1991

P.H.M. America, F.S. de Boer

A proof theory for process creation

Computer Science/Department of Software Technology Report CS-R9119 March

CWI, nationaal instituut voor onderzoek op het gebied van wiskunde en informatica

CWI is the research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a non-profit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands organization for scientific research (NWO).

A Proof Theory for Process Creation

Pierre America
Philips Research Laboratories
P.O. Box 80000, 5600 JA Eindhoven
The Netherlands
Frank de Boer

CWI
P.O. Box 4079, 1009 AB Amsterdam,
The Netherlands

Abstract

We develop a Hoare-style proof system for a parallel language with process creation. One of the main objectives of the proof system is to formalize reasoning about dynamically evolving process structures at an abstraction level at least as high as that of the programming language.

1980 Mathematics Subject Classification: 70A05.
1986 CR Categories: F.3.1.
Key Words and Phrases: proof theory, pre- and post-conditions, process creation, dynamically evolving process structures, cooperation test, soundness, completeness.

Report CS-R9119
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

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 a programming language, which we simply call P. The language P is a simplified relative of POOL, a parallel object-oriented language [Am2]. POOL makes use of the structuring mechanisms of object-oriented programming [Mey], integrated with concepts for expressing concurrency: processes and communication.

A program of our language P 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 local 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. It contains exactly one value; this can be an integer or a boolean, or it can be a reference to an object. (This is the only essential difference between P and POOL: in POOL communication proceeds by a rendezvous mechanism, where a method, a kind of procedure, is invoked in the receiving object in response to a message.) 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 the communication structure of the processes is determined dynamically, without any regular structure imposed on it a priori. This is in contrast to 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 have not (yet) been created do not play a role.

Strictly speaking, direct dereferencing is not even allowed in the programming language, because each object has access to its own instance variables only. 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 (see, e.g, [Am3]). Nevertheless, the proof system presented in this paper formalizes reasoning about dynamically evolving pointer structures at a more appropriate abstraction level than the system developed in [Bo] where those structures are described in terms of some particular coding of the objects.

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. (This is not uncommon in proof systems dealing with more data types than integers only [TZ].) 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]. 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 communication or object creation. These statements are proved correct with respect to pre- and postconditions in the usual manner of sequential programs [Ap1, Ba, Ho1]. At this stage, we use *assumptions* to describe the behaviour of the communication and creation statements. These will be verified in the next stage. In this local stage, a *local assertion language* is used, which only talks about the current object in isolation.

- An *intermediate* stage. In this stage the above assumptions about communication and creation statements are verified. Here a *global assertion language* is used, which reasons about all the objects in the system. For each creation statement and for each pair of possibly communicating send and receive 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 that the proof system is sound and complete with respect to a formally defined semantics. Soundness means that everything that can be proved using the proof system is indeed true in the semantics. On the other hand, completeness means that every true property of a program that can be expressed using our assertion language can also be proved formally in the proof system. Due to the abstraction level of the assertion language we had to modify considerably the standard techniques for proving completeness.

Our paper is organized as follows: In the following section we describe the programming language P. In section 3 we define two assertion languages, the local one and the global one. Then, in section 4 we describe the proof system. Section 5 presents an example of a correctness proof for a nontrivial program. Section 6 presents the semantics of the programming language, of the assertion languages, and of the correctness formulas. In section 7 we prove the soundness of the proof system and in section 8 we prove completeness. The expressibility of the assertion languages is studied in section 9. Finally, in section 10 we draw some conclusions.

2 The programming language

In this section we define the programming language P of which we shall study the proof theory. This language is related to CSP [Ho2], in that it describes a number of processes that communicate synchronously by transmitting values to each other, but it has the additional possibility of dynamically creating processes and manipulating references to processes. It can also be compared to Smalltalk [GR], since it describes a dynamically evolving collection of objects where each has its own private data, but in P each object is provided with an autonomous local process and it communicates with other objects simply by exchanging a value instead of invoking a method.

A system, the result of executing a program written in P, consists of *objects*. On the one hand these objects have the properties of processes, that is, each of them has an internal activity, which runs in parallel with all the other objects in the system. On the other hand, objects are in some way like data records: they contain some internal data, and they have the ability to act on these data. An important characteristic of the objects in the language P is that they can be created dynamically: Whenever required during the execution of a program, a new object can be called into existence.

An object stores its internal data in *variables* (also called *instance variables* to distinguish them from the other kinds of variables that we shall need in the assertion language). A variable can contain a reference to an object, which can be another object, or possibly the object under consideration itself. Alternatively, it can contain an element of a standard, built-in data type, of which our language P contains only integers and booleans. The contents of a variable can be changed by an assignment statement. The variables of one object cannot be accessed directly by other objects. They can only be read and changed by the object to which they belong.

Interaction between objects takes place by sending messages. The language P uses a synchronous communication mechanism, that is, the sender and receiver of a message perform the communication at the same time; the one that reaches its communication statement first will wait for its partner. The sender of a message must always specify the receiver explicitly. The receiver however, has the possibility of mentioning the sender that it wants to communicate with, but it can also omit the indication of its communication partner, in which case it is willing to communicate with any sender that sends a message of the correct type. A message consists of a data value, which is transferred from the sender to the receiver. This data value can be a reference to an object or it can be an integer or boolean.

In order to describe by a program the unbounded number of objects in a system, we group them into *classes*. All objects in one class (the *instances* of that class) have the same structure of variables (each object has its own private variables, but the variables of all instances of a class have the same names and types) and they execute

identical local processes. In this way a class can be considered as a blueprint for creating new instances.

Let us now give a formal definition of the language P. We assume as given a set C of *class names*, with typical element c. By this we mean that symbols like c, c', c_1, etc. will range over the set C of class names. (Appendix A.1 gives an overview of the sets used in this paper, together with their typical elements.) The set $C \cup \{\mathsf{Int}, \mathsf{Bool}\}$ of *data types*, with typical element d, we denote by C^+. Here Int and Bool denote the types of the integers and booleans, respectively. For each $c \in C$ and $d \in C^+$ we assume $IVar_d^c$ to be the set of instance variables of type d in class c, with typical elements x_d^c and y_d^c. Such a variable x_d^c occurs in each object of class c and it can refer to objects of type d only. We assume that $IVar_d^c \cap IVar_{d'}^{c'} = \emptyset$ whenever $c \neq c'$ or $d \neq d'$. In cases where no confusion arises we omit the subscripts and superscripts.

Definition 2.1
We define the set Exp_d^c of expressions of type d in class c, with typical element e_d^c. Such an expression e_d^c can be evaluated by an object of class c and the object to which it refers will be of type d.

These expressions are defined as follows:

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

An expression e_d^c will be evaluated by a certain object α of class c. An expression of the form x denotes the value of the variable x that belongs to the object α. The expression self denotes the object α itself. The expression nil denotes no object at all. It can be used for every type, including Int and Bool. The symbols true and false stand for the corresponding values of type Bool. Every integer n can occur as an expression of type Int; it simply denotes itself. We assume that the standard arithmetic and comparison operations on integers are available, but we list only the operator '+'. We assume that all these operations result in nil whenever an error occurs (e.g., division by zero or nil as an operand). Finally, for every type we have a test for equality. The expression $e_1 \doteq e_2$ evaluates to true whenever e_1 and e_2 denote the same object (or both denote *no* object, viz. nil). Note that in the programming

language we put a dot over the equality sign ($\doteq$) to distinguish it from the equality sign we use in the metalanguage.

