mathsf{l}_2$
- $m \Leftarrow R_1; \mathsf{l}_1; S; \mathsf{l}_2; R_2 \uparrow e$,
  where $Ass(R_2) \cap Var(e) = \emptyset$.

Here $Ass(R)$ is defined inductively as follows:

- $Ass(x \leftarrow s) = \{x\}$
- $Ass(\mathsf{answer}(m_1, \ldots, m_n)) = \bigcup_i Ass(S_i) \cap IVar$,
  where S_i is the body of m_i
- $Ass(S_1; S_2) = Ass(S_1) \cup Ass(S_2)$
- $Ass(\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}) = Ass(S_1) \cup Ass(S_2)$
- $Ass(\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}) = Ass(S)$

When m is declared as $R_1; \mathsf{l}_1; S; \mathsf{l}_2; R_2 \uparrow e$ we call R_1 its *prelude* and R_2 its *postlude*. Also we call the statement R_1 the prelude of the bracketed section $\mathsf{l}; R_1; R; R_2; \mathsf{l}'$ and R_2 its postlude. The restriction $Ass(R_1) \cap Var(e_0, \ldots, e_n) = \emptyset$ of the first clause is introduced to ensure that the execution of R_1 does not affect the values of the expressions $e_0, \ldots, e_n$, so that before the execution of R_1 we in fact know the object to which the request is made and the actual parameters. The restriction $Ass(R_2) \cap Var(e) = \emptyset$ of the last clause ensures that the execution of R_2 does not affect the result. Both restrictions will be used in the definition of the cooperation test.

In the following section we define a proof system for reasoning about local correctness formulas. The derivability from this system of a correctness formula $(A, C : \{p\}S\{q\})$ then amounts essentially to proving

$$\mathcal{A} \vdash \{p\}S'\{q\}$$

where $\mathcal{A} = \{\{C(\mathsf{l})\}R\{A(\mathsf{l}')\} : \mathsf{l}; R; \mathsf{l}'$ a bracketed section occurring in $S\}$, and S' results from S by removing all labels, using the usual proof system for sequential programs. Here, given a set of labeled assertions X such that with each label occurring in X there corresponds precisely one assertion, and a label l, we define

$$\begin{aligned} X(\mathsf{l}) &= p \quad \text{if } \mathsf{l}.p \in X \\ &= \mathsf{true} \quad \text{otherwise.} \end{aligned}$$

However, with respect to the soundness proofs, correctness formulas $(A, C : \{p\}S\{q\})$, as they essentially represent a *partial proof-outline* of the version of S without labels, are more convenient.

Next we define intermediate correctness formulas, which describe the behaviour of objects executing a local statement (that is, a statement not involving any new, answer, or send statements), or a bracketed section containing a new-statement, from a global point of view.

Definition 3.8
An *intermediate correctness formula* can have one of the following two forms:

- $\{P\}(z_c, R^c)\{Q\}$, where R^c is a local statement or a bracketed section containing a new-statement.
- $\{P\}(z_{c_1}, R_1^{c_1}) \parallel (z'_{c_2}, R_2^{c_2})\{Q\}$, where z_{c_1} and z'_{c_2} are distinct logical variables and $R_1^{c_1}$ and $R_2^{c_2}$ are local statements.

The logical variables z_c, z_{c_1} and z'_{c_2} in the above constructs denote the objects that are considered to be executing the corresponding statements. More precisely, the meaning of the intermediate correctness formula $\{P\}(z, R)\{Q\}$ is as follows:

> Every terminating execution of R by the object denoted by the logical variable z starting in a state satisfying P ends in a state satisfying Q.

The meaning of the second form of intermediate correctness formula, $\{P\}(z, R_1) \parallel (z', R_2)\{Q\}$, can be described as follows:

> Every terminating parallel execution of R_1 by the object denoted by the logical variable z and of R_2 by the object denoted by z' starting in a state satisfying P will end in a state satisfying Q.

In the cooperation test a correctness formula $\{P\}(z, R)\{Q\}$, R a bracketed section containing a new-statement, will be used to justify the assumption associated with R. A correctness formula $\{P\}(z_1, R_1) \parallel (z_2, R_2)\{Q\}$, with R_1 the prelude of a bracketed section containing a send-statement, and R_2 a prelude of an answer-statement, will be used to justify the assumption about the parameters. Information about the actual parameters will be coded in P. On the other hand a correctness formula $\{P\}(z_1, R_1) \parallel (z_2, R_2)\{Q\}$, with R_1 the postlude of a bracketed section containing a send-statement, and R_2 a postlude of an answer-statement, will be used to justify

the assumption about the result value and the assumption about the state after the execution of the answer-statement. Information about the result will be coded in P.

Finally, we have *global correctness formulas*, which describe a complete system:

Definition 3.9
A global correctness formula is of the form

$$\{p^c[z_c/\mathsf{self}]\}\rho\{Q\}$$

where ρ is a program and c is the root class in ρ, and $TVar(p) = \emptyset$.

The variable z_c in such a global correctness formula denotes the root object. Initially this root object is the only existing object, so it is sufficient for the precondition of a complete system to describe only its local state. We obtain such a precondition by transforming some local assertion p^{c_n} to a global one. On the other hand, the final state of an execution of a complete system is described an arbitrary global assertion. The meaning of the global correctness formula $\{p[z/\mathsf{self}]\}\rho\{Q\}$ can be rendered as follows:

> If the execution of the unit ρ starts with a root object denoted by z that satisfies the local assertion p and no other objects, and if moreover this execution terminates, then the final state will satisfy the global assertion Q.

4 The proof system

The proof system we present consists of three levels. The first level, called the *local* proof system, enables one to reason about the correctness of an object. Testing the assumptions, which are introduced at the first level to deal with answer, send and new-statements, is done at the second level, which is called the *intermediate* proof system. The third level, the *global* proof system, formalizes the reasoning about a complete system.

4.1 The local proof system

The proof system for local correctness formulas dealing with assignment, sequential composition, the **if** – **then** – **else** – **fi** and **while** – **do** – **od** construct equals the usual system for sequential programs:

Definition 4.1
We have the following well-known assignment axiom for instance variables:

$$(A, C : \{p^c[e_d^c/x_d^c]\}x_d^c := e_d^c\{p^c\}) \qquad \text{(LIASS)}$$

Definition 4.2
We have the following assignment axiom for temporary variables:

$$(A, C : \{p^c[e_d^c/u_d]\}u_d := e_d^c\{p^c\}) \qquad \text{(LTASS)}$$

The substitution operation occurring in the assignment axioms is the ordinary substitution, i.e., literal replacement of every occurrence of the variable x (u) by the expression e. Note that at this level we have no aliasing, i.e., there exist no two local expressions denoting the same variable.

Definition 4.3
The following rule formalizes reasoning about sequential composition:

$$\frac{(A, C : \{p^c\}S_1^c\{r^c\}),\ (A, C : \{r^c\}S_2^c\{q^c\})}{(A, C : \{p^c\}S_1^c; S_2^c\{q^c\})} \qquad \text{(LSC)}$$

Definition 4.4
Next we define the rule for the alternative command:

$$\frac{(A, C : \{p^c \wedge e\} S_1^c \{q^c\}),\ (A, C : \{p^c \wedge \neg e\} S_2^c \{q^c\})}{(A, C : \{p^c\} \textsf{if } e \textsf{ then } S_1^c \textsf{ else } S_2^c \textsf{ fi} \{q^c\})} \qquad \text{(LALT)}$$

Definition 4.5
We have the following rule for the iteration construct:

$$\frac{(A, C : \{p^c \wedge e\} S^c \{p^c\})}{(A, C : \{p^c\} \textsf{while } e \textsf{ do } S^c \textsf{ od} \{p^c \wedge \neg e\})} \qquad \text{(LIT)}$$

Definition 4.6
We have the following consequence rule:

$$\frac{p^c \rightarrow p_1^c,\ (A, C : \{p_1^c\} S^c \{q_1^c\}),\ q_1^c \rightarrow q^c}{(A, C : \{p^c\} S^c \{q^c\})} \qquad \text{(LCR)}$$

The following axioms and rule deal with bracketed sections.

Definition 4.7
We have the following axiom about bracketed sections containing new-statements:

$$(A, C : \{C(\mathsf{l}_1)\}\mathsf{l}_1; R_1; x \leftarrow \textsf{new}; R_2; \mathsf{l}_2\{A(\mathsf{l}_2)\}) \qquad \text{(BN)}$$

We have a similar axiom in case the identity of the newly created object is assigned to a temporary variable.

Definition 4.8
We have the following axiom about bracketed sections containing send-statements:

$$(A, C : \{C(\mathsf{l}_1)\}\mathsf{l}_1; R_1; \mathsf{l}; x \leftarrow e_0!m(e_1, \ldots, e_n); R_2; \mathsf{l}_2\{A(\mathsf{l}_2)\}) \qquad \text{(BS)}$$

where $x \notin IVar(C(\mathsf{l}))$. We have a similar axiom in case the result of the send-expression is assigned to a temporary variable.

Definition 4.9
We have the following rule about bracketed sections containing answer-statements:

$$\frac{(\emptyset, \emptyset : \{p[\bar{y}/\bar{u}]\} S_i \{p[\bar{y}/\bar{u}]\}),\ i = 1, \ldots, n}{(A, C : \{p \wedge C(\mathsf{l}_1)\}\mathsf{l}_1; \textsf{answer}(m_1, \ldots, m_n); \mathsf{l}_2\{p \wedge A(\mathsf{l}_2)\})} \qquad \text{(BA)}$$

where $\bar{u}$ is the sequence of the temporary variables occurring in p, and $\bar{y}$ is a corresponding sequence of new instance variables, and $TVar(C(\mathsf{l}_1), A(\mathsf{l}_2)) = \emptyset$. Furthermore, S_i denotes the body of the method m_i.

The axioms (BN), (BS) and the rule (BA) extract from the set C the precondition and from the set A the postcondition using the labels which mark the beginning and the end of the bracketed section. The rule (BA) additionally incorporates the derivation of some invariance property of the answer-statement involved. Note that in the derivation of this property we are not allowed to use the sets of assumptions A and commitments C. Reasoning about new, send, and answer statements occurring in the bodies of the methods specified by the answer statement can be solely done by means of the invariance axiom described below. In the cooperation test the applications of the axioms (BN), (BS) and the rule (BA) will be justified.

Definition 4.10
We have the following invariance axiom:

$$(A, C : \{p\}S\{p\}) \qquad \text{(INV)}$$

where $Ass(S) \cap ITvar(p) = \emptyset$.

4.2 The intermediate proof system

In this subsection we present the proof system for the intermediate correctness formulas. This proof system is derived from the proof system for the language SPOOL, a sequential version of POOl [AB2].

4.2.1 The assignment axiom

We have the following assignment axiom:

Definition 4.11
Let $x \leftarrow e \in Stat^c$ and $z \in LVar_c$. We define

$$\{P[e[z/\mathsf{self}]/z.x]\}(z, x \leftarrow e)\{P\} \qquad \text{(IASS).}$$

First note that we have to transform the expression e to the global expression $e[z/\mathsf{self}]$ and substitute this latter expression for $z.x$ because we consider the execution of

the assignment $x \leftarrow e$ by the object denoted by z. Furthermore we have to define this substitution operation $[e[z/\mathsf{self}]/z.x]$ because the usual one does not consider possible aliases of the expression $z.x$. For example, the expression $z'.x$, where z' differs syntactically from z, has to be substituted by $e[z/\mathsf{self}]$ if the variables z and z' both refer to the same object, i.e., if $z \doteq z'$ holds.

Definition 4.12
Given a global expression g'_d, a logical variable z_c and a variable x, $x \in IVar^c_d$, we define for an arbitrary global expression g the substitution of the expression g' for $z_c.x$ in g by induction on the complexity of g. The result of this substitution we denote by $g[g'_d/z_c.x]$. Let [.] abbreviate $[g'_d/z_c.x]$:

$$\begin{aligned}
z[.] &= z \\
g[.] &= g,\ g = n, \mathsf{nil}, \mathsf{self}, \mathsf{true}, \mathsf{false} \\
(g.y)[.] &= g[.].y,\ y \neq x \\
(g.x)[.] &= \mathsf{if}\ g[.] = z_c\ \mathsf{then}\ g'_d\ \mathsf{else}\ g[.].x\ \mathsf{fi} \\
&\vdots
\end{aligned}$$

The omitted cases are defined directly from the application of the substitution to the subexpressions. This substitution operation is generalized to a global assertion in a straightforward manner, notation: $P[g'_d/z_c.x]$.

The most important aspect of this substitution is certainly the conditional expression that turns up when we are dealing with an expression of the form $g.x$. This is necessary because a certain form of aliasing, as described by the example above, is possible: After substitution it is possible that g refers to the object denoted by the logical variable z_c, so that $g.x$ is the same variable as $z_c.x$ and should be substituted by g'. It is also possible that, after substitution, g does not refer to the object denoted by z_c, and in this case no substitution should take place. Since we can not decide between these possibilities by the form of the expression only, a conditional expression is constructed which decides "dynamically".

The intended meaning of this substitution operation is that the value of the substituted expression (assertion) in a state equals the value of the expression (assertion) in the state resulting from assigning the value of the expression g'_d to the variable x of the object denoted by z_c. A proof of the correctness of the substitution operation can be found in [Bo].

4.2.2 The creation of new objects

We describe the meaning of a new statement by the following axiom:

Definition 4.13
Let $x \leftarrow \mathsf{new} \in Stat^c$, the type of the variable x being $d \in C^+$. Furthermore let $z \in LVar_c$ and $z' \in LVar_d$ be two distinct variables. We define

$$\{P[z'/z.x][\mathsf{new}/z']\}(z, x \leftarrow \mathsf{new})\{P\} \qquad \text{(NEW)}$$

provided z' does not occur in P.

The calculation of the weakest precondition of an assertion with respect to the creation of a new object is done in two steps: The first of which consists of the substitution of a fresh variable z' for $z.x$. This substitution makes explicit all the possible aliases of the expression $z.x$. Next we carry out the substitution $[\mathsf{new}/z']$. This substitution interprets the variable z' as the new object.

The definition of this latter substitution operation is complicated by the fact that the newly created object does not exists in the state just before its creation, so that in this state we can not refer to it. Assuming the new object to be referred to by the logical variable z' (in the state just after its creation) we however are able to carry out the substitution due to the fact that this variable z' can essentially occur only in a context where either one of its instances variables is referenced, or it is compared for equality with another expression. In both of these cases we can predict the outcome without having to refer to the new object.

Definition 4.14
Let $[.]$ abbreviate $[\mathsf{new}/z_c]$, we first define $g[.]$ by induction on the complexity of g:

$$\begin{array}{lcll}
z[.] & = & z,\ z \neq z_c & \\
z_c[.] & & \text{is undefined} & \\
g[.] & = & g & g = n, \mathsf{nil}, \mathsf{self}, \mathsf{true}, \mathsf{false} \\
(z.x)[.] & = & z.x,\ z \neq z_c & \\
(z_c.x_d)[.] & = & \mathsf{nil} & \\
(g.y.x)[.] & = & (g.y)[.].x & \\
\left(\begin{array}{l}\mathsf{if}\ g_0 \\ \mathsf{then}\ g_1 \\ \mathsf{else}\ g_2 \\ \mathsf{fi}\end{array}\right).x\,[.] & = & \begin{array}{l}\mathsf{if}\ g_0[.] \\ \mathsf{then}\ (g_1.x)[.] \\ \mathsf{else}\ (g_2.x)[.] \\ \mathsf{fi}\end{array} & \\
(g : g')[.] & = & g[.] : g'[.] & \\
|g|[.] & = & |g[.]| &
\end{array}$$

$$\left(\begin{array}{l}\textsf{if } g_0\\ \textsf{then } g_1\\ \textsf{else } g_2\\ \textsf{fi}\end{array}\right)[.] \;=\; \begin{array}{l}\textsf{if } g_0[.]\\ \textsf{then } g_1[.]\\ \textsf{else } g_2[.]\\ \textsf{fi}\end{array} \qquad \text{if the substitutions are defined, undefined otherwise}$$

$$\begin{array}{lcll}(g_1 = g_2)[.] & = & g_1[.] = g_2[.] & g_1, g_2 \neq z_c, \textsf{if} \ldots \textsf{fi}\\ (g_1 = g_2)[.] & = & \textsf{false} & g_i = z_c, g_j \neq z_c, \textsf{if} \ldots \textsf{fi},\ i \neq j \in \{1,2\}\\ (g_1 = g_2)[.] & = & \textsf{true} & g_1 = g_2 = z_c\end{array}$$

$$\left(\begin{array}{l}\textsf{if } g_0\\ \textsf{then } g_1\\ \textsf{else } g_2\\ \textsf{fi}\end{array} = g_3\right)[.] \;=\; \begin{array}{l}\textsf{if } g_0[.] = \textsf{nil}\\ \textsf{then } (g_3 = \textsf{nil})[.]\\ \textsf{else if } g_0[.]\\ \qquad \textsf{then } (g_1 = g_3)[.]\\ \qquad \textsf{else } (g_2 = g_3)[.]\\ \qquad \textsf{fi}\\ \textsf{fi}\end{array}$$

$$\left(g_3 = \begin{array}{l}\textsf{if } g_0\\ \textsf{then } g_1\\ \textsf{else } g_2\\ \textsf{fi}\end{array}\right)[.] \;=\; \begin{array}{l}\textsf{if } g_0[.] = \textsf{nil}\\ \textsf{then } (g_3 = \textsf{nil})[.]\\ \textsf{else if } g_0[.]\\ \qquad \textsf{then } (g_1 = g_3)[.] \qquad g_3 \neq \textsf{if} \ldots \textsf{fi}\\ \qquad \textsf{else } (g_2 = g_3)[.]\\ \qquad \textsf{fi}\\ \textsf{fi}\end{array}$$

We have the following proposition about this substitution operation applied to global expressions:

Proposition 4.15
For every global expression g, logical variable z_c, $g[\textsf{new}/z_c]$ is defined iff g is not of the form gz:

$$gz \;::=\; z_c \mid \textsf{if } g_0 \textsf{ then } gz \textsf{ else } g_1 \textsf{ fi} \mid \textsf{if } g_0 \textsf{ then } g_1 \textsf{ else } gz \textsf{ fi}$$

In [Bo] we prove that the value of the application of this substitution operation to an expression g in a state equals the value of the expression g in the state resulting from

the creation of a new object of class c, assuming this new object in this new state to be referred to by the variable z_c.

Next we define $P[\mathsf{new}/z_c]$ by induction on the complexity of P.