Definition 2.2
We next define the set $Stat^c$ of statements in class c, with typical element S^c. These statements can be executed by an object of class c.

Statements can be of the following forms:

$$
\begin{array}{rcll}
S^c & ::= & x_d^c := e_d^c & \\
 & | & x_d^c := \mathsf{new}_d & \text{if } d \neq \mathsf{Int}, \mathsf{Bool} \\
 & | & x_{c'}^c ! e_d^c & \\
 & | & x_{c'}^c ? y_d^c & \\
 & | & ?y_d^c & \\
 & | & S_1^c ; S_2^c & \\
 & | & \text{if } e_{\mathsf{Bool}}^c \text{ then } S_1^c \text{ else } S_2^c \text{ fi} & \\
 & | & \text{while } e_{\mathsf{Bool}}^c \text{ do } S^c \text{ od} &
\end{array}
$$

A statement S^c can be executed by an object of class c. The object executes the assignment statement $x := e$ by first evaluating the expression e at the right-hand side and then storing the result in its own variable x. The execution of the new-statement $x := \mathsf{new}_d$ by the object α consists of creating a new object β of class d and making the variable x of the creator α refer to it. The instance variables of the new object β are initialized to nil and β will immediately start executing its local process. It is not possible to create new elements of the standard data types Int and Bool.

A statement $x_{c'}^c ! e_d^c$ is called an *output* statement and statements like $x_{c'}^c ? y_d^c$ and $?y_d^c$ are called *input* statements. Together they are called I/O statements. The execution of an output statement $x_1{}_{c'}^c ! e_d^c$ by an object α is always synchronized with the execution of a corresponding input statement $x_2{}_c^{c'} ? y_d^{c'}$ or $?y_d^{c'}$ by another object β. Such a pair of input and output statements are said to *correspond* if all the following conditions are satisfied:

- The variable x_1 of the sending object α should refer to the receiving object β (therefore necessarily the type of the variable x_1 coincides with the class c' of β).
- If the input statement to be executed is of the form $x_2{}_c^{c'} ? y_d^{c'}$, then the variable x_2 of the receiving object β should refer to the sending object α (again, this means that the type of the variable x_2 coincides with the class c of α).
- The type d of the expression e_d^c in the output statements should coincide with the type of the destination variable $y_d^{c'}$ in the input statement.

If an object tries to execute a I/O statement, but no other object is trying to execute a corresponding statement yet, it must wait until such a communication partner appears. If two objects are ready to execute corresponding I/O statements, the communication may take place. This means that the value of the expression e in the sending object α is assigned to the destination variable y in the receiving object β. When an object is ready to execute an input statement $?y$ there may be several objects ready to execute a corresponding output statement. One of them is chosen non-deterministically.

Statements are built up from these atomic statements by means of sequential composition, denoted by the semicolon ';', the conditional construct if-then-else-fi and the iterative construct while-do-od. The meaning of these constructs we shall assume to be known.

Definition 2.3
Finally we define the set *Prog* of *programs*, with typical element ρ, as follows:

$$\rho ::= \langle c_1 \leftarrow S_1^{c_1}, \ldots, c_{n-1} \leftarrow S_{n-1}^{c_{n-1}} : S_n^{c_n} \rangle$$

Here we require that all the class names $c_1, \ldots, c_n$ are different. Furthermore we require for every variable x_d^c occurring in ρ that its type d is among $c_1, \ldots, c_n$, Int, Bool and that in every new-statement $x := \mathsf{new}_d$ the type d of the newly created object is among $c_1, \ldots, c_{n-1}$.

The first part of a program consists of a finite number of class definitions $c_i \leftarrow S_i$, which determine the local processes of the instances of the classes $c_1, \ldots, c_{n-1}$. Whenever a new object of class c_i is created, it will begin to execute the corresponding statement S_i. The second part specifies the local process S_n of the *root class* c_n. The execution of a program starts with the creation of a single instance of this root class, the *root object*, which begins executing the statement S_n. This root object can create other objects in order to establish parallelism. Due to the above restriction on the types of new-statements, the root object will always be the only instance of its class.

An example program

We illustrate the programming language by giving a program that generates the prime numbers up to a certain constant n. The program uses the sieve method of Eratosthenes. It consists of two classes. The class G (for 'generator') describes the behaviour of the root object, which consists of generating the natural numbers from 2 to n and sending them to an object of the other class P. The objects of the class P (for 'prime sieve') essentially form a chain of filters. Each of these objects remembers the first number it is sent; this will always be a prime. From the numbers it receives

subsequently, it will simply discard the ones that are divisible by its local prime number, and it will send the others to the next P object in the chain.

The class G makes use of two instance variables: f (for 'first') of type P and c (for 'count') of type Int (note that n is not an instance variable but an integer constant). The class P has three instance variables: m ('my prime') and b ('buffer') of type Int and l ('link') of type P. Here is the complete program:

```
⟨P ← ?m;
     if m ≠ nil
     then l := new;
          ?b;
          while b ≠ nil
          do if m ∤ b then l ! b; fi;
             ?b
          od;
          l ! b
     fi
   : f := new; c := 2;
     while c ≤ n do f ! c; c := c + 1 od;
     f ! nil⟩
```

3 The assertion language

In this section we define two different assertion languages. An *assertion* describes 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 is called the *local* assertion language. It will be used in the local proof system. The other one, the *global* assertion language, describes a whole system of objects. It will be used in the intermediate and global proof systems.

3.1 The local assertion language

First we introduce a new kind of variables: For $d = \mathsf{Int}, \mathsf{Bool}$, let $LogVar_d$ 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 $LExp^c_d$ of *local expressions* of type d in class c, with typical element l^c_d, is defined as follows:

$$
\begin{array}{rcll}
l^c_d & ::= & z_d & \\
 & | & x^c_d & \\
 & | & \mathsf{self} & \text{if } d = c \\
 & | & \mathsf{nil} & \\
 & | & n & \text{if } d = \mathsf{Int} \\
 & | & \mathsf{true} \mid \mathsf{false} & \text{if } d = \mathsf{Bool} \\
 & | & l^c_{1\mathsf{Int}} + l^c_{2\mathsf{Int}} & \text{if } d = \mathsf{Int} \\
 & \vdots & & \\
 & | & l^c_{1d'} \doteq l^c_{2d'} & \text{if } d = \mathsf{Bool}
\end{array}
$$

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

$$
\begin{array}{rcll}
p^c & ::= & l^c_{\mathsf{Bool}} & \\
 & | & \neg p^c & \\
 & | & p^c_1 \wedge p^c_2 & \\
 & | & \exists z_d\, p^c & (d = \mathsf{Bool}, \mathsf{Int})
\end{array}
$$

We shall regard other logical connectives ($\vee, \rightarrow, \forall$) as abbreviations for combinations of the above ones.