Definition 4.16
Let, again, $[.]$ abbreviate $[\mathsf{new}/z_c]$.

$$
\begin{array}{lcl}
g[.] & & \text{defined as above} \\
(\neg P)[.] & = & \neg(P[.]) \\
(P_1 \wedge P_2)[.] & = & (P_1[.] \wedge P_2[.]) \\
(\forall z_a P)[.] & = & \forall z_a(P[.]),\ a \neq c, c^* \\
(\forall z'_c P)[.] & = & \forall z'_c(P[.]) \wedge P[z_c/z'_c][.]),\ z'_c \neq z_c \\
(\forall z_{c^*} P)[.] & = & \forall z_{c^*} \forall z_{\mathsf{Bool}^*}(|z_{c^*}| = |z_{\mathsf{Bool}^*}| \rightarrow P[z_{\mathsf{Bool}^*}, z_c/z_{c^*}][.]) \\
(\exists z_a P)[.] & = & \exists z_a(P[.]),\ a \neq c, c^* \\
(\exists z'_c P)[.] & = & \exists z'_c(P[.] \vee P[z_c/z'_c][.]),\ z'_c \neq z_c \\
(\exists z_{c^*} P)[.] & = & \exists z_{c^*} \exists z_{\mathsf{Bool}^*}(|z_{c^*}| = |z_{\mathsf{Bool}^*}| \wedge P[z_{\mathsf{Bool}^*}, z_c/z_{c^*}][.])
\end{array}
$$

Here we assume that z_{Bool^*} does not occur in P. The case of quantification over the type c of the newly created object can be explained as follows: Suppose we interpret the result of the substitution in a state in which the object denoted by z_c does not yet exists. In the first part of the substituted formula the bound variable z'_c thus ranges over all the old objects. In the second part the object to be created (the object denoted by z_c) is dealt with separately. This is done by first (literally) substituting the variable z_c for the quantified variable z'_c and then applying the substitution $[\mathsf{new}/z_c]$. In this way the second part of the substituted formula expresses that the assertion P is valid in the new state (the state after the creation of the object denoted by z_c) when this variable z'_c is interpreted as the newly created object. Together the two parts of the substituted formula express quantification over the whole range of existing objects in the new state.

The idea of the substitution operation $[z_{\mathsf{Bool}^*}, z_c/z_{c^*}]$ is that z_{Bool^*} and z_{c^*} together code a sequence of objects in the state just after the creation of the new object. At the places where z_{Bool^*} yields **true** the value of the coded sequence is the newly created object. Where z_{Bool^*} yields **false** the value of the coded sequence is the same as the value of z_{c^*} and where z_{Bool^*} delivers $\perp$ the sequence also yields $\perp$.

Now $g[z_{\mathsf{Bool}^*}, z_c/z_{c^*}]$ is defined as follows:

Definition 4.17

Let [.] abbreviate $[z_{\mathsf{Bool}^*}, z_c/z_{c^*}]$.

$$
\begin{array}{lcl}
z_{c^*}[.] & & \text{isundefined} \\
z[.] & = & z,\ z \neq z_{c^*} \\
g[.] & = & g,\ g = n, \mathsf{nil}, \mathsf{self}, \mathsf{tue}, \mathsf{false} \\
(g.x)[.] & = & g[.].x \\
(z_{c^*} : g)[.] & = & \begin{array}{l} \mathsf{if}\ z_{\mathsf{Bool}^*} : (g[.]) \\ \mathsf{then}\ z_c \\ \mathsf{else}\ z_{c^*} : (g[.]) \\ \mathsf{fi} \end{array} \\
(g_1 : g_2)[.] & = & g_1[.] : g_2[.],\ g_1 \neq z_{c^*} \\
(|z_{c^*}|)[.] & = & |z_{c^*}| \\
(|g|)[.] & = & |g[.]|,\ g \neq z_{c^*} \\
 & \ldots &
\end{array}
$$

We have the following proposition:

Proposition 4.18
For an arbitrary global expression g the expression $g[z_{\mathsf{Bool}^*}, z_c/z_{c^*}]$ is defined iff g is not of the form g':

$$g' ::= z_{c^*} \mid \mathsf{if}\ g_0\ \mathsf{then}\ g'\ \mathsf{else}\ g_1\ \mathsf{fi} \mid \mathsf{if}\ g_0\ \mathsf{then}\ g_1\ \mathsf{else}\ g'\ \mathsf{fi}$$

In [Bo] can be found a proof that the assertion $P[\mathsf{new}/z_c]$ holds in a state iff it holds in the state resulting from the creation of a new object of class c, assuming the newly created object in this new state to be referred to by the variable z.

4.2.3 Some other rules

The rules for sequential composition, the alternative, the iterative construct, and the consequence rule are straightforward translations of the corresponding rules of the local proof system.

Definition 4.19
Let $S_1, S_2 \in Stat^c$ and $z \in LVar_c$.

$$\frac{\{P\}(z, S_1)\{R\},\ \{R\}(z, S_2)\{Q\}}{\{P\}(z, S_1; S_2)\{Q\}} \qquad \text{(ISC)}$$

Definition 4.20
Let if e then S_1 else S_2 fi $\in Stat^c$ and $z \in LVar_c$.

$$\frac{\{P \wedge e[z/\mathsf{self}]\}(z, S_1)\{Q\},\ \{P \wedge \neg e[z/\mathsf{self}])\}(z, S_2)\{Q\}}{\{P\}(z, \text{if } e \text{ then } S_1 \text{ else } S_2 \text{ fi})\{Q\}} \qquad \text{(IALT)}$$

Definition 4.21
Let while e do S od $\in Stat^c$ and $z \in LVar_c$.

$$\frac{\{P \wedge e[z/\mathsf{self}]\}(z, S)\{P\}}{\{P\}(z, \text{while } e \text{ do } S \text{ od})\{P \wedge \neg e[z/\mathsf{self}]\}} \qquad \text{(IIT)}$$

Definition 4.22
Let $S \in Stat^c$ and $z \in LVar_c$.

$$\frac{P \to P_1,\ \{P_1\}(z, S)\{Q_1\},\ Q_1 \to Q}{\{P\}(z, S)\{Q\}} \qquad \text{(ICR)}$$

Finally, we have the following two rules describing the parallel execution of two objects:

Definition 4.23

$$\frac{\{P\}(z_1, S_1)\{R\},\ \{R\}(z_2, S_2)\{Q\}}{\{P\}(z_1, S_1) \parallel (z_2, S_2)\{Q\}} \qquad \text{(Par)}$$

Definition 4.24

$$\frac{P \to P_1,\ \{P_1\}(z_1, S_1) \parallel (z_2, S_2)\{Q_1\},\ Q_1 \to Q}{\{P\}(z_1, S_1) \parallel (z_2, S_2)\{Q\}} \qquad \text{(Cpar)}$$

4.3 The global proof system

In this section we describe the global proof system. We first define the notion of the cooperation test:

Definition 4.25
Let $\rho^c = \langle U^{c_1,\ldots,c_{n-1}}_{m_1,\ldots,m_k} | c_n : S^{c_n}_n \rangle$ be bracketed (that is, every new, send, and answer statement of ρ^c occurs in a bracketed section), with $U^{c_1,\ldots,c_{n-1}}_{m_1,\ldots,m_k} = D_1{}^{c_1}_{\bar{m}_1}, \ldots, D_{n-1}{}^{c_{n-1}}_{\bar{m}_{n-1}}$, where $D_i{}^{c_i}_{\bar{m}_i} = \langle m^i_1{}^{c_i}_{\bar{d}^i_1} \Leftarrow \mu^i_1{}^{c_i}_{\bar{d}^i_1}, \ldots, m^i_{n_i}{}^{c_i}_{\bar{d}^i_{n_i}} \Leftarrow \mu^i_{n_i}{}^{c_i}_{\bar{d}^i_{n_i}} \rangle : S^{c_i}_i$. (We define $D_n = \langle\rangle : S^{c_n}_n$.)
The specifications

$$(A_k, C_k : \{p^{c_k}_k\} S^{c_k}_k \{q^{c_k}_k\}),\ 1 \leq k \leq n$$

(with $TVar(p_k, q_k) = \emptyset$) cooperate with respect to some global invariant I iff

1. There are no occurrences in I of variables which occur at the left hand side of an assignment which is not contained in a bracketed section
2. $\vdash (A_k, C_k : \{p^{c_k}_k\} S^{c_k}_k \{q^{c_k}_k\}),\ 1 \leq k \leq n$
3. $\vdash (A_i, C_i : \{A_i(\mathsf{l}_1)\} R \{C_i(\mathsf{l}_2)\}),\ 1 \leq i \leq n$, where $m \Leftarrow R_1; \mathsf{l}_1; R; \mathsf{l}_2; R_2 \uparrow e \in D_i$.
4. Let $\mathsf{l}_1; R; \mathsf{l}_2$ be a bracketed section, occurring in D_i, containing the new-statement $x \leftarrow \mathsf{new}$. Furthermore, let $z \in LVar_{c_i}$ be a new variable. Then:

   $$\vdash \{I \wedge p[z, \bar{y}/\mathsf{self}, \bar{u}]\}(z, R[\bar{y}/\bar{u}])\{I \wedge q[z, \bar{y}/\mathsf{self}, \bar{u}] \wedge p^{c_j}_j[z.x/\mathsf{self}]\}$$

   where $\bar{u}$ is the sequence of the temporary variables occurring in $p = C_i(\mathsf{l}_1))$, $q = A_i(\mathsf{l}_2)$, and $\bar{y}$ is a corresponding sequence of new instance variables. Furthermore, we assume the variable x to be of type c_j.
5. For $\mathsf{l}_1; R_1; \mathsf{l}; x \leftarrow e_0!m(e_1, \ldots, e_k); R_2; \mathsf{l}_2$ occurring in D_i, and $\mathsf{l}'_1; \mathsf{answer}(\ldots, m, \ldots); \mathsf{l}'_2$ occurring in D_j, such that the type of e_0 is c_j, with m declared as $R'_1; \mathsf{l}''_1; R; \mathsf{l}''_2; R'_2 \uparrow e$, we have

   $$\begin{array}{rc} & \{I \wedge r_1[z_1, \bar{y}_1/\mathsf{self}, \bar{u}_1] \wedge r'_1[z_2/\mathsf{self}] \wedge P\} \\ \vdash & (z_1, R_1[\bar{y}_1/\bar{u}_1]) \parallel (z_2, R'_1[\bar{y}_2/\bar{u}_2]) \\ & \{I \wedge r[z_1, \bar{y}_1/\mathsf{self}, \bar{u}_1] \wedge r''_1[z_2, \bar{y}_2/\mathsf{self}, \bar{u}_2]\} \end{array}$$

   and

   $$\begin{array}{rc} & \{I \wedge r[z_1, \bar{y}_1/\mathsf{self}, \bar{u}_1] \wedge r''_2[z_2, \bar{y}_2/\mathsf{self}, \bar{u}_2] \wedge Q\} \\ \vdash & (z_1, R_2[\bar{y}_1/\bar{u}_1]) \parallel (z_2, R'_2[\bar{y}_2/\bar{u}_2]) \\ & \{I \wedge r_2[z_1, \bar{y}_1/\mathsf{self}, \bar{u}_1] \wedge r'_2[z_2/\mathsf{self}]\} \end{array}$$

where $\bar{u}_1$ is the sequence of the temporary variables occurring in r, r_1, r_2, R_1, R_2, and $\bar{y}_1$ is the corresponding sequence of new instance variables, $\bar{u}_2$ is a sequence of the temporary variables occurring in $r'_1, r'_2, r''_1, r''_2, R'_1, R'_2$, and $\bar{y}_2$ is a corresponding sequence of new instance variables. The variables z_1 and z_2 are new variables, z_1 being of type c_i and z_2 being of type c_j. Furthermore, we have

- $C_i(\mathsf{l}_1) = r_1,\ C_i(\mathsf{l}) = r,\ A_i(\mathsf{l}_2) = r_2$
- $C_j(\mathsf{l}'_1) = r'_1,\ A_j(\mathsf{l}'_2) = r'_2,\ A_j(\mathsf{l}''_1) = r''_1,\ C_j(\mathsf{l}''_2) = r''_2$

Finally, we have

- $P = e_0[z_1, \bar{y}_1/\mathsf{self}, \bar{u}_1] \doteq z_2 \wedge \bigwedge_i e_i[z_1, \bar{y}_1/\mathsf{self}, \bar{u}_1] \doteq z_2.y'_i \wedge \bigwedge_j z_2.y''_j \doteq \mathsf{nil}$
- $Q = z_1 \doteq z_2.y'_1 \wedge z_1.x \doteq e[z_2, \bar{y}_2/\mathsf{self}, \bar{u}_2]$

where $\bar{y}' \cup \bar{y}'' = \bar{y}_2$, with $\bar{y}'$ being the instance variables corresponding to the formal parameters and $\bar{y}''$ corresponding to the local variables of m.

6. The following assertion holds:

$$p_n^{c_n}[z/\mathsf{self}] \wedge \forall z'(z' \doteq z) \wedge \bigwedge_{1 \le i < n} (\forall z_i \mathsf{false}) \rightarrow I$$

Here the variables z and z' are assumed to be of type c_n, the type of the root-object, the variable z_i is assumed to be of type c_i.

The syntactic restriction on occurrences of variables in the global invariant I implies the invariance of this assertion over those parts of the program which are not contained in a bracketed section. The clauses 4 and 5 imply among others the invariance of the global invariant over the bracketed sections.

This global invariant expresses some invariant properties of the dynamically evolving pointer structures arising during a computation of ρ. These properties are invariant in the sense that they hold whenever the program counter of every existing object is at a location outside a bracketed section. The above method to prove the invariance of the global invariant is based on the following semantical property of bracketed sections: Every computation of ρ can be rearranged such that at every time there is at most one object executing a bracketed section containing a new-statement, or a bracketed section belonging to an answer-statement.

Clause 2 verifies in an uniform manner the behaviour of the objects belonging to a class defined by the program.

Clause 3 verifies the behaviour of the methods, more precisely, the part of the body of a method excluding its prelude and postlude.

Clause 4 discharges assumptions about bracketed sections containing new-statements. Additionally the truth of the precondition of the local process of the new object is established. Note that by definition of a bracketed section we know that immediately after the execution of a bracketed section containing a new-statement $x :=$ **new** the newly created object is referred to by x.

Clause 5 establishes the cooperation between two arbitrary matching bracketed sections, where two bracketed sections are said to match if they contain a send-statement and an **answer**-statement which match, i.e., the method name mentioned in the send-statement occurs in the **answer**-statement and the class of the object to which this method is sent equals that of the **answer**-statement.

The first correctness formula of clause 5 describes the activation of the rendezvous whilst the second one describes the termination of it.

The state before the rendezvous is characterized by

- the global invariant, which describes the complete system,
- the precondition of the **answer**-statement lifted to the global assertion language, which describes the local state of the receiver,
- the precondition of the bracketed section containing the send-statement also lifted to the global assertion language, which describes the local state of the sender, and, finally,
- a global assertion expressing that the sender indeed addresses the receiver and that the actual parameters are stored in the instance variables which denote at the level of the global assertion language the formal parameters.

Note that we have to introduce new instance variables which at the level of the global assertion language stand for the temporary variables of the sender and the receiver. The activation of the rendezvous then is described as the parallel execution of the prelude of the bracketed section containing the send-statement, and that of the method being executed. Note that the order in which these statements are executed does not matter because by definition the values of the expressions denoting the receiver and the actual parameters are not affected by the execution of the prelude of the bracketed section containing the **send**-statement. After the execution of these preludes the global invariant must hold and the local assertions lifted to the global assertion language, which are associated with the corresponding control points.

The state just before the termination of the rendezvous, which is described by the parallel execution of the postlude of the bracketed section containing the send-statement and that of the method being executed, is characterized by

- the global invariant,
- the local assertions lifted to the level of the global assertion language, which are associated with the corresponding control points,
- a global assertion expressing that this incarnation of the method has indeed been invoked by the object executing the bracketed section containing the send-statement (note that here we make use of the fact that the identity of the sender is sent as parameter) and, furthermore, that the result has been sent back.

Note that we may assume that the result has been sent back before the execution of the method has been completed because, by definition, the execution of the postlude of the method does not affected the result. Furthermore, the variable of the sender in which this result is to be stored, is not allowed to occur in the local assertion associated with the label marking the send-statement. After the execution of these postludes the global invariant must hold again together with the local assertions lifted to the global assertion language, which are associated with the corresponding control points.

Clause 6 establishes the truth of the global invariant in the initial state. Note that the assertion $\forall z_c \mathsf{false}$ expresses that there exist no objects of class c. The assertion $\forall z'(z' \doteq z)$ expresses that there exists precisely one object of class c_n.

Finally for $\rho^c = \langle U^{c_1,\ldots,c_{n-1}}_{m_1,\ldots,m_k} | c_n : S^{c_n}_n \rangle$, with $U^{c_1,\ldots,c_{n-1}}_{m_1,\ldots,m_k} = D_1{}^{c_1}_{\bar{m}_1}, \ldots, D_{n-1}{}^{c_{n-1}}_{\bar{m}_{n-1}}$, where $D_i{}^{c_i}_{\bar{m}_i} = \langle m^i_1{}^{c_i}_{\bar{d}^i_1} \Leftarrow \mu^i_1{}^{c_i}_{\bar{d}^i_1}, \ldots, m^i_{n_i}{}^{c_i}_{\bar{d}^i_{n_i}} \Leftarrow \mu^i_{n_i}{}^{c_i}_{\bar{d}^i_{n_i}} \rangle : S^{c_i}_i$ we have the following rules:

Definition 4.26
We have the following program rule:

$$\frac{(A_i, C_i : \{p^{c_i}_i\} S^{c_i}_i \{q^{c_i}_i\},\ 1 \leq i \leq n, \text{ cooperate w.r.t. } I}{\{p^{c_n}_n[z/\mathsf{self}]\}\rho\{I \wedge \bigwedge_{1 \leq i < n} \forall z_i q^{c_i}_i[z_i/\mathsf{self}] \wedge q^{c_n}_n[z/\mathsf{self}]\}} \qquad \text{(PR)}$$

where z is of type c_n and z_i is of type c_i.

Note that in the conclusion of the program rule (PR) we take as precondition the precondition of the local process of the root-object because initially only this object exists. The postcondition consists of a conjunction of the global invariant, the assertions $\forall z_i q^{c_i}_i[z_i/\mathsf{self}]$, which express that the final local state of every object of class c_i is characterized by the local assertion $q^{c_i}_i$, and the assertion $q^{c_n}_n[z/\mathsf{self}]$ expressing that the final local state of the root-object is characterized by the local assertion $q^{c_n}_n$.

Definition 4.27
We have the following consequence rule for programs:

$$\frac{p^{c_n} \to p_1^{c_n},\ \{p_1^{c_n}[z_{c_n}/\mathsf{self}]\}\rho\{Q_1\},\ Q_1 \to Q}{\{p^{c_n}[z_{c_n}/\mathsf{self}]\}\rho\{Q\}} \qquad \text{(PC)}$$

Definition 4.28
Next we have a substitution rule to initialize instance variables:

$$\frac{\{p^{c_n}[z_{c_n}/\mathsf{self}]\}\rho\{Q\}}{\{(p^{c_n}[l/x])[z_{c_n}/\mathsf{self}]\}\rho\{Q\}} \qquad \text{(S1)}$$

provided the instance variable x does not occur in ρ or Q.

Definition 4.29
The following substitution rule initializes logical variables:

$$\frac{\{p^{c_n}[z_{c_n}/\mathsf{self}]\}\rho\{Q\}}{\{(p^{c_n}[l/z])[z_{c_n}/\mathsf{self}]\}\rho\{Q\}} \qquad \text{(S2)}$$

provided the logical variable z does not occur in Q.