An example of a local assertion is

$$\neg(\mathsf{b} \doteq \mathsf{nil}) \rightarrow \forall i(2 \leq i \wedge i \leq \mathsf{m} \rightarrow \neg(i \mid \mathsf{b})),$$

which we might abbreviate to $\mathsf{b} \neq \mathsf{nil} \rightarrow \forall i(2 \leq i \leq \mathsf{m} \rightarrow i \not\mid \mathsf{b})$. Here i is a logical variable of type Int. (Most of our examples will be in the context of the program at the end of section 2.)

Local expressions l_d^c and local assertions p^c are evaluated with respect to the local state of an object of class c, determining the values of its instance variables, plus a logical environment, which assigns values to the logical variables. Therefore 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, so that only quantification over integers and booleans is possible. (By the way, the value nil is *not* included in the range of quantifications.) As we shall see below, quantification over other types would require knowledge of the set of existing objects, which is not available locally.

3.2 The global assertion language

Next we define the global assertion language. Since in the global assertion language we also want to quantify over objects of any class $c \in C^+$, we now need for every $c \in C$ a new set $LogVar_c$ 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 in addition we assume for every $d \in C^+$ the set $LogVar_{d^*}$ of logical variables of type d^*, which range over finite sequences of elements of type d. Therefore we now have a set $LogVar_a$ 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:

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

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. For sequence types the expression nil denotes the empty sequence. 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. $|g|$ denotes the length of the sequence denoted by g. The expression $g_1 : g_2$ denotes the nth element of the sequence denoted by g_1, where n is the value of g_2 (if g_2 is less than 1 or greater than $|g_1|$, the result is nil.) The conditional expression if g_0 then g_1 else g_2 fi is introduced to facilitate the handling of aliasing (see definition 4.3). If the condition is nil, then the result of the conditional expression is nil, too.

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

$$
\begin{array}{lcl}
P & ::= & g_{\mathsf{Bool}} \\
 & | & \neg P \\
 & | & P_1 \wedge P_2 \\
 & | & \exists z_a\, P
\end{array}
$$

Again, other logical connectives are regarded as abbreviations.

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

$\exists z_c$ true is false in some state iff there are no objects of class c in this state. More interestingly, the assertion

$$\forall p\, \exists s{\bullet}(s : 1 \doteq g.\mathsf{f} \wedge s : |s| \doteq p \wedge \forall i(1 \leq i < |s| \rightarrow (s : i).\mathsf{l} \doteq s : (i+1)))$$

expresses that every object of class P is a member of the l-linked chain that starts with g.f (where g is the generator object).

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 I/O 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_d^c and a global expression g_c we define a global expression $l_d^c \downarrow g_c$. This expression denotes the result of evaluating the local expression l in the object denoted by the global expression g. The definition proceeds by induction on the complexity of the local expression l:

$$\begin{array}{lcll} l \downarrow g & = & l & \text{if } l = z, \mathsf{nil}, n, \mathsf{true}, \mathsf{false} \\ x \downarrow g & = & g.x & \\ \mathsf{self} \downarrow g & = & g & \\ (l_1 + l_2) \downarrow g & = & (l_1 \downarrow g) + (l_2 \downarrow g) & \\ (l_1 \doteq l_2) \downarrow g & = & (l_1 \downarrow g) \doteq (l_2 \downarrow g) & \end{array}$$

For a local assertion p^c we define the global assertion $p^c \downarrow g_c$ as follows:

$$\begin{array}{lcl} (\neg p) \downarrow g & = & (\neg p \downarrow g) \\ (p_1 \wedge p_2) \downarrow g & = & (p_1 \downarrow g) \wedge (p_2 \downarrow g) \\ (\exists z\, p) \downarrow g & = & \exists z(p \downarrow g) \end{array}$$

As an example, note that $(\mathsf{b} \not\doteq \mathsf{nil} \rightarrow \forall i(2 \leq i \leq \mathsf{m} \rightarrow i \not| \mathsf{b})) \downarrow p$ is equal to $p.\mathsf{b} \not\doteq \mathsf{nil} \rightarrow \forall i(2 \leq i \leq p.\mathsf{m} \rightarrow i \not| p.\mathsf{b})$.

3.3 Correctness formulas

In this section we define how we specify an object and a complete system of objects, using the formalism of Hoare triples. We start with the specification of an object.

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

$$\{p^c\}S^c\{q^c\}.$$

Here the assertion p is called the *precondition* and the assertion q is called the *postcondition.* The meaning of such a correctness formula is described informally as follows:

> Every terminating execution of S by an object of class c starting from a state satisfying p will end in a state satisfying q.

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
A bracketed section is a construct of the form $\langle S_1^c; S^c; S_2^c \rangle$, where S^c denotes an I/O statement or a new-statement, and in S_1 and S_2 neither I/O statements nor new-statements occur (note that S_1 and S_2 can each be composed of several statements by means of sequential compositions, conditionals, or loops). Furthermore, if $S = x := \text{new}$ then the variable x must not occur at the left-hand side of an assignment occurring in S_2. The additional condition on the variable x, which is used to store the identity of the new object, is to ensure that after the execution of the corresponding bracketed section this new object is still referred to by the variable x.

Next we define intermediate correctness formulas, which describe the behaviour of an object executing a bracketed section containing a new-statement or a communication between two objects.

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

- $\{P\}(z_c, S^c)\{Q\}$, where S is a local statement or a bracketed section containing a new-statement.
- $\{P\}(z_{c_1}, S_1^{c_1}) \parallel (z'_{c_2}, S_2^{c_2})\{Q\}$, where z and z' are distinct logical variables and $\langle S_1 \rangle$ and $\langle S_2 \rangle$ are bracketed sections that contain I/O 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 $\{P\}(z, S)\{Q\}$ is as follows:

> Every terminating execution of the bracketed section S by the object denoted by the logical variable z starting in a (global) state satisfying P ends in a (global) state satisfying Q.

The meaning of the second form of intermediate correctness formula, $\{P\}(z, S_1) \parallel (z', S_2)\{Q\}$, can be described as follows:

> Every terminating parallel execution of the bracketed section S_1 by the object denoted by the logical variable z and of S_2 by the object denoted by z' starting in a (global) state satisfying P will end in a (global) state satisfying Q.

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

Definition 3.9
A global correctness formula is of the form

$$\{p^c \downarrow z_c\}\rho^c\{Q\}$$

The variable z 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 to a global one. On the other hand, the final state of an execution of a complete system is described by an arbitrary global assertion. The meaning of the global correctness formula $\{p \downarrow z\}\rho\{Q\}$ can be rendered as follows:

> If the execution of the program 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, allows us to reason about the correctness of a single object. Testing the assumptions that are introduced at this first level to deal with I/O statements 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 is similar to the usual system for sequential programs.

Definition 4.1
The local proof system consists of the following axiom and rules:

Assignment:

$$\{p[e/x]\}x := e\{p\} \qquad \text{(LASS)}$$

Sequential composition:

$$\frac{\{p\}S_1\{r\}, \quad \{r\}S_2\{q\}}{\{p\}S_1; S_2\{q\}} \qquad \text{(LSC)}$$

Conditional:

$$\frac{\{p \wedge e\}S_1\{q\}, \quad \{p \wedge \neg e\}S_2\{q\}}{\{p\}\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}\{q\}} \qquad \text{(LCOND)}$$

Iteration:

$$\frac{\{p \wedge e\}S\{p\}}{\{p\}\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}\{p \wedge \neg e\}} \qquad \text{(LIT)}$$

Consequence:

$$\frac{p \rightarrow p_1, \quad \{p_1\}S\{q_1\}, \quad q_1 \rightarrow q}{\{p\}S\{q\}} \qquad \text{(LCR)}$$

Reasoning about new-statements and I/O statements is done by introducing assumptions about them, where assumptions have the form of local correctness formulas.

The substitution operation $[e/x]$ occurring in the above assignment axiom is the ordinary substitution, i.e., literal replacement of every occurrence of the variable x by the expression e. This works because at this level we have no aliasing, i.e., it is not possible that different local expressions denote the same variable. In the intermediate and global proof systems we shall have to take special measures to deal with aliasing.

4.2 The intermediate proof system

In this section we present the proof system for intermediate correctness formulas.

4.2.1 The assignment axiom

Definition 4.2
The assignment axiom in the intermediate proof system has the form

$$\{P[e \downarrow z/z.x]\}(z, x := e)\{P\} \qquad \text{(IASS)}$$

where $x \in IVar_d^c$, $e \in Exp_d^c$, $z \in LogVar_c$, and $P \in GAss$.

First note that we have to transform the expression e to the global expression $e \downarrow z$ and substitute this latter expression for $z.x$ because we consider the execution of the assignment $x := e$ by the object denoted by z. Furthermore we have to pay special attention to the substitution $[e \downarrow z/z.x]$ because the usual substitution does not take into account that there are many different global expressions that may denote the same variable $z.x$. This problem is solved by the following definition.

Definition 4.3
Given a global expression g_d, a logical variable z_c and an instance variable x_d^c, we define for any global expression g' the substitution $g'[g/z.x]$ by induction on the complexity of g' as follows.

$$\begin{array}{lcll}
g'[g/z.x] & = & g' & \text{if } g' = z', n, \mathsf{nil}, \mathsf{self}, \mathsf{true}, \mathsf{false} \\
(g'.y)[g/z.x] & = & g'[g/z.x].y & \text{if } y \neq x \\
(g'.x)[g/z.x] & = & \mathsf{if}\ g'[g/z.x] = z\ \mathsf{then}\ g\ \mathsf{else}\ g'[g/z.x].x\ \mathsf{fi} & \\
 & \vdots & & \\
(g_1 \doteq g_2)[g/z.x] & = & (g_1[g/z.x]) \doteq (g_2[g/z.x]) &
\end{array}$$

The omitted cases are defined directly from the application of the substitution to the subexpressions, like the last one. This substitution operation is generalized to global assertions in a straightforward manner, with the notation $P[g/z.x]$.

As an example, consider the postcondition $\forall i(1 \leq i \leq |s| \rightarrow (s : i).x \doteq i)$ for the statement $x := y + 1$, executed by the object denoted by z. The precondition given by the axiom (IASS) is

$$\forall i(1 \leq i \leq |s| \rightarrow \mathsf{if}\ s : i \doteq z\ \mathsf{then}\ z.y + 1\ \mathsf{else}\ (s : i).x\ \mathsf{fi} \doteq i).$$

The best way to justify definition 4.3 is by comparing it to the ordinary substitution $[e/x]$, which is used in the local assignment axiom (LASS). The essential property of this substitution is that the substituted expression, evaluated in the state before the assignment, has the same value as the original expression in the state after the assignment. Quasi-formally, we could write this as

$$[\![l[e/x]]\!](\sigma) = [\![l]\!](\sigma')$$

where σ' is the state that results from executing the assignment $x := e$ in the state σ. We could say that the substitution is a way of predicting the value that an expression or assertion will have after performing an assignment.

It is easy to prove that the substitution operation defined in definition 4.3 has exactly the same property:

$$[\![g'[g/z.x]]\!](\sigma) = [\![g']\!](\sigma')$$

and

$$\sigma \models P[g/z.x] \iff \sigma' \models P$$

where σ' is the state that results from σ by changing the value of the variable x in the object denoted by z to the value $[\![g]\!](\sigma)$ that results from evaluating the expression g in the state σ.

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 is possible: After the assignment it may be the case that g' refers to the same object as the logical variable z, so that $g'.x$ is the same variable as $z.x$, which has the value $[\![g]\!](\sigma)$. It is also possible that, after the assignment, g' does not refer to the object denoted by z, so that the value of $g'.x$ does not change. Since we can not decide between these possibilities by the form of the expression only, a conditional expression is constructed which decides dynamically.

The notation [./.] may seem overloaded now, but it is always possible to determine the required operation from the form of the arguments (see appendix A.2).

4.2.2 The creation of new objects

Definition 4.4
We describe the new-statement by the following axiom of the intermediate proof system:

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

where $x \in IVar^c_d$, $d \in C$, $z \in LogVar_c$, $z' \in LogVar_d$, $z \neq z'$, $P \in GAss$, and z' does not occur in P.

This axiom reflects the fact that we can view the execution of the statement $x := \mathsf{new}$ as consisting of two steps: first the new object is created and temporarily stored in the logical variable z', and then the value of this logical variable z' is assigned to the instance variable x of the object denoted by z. The second step is dealt with by the substitution $[z'/z.x]$ of definition 4.3, which takes care of all possible complications of aliasing.

For the creation of a new object we have to define another substitution operation $[\mathsf{new}/z]$. This is complicated by the fact that the newly created object does not exist in the state just before its creation, so that in this state we can not refer to it. Fortunately, we do need this substitution for all possible global expressions, but primarily for assertions, and in an assertion the logical variable z can essentially occur in only two contexts: either one of its instance variables is referenced, or it is compared for equality with another expression. In both cases we can predict the outcome without having to refer to the new object.

Definition 4.5
Let $d \in C$ and $z \in \mathit{LogVar}_d$. For certain global expressions g we define $g[\mathsf{new}/z]$ by induction on the complexity of g. We only list the interesting cases.