Definition 4.30
The following rule is used to describe the initial state:

$$\frac{\{(p^{c_n} \wedge x \doteq \mathsf{nil})[z_{c_n}/\mathsf{self}]\}\rho\{Q\}}{\{p^{c_n}[z_{c_n}/\mathsf{self}]\}\rho\{Q\}} \qquad \text{(INIT)}$$

where $x \in \bigcup_c IVar_c^{c_n}$.

Definition 4.31
Finally, we have the following rule for *auxiliary variables*:

$$\frac{\{P\}\rho'\{Q\}}{\{P\}\rho\{Q\}} \qquad \text{(AUX)}$$

where ρ is obtained from ρ' by deleting all assignments to variables belonging to some set Aux, i.e. a set of auxiliary variables, such that for an arbitrary assignment $x \leftarrow e$ $(u \leftarrow e)$ occurring in ρ' we have that $ITvar(e) \cap Aux \neq \emptyset$ implies that $x \in Aux$ $(u \in Aux)$, moreover, the variables of the set Aux do not occur in tests of ρ' or in assignments $x \leftarrow s$ $(u \leftarrow s)$, s not a simple expression, and, finally, $IVar(Q) \cap Aux = \emptyset$.

The rule for auxiliary variables can be explained as follows: To be able to express some properties of a program ρ it may be necessary to add some assignments to new variables, which are called auxiliary variables. These assignments may not influence the flow of control of ρ, otherwise these auxiliary variables can not be used to express some properties of ρ. This requirement is formulated syntactically.

5 Semantics

In this section we define in a formal way the semantics of the programming language and the assertion languages. First, in section 5.1, we deal with the assertion languages on their own. Then, in section 5.2, we give a formal semantics to the programming language, making use of *transition systems*. Finally, section 5.3 formally defines the notion of truth of a correctness formula.

5.1 Semantics of the assertion languages

For every type $a \in C^\dagger$, we shall let $\mathbf{O}^a$ denote the set of objects of type a, with typical element α^a. To be precise, we define $\mathbf{O} = \mathbf{Z}$ and $\mathbf{O} = \mathbf{B}$, whereas for every class $c \in C$ we just take for $\mathbf{O}^c$ an arbitrary infinite set. With $\mathbf{O}^d_\perp$ we shall denote $\mathbf{O}^d \cup \{\perp\}$, where $\perp$ is a special element not in $\mathbf{O}^d$, which will stand for 'undefined', among others the value of the expression nil. Now for every type $d \in C^+$ we let $\mathbf{O}^{d^*}$ denote the set of all finite sequences of elements from $\mathbf{O}^d_\perp$ and we take $\mathbf{O}^{d^*}_\perp = \mathbf{O}^{d^*}$. This means that sequences can contain $\perp$ as a component, but a sequence can never be $\perp$ itself (as an expression of a sequence type, nil just stands for the empty sequence).

Definition 5.1
We shall often use generalized Cartesian products of the form

$$\prod_{i \in A} B(i).$$

As usual, the elements of this set are the functions f with domain A such that $f(i) \in B(i)$ for every $i \in A$.

Definition 5.2
Given a function $f \in A \to B$, $a \in A$, and $b \in B$, we use the *variant notation* $f\{b/a\}$ to denote the function in $A \to B$ that satisfies

$$f\{b/a\}(a') = \begin{cases} b & \text{if } a' = a \\ f(a') & \text{otherwise.} \end{cases}$$

Definition 5.3
The set *GState* of *global states*, with typical element σ, is defined as follows:

$$GState = (\prod_d P^d) \times \prod_c (\mathbf{O}^c \to \prod_d (IVar^c_d \to \mathbf{O}^d_\perp)) \times \prod_c (\mathbf{O}^c \to (\prod_d (TVar_d \to \mathbf{O}^d_\perp))^*)$$

where P^c, for every $c \in C$, denotes the set of finite subsets of $\mathbf{O}^c$, and for $d =$ Int, Bool we define $P^d = \{\mathbf{O}^d\}$.

A global state describes the situation of a complete system of objects at a certain moment during program execution. The first component specifies for each class the set of *existing* objects of that class, that is, the set of objects that have been created up to this point in the execution of the program. Relative to some global state σ an object $\alpha \in \mathbf{O}^d$ can be said to exist if $\alpha \in \sigma_{(1)(d)}$. For the built-in data types we have for every global state σ that $\sigma_{(1)}(\mathsf{Int}) = \mathbf{Z}$ and $\sigma_{(1)}(\mathsf{Bool}) = \mathbf{B}$. Note that $\perp \notin \sigma_{(1)(d)}$ for every $d \in C^+$. The second component of a global state specifies for each object the values of its instance variables. The third component specifies for each object a *stack* of local environments, i.e., functions assigning objects to temporary variables.

We introduce the following abbreviations:

- We abbreviate $\sigma_{(1)(d)}$ to $\sigma^{(d)}$.
- The local state of an object α in σ we will denote by $\sigma(\alpha)$, it consists of the assignment of objects to the instance variables and the temporary variables as given by $\sigma_{(2)}(\alpha)$, $Top(\sigma_{(3)}(\alpha))$, respectively. Furthermore, $\sigma(\alpha)(x)$ $(\sigma(\alpha)(u))$ will abbreviate $\sigma_{(2)(c,d)}(\alpha)(x)$ $(Top(\sigma_{(3)(c)}(\alpha))(d)(u))$, assuming the type of α to be c and that of x (u) to be d. Here $Top(< f_1, \ldots, f_n >) = f_n$.
- Furthermore, $\sigma\{\beta/\alpha, x\}$ will denote the state resulting from σ by assigning β to the variable x of α, and $\sigma\{\beta/\alpha, u\}$ will denote the state resulting from assigning β to the variable u of the top local environment of α, i.e., $Top(\sigma_{(3)}(\alpha))$.

Definition 5.4
The set $LState^c$ of *local states* of class c, with typical element θ, is defined by

$$LState^c = \mathbf{O}^c \times GState$$

Definition 5.5
We now define the set $LEnv$ of *logical environments*, with typical element ω, by

$$LEnv = \prod_a (LVar_a \to \mathbf{O}^a_\perp).$$

A logical environment assigns values to logical variables. We abbreviate $\omega_{(a)}(z_a)$ to $\omega(z_a)$.

Definition 5.6
The following semantic functions are defined in a straightforward manner. We omit most of the detail and only give the most important cases:

1. The function $\mathcal{E}_d^c \in Exp_d^c \to LState^c \to \mathbf{O}_\perp^d$ assigns a value $\mathcal{E}[\![e]\!](\theta)$ to the expression e_d^c in the local state θ^c. For example, $\mathcal{E}_d^c[\![\mathsf{nil}]\!](\theta) = \perp$ and $\mathcal{E}_d^c[\![x_d^c]\!](\langle\alpha,\sigma\rangle) = \sigma(\alpha)(x_d^c)$.

2. The function $\mathcal{L}_d^c \in LExp_d^c \to LEnv \to LState^c \to \mathbf{O}_\perp^d$ assigns a value $\mathcal{L}[\![l]\!](\omega)(\theta)$ to the local expression l_d^c in the logical environment ω and the local state θ^c.

3. The function $\mathcal{G}_a \in GExp_a \to LEnv \to GState \to \mathbf{O}_\perp^a$ assigns a value $\mathcal{G}[\![g]\!](\omega)(\sigma)$ to the global expression g_a in the logical environment ω and the global state σ.

4. The function $\mathcal{A}^c \in LAss^c \to LEnv \to LState^c \to \mathbf{B}$ assigns a value $\mathcal{A}[\![p]\!](\omega)(\theta)$ to the local assertion p^c in the logical environment ω and the local state θ^c. Here the following cases are special:

$$\mathcal{A}[\![l]\!](\omega)(\theta) = \begin{cases} \mathbf{t} & \text{if } \mathcal{L}[\![l]\!](\omega)(\theta) = \mathbf{t} \\ \mathbf{f} & \text{if } \mathcal{L}[\![l]\!](\omega)(\theta) = \mathbf{f} \text{ or } \mathcal{L}[\![l]\!](\omega)(\theta) = \perp \end{cases}$$

$$\mathcal{A}[\![\exists z_d\, p]\!](\omega)(\theta) = \begin{cases} \mathbf{t} & \text{if there is an } \alpha^d \in \mathbf{O}^d \text{ such that } \mathcal{A}[\![p]\!](\omega\{\alpha/z\})(\theta) = \mathbf{t} \\ \mathbf{f} & \text{otherwise} \end{cases}$$

Note that in the latter case $d = \mathsf{Int}$ or $d = \mathsf{Bool}$ and that the range of quantification *does not include* $\perp$.

5. The function $\mathcal{A} \in GAss \to LEnv \to GState \to \mathbf{B}$ assigns a value $\mathcal{A}[\![P]\!](\omega)(\sigma)$ to the global assertion P in the logical environment ω and the global state σ. The following cases are special:

$$\mathcal{A}[\![g]\!](\omega)(\sigma) = \begin{cases} \mathbf{t} & \text{if } \mathcal{L}[\![g]\!](\omega)(\sigma) = \mathbf{t} \\ \mathbf{f} & \text{if } \mathcal{L}[\![g]\!](\omega)(\sigma) = \mathbf{f} \text{ or } \mathcal{L}[\![g]\!](\omega)(\sigma) = \perp \end{cases}$$

$$\mathcal{A}[\![\exists z_d\, P]\!](\omega)(\sigma) = \begin{cases} \mathbf{t} & \text{if there is an } \alpha^d \in \sigma^{(d)} \text{ such that } \mathcal{A}[\![P]\!](\omega\{\alpha/z\})(\sigma) = \mathbf{t} \\ \mathbf{f} & \text{otherwise} \end{cases}$$

Note that here d can be any type in C^+ and that the quantification ranges over $\sigma^{(d)}$, the set of *existing* objects of type d (which does not include $\perp$).

$$\mathcal{A}[\![\exists z_{d^*}\, P]\!](\omega)(\sigma) = \begin{cases} \mathbf{t} & \text{if there is an } \alpha^{d^*} \in \mathbf{O}^{d^*} \text{ such} \\ & \text{that } \alpha(n) \in \sigma^{(d)} \cup \perp \text{ for all } n \in \mathbf{N} \\ & \text{and } \mathcal{A}[\![P]\!](\omega\{\alpha/z\})(\sigma) = \mathbf{t} \\ \mathbf{f} & \text{otherwise} \end{cases}$$

For sequence types, quantification ranges over those sequences of which every element is either $\perp$ or an existing object.

The values $\mathcal{G}[\![g_a]\!](\omega)(\sigma)$ of the global expression g_a and $\mathcal{A}[\![g]\!](\omega)(\sigma)$ of the global assertion P are in fact only meaningful for those ω and σ that are consistent and compatible:

Definition 5.7
We define the global state σ to be *consistent*, for which we use the notation $OK(\sigma)$ iff

$$\forall c \in C \forall \alpha \in \sigma^{(c)} \forall d \in C \forall x \in IVar_d^c \, \sigma(\alpha)(x) \in \sigma^{(d)} \cup \bot.$$

In other words, the value in σ of a variable of an existing object is either $\bot$ or an existing object itself.

Furthermore we define the logical environment ω to be *compatible* with the global state σ, with the notation $OK(\omega,\sigma)$, iff $OK(\sigma)$ and, additionally,

$$\forall d \in C \forall z \in LVar_d \, (\omega(z) \in \sigma^{(d)} \cup \{\bot\})$$

and

$$\forall d \in C \forall z \in LVar_{d^*} \forall a \in \mathbf{N} \, (\omega(z)(n) \in \sigma^{(d)} \cup \{\bot\}).$$

In other words, ω assigns to every logical variable z_d of a simple type the value $\bot$ or an existing object, and to every sequence variable z_{d^*} a sequence of which each element is an existing object or equals $\bot$.

5.2 The transition system

We will describe the internal behaviour of an object by means of a transition system. A *local configuration* we define to be a pair (S^c, θ^c). The set of local configurations is denoted by $LConf$. Let $Rec = \{< \alpha,\beta >, < m,\bar{\beta} >, < m^O,\bar{\beta} >, < m^I,\bar{\beta} >: \alpha,\beta \in \bigcup_c \mathbf{O}^c$, $\bar{\beta}$ denoting a sequence of objects$\} \cup \{\epsilon\}$. A pair $< \alpha,\beta >$ is called an *activation record.* It records the information that the object α created β. Sequences of the form $< m,\bar{\beta} >, < m^I,\bar{\beta} >$, and $< m^O,\bar{\beta} >$ are called *communication records.* A record $< m,\beta_0,\ldots,\beta_n >$, with n the number of formal parameters of m, records the information that the method m has been sent by β_1 to β_0 with actual parameters $\beta_1,\ldots,\beta_n$. (Remember that the identity of the sender is sent as the first actual parameter.) On the other hand a record $< m,\beta_1,\ldots,\beta_n >$, with again n the number of formal parameters, records the information that the method m has been received with actual parameters $\beta_1,\ldots,\beta_n$. A record $< m^O,\beta_1,\beta_2 >$ records the information that the result of the method m, the object β_2, has been sent to β_1. A record $< m^I,\beta_1,\beta_2 >$ records the information that the result of the method m, the object β_2, has been received from β_1.

We define for every $r \in Rec$ a transition relation $\rightarrow^r \subseteq LConf \times LConf$. (In fact we define $\rightarrow^r$ given a unit U.) To facilitate the semantics we introduce the auxiliary statement E, the empty statement, to denote termination, and the statements

$\mathsf{send}(m, e, \alpha)$ and $\mathsf{wait}(m, \beta)$. The statement $\mathsf{send}(m, e, \alpha)$ will model the process of sending the result of m, the value of e, to α. The statement $\mathsf{wait}(m, \beta)$ will model the process of waiting for the object β to send the result of m. Furthermore, we introduce the operations *Push* and *Pop*:

$$\begin{aligned} Push(< f_1, \ldots, f_n >, f) &=< f_1, \ldots, f_n, f > \\ Pop(< f_1, \ldots, f_n >) &=< f_1, \ldots, f_{n-1} > . \end{aligned}$$

Definition 5.8
Let $\theta = \langle \alpha, \sigma \rangle$. We define

- $(x_d \leftarrow e_d, \theta) \rightarrow^{\epsilon} (E, \theta')$,
  where $\theta' = \langle \alpha, \sigma\{\mathcal{E}[\![e_d]\!]\ (\theta)/\alpha, x\} \rangle$.
- $(u_d \leftarrow e_d, \theta) \rightarrow^{\epsilon} (E, \theta')$,
  where $\theta' = \langle \alpha, \sigma\{\mathcal{E}[\![e_d]\!]\ (\theta)/\alpha, u\} \rangle$.
- $(x_d := \mathsf{new}, \theta) \rightarrow^{<\alpha,\beta>} (E, \theta')$,
  where $\theta' = \langle \alpha, \sigma'\{\beta/\alpha, x_d\}\{\bot/\beta, y\}_{y \in IVar^d} \rangle$, and

  $$\sigma' = (\sigma_{(1)}\{\sigma^{(d)} \cup \{\beta\}\}, \sigma_{(2)}, \sigma_{(3)}),$$

  and $\beta \in \mathbf{O}^d \setminus \sigma^{(d)}$.
- $(u_d := \mathsf{new}, \theta) \rightarrow^{<\alpha,\beta>} (E, \theta')$,
  where $\theta' = \langle \alpha, \sigma'\{\beta/\alpha, u_d\}\{\bot/\beta, y\}_{y \in IVar^d} \rangle$, and

  $$\sigma' = (\sigma_{(1)}\{\sigma^{(d)} \cup \{\beta\}\}, \sigma_{(2)}, \sigma_{(3)}),$$

  and $\beta \in \mathbf{O}^d \setminus \sigma^{(d)}$.
- $(x \leftarrow e_0!m(e_1, \ldots, e_n), \theta) \rightarrow^{<m,\bar{\beta}>} (x \leftarrow \mathsf{wait}(m, \beta_0), \theta)$,
  where $\beta_i = \mathcal{E}[\![e_i]\!]\ (\theta)$, and $\bar{\beta} = \beta_0, \ldots, \beta_n$.
- $(u \leftarrow e_0!m(e_1, \ldots, e_n), \theta) \rightarrow^{<m,\bar{\beta}>} (u \leftarrow \mathsf{wait}(m, \beta_0), \theta)$,
  where $\beta_i = \mathcal{E}[\![e_i]\!]\ (\theta)$, and $\bar{\beta} = \beta_0, \ldots, \beta_n$.
- $(x \leftarrow \mathsf{wait}(m, \beta_0), \theta) \rightarrow^{<m^I,\beta_0,\gamma>} (E, \theta')$,
  where $\theta' = \langle \alpha, \sigma\{\gamma/\alpha, x_d\} \rangle$, with γ an arbitrary element of $\mathbf{O}^d$.
- $(u \leftarrow \mathsf{wait}(m, \beta_0), \theta) \rightarrow^{<m^I,\beta_0,\gamma>} (E, \theta')$,
  where $\theta' = \langle \alpha, \sigma\{\gamma/\alpha, x_d\} \rangle$.

- $(\mathsf{answer}(\ldots, m, \ldots), \theta) \rightarrow^{<m,\bar{\beta}>} (S; \mathsf{send}(m, e, \beta_1), \theta')$,
  where $\theta' = \langle \alpha, \sigma\{Push(\sigma_{(3)}(\alpha), f)/\alpha\}\rangle$, and

$$f(u_i) = \beta_i \quad u_i \in \{u_1, \ldots, u_n\}$$
$$f(u) = \perp \quad u \notin \{u_1, \ldots, u_n\}.$$

  Furthermore,

$$\sigma\{Push(\sigma_{(3)}(\alpha), f)/\alpha\} = (\sigma_{(1)}, \sigma_{(2)}, \sigma_{(3)}\{Push(\sigma_{(3)}(\alpha), f)/\alpha\}).$$

  The sequence of actual parameters is chosen arbitrarily. (Here we assume $u_1, \ldots, u_n$ to be the formal parameters of m, The statement S to be its body, and e to be its result expression.)

- $(\mathsf{send}(m, e, \beta), \theta) \rightarrow^{<m^O,\beta,\gamma>} (E, \theta')$,
  where $\theta' = \langle \alpha, \sigma\{Pop(\sigma_{(3)}(\alpha)/\alpha\}\rangle$, and $\gamma = \mathcal{E}[\![e]\!](\theta)$. Here

$$\sigma\{Pop(\sigma_{(3)}(\alpha))/\alpha\} = (\sigma_{(1)}, \sigma_{(2)}, \sigma_{(3)}\{Pop(\sigma_{(3)}(\alpha))/\alpha\}).$$

- $(\mathsf{l}; S, \theta) \rightarrow^{\epsilon} (S, \theta)$.

- $$\frac{(S_1, \theta) \rightarrow^{r} (S_2, \theta') \mid (E, \vartheta')}{(S_1; S, \theta) \rightarrow^{r} (S_2; S, \theta') \mid (S, \vartheta')}$$

- $(\mathsf{if}\ e_{\mathsf{Bool}}\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}, \theta) \rightarrow^{\epsilon} (S_1, \theta)$,
  if $\mathcal{E}[\![e_{\mathsf{Bool}}]\!]\ (\theta) = \mathsf{true}$.

- $(\mathsf{if}\ e_{\mathsf{Bool}}\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}, \theta) \rightarrow^{\epsilon} (S_2, \theta)$,
  if $\mathcal{E}[\![e_{\mathsf{Bool}}]\!]\ (\theta) = \mathsf{false}$.