$z'[\mathsf{new}/z]$	$=$	z'	if $z' \neq z$
$z[\mathsf{new}/z]$		is undefined	
$g[\mathsf{new}/z]$	$=$	g	if $g = n, \mathsf{nil}, \mathsf{self}, \mathsf{true}, \mathsf{false}$
$(z'.x)[\mathsf{new}/z]$	$=$	$z'.x$	if $z' \neq z$
$(z.x)[\mathsf{new}/z]$	$=$	nil	
$(g.y.x)[\mathsf{new}/z]$	$=$	$(g.y)[\mathsf{new}/z].x$	
$(g_1 \doteq g_2)[\mathsf{new}/z]$	$=$	$g_1[\mathsf{new}/z] \doteq g_2[\mathsf{new}/z]$	if $g_1, g_2 \neq z$, if ... fi
$(g_1 \doteq z)[\mathsf{new}/z]$	$=$	false	if $g_1 \neq z$, if ... fi
$(z \doteq g_2)[\mathsf{new}/z]$	$=$	false	if $g_2 \neq z$, if ... fi
$(z \doteq z)[\mathsf{new}/z]$	$=$	true	

Here we have ignored the conditional expressions (these can be removed before substituting). In all the other cases not listed above, the substitution $[\mathsf{new}/z]$ can be applied to an expression by applying it to the constituent expressions. In this way $g[\mathsf{new}/z]$ is defined for all global expressions g except for z itself (and for certain conditional expressions). Again it is rather easy to prove that this substitution has the desired property:

$$[\![g[\mathsf{new}/z]]\!](\sigma) = [\![g]\!](\sigma')$$

where σ' can be obtained from σ by creating a new object (with all its variables initialized to nil) and storing it in the variable z.

Definition 4.6
We define $P[\mathsf{new}/z_c]$ by induction on the complexity of the global assertion P.

$$
\begin{array}{lcll}
g_{\mathsf{Bool}}[\mathsf{new}/z_c] & & \text{as in definition 4.5} & \\
(\neg P)[\mathsf{new}/z_c] & = & \neg(P[\mathsf{new}/z_c]) & \\
(P_1 \wedge P_2)[\mathsf{new}/z_c] & = & (P_1[\mathsf{new}/z_c] \wedge P_2[\mathsf{new}/z_c]) & \\
(\exists z_a P)[\mathsf{new}/z_c] & = & \exists z_a(P[\mathsf{new}/z_c]) & \text{if } a \neq c, c^* \\
(\exists z'_c P)[\mathsf{new}/z_c] & = & \exists z'_c(P[\mathsf{new}/z_c]) \vee (P[z_c/z'_c][\mathsf{new}/z_c]) & \text{if } z'_c \neq z_c \\
(\exists z_{c^*} P)[\mathsf{new}/z_c] & = & \exists z_{c^*} \exists z_{\mathsf{Bool}^*}(|z_{c^*}| \doteq |z_{\mathsf{Bool}^*}| \wedge P[z_{\mathsf{Bool}^*}, z_c/z_{c^*}][\mathsf{new}/z_c]) &
\end{array}
$$

Here we assume that z_{Bool^*} does not occur in P. The substitution $[z_{\mathsf{Bool}^*}, z_c/z_{c^*}]$ will be defined in definition 4.7.

The case of quantification over the type c of the newly created object can be explained as follows: Remember that we interpret the result of the substitution in a state in which the object denoted by z does not yet exist. In the first part $\exists z'(P[\mathsf{new}/z])$ of the substituted formula the bound variable z' thus ranges over all the old objects. In the second part the object to be created is dealt with separately. This is done by first substituting z for z' (expressing that the quantified variable z' takes the same value as the variable z) and then applying the substitution $[\mathsf{new}/z]$ (note that simply applying $[\mathsf{new}/z']$ does not give the right result in the case that z occurs in P). Together the two parts of the substituted formula express quantification over the whole range of existing objects in the new state.

Let us consider, for example, the statement $x := \mathsf{new}$, to be executed by the object indicated by the logical variable z, and the given postcondition

$$\forall v(v.x \neq \mathsf{nil} \rightarrow v.y \doteq 1).$$

In order to determine the corresponding precondition given by the axiom (NEW), we first apply the substitution $[z'/z.x]$, which leads to

$$\forall v(\mathsf{if}\ v \doteq z\ \mathsf{then}\ z'\ \mathsf{else}\ v.x\ \mathsf{fi} \neq \mathsf{nil} \rightarrow v.y \doteq 1).$$

We can remove the conditional expression by taking the equivalent assertion

$$\forall v((v \doteq z \wedge z' \neq \mathsf{nil}) \vee (v \neq z \wedge v.x \neq \mathsf{nil}) \rightarrow v.y \doteq 1).$$

Now to this we apply the substitution $[\mathsf{new}/z']$, resulting in

$$\begin{array}{c}
\forall v((v \doteq z \wedge \mathsf{true}) \vee (v \neq z \wedge v.x \neq \mathsf{nil}) \rightarrow v.y \doteq 1) \\
\wedge\ ((\mathsf{false} \wedge \mathsf{true}) \vee (\mathsf{true} \wedge \mathsf{nil} \neq \mathsf{nil}) \rightarrow \mathsf{nil} \doteq 1),
\end{array}$$

which can be simplified to

$$\forall v(v \doteq z \vee v.x \neq \mathsf{nil} \rightarrow v.y \doteq 1).$$

For quantification about sequences of objects of class c, we need a slightly more elaborate mechanism. The sequences over which we quantify in the old state cannot contain the new object as an element. Therefore we use two sequence variables $z_{c^\bullet}$ and $z_{\mathsf{Bool}^\bullet}$ to code one sequence of objects in the new state. The idea of the substitution operation $[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}]$ is that at the places where $z_{\mathsf{Bool}^\bullet}$ yields true, the value of the coded sequence is the newly created object, here denoted by the variable z_c. Where $z_{\mathsf{Bool}^\bullet}$ yields false, the value of the coded sequence is the same as the value of $z_{c^\bullet}$ and where $z_{\mathsf{Bool}^\bullet}$ delivers nil the sequence also yields nil. This is formalized in the following definition.

Definition 4.7
For certain global expressions g we define $g[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}]$ as follows (again we list only the interesting cases):

$$
\begin{array}{lcll}
z_{c^\bullet}[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}] & & \text{is undefined} & \\
z[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}] & = & z \quad \text{if } z \neq z_{c^\bullet} & \\
g[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}] & = & g \quad \text{if } g = n, \mathsf{nil}, \mathsf{self}, \mathsf{true}, \mathsf{false} & \\
(g.x)[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}] & = & g[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}].x & \\
(z_{c^\bullet} : g)[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}] & = & \text{if } z_{\mathsf{Bool}^\bullet} : (g[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}]) & \\
 & & \text{then } z_c & \\
 & & \text{else } z_{c^\bullet} : (g[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}]) & \\
 & & \text{fi} & \\
(g_1 : g_2)[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}] & = & g_1[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}] : g_2[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}] & \text{if } g_1 \neq z_{c^\bullet} \\
(|z_{c^\bullet}|)[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}] & = & |z_{c^\bullet}| & \\
(|g|)[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}] & = & |g[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}]| \quad \text{if } g \neq z_{c^\bullet} &
\end{array}
$$

The generalization of the above to other global expressions and to global assertions is straightforward.

Again we have the desired property:

$$\sigma \models P[\mathsf{new}/z] \iff \sigma' \models P$$

where σ' can be obtained from σ by creating a new object (with all its variables initialized to nil) and storing it in the variable z.

We illustrate the last definition by another example of the use of axiom (NEW). Again we take the statement $x := \mathsf{new}$ executed by the object in z, but this time the postcondition is $\exists s \, \forall p \, \exists i (s : i \doteq p)$. Applying the substitution $[z'/z.x]$ leaves this

assertion unchanged, so we can directly apply the substitution [new/z']. We calculate in the following few steps:

$$
\begin{aligned}
&\Big(\exists s\, \forall p\, \exists i(s : i \doteq p)\Big)[\mathsf{new}/z'] \\
&= \exists s\, \exists b_{\mathsf{Bool}^\bullet}\, |s| \doteq |b| \wedge \Big(\forall p\, \exists i(s : i \doteq p)\Big)[b, z'/s][\mathsf{new}/z'] \\
&= \exists s\, \exists b\, |s| \doteq |b| \wedge \Big(\forall p\, \exists i(\mathsf{if}\ b : i\ \mathsf{then}\ z'\ \mathsf{else}\ s : i\ \mathsf{fi} \doteq p)\Big)[\mathsf{new}/z'] \\
&\equiv \exists s\, \exists b\, |s| \doteq |b| \wedge \Big(\forall p\, \exists i(b : i \wedge z' \doteq p) \vee (\neg b : i \wedge s : i \doteq p)\Big)[\mathsf{new}/z'] \\
&= \exists s\, \exists b\, |s| \doteq |b| \wedge \forall p\, \exists i\Big((b : i \wedge \mathsf{false}) \vee (\neg b : i \wedge s : i \doteq p)\Big) \\
&\qquad\qquad \wedge\, \exists i(b : i \wedge \mathsf{true}) \vee (\neg b : i \wedge \mathsf{false}) \\
&\equiv \exists s\, \exists b\Big(|s| \doteq |b| \wedge \forall p\, \exists i(\neg b : i \wedge s : i \doteq p) \wedge \exists i(b : i)\Big)
\end{aligned}
$$

Here $\equiv$ denotes semantic equivalence, which means that the assertions on both sides will have the same truth value in every environment. (By the way, in any state that can occur in the execution of a P program, the above assertions will be true, since there is only a finite number of objects of any class.)

4.2.3 Communication

Now we define an axiom and some rules which together describe the communication between objects.

Definition 4.8
Let $S_1 \in Stat^c$ be of the form $x?y$ or $?y$ and $S_2 = x'!e \in Stat^{c'}$ such that $x \in IVar^c_{c'}$, $x' \in IVar^{c'}_c$, and the variable y is of the same type as the expression e. Such a pair of I/O statements are said to *match*. Furthermore let $z \in LogVar_c$ and $z' \in LogVar_{c'}$ be two distinct variables and let P be a global assertion. Then the following is an instance of the communication axiom:

$$\{P[e \downarrow z'/z.y]\}(z, S_1) \parallel (z', S_2)\{P\} \qquad \text{(COMM)}$$

Note that the communication is described by a substitution that expresses the assignment of $e \downarrow z'$ to $z.y$ (definition 4.3).

The following two rules show how one can use the information about the relationship between the receiver and the sender which must hold for the communication to take place, that is, the sender must refer to the receiver, and if the receiver executes an input statement of the form $x?y$, it must refer to the sender.

Definition 4.9
Let $x?y, ?y \in Stat^c$ and $x'!e \in Stat^{c'}$ be such that both input statements match with the output statement. Furthermore let $z \in LogVar_c$ and $z' \in LogVar_{c'}$ be two distinct variables. Then we have the following two rules:

$$\frac{\{P \wedge z.x \doteq z' \wedge z' \neq \mathsf{nil} \wedge z'.x' \doteq z \wedge z \neq \mathsf{nil} \wedge R\}(z, x?y) \parallel (z', x'!e)\{Q\}}{\{P\}(z, x?y) \parallel (z', x'!e)\{Q\}} \quad \text{(SR1)}$$

and

$$\frac{\{P \wedge z'.x' \doteq z \wedge z \neq \mathsf{nil} \wedge R\}(z, ?y) \parallel (z', x'!e)\{Q\}}{\{P\}(z, ?y) \parallel (z', x'!e)\{Q\}} \quad \text{(SR2)}$$

where $R = (z \neq z')$ if $c = c'$ and $R = \mathsf{true}$ otherwise.

The following rule describes the independent parallel execution of two objects.

Definition 4.10
Suppose that $S_1 \in Stat^c$ and $S_2 \in Stat^{c'}$ do not contain any I/O or new-statements. Furthermore let $z \in LogVar_c$ and $z' \in LogVar_{c'}$ be two distinct variables. Then we have the following rule:

$$\frac{\{P\}(z, S_1)\{R\}, \quad \{R\}(z', S_2)\{Q\}}{\{P\}(z, S_1) \parallel (z', S_2)\{Q\}} \quad \text{(PAR1)}$$

Note that this rule models the fact that the parallel execution of two local computations can be sequentialized. The next rule takes care of the case where the two bracketed sections do contain I/O statements.

Definition 4.11
Let $\langle S_1; S; S_2\rangle \in Stat^c$ and $\langle S_1'; S'; S_2'\rangle \in Stat^{c'}$ be two bracketed sections containing I/O statements. Furthermore let $z \in LogVar_c$ and $z' \in LogVar_{c'}$ be two distinct variables. Then we have the rule

$$\frac{\{P\}(z, S_1) \parallel (z', S_1')\{P_1\}, \quad \{P_1\}(z, S) \parallel (z', S')\{Q_1\}, \quad \{Q_1\}(z, S_2) \parallel (z', S_2')\{Q\}}{\{P\}(z, S_1; S; S_2) \parallel (z', S_1'; S'; S_2')\{Q\}} \quad \text{(PAR2)}$$

Finally, the intermediate proof system contains rules for sequential composition (ISC), the conditional statement (ICOND), the iterative construct (IIT), and a consequence rule (ICR). These are straightforward modifications of the corresponding rules of the local proof system, so we only give the iteration rule as an example.

Definition 4.12
Let while t do S od $\in Stat^c$ and $z \in LogVar_c$. Then we have the following rule:

$$\frac{\{P \wedge e \downarrow z\}(z, S)\{Q\}}{\{P\}(z, \mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od})\{P \wedge \neg e \downarrow z\}} \quad \text{(IIT)}$$

4.3 The global proof system

In this section we describe the global proof system. Two bracketed sections containing I/O statements are said to *match* if the corresponding I/O statements match. A program ρ is said to be *bracketed* if every I/O statement and new-statement occurs in a bracketed section. An *assumption* is defined to be a local correctness assertion about a bracketed section. Given a set A of assumptions, the construct $A \vdash \{p^c\}S^c\{q^c\}$ denotes the derivability of the local correctness formula $\{p^c\}S^c\{q^c\}$ from the local proof system using the assumptions of A as additional axioms. Now we are ready to define the notion of the cooperation test:

Definition 4.13
Let $\rho = \langle c_1 \leftarrow S^{c_1}, \ldots, c_{n-1} \leftarrow S^{c_{n-1}} : S^{c_n}\rangle$ be bracketed. Furthermore, for each k, $1 \leq k \leq n$, let A^{c_k} denote a set of local correctness formulas about the bracketed sections occurring in S^{c_k} such that $A^{c_k} \vdash \{p^{c_k}\}S^{c_k}\{q^{c_k}\}$. Finally, let I be some global assertion, which we shall call the *global invariant*. Now we say that the *cooperation test* holds, notation $Coop(A^{c_1}, \ldots, A^{c_n}, I, \rho)$, if the following conditions are fulfilled:

1. The invariant I does not contain any instance variable that occurs at the left-hand side of an assignment outside a bracketed section.