- $(\mathsf{while}\ e_{\mathsf{Bool}}\ \mathsf{do}\ S\ \mathsf{od}, \theta) \rightarrow^{\epsilon} (S; \mathsf{while}\ e_{\mathsf{Bool}}\ \mathsf{do}\ S\ \mathsf{od}, \theta)$,
  if $\mathcal{E}[\![e_{\mathsf{Bool}}]\!]\ (\theta) = \mathsf{true}$.

- $(\mathsf{while}\ e_{\mathsf{Bool}}\ \mathsf{do}\ S\ \mathsf{od}, \theta) \rightarrow^{\epsilon} (E, \theta)$,
  if $\mathcal{E}[\![e_{\mathsf{Bool}}]\!]\ (\theta) = \mathsf{false}$.

We define $\rightarrow^{h} = TC(\bigcup_{r \in Rec} \rightarrow^{r})$. Here the operation TC denotes the transitive and reflexive closure which composes additionally the communication records and activation records into a *history* h, a sequence of communication records and activation records.

Using the above transition system we define another transition relation $\rightarrow_L$ which *hides* the computations within the bracketed sections.

Definition 5.9

- $\dfrac{(S_1,\theta_1) \rightarrow^r (S_2,\theta_2)}{(S_1,\theta_1) \rightarrow_L (S_2,\theta_2)}$,
  where S_1 is not of the form $\mathsf{l}; S; \mathsf{l}'$.
- $\dfrac{(S,\theta_1) \rightarrow^h (E,\theta_2)}{(\mathsf{l}; S; \mathsf{l}',\theta_1) \rightarrow_L (\mathsf{l}',\theta_2)}$.

The semantics of statements and local correctness formulas will be defined with respect to this transition relation $\rightarrow_L$.

Next we describe the behaviour of several objects working in parallel. The local behaviour of the objects we shall derive from the local transition system as described above. But at this level we have the necessary information to select the right choices concerning the communications.

We define an *intermediate configuration* to be a tuple $(\sigma,(\alpha_i,S_i^{c_i})_i)$, where $\alpha_i \in \sigma^{(c_i)}$, assuming all the α_i to be distinct. The set of intermediate configurations will be denoted by *IConf*. We define $\rightarrow^r \subseteq \mathit{IConf} \times \mathit{IConf}$, with $r =< m,\vec{\beta} >$, as follows (note that we use the same notation as for the local transition relation, however this will cause no harm):

Definition 5.10
We define

- $\dfrac{(S_j,\langle\alpha_j,\sigma\rangle) \rightarrow^r (S_j',\langle\alpha_j,\sigma'\rangle)}{(\sigma,(\alpha_i,S_i)_i) \rightarrow^r (\sigma',(\alpha_i,S_i')_i)}$
  where $r = \epsilon, < \alpha_j,\beta >$
  $$\begin{aligned} S_i' &= S_i \quad i \neq j \\ &= S_i' \quad \text{otherwise.} \end{aligned}$$
- $\dfrac{(S_j,\langle\alpha_j,\sigma\rangle) \rightarrow^r (S_j',\langle\alpha_j,\sigma_1\rangle),\ (S_k,\langle\alpha_k,\sigma\rangle) \rightarrow^{r'} (S_k',\langle\alpha_k,\sigma_2\rangle)}{(\sigma,(\alpha_i,S_i)_i) \rightarrow^{r'} (\sigma',(\alpha_i,S_i')_i)}$
  where $j \neq k$ and $r =< m,\beta_1,\ldots,\beta_n >$, $r' =< m,\alpha_j,\beta_1,\ldots,\beta_n >$, furthermore, we have
  $$\begin{aligned} S_i' &= S_i \quad i \neq j,k \\ &= S_i' \quad i = j,k, \end{aligned}$$
  and $\sigma' = \sigma\{\sigma_1(\alpha_j)/\alpha_j\}\{\sigma_2(\alpha_k)/\alpha_k\}$. (Here $\sigma\{\sigma_1(\alpha_j)/\alpha_j\}\{\sigma_2(\alpha_k)/\alpha_k\}$ denotes the state resulting from changing the local state of α_j (α_k) to $\sigma_1(\alpha_j)$ $(\sigma_2(\alpha_k))$.)
- $\dfrac{(S_j,\langle\alpha_j,\sigma\rangle) \rightarrow^r (S_j',\langle\alpha_j,\sigma_1\rangle),\ (S_k,\langle\alpha_k,\sigma\rangle) \rightarrow^{r'} (S_k',\langle\alpha_k,\sigma_2\rangle)}{(\sigma,(\alpha_i,S_i)_i) \rightarrow^{r''} (\sigma',(\alpha_i,S_i')_i)}$
  where $j \neq k$ and $r =< m^O,\alpha_k,\gamma >$, $r' =< m^I,\alpha_j,\gamma >$ and $r'' =< m,\alpha_j,\gamma >$,

furthermore, we have

$$\begin{aligned} S_i' &= S_i \quad i \neq j,k \\ &= S_i' \quad i = j,k, \end{aligned}$$

and $\sigma' = \sigma\{\sigma_1(\alpha_j)/\alpha_j\}\{\sigma_2(\alpha_k)/\alpha_k\}$.

The first rule above selects one object and its local state and uses the local transition system to derive one local step of this object. The second and third rule select two objects which are ready to communicate with each other. Note that we use a different interpretation of a communication record now: A record $< m, \alpha, \beta_1, \ldots, \beta_n >$, with n the number of formal parameters of m, records the information that α has received the method m with actual parameters $\beta_1, \ldots, \beta_n$, where β_1 in fact denotes the sender. Furthermore, $< m, \alpha, \gamma >$ will now record the information that α has received the result of m, the object γ.

We define $\rightarrow^h = TC(\bigcup_{r \in Rec} \rightarrow^r)$, again using the same notation as for the transitive and reflexive closure of the local transition relation, from the context however it should be clear which one is meant. This new transition relation will be used to define the semantics of intermediate correctness formulas.

To describe the behaviour of a complete system we introduce the notion of a *global configuration*: a pair (X, σ), where $X \in \prod_c \mathbf{O}^c \rightarrow \mathit{Stat}^c$, and a transition relation $\rightarrow^r \subseteq \mathit{GConf} \times \mathit{GConf}$. We note again that we do not notationally distinguish between the different transition relations, from the context however it will be clear which one is meant. The set of all global configurations we denote by *GConf*. We will abbreviate in the sequel $X_{(c)}(\alpha)$, for $\alpha \in \mathbf{O}^c$, by $X(\alpha)$. The idea is that $X(\alpha)$ denotes the statement to be executed by α.

Definition 5.11
We have the following rule

$$\frac{(\sigma, (\alpha_i, X(\alpha_i))_i) \rightarrow^r (\sigma', (\alpha_i, S_i^{c_i})_i)}{(X, \sigma) \rightarrow^r (X', \sigma')}$$

where $\alpha_i \in \sigma^{(c_i)}$ (all the α_i distinct) and $X' = X\{S_i^{c_i}/\alpha_i\}_i$.

This rule selects some finite set of objects which execute in parallel according to the previous transition system.

We define $\rightarrow^h = TC(\bigcup_{r \in Rec} \rightarrow^r)$. This transition relation will be used to define the semantics of programs and global correctness formulas.

We proceed with the following definition which characterizes the set of *initial* and *final* global configurations of a given program ρ:

Definition 5.12
Let $\rho^c = \langle U^{c_1,\ldots,c_{n-1}}_{m_1,\ldots,m_k} | c_n : S^{c_n}_n \rangle$, with $U^{c_1,\ldots,c_{n-1}}_{m_1,\ldots,m_k} = D_1{}^{c_1}_{\bar{m}_1}, \ldots, D_{n-1}{}^{c_{n-1}}_{\bar{m}_{n-1}}$, where $D_i{}^{c_i}_{\bar{m}_i} = \langle m^i_1{}^{c_i}_{\bar{d}^i_1} \Leftarrow \mu^i_1{}^{c_i}_{\bar{d}^i_1}, \ldots, m_{n_i}{}^{c_i}_{\bar{d}^i_{n_i}} \Leftarrow \mu_{n_i}{}^{c_i}_{\bar{d}^i_{n_i}} \rangle : S^{c_i}_i$. Furthermore let $X \in \prod_c \mathbf{O}^c \rightarrow Stat^c$. We define

$$Init_\rho(X) \text{ iff}$$

- $X(\alpha) = S^{c_i}_i$, $c_i \in \{c_1, \ldots, c_n\}$, $\alpha \in \mathbf{O}^{c_i}$.
- $X(\alpha) = E$, $\alpha \in \mathbf{O}^c$, $c \notin \{c_1, \ldots, c_n\}$.

We define for a state σ such that $OK(\sigma)$:

$$Init_\rho(\sigma) \text{ iff}$$

- $\sigma^{(c)} = \emptyset \quad c \in \{c_1, \ldots, c_{n-1}\}$
  $= \{\alpha\} \quad c = c_n$, for some $\alpha \in \mathbf{O}^{c_n}$
- $\sigma(\alpha)(x) = \perp$, for $\alpha \in \sigma^{(c_n)}$ and $x \in IVar^{c_n}_{c_n}$.

We next define $Init_\rho((X,\sigma))$ iff $Init_\rho(X)$ and $Init_\rho(\sigma)$. We define

$$Final_\rho((X,\sigma)) \text{ iff } X(\alpha) = E, c_i \in \{c_1, \ldots, c_n\}, \text{ for } \alpha \in \sigma^{(c_i)}.$$

The predicate $Init_\rho(Final_\rho)$ characterizes the set of *initial* (*final*) configurations of ρ. Note that the value of a variable $x^{c_n}_c$ of the root-object, $c \in \{c_1, \ldots, c_n\}$, is undefined initially. This follows for $c \neq c_n$ from the fact that we consider only consistent states and that initially only the root-object exists (with respect to the classes $c_1, \ldots, c_n$). But the consistency of the initial state would also allow the value of a variable $x \in IVar^{c_n}_{c_n}$ to be the root-object itself. However, as it will appear to be convenient with respect to the formulation of some rules which formalize reasoning about the initial state, we define the initial state to be completely specified by the variables ranging over the standard objects.

Now we are able to define the meaning of the following programming constructs: S^c, (z_c, S^c), $(z_{c_i}, S^{c_i}_1) \parallel (z_{c_j}, S^{c_j}_2)$ and ρ.

Definition 5.13
We define

$$\mathcal{S}[\![S^c]\!](\theta) = \{< (S_1,\theta_1), \ldots, (S_n,\theta_n) >: (S_i,\theta_i) \rightarrow_L (S_{i+1},\theta_{i+1}),\ 1 \leq i < n,\ S_1 = S, \theta_1 = \theta\}$$

Definition 5.14
We define $\mathcal{I}[\![(z,S)]\!](\omega)(\sigma_1) = \emptyset$ if not $OK(\omega,\sigma_1)$, and $\mathcal{I}[\![(z_1,S_1) \parallel (z_2,S_2)]\!](\omega)(\sigma_1) = \emptyset$ if not $OK(\omega,\sigma_1)$ or $\omega(z_1) = \omega(z_2)$. So assume from now on that $OK(\omega,\sigma_1)$, furthermore that $\omega(z) = \alpha$, $\omega(z_i) = \alpha_i$.

$$\mathcal{I}[\![(z,S)]\!](\omega)(\sigma_1) = \{\sigma_2 : \text{ for some } h\ (\sigma_1,(\alpha,S)) \rightarrow^h (\sigma_2,(\alpha,E))\}$$

Assuming furthermore that $\alpha_1 \neq \alpha_2$:

$$\mathcal{I}[\![(z_1,S_1) \parallel (z_2,S_2)]\!](\omega)(\sigma_1) =$$
$$\{\sigma_2 : \text{ for some } h\ (\sigma_1,(\alpha_1,S_1),(\alpha_2,S_2)) \rightarrow^h (\sigma_2,(\alpha_1,E),(\alpha_2,E))\}$$

Definition 5.15
The semantics of programs is defined as follows:

$$\mathcal{P}[\![\rho]\!](\sigma_1) = \{\sigma_2 : \text{for some } h\ (X_1,\sigma_1) \rightarrow^h (X_2,\sigma_2)\}$$

where $Init_\rho((X_1,\sigma_1))$ and $Final_\rho((X_2,\sigma_2))$.

Note that $\mathcal{P}[\![\rho]\!](\sigma) = \emptyset$ if it is not the case that $Init_\rho(\sigma)$.

5.3 Truth of correctness formulas

In this section we define formally the truth of the local, intermediate, and global correctness formulas, respectively. First we define the truth of local correctness formulas.

Definition 5.16
We define

$$\models (A,C : \{p^c\}S^c\{q^c\}) \text{ iff}$$

for an arbitrary ω and $< (S_1,\theta_1),\ldots,(S_n,\theta_n) > \in \mathcal{S}[\![S]\!](\theta_1)$:

- $\theta_1,\omega \models p$
- $\theta_i,\omega \models A(Lab(S_i))$, $1 \leq i \leq n$

implies

$$\theta_n,\omega \models C(Lab(S_n)) \text{ and if } S_n = E \text{ then } \theta_n,\omega \models q.$$

Here

$$\begin{aligned} Lab(S) &= \mathsf{l} \quad \text{if } S = \mathsf{l}; S' \\ &= \emptyset \quad \text{otherwise} \end{aligned}$$

(Note that for X a set of labeled assertions we have $X(\emptyset) = \textsf{true}$.)

Next we define the truth of intermediate correctness formulas.

Definition 5.17
We define

$$\models \{P\}(z_c, S^c)\{Q\} \text{ iff}$$
$$\forall\omega\forall\sigma_1\forall\sigma_2 \in \mathcal{I}[\![(z_c, S^c)]\!](\omega)(\sigma_1) : \ \sigma_1, \omega \models P \ \Rightarrow \ \sigma_2, \omega \models Q.$$

And

$$\models \{P\}(z_{c_i}, S^{c_i}) \parallel (z'_{c_j}, S^{c_j})\{Q\} \text{ iff}$$
$$\forall\omega\forall\sigma_1\forall\sigma_2 \in \mathcal{I}[\![(z_{c_i}, S^{c_i}) \parallel (z'_{c_j}, S^{c_j})]\!](\omega)(\sigma_1) : \ \sigma_1, \omega \models P \ \Rightarrow \ \sigma_2, \omega \models Q.$$

Finally, we define the truth of global correctness formulas.

Definition 5.18
We define

$$\models \{P\}\rho\{Q\} \text{ iff } \forall\omega\forall\sigma_1\forall\sigma_2 \in \mathcal{P}[\![\rho]\!](\sigma_1) : \ \sigma_1, \omega \models P \ \Rightarrow \ \sigma_2, \omega \models Q.$$

6 Soundness

In this section we prove the soundness of the proof system as presented in the previous section. We first discuss the soundness of the local proof system.

6.1 The local proof system

The soundness of the local proof system is proved by induction on the length of the derivation. We treat the rule BA, the other axioms and rules being straightforward to deal with.

Lemma 6.1
If

$$\models (\emptyset,\emptyset : \{p[\bar{y}/\bar{u}]\}S_i\{p[\bar{y}/\bar{u}]\})$$

then

$$\models (A, C : \{p \wedge C(\mathsf{l}_1)\}\mathsf{l}_1; \mathsf{answer}(m_1,\ldots,m_n); \mathsf{l}_2\{p \wedge A(\mathsf{l}_2)\}).$$

Proof
Let $R = \mathsf{l}_1; \mathsf{answer}(m_1,\ldots,m_n); \mathsf{l}_2$ and let $< (R,\theta),(\mathsf{l}_2,\theta') > \in \mathcal{S}[\![R]\!](\theta)$ such that

- $\theta,\omega \models p \wedge C(l_1)$,
- $\theta',\omega \models A(l_2)$.

It suffices to prove that $\theta',\omega \models p$. Let $(R,\theta_1) \rightarrow^{r_1} \ldots \rightarrow^{r_{k-1}} (\mathsf{l}_2,\theta_k)$, with $\theta_1 = \theta$ and $\theta_k = \theta'$. We next define for $1 \leq i \leq k$, $\theta'_i = \langle \alpha, \sigma'_i \rangle$ (assuming $\theta_i = \langle \alpha, \sigma_i \rangle$), with $\sigma'_i = \sigma_i\{\sigma_1(\alpha)(\bar{u})/\alpha,\bar{y}\}$. It then follows that: $(R,\theta'_1) \rightarrow^{r_1} \ldots \rightarrow^{r_{k-1}} (\mathsf{l}_2,\theta'_k)$ (note that $\bar{y}$ are some new variables not occurring in the body of m_i, $1 \leq i \leq n$). We have for some $1 \leq i \leq n$ that $(S_i,\theta'_2) \rightarrow^{r_2} \ldots \rightarrow^{r_{k-2}} (E,\theta'_{k-1})$ and $\theta'_2,\omega \models p[\bar{y}/\bar{u}]$, with S_i the body of method m_i: σ'_2 results from σ'_1 by creating a new local environment for the execution of S_i by α, and σ'_k is obtained from σ'_{k-1} by popping the stack of local environments associated with α. (Note that σ'_2 and σ'_1 agree with respect to the instance variables of α). From this and $\models (\emptyset,\emptyset : \{p[\bar{y}/\bar{u}]\}S_i\{p[\bar{y}/\bar{u}]\})$ we infer that $\theta'_{k-1},\omega \models p[\bar{y}/\bar{u}]$. Since $p[\bar{y}/\bar{u}]$ contains no temporary variables and σ'_k and σ'_{k-1} agree with respect to the instance variables of α we have $\theta'_k,\omega \models p[\bar{y}/\bar{u}]$, or, equivalently, $\theta_k,\omega \models p$ (note that $\sigma'_k(\alpha)(\bar{y}) = \sigma'_1(\alpha)(\bar{y}) = \sigma_1(\alpha)(\bar{u}) = \sigma'_1(\alpha)(\bar{u}) = \sigma'_k(\alpha)(\bar{u})$). We conclude $\theta',\omega \models p$. □

6.2 The intermediate proof system

In this subsection we discuss the soundness of the intermediate proof system. We prove the soundness of the assignment axiom (IASS) and the axiom (NEW). The soundness of the intermediate proof system then follows by a straightforward induction argument. To prove the soundness of the assignment axiom (IASS) we need the following lemma about the correctness of the corresponding substitution operation. This lemma states that semantically substituting the expression g' for $z.x$ in an assertion (expression) yields the same result when evaluating the assertion (expression) in the state where the value of g' is assigned to the variable x of the object denoted by z.

Lemma 6.2
For an arbitrary σ, ω such that $OK(\omega, \sigma)$ we have:

$$\mathcal{G}[\![g[g'_d/z_c.x_d]]\!](\omega)(\sigma) = \mathcal{G}[\![g]\!](\omega)(\sigma')$$

and

$$\mathcal{A}[\![P[g'_d/z_c.x_d]]\!](\omega)(\sigma) = \mathcal{A}[\![P]\!](\omega)(\sigma')$$

where $\sigma' = \sigma\{\mathcal{G}[\![g'_d]\!](\omega)(\sigma)/\omega(z_c), x_d\}$.