2. Let $\langle S_1\rangle$ and $\langle S_2\rangle$ be two matching bracketed sections such that $\{p_1\}S_1\{q_1\} \in A^{c_i}$ and $\{p_2\}S_2\{q_2\} \in A^{c_j}$, where $1 \leq i \leq n-1$, $1 \leq j \leq n$. Furthermore let $z \in LogVar_{c_i}$ and $z' \in LogVar_{c_j}$ be two new distinct variables. Then we require

$$\vdash \{I \wedge p_1 \downarrow z \wedge p_2 \downarrow z'\}(z, S_1) \parallel (z', S_2)\{I \wedge q_1 \downarrow z \wedge q_2 \downarrow z'\}.$$

3. Let $\langle S\rangle$ be a bracketed section containing the new-statement $x := \mathsf{new}_{c_j}$ such that $\{p_1\}S\{q_1\} \in A^{c_i}$. Furthermore let $z \in LogVar_{c_i}$ be a new variable. Then we require that

$$\vdash \{I \wedge p_1 \downarrow z\}(z, S)\{I \wedge q_1 \downarrow z \wedge p^{c_j} \downarrow z.x\}.$$

4. Let z_{c_n} be a new variable. Then the following assertion should hold:

$$p^{c_n} \downarrow z_{c_n} \wedge \forall z'_{c_n}(z'_{c_n} \doteq z_{c_n}) \wedge \bigwedge_{1 \leq i < n} (\forall z_{c_i}\, \mathsf{false}) \rightarrow I.$$

The syntactic restriction in clause 1 on occurrences of variables in the global invariant I implies the invariance of this assertion over those parts of the program that are not contained in a bracketed section. The clauses 2 and 3 imply, among others, the invariance of the global invariant over the bracketed sections.

This global invariant expresses some invariant properties of the global states 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, and an object is allowed to enter a bracketed section containing a I/O statement only if there is at most one other object executing a bracketed section.

Clause 2 establishes the cooperation between two arbitrary matching assumptions, where two assumptions are said to match if their corresponding bracketed sections contain matching I/O statements. Note that since there exists only one object of class c_n, the root object, we do not have to apply the cooperation test between two matching assumptions of the set A^{c_n}.

Clause 3 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 the variable x of the new-statement cannot occur as the left-hand side of an assignment after the new-statement. Therefore it refers to the newly created object immediately after the execution of such a bracketed section.

Clause 4 establishes the truth of the global invariant in the initial state. Note that the assertion $\forall z_c$ false expresses that there exist no objects of class c. The assertion $\forall z'_{c_n}(z'_{c_n} \doteq z_{c_n})$ expresses that there exists precisely one object of class c_n.

In the following definitions, let $\rho = \langle c_1 \leftarrow S^{c_1}, \ldots, c_{n-1} \leftarrow S^{c_{n-1}} : S^{c_n} \rangle$.

Definition 4.14
The program rule of the global proof system has the following form:

$$\frac{Coop(A^{c_1}, \ldots, A^{c_n}, I, \rho) \qquad A^{c_k} \vdash \{p^{c_k}\} S^{c_k} \{q^{c_k}\}, 1 \leq k \leq n}{\{p^{c_n} \downarrow z_{c_n}\} \rho \{I \wedge q^{c_n} \downarrow z_{c_n} \wedge \bigwedge_{1 \leq i < n} \forall z_{c_i}\, q^{c_i} \downarrow z_{c_i}\}} \qquad \text{(PR)}$$

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 assertion $q^{c_n} \downarrow z_{c_n}$ expressing that the final local state of the root object is characterized by the local assertion q^{c_n}, and the assertions $\forall z_{c_i}\, q^{c_i} \downarrow z_{c_i}$, which express that the final local state of every object of class c_i is characterized by the local assertion q^{c_i}.

Definition 4.15
We have the following consequence rule for programs:

$$\frac{p^{c_n} \rightarrow p_1^{c_n}, \quad \{p_1^{c_n} \downarrow z_{c_n}\} \rho \{Q_1\}, \quad Q_1 \rightarrow Q}{\{p^{c_n} \downarrow z_{c_n}\} \rho \{Q\}} \qquad \text{(PCR)}$$

Definition 4.16
Furthermore, we have a rule for *auxiliary variables*: For a given program ρ and a (global) postcondition Q, let Aux be a set of instance variables such that

- for any assignment $x := e$ occurring in ρ we have that $IVar(e) \cap Aux \neq \emptyset$ implies that $x \in Aux$,
- the variables of the set Aux do not occur as tests in conditionals or loops of ρ,
- $IVar(Q) \cap Aux = \emptyset$.

Let ρ' be the program that can be obtained from ρ by deleting all assignments to variables belonging to the set Aux. Then we have the following rule:

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

The rule for auxiliary variables can be explained as follows: To be able to prove some properties of a program ρ' it may be necessary to add a number of extra variables and statements to do some bookkeeping. If these additions satisfy the above conditions, we are sure that they do not influence the flow of control of the program, and therefore they can be deleted after the proof of the enlarged program ρ is completed.

Definition 4.17
Next we have a substitution rule to remove instance variables from preconditions:

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

provided the instance variable x does not occur in ρ or Q. In practice, this rule is mainly used for auxiliary variables.

Definition 4.18
The following substitution rule removes logical variables from the precondition:

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

provided the logical variable z does not occur in Q.

Definition 4.19
Finally, the following rule makes explicit the knowledge that in the initial state all instance variables are nil:

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

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

5 An example proof

As an illustration of the proof system we shall now formally prove a property of the program listed at the end of section 2. We want to prove that exactly the primes up to n are generated. This amounts to proving the postcondition

$$\forall i(\mathit{Prime}(i) \wedge i \leq n \leftrightarrow \exists p\, p.\mathsf{m} \doteq i) \tag{5.1}$$

Here i and p are logical variables ranging over integers and objects of class P, respectively. The predicate *Prime* holds for an integer if and only if it is a (positive) prime number.