Proof
By induction on the complexity of g and P. We treat only the case $g = g_1.x$, all the other ones following directly from the induction hypothesis. Now:

$$\begin{aligned}&\mathcal{G}[\![g[g'_d/z_c.x_d]]\!](\omega)(\sigma) = \\ &\mathcal{G}[\![\textsf{if } g_1[g'_d/z_c.x_d] \doteq z_c \textsf{ then } g' \textsf{ else } g_1[g'/z_c.x].x \textsf{ fi}]\!](\omega)(\sigma)\end{aligned}$$

Suppose that $\mathcal{G}[\![g_1[g'_d/z_c.x_d]]\!](\omega)(\sigma) = \omega(z_c)$. We have: $\mathcal{G}[\![g_1.x]\!](\omega)(\sigma') = \sigma'(\mathcal{G}[\![g_1]\!](\omega)(\sigma'))(x)$. So by the induction hypothesis we have that:

$$\mathcal{G}[\![g_1.x]\!](\omega)(\sigma') = \sigma'(\omega(z_c))(x) = \mathcal{G}[\![g'_d]\!](\omega)(\sigma).$$

On the other hand if $\mathcal{G}[\![g_1[g'_d/z_c.x_d]]\!](\omega)(\sigma) \neq \omega(z_c)$ then:

$$\begin{aligned}&\mathcal{G}[\![g_1[g'_d/z_c.x_d].x_d]\!](\omega)(\sigma) && = \\ &\sigma(\mathcal{G}[\![g_1[g'_d/z_c.x_d]]\!](\omega)(\sigma))(x_d) && = \text{ (definition of } \sigma') \\ &\sigma'(\mathcal{G}[\![g_1[g'_d/z_c.x_d]]\!](\omega)(\sigma))(x_d) && = \text{ (induction hypothesis)} \\ &\sigma'(\mathcal{G}[\![g_1]\!](\omega)(\sigma'))(x_d) && = \\ &\mathcal{G}[\![g_1.x_d]\!](\omega)(\sigma'). &&\end{aligned}$$

□

The following lemma states the soundness of the axiom (IASS)

Lemma 6.3
We have

$$\models \Big\{P[e[z/\mathsf{self}]/z.x]\Big\}(z, x \leftarrow e)\Big\{P\Big\},$$

where we assume $x \leftarrow e \in Stat^c$ and $z \in LVar_c$.

Proof
Let σ,ω, with $OK(\omega,\sigma)$, such that $\sigma,\omega \models P[e[z/\mathsf{self}]/z.x]$ and $\sigma' \in \mathcal{I}[\![(z, x \leftarrow e)]\!](\omega)(\sigma)$. It follows that $\sigma' = \sigma\{\mathcal{G}[\![e[z/\mathsf{self}]]\!](\omega)(\sigma)/\omega(z), x\}$ (note that $\mathcal{G}[\![e[z/\mathsf{self}]]\!](\omega)(\sigma) = \mathcal{E}[\![e]\!](\langle\alpha, \sigma(\alpha)\rangle)$, with $\alpha = \omega(z)$). Thus by the previous lemma we conclude $\sigma', \omega \models P$.
□

To prove the soundness of the axiom describing the **new** statement we need the following lemma which states the correctness of the corresponding substitution operation. This lemma states that semantically the substitution $[\mathsf{new}/z]$ applied to an assertion yields the same result when evaluating the assertion in the state resulting from the creation of a new object, interpreting the variable z as the newly created object.

Lemma 6.4
For an arbitrary $\omega, \omega', \sigma, \sigma', \beta \in \mathbf{O}^c \backslash \sigma^{(c)}$ such that $OK(\omega,\sigma)$ and

$$\sigma' = (\sigma_{(1)}\{\sigma^{(c)} \cup \{\beta\}/c\}, \sigma_{(2)}\{\perp/\beta, y\}_{y \in IVar^c}, \sigma_{(3)})$$

and $\omega' = \omega\{\beta/z_c\}$, we have for an arbitrary assertion P:

$$\mathcal{A}[\![P[\mathsf{new}/z_c]]\!](\omega)(\sigma) = \mathcal{A}[\![P]\!](\omega')(\sigma').$$

The proof of this lemma proceeds by induction on the structure of P. To carry out this induction argument, which we trust the interested reader to be able to perform, we need the following two lemmas. The first of which is applied to the case $P = g$ and the second of which is applied to the case $P = \exists z P'$, $z \in LVar_a$, $a = c, c^*$.

Lemma 6.5
For an arbitrary σ, ω, with $OK(\omega,\sigma)$, global expression g and logical variable z_c such that $g[\mathsf{new}/z_c]$ is defined we have:

$$\mathcal{G}[\![g]\!](\omega')(\sigma') = \mathcal{G}[\![g[\mathsf{new}/z_c]]\!](\omega)(\sigma)$$

where $\sigma' = (\sigma_{(1)}\{\sigma^{(c)} \cup \{\beta\}/c\}, \sigma_{(2)}\{\perp/\beta, y\}_{y \in IVar^c}, \sigma_{(3)})$ and $\omega' = \omega\{\beta/z_c\}$, $\beta \notin \sigma^{(c)}$.

Proof
Induction on the structure of g. □

The following lemma states that semantically the substitution $[z_{\mathsf{Bool}^*}, z_c/z_{c^*}]$ applied to an assertion (expression) yields the same result when updating the sequence denoted by the variable z_{c^*} to the value of z_c at those positions for which the sequence denoted by z_{Bool^*} gives the value **true**.

Lemma 6.6
Let ω, σ, $\alpha = \omega(z_{c^*}), \alpha' = \omega(z_{\mathsf{Bool}^*})$ such that $|\alpha| = |\alpha'|$ and $OK(\omega, \sigma)$.

Let $\alpha'' \in \mathbf{O}^{c^*}$ such that

- $|\alpha''| = |\alpha|$
- for $n \in \mathbf{N}$: $\alpha''(n) = \omega(z_c)$ if $\alpha'(n) = \mathsf{true}$
  $= \alpha(n)$ if $\alpha'(n) = \mathsf{false}$
  $= \perp$ if $\alpha'(n) = \perp$

Let $\omega' = \omega\{\alpha''/z_{c^*}\}$. Then:

1. For every g such that $g[z_{\mathsf{Bool}^*}, z_c/z_{c^*}]$ is defined:
$$\mathcal{G}[\![g[z_{\mathsf{Bool}^*}, z_c/z_{c^*}]]\!](\omega)(\sigma) = \mathcal{G}[\![g]\!](\omega')(\sigma)$$
2. For every P such that z_{Bool^*} does not occur in it:
$$\mathcal{A}[\![P[z_{\mathsf{Bool}^*}, z_c/z_{c^*}]]\!](\omega)(\sigma) = \mathcal{A}[\![P]\!](\omega')(\sigma)$$

Proof
Induction on the structure of g and P. □

Now we are ready to prove the soundness of the axiom (NEW).

Lemma 6.7
We have
$$\models \Big\{P[z'/z.x][\mathsf{new}/z']\Big\}(z, x := \mathsf{new})\Big\{P\Big\},$$
where $x := \mathsf{new} \in \mathit{Stat}^c$, $z \in \mathit{LVar}_c$, and z' is a new logical variable of the same type as x.

Proof
Let σ, ω, with $OK(\sigma, \omega)$, such that $\sigma, \omega \models P[z'/z.x][\mathsf{new}/z']$ and $\sigma' \in \mathcal{I}[\![(z, x := \mathsf{new})]\!](\omega)(\sigma)$. We have by lemma 6.4 that $\sigma'', \omega' \models P[z'/z.x]$, where $\omega' = \omega\{\beta/z'\}$, with $\beta \in \mathbf{O}^d \setminus \sigma^d$, assuming d to be the type of the variable x, and $\sigma'' = (\sigma_{(1)}\{\sigma^{(d)} \cup \{\beta\}/d\}, \sigma_{(2)}\{\perp/\beta, y\}_{y \in \mathit{IVar}^c}, \sigma_{(3)})$. Now by lemma 6.2 it follows that $\sigma', \omega' \models P$. Finally, as z' does not occur in P we have $\sigma', \omega \models P$. □

6.3 The global proof system

In this subsection we prove the soundness of the global proof system. We will prove only the soundness of the rule (PR), the other rules being straightforward to deal with. We first introduce some definitions.

Definition 6.8
We call a global configuration (X, σ) *stable* iff there exists no object executing inside a bracketed section, more precisely, no object is executing a prelude or postlude of a bracketed section containing a new or send-statement, or the prelude, postlude of the body of a method. Finally, there exists no object which has finished executing the prelude of a bracketed section containing a send-statement but has not yet send its actual parameters.

Definition 6.9
We call a global computation of ρ, i.e., a sequence $(X_1, \sigma_1), \ldots, (X_n, \sigma_n)$ such that $Init_\rho((X_1, \sigma_1))$, and for $1 \leq i < n$ we have $(X_i, \sigma_i) \rightarrow^{r_i} (X_{i+1}, \sigma_{i+1})$, for some record r_i, *regular* if in every configuration (X_i, σ_i) at most one object is executing a bracketed section containing a **new**-statement or belonging to an **answer**-statement (the prelude or postlude of the body of one of its methods).

We observe that every terminating computation of a program ρ (ρ arbitrary) can be rearranged into an equivalent (with respect to the local behaviour of the objects) regular one.

It is not difficult to see that the following lemma implies the soundness of the rule (PR):

Lemma 6.10
Let $\rho^c = \langle U^{c_1,\ldots,c_{n-1}}_{m_1,\ldots,m_k} | c_n : S^{c_n}_n \rangle$ be bracketed, with $U^{c_1,\ldots,c_{n-1}}_{m_1,\ldots,m_k} = D_1{}^{c_1}_{\bar{m}_1}, \ldots, D_{n-1}{}^{c_{n-1}}_{\bar{m}_{n-1}}$, where $D_i{}^{c_i}_{\bar{m}_i} = \langle m^{i\ c_i}_{1\bar{d}^i_1} \Leftarrow \mu^{i\ c_i}_{1\bar{d}^i_1}, \ldots, m^{i\ c_i}_{n_i\bar{d}^i_{n_i}} \Leftarrow \mu^{i\ c_i}_{n_i\bar{d}^i_{n_i}} \rangle : S^{c_i}_i$. Let $(A_i, C_i : \{p^{c_i}_i\} S^{c_i}_i \{q^{c_i}_i\})$, $1 \leq i \leq n$, be some cooperating (with respect to some global invariant I) specifications. Then for an arbitrary regular computation $(X_0, \sigma_0), \ldots, (X_k, \sigma_k)$ of ρ such that $\langle \alpha, \sigma_0 \rangle, \omega \models p_n$ (α being the root-object), and (X_k, σ_k) is stable, we have

- for every object $\alpha \in \sigma_k^{(c_i)}$ we have $\langle \alpha, \sigma_k \rangle, \omega \models A_i(Lab(X_k(\alpha)))$,
- $\sigma_k, \omega \models I$,
- if $\alpha \in \mathbf{O}^{c_i}$ is a newly created object of σ_k then $\langle \alpha, \sigma_k \rangle, \omega \models p^{c_i}_i$.

Proof
The proof proceeds by induction on the length of the computation. We first consider

the following case: Let

$$(X_0, \sigma_0) \rightarrow \ldots \rightarrow (X_m, \sigma_m) \rightarrow \ldots \rightarrow (X_k, \sigma_k)$$

such that

- $X_m(\alpha) = \mathsf{l}_1; R_1; \mathsf{l}; x \leftarrow e_0!m(\bar{e}); R_2; \mathsf{l}_2; S$, for some S, with $\alpha \in \mathbf{O}^{c_i}$
- $X_m(\beta) = \mathsf{l}'_1; \mathsf{answer}(\ldots, m, \ldots); \mathsf{l}'_2; S'$, for some S', with $\beta \in \mathbf{O}^{c_j}$
- From (X_m, σ_m) to (X_k, σ_k) only α and β are executing; α is executing the statement R_1 and sending β the actual parameters, and β is executing the statement R'_1, assuming m to be declared as $R'_1; \mathsf{l}''_1; R; \mathsf{l}''_2; R'_2 \uparrow e$.

Let

$$r_1 = C_i(\mathsf{l}_1), \quad r'_1 = C_j(\mathsf{l}'_1),$$
$$r = C_i(\mathsf{l}), \quad r''_1 = A(\mathsf{l}''_1).$$

Next suppose that $\mathsf{l}_1; R_1; \mathsf{l}; x \leftarrow e_0!m(\bar{e}); R_2; \mathsf{l}_2$ occurs in the statement $S_i^{c_i}$. Let k' be such that from $(X_{k'}, \sigma_{k'})$ to (X_m, σ_m) α is executing $S_i^{c_i}$. From the induction hypothesis it then follows that $\langle \alpha, \sigma_{k'} \rangle, \omega \models p_i^{c_i}$. We are given that $\models (A_i, C_i : \{p_i^{c_i}\} S_i^{c_i} \{q_i^{c_i}\})$ (by the second clause of the cooperation test and the soundness of the local proof system), so applying again the induction hypothesis we have that $\langle \alpha, \sigma_m \rangle, \omega \models r_1$. On the other hand if $\mathsf{l}_1; R_1; \mathsf{l}; x \leftarrow e_0!m(\bar{e}); R_2; \mathsf{l}_2$ occurs in the body of some method m' it follows in a similar way from $\models (A_i, C_i : \{A(\bar{\mathsf{l}}_1)\} R \{C_i(\bar{\mathsf{l}}_2)\})$, assuming m' to be declared as $\bar{R}_1; \bar{\mathsf{l}}_1; \bar{R}; \bar{\mathsf{l}}_2; \bar{R}_2 \uparrow e'$, and the induction hypothesis, that $\langle \alpha, \sigma_m \rangle, \omega \models r_1$.

Analogously we have that $\langle \beta, \sigma_m \rangle, \omega \models r'_1$.

Furthermore, as X_m is stable we have by the induction hypothesis that $\sigma_m, \omega \models I$.

Next we define

$$\sigma'_m = \sigma_m \{\sigma_m(\alpha)(\bar{u})/\alpha, \bar{y}_1\} \{\mathcal{E}[\![e_i]\!](\langle \alpha, \sigma_m \rangle)/\beta, y'_i\}_{1 \leq i \leq n} \{\perp/\beta, y''_j\}_j,$$

where $\bar{y}_2 = \bar{y}' \cup \bar{y}''$, with $\bar{y}'$ being a sequence of instance variables corresponding to the formal parameters of m and $\bar{y}''$ to the local variables of m. It then follows that

$$\sigma'_m, \omega' \models I \wedge r_1[z_1, \bar{y}_1/\mathsf{self}, \bar{u}] \wedge r'_1[z_2/\mathsf{self}] \wedge P,$$

where $\omega' = \omega\{\alpha, \beta/z_1, z_2\}$ and $P = e_0[z_1, \bar{y}_1/\mathsf{self}, \bar{u}_1] \doteq z_2 \wedge \bigwedge_i e_i[z_1, \bar{y}_1/\mathsf{self}, \bar{u}_1] \doteq z_2.y'_i \wedge \bigwedge_j z_2.y''_j \doteq \mathsf{nil}$. Next, let

$$(\sigma'_m, (\alpha, R_1[\bar{y}_1/\bar{u}]), (\beta, R'_1[\bar{y}_2/\bar{u}])) \rightarrow^* (\sigma', (\alpha, E), (\beta, E)).$$

It then follows by the cooperation test and the soundness of the intermediate proof system that

$$\sigma', \omega \models I \wedge \bar{r}[z_1, \bar{y}_1/\mathsf{self}, \bar{u}] \wedge r_1''[z_2, \bar{y}_2/\mathsf{self}, \bar{u}].$$

Now it is not difficult to see that

$$\sigma_k = \sigma'\{\sigma_m(\alpha)(\bar{y}_1)/\alpha, \bar{y}_1\}\{\sigma_m(\beta)(\bar{y}_2)/\beta, \bar{y}_2\}\{\sigma'(\alpha)(\bar{y}_1)/\alpha, \bar{u}\}\{Push(\sigma'_{(2)(\beta)}, f)/\beta\},$$

where $f(\bar{u}) = \sigma'(\beta)(\bar{y}_2)$. So we concude that

- $\sigma_k, \omega \models I$
- $\langle \alpha, \sigma_k \rangle, \omega \models r$
- $\langle \beta, \sigma_k \rangle, \omega \models r_1''$.

Next we consider the following case. Let

$$(X_0, \sigma_0) \rightarrow \ldots (X_m, \sigma_m) \rightarrow \ldots \rightarrow (X_k, \sigma_k)$$

such that

- $X_m(\alpha) = x \leftarrow \mathsf{wait}(m, \beta); R_2; \mathsf{l}_2; S$, for some S, with $\alpha \in \mathbf{O}^{c_i}$
- $X_m(\beta) = \mathsf{l}_2''; R_2'; \mathsf{send}(m, e, \alpha); S'$, for some S', assuming R_2' to be the postlude of the method m, with m declared as $R_1'; \mathsf{l}_1''; R; \mathsf{l}_2''; R_2' \uparrow e$, and $\beta \in \mathbf{O}^{c_j}$
- From (X_m, σ_m) to (X_k, σ_k) only α and β are executing; α is executing $x \leftarrow \mathsf{wait}(m, \beta); R_2$ and β is executing $R_2'; \mathsf{send}(m, e, \alpha)$.

Let $C_j(\mathsf{l}_2'') = r_2''$ and $A_j(\mathsf{l}_2') = r_2'$, where l_2' marks the end of the answer statement which gave rise to the activation of the method m. Furthermore, let $A_i(\mathsf{l}_2) = r_2$. From $\models (A_j, C_j : \{r_1''\}R\{r_2''\})$ and the induction hypothesis we infer that $\langle \beta, \sigma_m \rangle, \omega \models r_2''$. Moreover, by the induction hypothesis we have $\sigma_m, \omega \models I$. From the previous case it follows that we may assume $\langle \alpha, \sigma_m \rangle, \omega \models r$, where $r = C_i(\mathsf{l})$, with l the label marking the send statement which activated the method m. Next, we define

$$\sigma'_m = \sigma_m\{\sigma_m(\alpha)(\bar{u})/\alpha, \bar{y}_1\}\{\mathcal{E}[\![e]\!](\langle \beta, \sigma_m \rangle)/\alpha, x\}\{\sigma_m(\beta)(\bar{u})/\beta, \bar{y}_2\}.$$

It then follows that

$$\sigma'_m, \omega' \models I \wedge r[z_1, \bar{y}_1/\mathsf{self}, \bar{u}] \wedge r_2''[z_2, \bar{y}_2/\mathsf{self}, \bar{u}] \wedge Q,$$

where $\omega' = \omega\{\alpha, \beta/z_1, z_2\}$, and $Q = z_1 \doteq z_2.y_1' \wedge z_1.x \doteq e[z_2, \bar{y}_2/\mathsf{self}, \bar{u}_2]$. (Note that x does not occur in r.) Next, let

$$(\sigma'_m, (\alpha, R_2[\bar{y}_1/\bar{u}]), (\beta, R_2'[\bar{y}_2/\bar{u}])) \rightarrow^* (\sigma', (\alpha, E)(, \beta, E)).$$

Doc. No.

It then follows by the cooperation test and the soundness of intermediate proof system that

$$\sigma', \omega' \models I \wedge r_2[z_1, \bar{y}_1/\mathsf{self}, \bar{u}] \wedge r_2'[z_2/\mathsf{self}].$$

It is not difficult to see that

$$\sigma_k = \sigma'\{\sigma_m(\alpha)(\bar{y}_1)/\alpha, \bar{y}_1\}\{\sigma_m(\beta)(\bar{y}_2)/\beta, \bar{y}_2\}\{\sigma'(\alpha)(\bar{y}_1)/\alpha, \bar{u}\}\{Pop(\sigma'_{(2)(\beta)})/\beta\}.$$

So we conclude that

- $\sigma_k, \omega \models I$
- $\langle \alpha, \sigma_k \rangle, \omega \models r_2$
- $\langle \beta, \sigma_k \rangle, \omega \models r_2'$.

(Note that $TVar(r_2') = \emptyset$.)

Finally, we have the following case to consider: Let

$$(X_0, \sigma_0) \rightarrow \ldots \rightarrow (X_m \sigma_m) \rightarrow \ldots \rightarrow (X_k, \sigma_k)$$

such that

- $X_m(\alpha) = \mathsf{l}_1; R_1; x \leftarrow \mathsf{new}; R_2; \mathsf{l}_2; S$, for some S, with $\alpha \in \mathbf{O}^{c_i}$.
- From (X_m, σ_m) to (X_k, σ_k) α is executing $\mathsf{l}_1; R_1; x \leftarrow \mathsf{new}; R_2; \mathsf{l}_2$.

Let $C_i(\mathsf{l}_1) = r_1$ and $A_i(\mathsf{l}_2) = r_2$. As in the previous cases, by the induction hypothesis we have

$$\langle \alpha, \sigma_m \rangle, \omega \models r_1 \text{ and } \sigma_m, \omega \models I.$$

Next, we define

$$\sigma'_m = \sigma_m\{\sigma_m(\alpha)(\bar{u})/\alpha, \bar{y}\}.$$

It then follows that

$$\sigma'_m, \omega \models I \wedge r_1[z, \bar{y}/\mathsf{self}, \bar{u}].$$

where $\omega' = \omega\{\alpha/z\}$. Now, let

$$(\sigma'_m, (\alpha, (R_1; x \leftarrow \mathsf{new}; R_2)[\bar{y}/\bar{u}])) \rightarrow^* (\sigma', (\alpha, E)).$$

It then follows by the fourth clause of the cooperation test and the soundness of the intermediate proof system that

$$\sigma', \omega' \models I \wedge r_2[z, \bar{y}/\mathsf{self}, \bar{u}] \wedge p[z \cdot x/\mathsf{self}],$$

where p is the precondition of the newly created object. It is not difficult to see that

$$\sigma_k = \sigma'\{\sigma_m(\alpha)(\bar{y})/\alpha, \bar{y}\}\{\sigma'(\alpha)(\bar{y})/\alpha, \bar{u}\}.$$

So we conclude that $\langle \alpha, \sigma_k \rangle, \omega \models r_2$, $\langle \beta, \sigma_k \rangle, \omega \models p$ (here β is the newly created object), and $\sigma_k, \omega \models I$. □

7 Completeness

In this section we prove that an arbitrary valid correctness formula about a program is derivable. To be more specific, let $\{p[z/\mathsf{self}]\rho\{Q\}$ be a valid correctness formula, with $\rho^c = \langle U^{c_1,\ldots,c_{n-1}}_{m_1,\ldots,m_k} | c_n : S^{c_n}_n \rangle$ where $U^{c_1,\ldots,c_{n-1}}_{m_1,\ldots,m_k} = D_1{}^{c_1}_{\bar{m}_1}, \ldots, D_{n-1}{}^{c_{n-1}}_{\bar{m}_{n-1}}$, and $D_i{}^{c_i}_{\bar{m}_i} = \langle m^i_1{}^{c_i}_{\bar{d}^i_1} \Leftarrow \mu^i_1{}^{c_i}_{\bar{d}^i_1}, \ldots, m^i_{n_i}{}^{c_i}_{\bar{d}^i_{n_i}} \Leftarrow \mu^i_{n_i}{}^{c_i}_{\bar{d}^i_{n_i}} \rangle : S^{c_i}_i$. (Here we put $D_n = \langle\rangle : S^{c_n}_n$.) Without loss of generality we assume that $IVar(p,Q) \subseteq IVar(\rho)$ and that every logical variable occurring in Q has a type defined by ρ.

First we want to modify ρ by adding to it assignments to socalled *history* variables, i.e., auxiliary variables which record for every object its history, the sequence of communication records and activation records the object participates in. In languages like CSP such histories can be coded by integers: In CSP we can associate with each process an unique integer and thus code a communication record by an integer [Ap]. As there is no dynamic process creation in CSP a history is a sequence of communication records, which, given the coding of these records, can be coded too.

Given some coding of objects, it is not possible in our language to program an internal computation, using auxiliary variables, which computes the code of an object. That is, we cannot program a mapping of histories into integers. Therefore to be able to prove completeness, using the technique applied to the proof theory of CSP, we have to extend our programming language. We do so by introducing for each $d \in C^+$ a new finite set of instance variables $IVar^c_{d*}$, for an arbitrary c. It is not difficult to see how to code histories using these variables. However in the completeness proof we will not go into the details of coding these histories but simply assume to be given for each object of a class c_i defined by ρ a history variable h_i, for the details we refer to [AB]. We transform ρ to ρ' as follows:

Definition 7.1
Let $\bar{x}^i$ be the instance variables occurring in D_i and $\bar{y}^i$ be some new corresponding instance variables.

- Prefix every occurrence of an **answer**-statement, occurring in, say, D_i, by the multiple assignment $\bar{y}^i \leftarrow \bar{x}^i$.
- Let the method m be declared in D_i as $S \uparrow e$; replace $S \uparrow e$ by

  $$\bar{v} \leftarrow \bar{y}^i; h_i \leftarrow h_i \circ < m, \bar{u} >; \mathsf{l}; S; \mathsf{l}'; \bar{y}^i \leftarrow \bar{v}; h_i \leftarrow h_i \circ < m, u_1, e > \uparrow e,$$

  where $\bar{v}$ is a sequence of new temporary variables corresponding to the variables of the sequence $\bar{y}^i$, and $\bar{u}$ are the formal parameters of m.
- Replace every occurrence of a statement $x \leftarrow e_0!m(\bar{e})$ in, say, D_i, by

  $$\mathsf{l}; h_i \leftarrow h_i \circ < m, e_0, \bar{e} >; \bar{\mathsf{l}}; x \leftarrow e_0!m(\bar{e}); h_i \leftarrow h_i \circ < m, \mathsf{self}, x >; \mathsf{l}'.$$

- Repace every occurrence of a statement $x \leftarrow \mathsf{new}$ in, say, D_i, by

$$\mathsf{l}; x \leftarrow \mathsf{new}; h_i \leftarrow h_i \circ < \mathsf{self}, x >; \mathsf{l}'.$$

We assume the labels introduced to be distinct. It is important to keep in mind that the assignments to the history variables are in fact abbreviations of statements which compute the corresponding code. To be able to reason about invariance properties of an answer-statement we introduced some new instance variables $\bar{y}^i$ to freeze the state just before the execution of the answer-statement. To ensure that after the execution of an answer-statement these variables still refer to the state just before the execution we assign the values of these variables to some new temporary variables when entering the body of a method. After the execution of a body of a method we then can recover the initial values of $\bar{y}^i$ from these temporary variables. We assume that $\rho' = \langle U'^{c_1,\ldots,c_{n-1}}_{m_1,\ldots,m_k} | c_n : S'^{\,c_n}_n \rangle$, with $U'^{c_1,\ldots,c_{n-1}}_{m_1,\ldots,m_k} = D'^{\,c_1}_{1\bar{m}_1}, \ldots, D'^{\,c_{n-1}}_{n-1\bar{m}_{n-1}}$, where $D'^{c_i}_{i\bar{m}_i} = \langle m^{i\,c_i}_{1\bar{d}^i_1} \Leftarrow \mu'^{i\,c_i}_{1\bar{d}^i_1}, \ldots, m^{i\,c_i}_{n_i\bar{d}^i_{n_i}} \Leftarrow \mu'^{i\,c_i}_{n_i\bar{d}^i_{n_i}} \rangle : S'^{c_i}_i$.

Definition 7.2
Let R occur in D'_i, we define the set $After(R, D'_i)$ as follows: First, assume that R occurs in $S'^{c_i}_i$, we put $After(R, D'_i) = After(R, S'^{c_i}_i)$, where $After(R, S)$ is defined as follows:

- If $R = S$ then $After(R, S) = \{E\}$
- If $S = S_1; S_2$

  then $After(R, S) = \{R'; S_2 : R' \in After(R, S_1)\}$ if R occurs in S_1
  $After(R, S) = After(R, S_2)$ if R occurs in S_2

- If $S = \mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}$ then $After(R, S) = After(R, S_1)$ if R occurs in S_1
  $After(R, S) = After(R, S_2)$ if R occurs in S_2
- If $S = \mathsf{while}\ e\ \mathsf{do}\ S_1\ \mathsf{od}$ then $After(R, S) = \{R'; S : R' \in After(R, S_1)\}$

Next, let R occur in S, S being the body of some method declared in D'_i, we define $After(R, D'_i)$ as follows:

$$After(R, D'_i) =$$
$$\bigcup\{After(R_1, R'_1); \ldots; After(R_n, R'_n) : \quad R_1 = R, R'_1 = S, R'_n = S'^{c_i}_i, \forall 1 < i \leq n :$$
$$R_i = \mathsf{answer}(\ldots, m_i, \ldots), m_i \Leftarrow R'_{i-1} \uparrow e_i \in D'_i\}.$$

(Here, $X_1; \ldots; X_n$, X_i being a set of statements, is defined by $\{R_1; \ldots; R_n : R_i \in X_i\}$.)

Furthermore we define $Before(R, D_i')$ as follows:

$$Before(R, D_i') = \{R; R' : E; R' \in After(R, D_i')\}.$$

Finally, for $R = x \leftarrow \mathsf{wait}(m, e_0)$, associated with the send-statement $R' = x \leftarrow e_0!m(\bar{e})$, we define

$$Before(R, D_i') = \{R; R'' E; R'' \in After(R', D_i')\}.$$

The intuition formalized by this definition should be clear: $After(R, D_i')$ characterizes control when R has just been executed, while $Before(R, D_i')$ characterizes control in those cases that R is about to be executed. The complication arising when R occurs in the body of some method is due to the fact that we have to take into account chains of answered methods of arbitrary length. We note that we assume some mechanism to distinguish between different occurrences of a statement.

Next we modify the precondition p of the valid correctness formula $\{p[z/\mathsf{self}]\rho\{Q\}$ as follows:

Definition 7.3
We define

$$\bar{p}^{c_n} = p^{c_n} \wedge \bigwedge_{x \in W} (x \doteq z_x) \wedge |h_n| \doteq 0,$$

where $W = IVar^{c_n} \cup IVar^{c_n}$, z_x being a new logical variable uniquely associated with the instance variable x. These newly introduced variables are used to "freeze" that part of the initial state as specified by the integer and boolean variables.

Note that the assertion $|h_n| \doteq 0$ should be interpreted as an abbreviation of an assertion expressing the same fact, i.e., that there is no history yet, in terms of some particular coding of the histories.

To define the *expressibility* of some set of states we have the following definition:

Definition 7.4
For R occurring in, say, D_i', we define

$$\begin{aligned} V(R) &= IVar(D_i') \cup TVar(D_i') && \text{if } R \text{ occurs in the body of some method} \\ &= IVar(D_i') && \text{if } R \text{ occurs in } S_i'^{c_i}. \end{aligned}$$

Furthermore, for $Y \subseteq IVar^c \cup TVar$, σ, σ', and $\alpha \in \sigma^{(c)} \cap \sigma'^{(c)}$, we define

$$\sigma(\alpha) =_Y \sigma'(\alpha) \text{ iff } \sigma(\alpha)(v) = \sigma'(\alpha)(v), \text{ for } v \in Y.$$

Each of the following lemmas state the expressibility of a set of states which collects all those states occurring during a particular computation whenever control is at some specific point.

Lemma 7.5
Let R be a substatement occurring in ρ', say, R occurs in D'_i. There exists an assertion $Pre(R)$ with $IVar(Pre(R)) \cup TVar(Pre(R)) \subseteq V(R)$, describing the local state of objects of class c_i, such that

$$\langle \alpha, \sigma \rangle, \omega \models Pre(R) \text{ iff}$$

- $\exists (X_0, \sigma_0) \rightarrow^* (X', \sigma')$, such that $Init_{\rho'}((X_0, \sigma_0))$,
- $X'(\alpha) \in Before(R, D'_i)$,
- $\langle \beta, \sigma_0 \rangle, \omega \models \bar{p}$, β being the root-object,
- $\sigma'(\alpha) =_{V(R)} \sigma(\alpha)$.

This local assertion $Pre(R)$ describes all the local states $\langle \alpha, \sigma \rangle$ for which there exists an intermediate configuration (X', σ') of a computation of ρ', starting from an initial state of the root-object which satisfies $\bar{p}$, such that $\sigma'(\alpha)$ equals $\sigma(\alpha)$ with respect to the variables of $V(R)$, and furthermore, in the configuration (X', σ') the object α is about to execute R.

Using the techniques of [AB] one can show that the assertion $Pre(R)$ exists. Note that the set $Before(R, D'_i)$ (and $After(R, D'_i)$ as well) is recursive. (To decide if a statement occurs in, say, $After(R, D'_i)$, we only have to look at the sets $After(R_1, R'_1); \ldots; After(R_n, R'_n)$, with n the length of the statement. But these sets are finite and there are only finitely many of them.)

Lemma 7.6
Let R be a substatement occurring in ρ', say, R occurs in D'_i. There exists an assertion $Post(R)$ with $IVar(Pre(R)) \cup TVar(Pre(R)) \subseteq V(R)$, describing the local state of objects of class c_i such that

$$\langle \alpha, \sigma \rangle, \omega \models Post(R) \text{ iff}$$

- $\exists (X_0, \sigma_0) \rightarrow^* (X', \sigma')$, such that $Init_{\rho'}((X_0, \sigma_0))$,
- $X'(\alpha) \in After(R, D'_i)$,
- $\langle \beta, \sigma_0 \rangle, \omega \models \bar{p}$, β being the root-object,

- $\sigma'(\alpha) =_{V(R)} \sigma(\alpha)$.

This local assertion describes all the local states $\langle \alpha, \sigma \rangle$ for which there exists an intermediate configuration (X', σ') of a computation of ρ', starting from an initial state of the root-object which satisfies $\bar{p}$, such that $\sigma'(\alpha)$ equals $\sigma(\alpha)$ with respect to the variables of $V(R)$, and furthermore, in the configuration (X', σ') the object α has just finished executing R. The expressibility of this assertion is proved in the same way as the assertion $Pre(R)$.

Lemma 7.7
Again, let R be a substatement occurring in, say, D_i'. There exists an assertion $Pre(R)_Y$ with $IVar(Pre(R)_Y) \cup TVar(Pre(R)_Y) \subseteq Y \cap V(R)$, where $Y \subseteq IVar^{c_i} \cup TVar$, such that

$$\langle \alpha, \sigma \rangle, \omega \models Pre(R)_Y \text{ iff}$$

there exists σ' such that $\sigma'(\alpha) =_Y \sigma(\alpha)$ and $\langle \alpha, \sigma' \rangle, \omega \models Pre(R)$.

The expressibility of this assertion is proved in the same way as the assertion $Pre(R)$.

Lemma 7.8
Let R be a substatement occurring in, say, D_i', and p^{c_i} be such that $TVar(p) = \emptyset$. There exists an assertion $SP(p, R)$ such that $TVar(SP(p, R)) = \emptyset$ and

$$\langle \alpha, \sigma \rangle, \omega \models SP(p, R) \text{ iff}$$

- $\exists (R, \theta_0) \rightarrow_L^* (E, \theta_1)$,
- $\theta_0, \omega \models p$,
- $\sigma_1(\alpha) =_{IVar(D_i')} \sigma(\alpha)$, where $\theta_1 = \langle \alpha, \sigma_1 \rangle$.

The assertion $SP(p, R)$ expresses what is called the strongest postcondition of the statement R with respect to the precondition p. Note that $SP(p, R)$ specifies the behaviour of R only with respect to the instance variables of D_i'. The expressibility of this assertion is proved in the same way as the assertion $Pre(R)$. Next we have the following lemma:

Lemma 7.9
For every class c_i defined by ρ' there exists a local assertion $Lhist_i^{c_i}$, with $TVar(Lhist_i) = \emptyset$ and $IVar(Lhist_i) = \{h_i\}$, such that

$$\langle \alpha, \sigma \rangle, \omega \models Lhist_i \text{ iff}$$

- $\exists (X_0, \sigma_0) \rightarrow^* (X', \sigma')$, $Init_{\rho'}((X_0, \sigma_0))$,
- $\langle \beta, \sigma_0 \rangle, \omega \models \bar{p}$, β being the root-object,
- $\sigma(\alpha)(h_i) = \sigma'(\alpha)(h_i)$.

The expressibility of this assertion is proved among the same lines as the assertions defined above. The assertion *Lhist* holds in a local state $\langle \alpha, \sigma \rangle$ iff there exists an intermediate configuration (X', σ') of a computation of ρ', starting from an initial state of the root-object which satisfies $\bar{p}$, such that σ and σ' agree with respect to the history of α. Finally, we have the following lemma,

Lemma 7.10
There exists a global assertion I such that $IVar(I) \subseteq \{h_1, \ldots, h_n\}$ and

$$\sigma, \omega \models I \text{ iff}$$

- $\exists (X_0, \sigma_0) \rightarrow^* (X', \sigma')$, with $Init_{\rho'}((X_0, \sigma_0))$,
- X' is stable (see definition 6.8),
- $\langle \alpha, \sigma_0 \rangle, \omega \models \bar{p}$,
- $\sigma'(\alpha)(h_i) = \sigma(\alpha)(h_i)$, for $\alpha \in \sigma'^{(c_i)}$, $c_i \in \{c_1, \ldots, c_n\}$,
- $\sigma^{(c_i)} = \sigma'^{(c_i)}$, $c_i \in \{c_1, \ldots, c_n\}$.

This assertion I, which we call the global invariant, describes all states σ for which there exists an intermediate configuration (X', σ') of a computation of ρ', starting from an initial state of the root-object which satisfies $\bar{p}$, such that σ and σ' agree with respect to the existing objects and with respect to the histories of these objects. Note that as X' is stable we have that the history recorded by an existing object of σ' equals the history obtained from h by deleting from it all the communication and activation records not involving this object. The expressibility of this assertion is shown along the lines of [AB].

Using the above lemmas we now define the set of assumptions and commitments for each class $c_i \in \{c_1, \ldots, c_n\}$.

Definition 7.11
We define

- $A_i = \{\mathsf{l}'.Post(R) : R = \mathsf{l}; R'; \mathsf{l}' \in D_i',$ with R' containing a **new**-statement$\}$ $\cup$
  $\{\mathsf{l}'.Post(R) : R = \mathsf{l}; R'; \mathsf{l}' \in D_i',$ with R' containing a **send**-statement$\}$ $\cup$
  $\{\mathsf{l}.Post(R) : m \Leftarrow R; \mathsf{l}; S; \mathsf{l}'; R' \uparrow e \in D_i'\}$ $\cup$
  $\{\mathsf{l}'.SP(\bar{y}^i \doteq \bar{x}^i, \mathsf{answer}(\bar{m})) \wedge Lhist_i : \mathsf{l}; \mathsf{answer}(\bar{m}); \mathsf{l}' \in D_i'\}$

- $C_i =$ $\{\mathsf{l}.Pre(R) :\ R = \mathsf{l}; R'; \mathsf{l}' \in D_i',$ with R' containing a **new**-statement$\}$ $\cup$
  $\{\mathsf{l}.Pre(R) :\ R = \mathsf{l}; R'; \mathsf{l}' \in D_i',$ with R' containing a **send**-statement$\}$ $\cup$
  $\{\mathsf{l}.Pre(R)_{IVar} :\ R = \mathsf{l}; \mathsf{answer}(\bar{m}); \mathsf{l}' \in D_i'\}$ $\cup$
  $\{\mathsf{l}.Pre(R)_{ITvar\setminus\{x\}} :\ R = x \leftarrow \mathsf{wait}(m, e_0),$ with $\mathsf{l}; x \leftarrow e_0!m(\bar{e}) \in D_i'\}$

Here $R \in D_i'$ should be interpreted to mean that R occurs in D_i'.

It is important to note that given a statement $\mathsf{l}; \mathsf{answer}(\bar{m}); \mathsf{l}'$ we did not associate the assertion $Post(\mathsf{l}; \mathsf{answer}(\bar{m}); \mathsf{l}')$ with the label l'. The reason for this being that this assumption cannot be justified in the cooperation test, because when exiting the body of one of the methods of $\bar{m}$ we have no information about the particular answer-statement which gave rise to the execution of this method. However we will see in the following lemma how we can strengthen the assumption $SP(\bar{y}^i \doteq \bar{x}^i, \mathsf{answer}(\bar{m})) \wedge Lhist_i$ by some reasoning within the local proof system to the assertion $Post(\mathsf{l}; \mathsf{answer}(\bar{m}); \mathsf{l}')$. We are now ready for the following lemma:

Lemma 7.12
We have

$$\vdash (A_i, C_i : \{Pre(R)\}R\{Post(R)\}),$$

where R occurs in D_i' such that R is normal, i.e., R does not occur in a bracketed section.

Proof
The proof proceeds by induction on the structure of R. We treat the case $R = \mathsf{l}; \mathsf{answer}(m_1, \ldots, m_n); \mathsf{l}'$, the other cases being straightforward.

Let m_j be declared as $R_j'; \mathsf{l}_j'; R_j; \mathsf{l}_j''; R_j'' \uparrow e_j$, furthermore, let $p = Pre(\mathsf{l}; \mathsf{answer}(m_1, \ldots, m_n); \mathsf{l}')$ and $p' = p[\bar{y}^i/\bar{x}^i]$. We first show that

$$\vdash (\emptyset, \emptyset : \{p'[\bar{y}/\bar{u}]\}R_j'; R_j; R_j''\{p'[\bar{y}/\bar{u}]\}),$$

where $TVar(p') = \bar{u}$ and $\bar{y}$ are some new instance variables:

1. $(\emptyset, \emptyset : \{p'[\bar{y}/\bar{u}][\bar{v}/\bar{y}^i]\}R_j''\{p'[\bar{y}/\bar{u}]\})$, by (LIASS).
2. $(\emptyset, \emptyset : \{p'[\bar{y}/\bar{u}][\bar{v}/\bar{y}^i]\}R_j\{p'[\bar{y}/\bar{u}][\bar{v}/\bar{y}^i]\})$, by (INV).
3. $(\emptyset, \emptyset : \{p'[\bar{y}/\bar{u}]\}R_j'\{p'[\bar{y}/\bar{u}][\bar{v}/\bar{y}^i]\})$, by (LTASS).

So applying the rule (BA) gives us

$$\vdash (A_i, C_i : \{p' \wedge p_1\}\mathsf{l}; \mathsf{answer}(m_1, \ldots, m_n); \mathsf{l}'\{p' \wedge q_1\}),$$

where

- $p_1 = C(\mathsf{I}) = Pre(\mathsf{I}; \mathsf{answer}(m_1, \ldots, m_n); \mathsf{I}')_{IVar}$,
- $q_1 = A(\mathsf{I}') = SP(\bar{y}^i \doteq \bar{x}^i, \mathsf{answer}(m_1, \ldots, m_n)) \wedge Lhist_i$.

It is not difficult to check that

$$\models Pre(\mathsf{I}; \mathsf{answer}(m_1, \ldots, m_n); \mathsf{I}') \rightarrow p' \wedge p_1.$$

(Note that $\models Pre(\mathsf{I}; \mathsf{answer}(m_1, \ldots, m_n); \mathsf{I}') \rightarrow \bar{y}^i \doteq \bar{x}^i$.) So applying the consequence rule gives us

$$\vdash (A_i, C_i : \{Pre(\mathsf{I}; \mathsf{answer}(m_1, \ldots, m_n); \mathsf{I}')\}\mathsf{I}; \mathsf{answer}(m_1, \ldots, m_n); \mathsf{I}'\{p' \wedge q_1\}).$$

By an application of the consequence rule it thus suffices to show that the following implication holds

$$p' \wedge q_1 \rightarrow Post(\mathsf{I}; \mathsf{answer}(m_1, \ldots, m_n); \mathsf{I}').$$

Here we go. Let

$$\langle \alpha, \sigma \rangle, \omega \models p' \wedge q_1.$$

From $\langle \alpha, \sigma \rangle, \omega \models Lhist_i$ it follows that there exists a computation

$$(X_0, \sigma_0) \rightarrow^* (X_m, \sigma_m)$$

of ρ' such that $\sigma_m(\alpha)(h_i) = \sigma(\alpha)(h_i)$. Furthermore, we may assume without loss of generality that α has just finished executing a bracketed section.

Let $\sigma' = \sigma\{\sigma(\alpha)(\bar{y}^i)/\alpha, \bar{x}^i\}$. By $\langle \alpha, \sigma' \rangle, \omega \models p$ it then follows that there exists a computation

$$(X_0, \sigma_0) \rightarrow^* (X_n, \sigma_n)$$

of ρ' such that $X_n(\alpha) \in Before(\mathsf{I}; \mathsf{answer}(m_1, \ldots, m_n); \mathsf{I}', D_i')$ and $\sigma_n(\alpha) =_{V(R)} \sigma'(\alpha)$ $(R = \mathsf{I}; \mathsf{answer}(m_1, \ldots, m_n); \mathsf{I}')$. Note that we may assume that this computation starts from the same initial configuration (X_0, σ_0) as the computation which exists according to $\sigma, \omega \models Lhist_i$ because of the use of freeze variables in definition 7.3.

Finally, by $\langle \alpha, \sigma \rangle, \omega \models q_1$ we have for some history h'

$$(\mathsf{answer}(m_1, \ldots, m_n), \theta) \rightarrow^{h'} (E, \theta'),$$

with $\theta_{(1)} = \theta'_{(1)} = \alpha$, $\theta'_{(2)}(\alpha) =_{IVar(D_i')} \sigma(\alpha)$, and $\theta, \omega \models \bar{y}^i \doteq \bar{x}^i$.

As $\sigma_m(\alpha)(h_i) = \sigma(\alpha)(h_i) = \theta'_{(2)}(\alpha)(h_i)$ we have $\theta'_{(2)}(\alpha)(h_i) = \sigma_m(\alpha)(h_i) = h'' \circ h'$ for some history h''. Now let

$$(X_0, \sigma_0) \rightarrow^* (X_k, \sigma_k) \rightarrow^* (X_m, \sigma_m)$$

such that $\sigma_k(\alpha)(h_i) = h''$ and α is about to execute a bracketed section. Now as $\theta'_{(2)}(\alpha)(h_i) = \theta_{(2)}(\alpha)(h_i) \circ h' = h'' \circ h'$ we have

$$\theta_{(2)}(\alpha)(h_i) = h'' = \sigma_k(\alpha)(h_i). \tag{7.1}$$

From

$$\begin{aligned} \theta_{(2)}(\alpha)(\bar{y}^i) &= \\ \theta'_{(2)}(\alpha)(\bar{y}^i) &= \\ \sigma(\alpha)(\bar{y}^i) &= \\ \sigma'(\alpha)(\bar{x}^i) &= \\ \sigma_n(\alpha)(\bar{x}^i) & \end{aligned}$$

and

$$\theta, \omega \models \bar{y}^i \doteq \bar{x}^i$$

it follows that

$$\theta_{(2)}(\alpha)(h_i) = \sigma_n(\alpha)(h_i). \tag{7.2}$$

So from 7.1 and 7.2 we infer that $\sigma_n(\alpha)(h_i) = \sigma_k(\alpha)(h_i)$. From this in turn we derive that $X_k(\alpha) = X_n(\alpha)$ and $\sigma_k(\alpha) = \sigma_n(\alpha)$. (Note that the behaviour of an object is uniquely determined with respect to the behaviour of the environment.) Now, from $\sigma_k(\alpha) = \sigma_n(\alpha)$ (using the above sequence of identities) it follows that $\sigma_k(\alpha) =_{IVar(D'_i)} \theta_{(2)}(\alpha)$. From which it is not difficult to derive that $X_m(\alpha) \in After(l.\mathsf{answer}(m_1, \ldots, m_n).l', D'_i)$ and $\sigma_m(\alpha) =_{IVar(D'_i)} \theta'_{(2)}(\alpha) =_{IVar(D'_i)} \sigma(\alpha)$ (use again that the behaviour of an object is uniquely determined with respect to the behaviour of the environment). Furthermore, we have

$$\begin{aligned} \sigma_m(\alpha)(u) &= \\ \sigma_k(\alpha)(u) &= \\ \sigma_n(\alpha)(u) &= \\ \sigma'(\alpha)(u) &= \\ \sigma(\alpha)(u), & \end{aligned}$$

where $u \in V(R)$ is a temporary variable. Note that the first identity follows from our assumption that α in (X_m, σ_m) has just finished executing a bracketed section, because then, as $\sigma_m(\alpha)(h_i) = \sigma(\alpha)(h_i) = \theta'_{(2)}(\alpha)(h_i)$, we have that α (in (X_m, σ_m)) has just finished executing $\mathsf{l}; \mathsf{answer}(m_1, \ldots, m_n); \mathsf{l}'$. Thus we conclude that $\sigma(\alpha) =_{V(R)} \sigma_m(\alpha)$, so

$$\langle \alpha, \sigma \rangle, \omega \models Post(\mathsf{l}; \mathsf{answer}(m_1, \ldots, m_n); \mathsf{l}').$$

□

In the following lemmas we show that the other requirements of the cooperation test are satisfied.

Lemma 7.13
Let $\mathsf{l}; \mathsf{answer}(\ldots, m, \ldots); \mathsf{l}'$ occur in D_i', with m declared as $R_1; \mathsf{l}_1; R; \mathsf{l}_2; R_2 \uparrow e$. Furthermore, let $\mathsf{l}_1'; R_1'; \bar{\mathsf{l}}; x \leftarrow e_0!m(e_1, \ldots, e_n); R_2'; \mathsf{l}_2'$ occur in D_j'. Then the following intermediate correctness formulas hold:

$$\{I \wedge p_1'[z_1, \bar{y}_1/\mathsf{self}, \bar{u}] \wedge p[z_2/\mathsf{self}] \wedge P\}$$
$$(z_1, R_1'[\bar{y}_1/\bar{u}]) \parallel (z_2, R_1[\bar{y}_2/\bar{u}])$$
$$\{I \wedge \bar{p}[z_1, \bar{y}_1/\mathsf{self}, \bar{u}] \wedge p_1[z_2, \bar{y}_2/\mathsf{self}, \bar{u}]\}$$

and

$$\{I \wedge \bar{p}[z_1, \bar{y}_1/\mathsf{self}, \bar{u}] \wedge p_2[z_2, \bar{y}_2/\mathsf{self}, \bar{u}] \wedge Q\}$$
$$(z_1, R_2'[\bar{y}_1/\bar{u}]) \parallel (z_2, R_2[\bar{y}_2/\bar{u}])$$
$$\{I \wedge p_2'[z_1, \bar{y}_1/\mathsf{self}, \bar{u}] \wedge p'[z_2, \bar{y}_2/\mathsf{self}, \bar{u}]\}$$

where

- $p = C_i(\mathsf{l}) = Pre(\mathsf{l}; \mathsf{answer}(\ldots, m, \ldots); \mathsf{l}')_{IVar}$,
- $p' = A_i(\mathsf{l}') = SP(\bar{y}^i \doteq \bar{x}^i, \mathsf{answer}(\ldots, m, \ldots)) \wedge Lhist_i$,
- $p_1 = A_i(\mathsf{l}_1) = Post(R_1)$, $p_2 = C_i(\mathsf{l}_2) = Pre(R_2)$,
- $p_1' = C_j(\mathsf{l}_1') = Pre(\mathsf{l}_1'; R_1'; \bar{\mathsf{l}}; x \leftarrow e_0!m(e_1, \ldots, e_n); R_2'; \mathsf{l}_2')$,
- $\bar{p} = C_j(\bar{\mathsf{l}}) = Pre(x \leftarrow \mathsf{wait}(m, e_0))_{ITvar \setminus \{x\}}$,
- $p_2' = A_j(\mathsf{l}_2') = Post(\mathsf{l}_1'; R_1'; \bar{\mathsf{l}}; x \leftarrow e_0!m(e_1, \ldots, e_n); R_2'; \mathsf{l}_2')$,
- $P = e_0[z_1, \bar{y}_1/\mathsf{self}, \bar{u}_1] \doteq z_2 \wedge \bigwedge_i e_i[z_1, \bar{y}_1/\mathsf{self}, \bar{u}_1] \doteq z_2.y_i' \wedge \bigwedge_j z_2.y_j'' \doteq \mathsf{nil}$,
- $Q = z_1 \doteq z_2.y_1' \wedge z_1.x \doteq e[z_2, \bar{y}_2/\mathsf{self}, \bar{u}_2]$.

Here $\bar{y}' \cup \bar{y}'' = \bar{y}_2$, with $\bar{y}'$ being the instance variables corresponding to the formal parameters and $\bar{y}''$ corresponding to the local variables of m.

Proof

We start with the proof of the validity of the first correctness formula.

Let

$$\sigma, \omega \models I \wedge p_1'[z_1, \bar{y}_1/\mathsf{self}, \bar{u}] \wedge p[z_2/\mathsf{self}] \wedge P.$$

Furthermore let

$$(\sigma, (\alpha_1, R_1'[\bar{y}_1/\bar{u}]), (\alpha_2, R_1[\bar{y}_2/\bar{u}])) \rightarrow^* (\sigma', (\alpha_1, E), (\alpha_2, E)),$$

where $\alpha_1 = \omega(z_1)$ and $\alpha_2 = \omega(z_2)$. Next we define $\sigma'' = \sigma\{\sigma(\alpha_1)(\bar{y}_1)/\alpha_1, \bar{u}\}$. By $\sigma'', \omega \models I$, $\langle \alpha_1, \sigma'' \rangle, \omega \models p'_1$ and $\langle \alpha_2, \sigma'' \rangle, \omega \models p$ it then follows that there exists a computation of ρ'

$$(X_0, \sigma_0) \rightarrow^* (X_n, \sigma_n)$$

such that

- $X_n(\alpha_1) \in \mathit{Before}(R'_1, D'_j)$ and $X_n(\alpha_2) \in \mathit{Before}(\mathsf{I}; \mathsf{answer}(\ldots, m, \ldots); \mathsf{I}', D'_i)$,
- $\sigma_n(\alpha_1) =_{V(R)} \sigma''(\alpha_1)$ $(R = R'_1; x \leftarrow e_0!m(e_1, \ldots, e_n); R'_2)$ and $\sigma_n(\alpha_2) =_{IVar(D'_i)} \sigma''(\alpha_2)$,
- $\sigma_n^{(c)} = \sigma''^{(c)}$, $c \in \{c_1, \ldots, c_n\}$,
- $\sigma_n(\alpha)(h_i) = \sigma''(\alpha)(h_i)$, $\alpha \in \sigma''^{(c_i)}$, $c_i \in \{c_1, \ldots, c_n\}$.

Note that, as the history of α_1 (α_2) in the computation which exists according to $\langle \alpha_1, \sigma'' \rangle, \omega \models p'_1$ $(\langle \alpha_2, \sigma'' \rangle, \omega \models p)$ equals that of α_1 (α_2) in the computation which exists according to $\sigma'', \omega \models I$, we have that in this latter computation α_1 is about to execute R'_1 (and α_2 is about to execute $\mathsf{I}; \mathsf{answer}(\ldots, m, \ldots); \mathsf{I}'$) and the local states of α_1 (α_2) in the final states of these computations coincide. (Again, the above observation is based on the fact that the behaviour of an object is completely determined given its interactions with the environment, furthermore we make use of that the computations which exists according to $\sigma'', \omega \models I$, $\langle \alpha_1, \sigma'' \rangle, \omega \models p'_1$ and $\langle \alpha_2, \sigma'' \rangle, \omega \models p$ all start from the same initial state, which is due to the use of the logical variables z_x, introduced in definition 7.3, as freeze variables.)

Now let

$$(X_n, \sigma_n) \rightarrow^* (X_k, \sigma_k)$$

such that

- $X_k(\alpha_1) \in \mathit{After}(R'_1, D'_j)$ and $X_k(\alpha_2) \in \mathit{Before}(x \leftarrow \mathsf{wait}(m, e_0), D'_i)$,
- from $< X_n, \sigma_n >$ to $< X_k, \sigma_k >$ only α_1 and α_2 are executing; α is executing R'_1 and α_2 is executing R_2.

Note that such a computation exists because $\sigma'', \omega \models e_0[z_1/\mathsf{self}] \doteq z_2$, so we have $\mathcal{E}[\![e_0]\!](\langle \alpha_1, \sigma_n \rangle) = \mathcal{E}[\![e_0]\!](\langle \alpha_1, \sigma'' \rangle) = \alpha_2$.

It is not difficult to see that

$$\sigma_k(\alpha_1)(\bar{u}) = \sigma'(\alpha_1)(\bar{y}_1),$$

$$\sigma_k(\alpha_2)(\bar{u}) = \sigma'(\alpha_2)(\bar{y}_2),$$
$$\sigma_k(\alpha_1)(x) = \sigma'(\alpha_1)(x),\ x \in IVar(D'_j),$$
$$\sigma_k(\alpha_2)(x) = \sigma'(\alpha_2)(x),\ x \in IVar(D'_i).$$

Furthermore we have

$$\sigma_k^{(c)} = \sigma'^{(c)},\ c \in \{c_1, \ldots, c_n\}$$

and

$$\sigma_k(\alpha)(h_i) = \sigma'(\alpha)(h_i),\ \alpha \in \sigma_k^{(c_i)},\ c_i \in \{c_1, \ldots, c_n\}.$$

So we conclude that

$$\sigma', \omega \models I,$$
$$\langle \alpha_1, \sigma' \rangle, \omega \models Pre(x \leftarrow \mathsf{wait}(m, e_0))[\bar{y}_1/\bar{u}],$$
$$\langle \alpha_2, \sigma' \rangle, \omega \models Post(R_1)[\bar{y}_2/\bar{u}].$$

Now it is not difficult to see that

$$\models Pre(x \leftarrow \mathsf{wait}(m, e_0)) \rightarrow \bar{p}.$$

So we conclude

$$\sigma', \omega \models I \wedge \bar{p}[z_1, \bar{y}_1/\mathsf{self}, \bar{u}] \wedge p_1[z_2, \bar{y}_2/\mathsf{self}, \bar{u}].$$

The proof of the validity of the second correctness formula is similar. □

Lemma 7.14
Let $\mathsf{l}; x \leftarrow \mathsf{new}; R_2; \mathsf{l}'$ occur in D'_i. Then the following correctness assertion holds:

$$\{I \wedge p[z, \bar{y}/\mathsf{self}, \bar{u}]\}$$
$$(z, (x \leftarrow \mathsf{new}; R_2)[\bar{y}/\bar{u}])$$
$$\{I \wedge p'[z, \bar{y}/\mathsf{self}, \bar{u}] \wedge q[z.x/\mathsf{self}]\}$$

where

- $p = Pre(\mathsf{l}; x \leftarrow \mathsf{new}; R_2; \mathsf{l}')$, $p' = Post(\mathsf{l}; x \leftarrow \mathsf{new}; R_2; \mathsf{l}')$, and $q = Pre(S_j'^{c_j})$, assuming the type of x to be c_j,
- $\bar{y}$ are some new instance variables corresponding to the temporary variables $\bar{u}$.

Proof
Let

$$\sigma, \omega \models I \wedge p[z, \bar{y}/\mathsf{self}, \bar{u}],$$

and

$$(\sigma, (\alpha, (x \leftarrow \mathsf{new}; R_2)[\bar{y}/\bar{u}])) \rightarrow^* (\sigma', (\alpha, E)),$$

where $\alpha = \omega(z)$. Let $\sigma'' = \sigma\{\sigma(\alpha)(\bar{y})/\bar{u}\}$. It follows that there exists a computation of ρ'

$$(X_0, \sigma_0) \rightarrow^* (X_n, \sigma_n)$$

(the existence of such a computation is justified as in the proof of the previous lemma) such that

- $X_n(\alpha) \in \mathit{Before}(\mathsf{l}; x \leftarrow \mathsf{new}; R_2; \mathsf{l}', D_i')$,
- $\sigma_n(\alpha) =_{V(R)} \sigma''(\alpha)$, $R = x \leftarrow \mathsf{new}; R_2$,
- $\sigma_n^{(c)} = \sigma''^{(c)}$, $c \in \{c_1, \ldots, c_n\}$,
- $\sigma_n(\beta)(h_i) = \sigma''(\beta)(h_i)$, $\beta \in \sigma''^{(c_i)}, c_i \in \{c_1, \ldots, c_n\}$.

Next let

$$(X_n, \sigma_n) \rightarrow^* (X_k, \sigma_k)$$

such that $X_k(\alpha) \in \mathit{After}(\mathsf{l}; x \leftarrow \mathsf{new}; R_2; \mathsf{l}', D_i')$ and from (X_n, σ_n) to (X_k, σ_k) only α is executing. Now from

$$\sigma_k(\beta) = \sigma'(\beta),$$

with β the newly created object,

$$\sigma_k(\alpha)(\bar{u}) = \sigma'(\alpha)(\bar{y}),$$

$$\sigma_k(\alpha)(x) = \sigma'(\alpha)(x),\ x \in \mathit{IVar}(D_i'),$$

and, finally,

$$\langle \beta, \sigma_k \rangle, \omega \models \mathit{Pre}(S_j'^{c_j})$$

it follows that

$$\sigma', \omega \models I \wedge p'[\bar{y}/\bar{u}][z/\mathsf{self}] \wedge q[z.x/\mathsf{self}].$$

□

Theorem 7.15
The following formula about ρ', which is called the most general correctness formula about ρ', is derivable:

$$\{\mathit{Pre}(S_n'^{c_n})[z_n/\mathsf{self}]\}$$
$$\rho'$$
$$\{I \wedge \textstyle\bigwedge_{i \neq n} \forall z_i \mathit{Post}(S_i'^{c_i})[z_i/\mathsf{self}] \wedge \mathit{Post}(S_n'^{c_n})[z_n/\mathsf{self}]\}.$$

Proof
From lemma 7.12 it follows that

$$\vdash (A_i, C_i : \{Pre(S_i'^{c_i})\} S_i'^{c_i} \{Post(S_i'^{c_i})\}),$$

and

$$\vdash (A_i, C_i : \{A_i(\mathsf{I})\} R \{C_i(\mathsf{I}')\}),$$

where R occurs in the body $R_1; \mathsf{I}; R; \mathsf{I}'.R_2 \uparrow e$ of some method m defined in D_i'. Furthermore it is not difficult to prove that

$$\models Pre(S_n'^{c_n})[z_n/\mathsf{self}] \wedge \forall z'(z' \doteq z_n) \wedge \bigwedge_{1 \leq i < n} (\forall z_i \ \mathsf{false}) \rightarrow I.$$

(The variables z_i are assumed to be of type c_i.) From the completeness of the intermediate proof system and the above lemmas it follows that the cooperation test holds. (The completeness of the intermediate proof system is proved in a similar way as the completeness of the usual Hoare-style proof system for sequential programs.) An application of the rule (PR) then finishes the proof. □

We are now ready for the completeness theorem.

Theorem 7.16
The valid correctness formula $\{p[z_{c_n}/\mathsf{self}]\}\rho\{Q\}$ is derivable.

$$\vdash \{p[z_{c_n}/\mathsf{self}]\}\rho\{Q\}.$$

Proof
By the previous theorem we have the derivability of the correctness formula

$$\{Pre(S_n'^{c_n})[z_n/\mathsf{self}]\}$$
$$\rho'$$
$$\{I \wedge \textstyle\bigwedge_{i \neq n} \forall z_i Post(S_i'^{c_i})[z_i/\mathsf{self}] \wedge Post(S_n'^{c_n})[z_n/\mathsf{self}]\}.$$

It is not difficult to prove that $\models p'^{c_n} \wedge \bigwedge_{x \in W} x \doteq \mathsf{nil} \rightarrow Pre(S_n'^{c_n})$, where $W = \bigcup_c IVar_c^{c_n}$. Furthermore, as $I \wedge \bigwedge_{i \neq n} \forall z_i Post(S_i'^{c_i})[z_i/\mathsf{self}] \wedge Post(S_n'^{c_n})[z_n/\mathsf{self}]$ can be easily seen to characterize precisely the set of final states, we have

$$\models I \wedge \bigwedge_{i \neq n} \forall z_i Post(S_i'^{c_i})[z_i/\mathsf{self}] \wedge Post(S_n'^{c_n})[z_n/\mathsf{self}] \rightarrow Q.$$

By the consequence rule and the rule (INIT) we thus have

$$\vdash \{p'[z_n/\mathsf{self}]\}\rho'\{Q\}.$$

Applying the substitution rules (S1) and (S2), substituting every logical variable z_x by the corresponding instance variable x, and substituting every history variable by the empty sequence, denoted by nil, then gives us, after an trivial application of the consequence rule,

$$\vdash \{p[z_n/\mathsf{self}]\}\rho'\{Q\}.$$

An application of the rule (AUX) then finishes the proof. □

8 Conclusion

We have developed a formal proof system for reasoning about the partial correctness of programs written in the language POOL. In [Bo] we proved the system to be sound and complete with respect to a formal semantics.

We mention the following topics for future research: First, we have the problem of *compositionality*, i.e., the development of a proof system along the lines of [ZREB] in which the history variables introduced in the completeness proof as auxiliary variables are incorporated in the system itself.

Another interesting subject is the problem how to formalize reasoning about *deadlock* behaviour. Due to the presence of dynamic object creation the standard techniques developed for languages like CSP do not apply.

Finally, in the full language POOL an object can call its own methods. We did not study this feature because we wanted to focus on the remote procedure call mechanism in POOL. But we think we can incorporate the proof theory for recursive procedures ([Ap2]) in our assumption/commitment formalism.

Acknowledgements We are much indebted to the work of Pierre America on the proof theory for the language SPOOL. Furthermore we thank the members of the Amsterdam Concurrency Group, J.W. de Bakker, A. de Bruin, P.M.W. Knijnenburg, J.N. Kok, J.J.M.M. Rutten, E. de Vink en J.H.A. Warmerdam for their fruitfull comments.

References

[AB] P. America, F.S. de Boer: *A proof system for a parallel programming language with dynamic process creation.* Proceedings of the seventeenth International Colloquium on Automata, Languages and Programming, University of Warwick, England, July 16-20, 1990.

[AB2] P. America, F.S. de Boer: *A proof theory for a sequential version of POOL*, C.W.I. Report, to appear.

[Am] P. America: *Definition of the programming language POOL-T.* ESPRIT project 415A, Doc. No. 0091, Philips Research Laboratories, Eindhoven, the Netherlands, September 1985.

[Ap] K.R. Apt: *Formal justification of a proof system for CSP.* Journal ACM, Vol. 30, No. 1, 1983, pp. 197–216.

[Ap2] K.R. Apt: *Ten years of Hoare logic: a survey-part 1.* Journal ACM, Vol. 3, No. 4, 1981, pp. 431–483.

[AFR] K.R. Apt, N. Francez, W.P. de Roever: *A proof system for communicating processes*, ACM Transactions on Programming Languages and Systems, Vol. 2, No. 3, 1980, pp. 359–385.

[Ba] J.W. de Bakker: *Mathematical theory of program correctness*, Prentice-Hall International, Englewood Cliffs, New Jersey, 1980.

[Bo] F. S. de Boer: *A proof system for the language POOL*, CWI-report, to appear.

[GR] R. Gerth, W.P. de Roever: *A proof system for concurrent Ada programs.* Science of Computer Programming 4, pp. 159-204.

[Go] Adele Goldberg, David Robson: *Smalltalk-80, The Language and its Implementation.* Addison-Wesley, 1983.

[Ho] C.A.R. Hoare: *An axiomatic basis for computer programming.* Communications of the ACM, Vol. 12, No. 10, 1969, pp. 567–580,583.

[HR] J. Hooman, W.P. de Roever: *The quest goes on: towards compositional proof systems for CSP.* J.W. de Bakker, W.P. de Roever, G. Rozenberg (eds.): Current trends in concurrency, Proc. LPC/ESPRIT Advanced School, Springer Lecture Notes in Computer SCIENCE, Vol. 224, 1986.

[Sc] Dana S. Scott: *Identity and existence in intuitionistic logic.* M.P. Fourman, C.J. Mulvey, D.S. Scott (eds.): Applications of Sheaves. Proceedings, Durham 1977, Springer-Verlag, 1979, pp. 660–696 (Lecture Notes in Mathematics 753).

[ZREB] J. Zwiers, W.P. de Roever, P. van Emde Boas: *Compositionality and concurrent networks: soundness and completeness of a proof system.* Proceedings of the twelfth International Colloquium on Automata, Languages and Programming, Nafplion, Greece, July 15–19, 1985, Springer Lecture Notes in Computer Science 194, pp. 509–519.

In this series appeared :

No.	Author(s)	Title
85/01	R.H. Mak	The formal specification and derivation of CMOS-circuits.
85/02	W.M.C.J. van Overveld	On arithmetic operations with M-out-of-N-codes.
85/03	W.J.M. Lemmens	Use of a computer for evaluation of flow films.
85/04	T. Verhoeff H.M.L.J.Schols	Delay insensitive directed trace structures satisfy the foam the foam rubber wrapper postulate.
86/01	R. Koymans	Specifying message passing and real-time systems.
86/02	G.A. Bussing K.M. van Hee M. Voorhoeve	ELISA, A language for formal specification of information systems.
86/03	Rob Hoogerwoord	Some reflections on the implementation of trace structures.
86/04	G.J. Houben J. Paredaens K.M. van Hee	The partition of an information system in several systems.
86/05	J.L.G. Dietz K.M. van Hee	A framework for the conceptual modeling of discrete dynamic systems.
86/06	Tom Verhoeff	Nondeterminism and divergence created by concealment in CSP.
86/07	R. Gerth L. Shira	On proving communication closedness of distributed layers.
86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987).
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language.
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing.
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86).
86/12	A. Boucher R. Gerth	A timed failures model for extended communicating processes.
86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards VXrifying object oriented systems (Fund. Informatica

86/14	R. Koymans	Specifying passing systems requires extending temporal logic.
87/01	R. Gerth	On the existence of sound and complete axiomati zations of the monitor concept.
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases.
87/03	G.J. Houben J.Paredaens	A formal approach to distributed information systems.
87/04	T.Verhoeff	Delay-insensitive codes - An overview.
87/05	R.Kuiper	Enforcing non-determinism via linear time temporal logic specification.
87/06	R.Koymans	Temporele logica specificatie van message passing en real-time systemen (in Dutch).
87/07	R.Koymans	Specifying message passing and real-time systems with real-time temporal logic.
87/08	H.M.J.L. Schols	The maximum number of states after projection.
87/09	J. Kalisvaart L.R.A. Kessener W.J.M. Lemmens M.L.P. van Lierop F.J. Peters H.M.M. van de Wetering	Language extensions to study structures for raster graphics.
87/10	T.Verhoeff	Three families of maximally nondeterministic automata.
87/11	P.Lemmens	Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model.
87/12	K.M. van Hee and A.Lapinski	OR and AI approaches to decision support systems.
87/13	J.C.S.P. van der Woude	Playing with patterns - searching for strings.
87/14	J. Hooman	A compositional proof system for an occam-like real-time language.
87/15	C. Huizing R. Gerth W.P. de Roever	A compositional semantics for statecharts.
87/16	H.M.M. ten Eikelder J.C.F. Wilmont	Normal forms for a class of formulas.
87/17	K.M. van Hee G.-J.Houben J.L.G. Dietz	Modelling of discrete dynamic systems framework and examples.

87/18	C.W.A.M. van Overveld	An integer algorithm for rendering curved surfaces.
87/19	A.J.Seebregts	Optimalisering van file allocatie in gedistribueerde database systemen.
87/20	G.J. Houben J. Paredaens	The R^2 -Algebra: An extension of an algebra for nested relations.
87/21	R. Gerth M. Codish Y. Lichtenstein E. Shapiro	Fully abstract denotational semantics for concurrent PROLOG.
88/01	T. Verhoeff	A Parallel Program That Generates the Möbius Sequence.
88/02	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specification for Information Systems.
88/03	T. Verhoeff	Settling a Question about Pythagorean Triples.
88/04	G.J. Houben J.Paredaens D.Tahon	The Nested Relational Algebra: A Tool to Handle Structured Information.
88/05	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specifications for Information Systems.
88/06	H.M.J.L. Schols	Notes on Delay-Insensitive Communication.
88/07	C. Huizing R. Gerth W.P. de Roever	Modelling Statecharts behaviour in a fully abstract way.
88/08	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	A Formal model for System Specification.
88/09	A.T.M. Aerts K.M. van Hee	A Tutorial for Data Modelling.
88/10	J.C. Ebergen	A Formal Approach to Designing Delay Insensitive Circuits.
88/11	G.J. Houben J.Paredaens	A graphical interface formalism: specifying nested relational databases.
88/12	A.E. Eiben	Abstract theory of planning.
88/13	A. Bijlsma	A unified approach to sequences, bags, and trees.
88/14	H.M.M. ten Eikelder R.H. Mak	Language theory of a lambda-calculus with recursive types.

88/15	R. Bos C. Hemerik	An introduction to the category theoretic solution of recursive domain equations.
88/16	C.Hemerik J.P.Katoen	Bottom-up tree acceptors.
88/17	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable specifications for discrete event systems.
88/18	K.M. van Hee P.M.P. Rambags	Discrete event systems: concepts and basic results.
88/19	D.K. Hammer K.M. van Hee	Fasering en documentatie in software engineering.
88/20	K.M. van Hee L. Somers M.Voorhoeve	EXSPECT, the functional part.
89/1	E.Zs.Lepoeter-Molnar	Reconstruction of a 3-D surface from its normal vectors.
89/2	R.H. Mak P.Struik	A systolic design for dynamic programming.
89/3	H.M.M. Ten Eikelder C. Hemerik	Some category theoretical properties related to a model for a polymorphic lambda-calculus.
89/4	J.Zwiers W.P. de Roever	Compositionality and modularity in process specification and design: A trace-state based approach.
89/5	Wei Chen T.Verhoeff J.T.Udding	Networks of Communicating Processes and their (De-)Composition.
89/6	T.Verhoeff	Characterizations of Delay-Insensitive Communication Protocols.
89/7	P.Struik	A systematic design of a paralell program for Dirichlet convolution.
89/8	E.H.L.Aarts A.E.Eiben K.M. van Hee	A general theory of genetic algorithms.
89/9	K.M. van Hee P.M.P. Rambags	Discrete event systems: Dynamic versus static topology.
89/10	S.Ramesh	A new efficient implementation of CSP with output guards.
89/11	S.Ramesh	Algebraic specification and implementation of infinite processes.
89/12	A.T.M.Aerts K.M. van Hee	A concise formal framework for data modeling.

89/13	A.T.M.Aerts K.M. van Hee M.W.H. Hesen	A program generator for simulated annealing problems.
89/14	H.C.Haesen	ELDA, data manipulatie taal.
89/15	J.S.C.P. van der Woude	Optimal segmentations.
89/16	A.T.M.Aerts K.M. van Hee	Towards a framework for comparing data models.
89/17	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra.
90/1	W.P.de Roever-H.Barringer C.Courcoubetis-D.Gabbay R.Gerth-B.Jonsson-A.Pnueli M.Reed-J.Sifakis-J.Vytopil P.Wolper	Formal methods and tools for the development of distributed and real time systems, pp. 17.
90/2	K.M. van Hee P.M.P. Rambags	Dynamic process creation in high-level Petri nets, pp. 19.
90/3	R. Gerth	Foundations of Compositional Program Refinement - safety properties - , p. 38.
90/4	A. Peeters	Decomposition of delay-insensitive circuits, p. 25.
90/5	J.A. Brzozowski J.C. Ebergen	On the delay-sensitivity of gate networks, p. 23.
90/6	A.J.J.M. Marcelis	Typed inference systems : a reference document, p. 17.
90/7	A.J.J.M. Marcelis	A logic for one-pass, one-attributed grammars, p. 14.
90/8	M.B. Josephs	Receptive Process Theory, p. 16.
90/9	A.T.M. Aerts P.M.E. De Bra K.M. van Hee	Combining the functional and the relational model, p. 15.
90/10	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17).
90/11	P. America F.S. de Boer	A proof system for process creation, p. 84.
90/12	P.America F.S. de Boer	A proof theory for a sequential version of POOL, p. 110.
90/13	K.R. Apt F.S. de Boer E.R. Olderog	Proving termination of Parallel Programs, p. 7.
90/14	F.S. de Boer	A proof system for the language POOL, p. 70.
90/15	F.S. de Boer	Compositionality in the temporal logic of concurrent systems,

p. 17.

90/16 F.S. de Boer
C. Palamidessi

A fully abstract model for concurrent logic languages, p. 23.

90/17 F.S. de Boer
C. Palamidessi

On the asynchronous nature of communication in concurrent logic languages: a fully abstract model based on sequences, p. 29.