To establish this postcondition we first introduce an auxiliary variable v (for 'valid') of type Bool in class P, which indicates that the value stored in the variable b is valid. The variable v is false precisely from the moment that the object sends the value of its b variable to its neighbour until the moment that it receives a new value for the variable b. Now we take a global invariant I which is the conjunction of the following assertions:

I_1: $\forall p(p.\mathsf{m} \neq \mathsf{nil} \rightarrow \mathit{Prime}(p.\mathsf{m}))$

I_2: $\forall p\, \forall p'(p.\mathsf{m} \neq \mathsf{nil} \rightarrow (p'.\mathsf{m} \doteq \mathit{nextpr}(p.\mathsf{m}) \rightarrow p.\mathsf{l} \doteq p'))$

I_3: $\forall p\, \forall p'(p.\mathsf{b} < p'.\mathsf{b} \rightarrow p.\mathsf{m} > p'.\mathsf{m})$

I_4: $\forall p\, \forall p'(p.\mathsf{m} \doteq p'.\mathsf{m} \rightarrow p \doteq p')$

I_5: $\forall g\, \forall i(\mathit{Prime}(i) \wedge i < g.\mathsf{c} \rightarrow \exists p(p.\mathsf{m} \doteq i \vee p.\mathsf{b} \doteq i))$

I_6: $\forall g\, \forall p((p.\mathsf{m} \neq \mathsf{nil} \rightarrow 2 \leq p.\mathsf{m} < g.\mathsf{c}) \wedge (p.\mathsf{b} \neq \mathsf{nil} \rightarrow 2 \leq p.\mathsf{b} < g.\mathsf{c})$

I_7: $\forall p\, \forall p'(p.\mathsf{m} \neq \mathsf{nil} \wedge p'.\mathsf{b} \neq \mathsf{nil} \wedge p'.\mathsf{v} \rightarrow p.\mathsf{m} < p'.\mathsf{b})$

I_8: $\forall p(p.\mathsf{m} \neq \mathsf{nil} \wedge p.\mathsf{b} \neq \mathsf{nil} \rightarrow p.\mathsf{m} < p.\mathsf{b})$

In the above definition g is a logical variable of type G, so it will always denote the root object. The function *nextpr* applied to an integer gives the next prime number.

The assertion I_2 expresses that successive primes are stored in successive P objects in the chain. Assertion I_3 states that the prime candidates flow through the chain in their natural order. Assertion I_4 states that each prime number is uniquely represented. Next, assertion I_5 states that all the prime numbers among the candidates sent are represented or being processed. On the other hand assertion I_6 essentially expresses that all the numbers which occur in the chain have been sent by the generator. Finally, I_7 and I_8 says that a candidate is always greater than any prime number already found: I_7 states this globally, but only if the variable b containing the candidate is marked as valid by the flag v, and I_8 expresses it locally.

The easiest way to represent the outcome of the local proof system is by means of a *proof outline*. This consists of an expanded version of the program: the statements

concerning auxiliary variables are added, the bracketed sections are indicated by angle brackets, and the assertions that play a role in the local proof are inserted at the proper places, surrounded by braces. The proof outline for our example program is given in figure 1.

Now we have to check the assumptions of the cooperation test. Here we shall deal with one pair of I/O statements and one new-statement. We first treat the following case:

$$\{I \wedge \psi_0 \downarrow r \wedge \psi_3 \downarrow s\}(r, ?\mathsf{m}) \parallel (s, \mathsf{I}!\mathsf{b};\ \mathsf{v} := \mathsf{false})\{I \wedge \psi_1 \downarrow r \wedge (\neg \mathsf{v}) \downarrow s\} \qquad (5.2)$$

Here s and r are logical variables of type P denoting the sender and the receiver.

We prove (5.2) using the rule (PAR2). The following nontrivial premisses are needed:

$$\{I \wedge \psi_1 \downarrow r\}(s, \mathsf{v} := \mathsf{false})\{I \wedge \psi_1 \downarrow r \wedge (\neg \mathsf{v}) \downarrow s\} \qquad (5.3)$$

$$\{I \wedge \psi_0 \downarrow r \wedge \psi_3 \downarrow s\}(r, ?\mathsf{m}) \parallel (s, \mathsf{I}!\mathsf{b})\{I \wedge \psi_1 \downarrow r\} \qquad (5.4)$$

In order to prove (5.3), by the rule (IASS) and the consequence rule, we must show that the postcondition, after applying the substitution $[\mathsf{false}/s.\mathsf{v}]$, is implied by the precondition. Now it is clear that I_1–I_6, I_8, and $\psi_1 \downarrow r$ are not changed by this substitution, so they are subsumed by the precondition. For the other parts we first calculate as follows:

$$\begin{aligned} ((\neg \mathsf{v}) \downarrow s)[\mathsf{false}/s.\mathsf{v}] &= (\neg s.\mathsf{v})[\mathsf{false}/s.\mathsf{v}] \\ &= \neg \mathsf{if}\ s \doteq s\ \mathsf{then\ false\ else}\ s.\mathsf{v}\ \mathsf{fi} \\ &\equiv \mathsf{true} \end{aligned}$$

Finally, we observe that $I_7[\mathsf{false}/s.\mathsf{v}]$ is the formula

$$\forall p \forall p'(p.\mathsf{m} \not\doteq \mathsf{nil} \wedge p'.\mathsf{b} \not\doteq \mathsf{nil} \wedge \mathsf{if}\ p' \doteq s\ \mathsf{then\ false\ else}\ p'.\mathsf{v}\ \mathsf{fi} \rightarrow p.\mathsf{m} < p'.\mathsf{b})$$

and this is clearly implied by I_7.

By the rule (SR2) and the axiom (COMM) the proof of the second premiss (5.4) amounts to establishing the truth of the formula

$$I \wedge \psi_0 \downarrow r \wedge \psi_3 \downarrow s \wedge s.\mathsf{I} \doteq r \rightarrow (I \wedge \psi_1 \downarrow r)[s.\mathsf{b}/r.\mathsf{m}].$$

Here we only show the following part of this:

$$I \wedge \psi_0 \downarrow r \wedge \psi_3 \downarrow s \wedge s.\mathsf{I} \doteq r \rightarrow I_1[s.\mathsf{b}/r.\mathsf{m}].$$

Now $I_1[s.\mathsf{b}/r.\mathsf{m}]$ is the formula

$$\forall p(\mathsf{if}\ p \doteq r\ \mathsf{then}\ s.\mathsf{b}\ \mathsf{else}\ p.\mathsf{m}\ \mathsf{fi} \not\doteq \mathsf{nil} \rightarrow \mathit{Prime}(\mathsf{if}\ p \doteq r\ \mathsf{then}\ s.\mathsf{b}\ \mathsf{else}\ p.\mathsf{m}\ \mathsf{fi})),$$

$\langle$P $\leftarrow$ $\{\psi_0 : \mathsf{l} \doteq \mathsf{nil} \wedge \mathsf{m} \doteq \mathsf{nil} \wedge b \doteq \mathsf{nil}\}$
$\langle ?\mathsf{m} \rangle$; $\{\psi_1 : \mathsf{l} \doteq \mathsf{nil} \wedge b \doteq \mathsf{nil}\}$
if $\mathsf{m} \not\doteq \mathsf{nil}$
then $\{\mathsf{l} \doteq \mathsf{nil}\}$
$\langle \mathsf{l} := \mathsf{new} \rangle$; $\{\mathsf{true}\}$
$\langle ?\mathsf{b};\ \mathsf{v} := \mathsf{true} \rangle$;
$\{\psi_2 : \mathsf{v} \wedge (\mathsf{b} \not\doteq \mathsf{nil} \rightarrow \forall i(2 \leq i < \mathsf{m} \rightarrow i \not| \mathsf{b}))\}$;
while $\mathsf{b} \not\doteq \mathsf{nil}$
do $\{\mathsf{v} \wedge \forall i(2 \leq i < \mathsf{m} \rightarrow i \not| \mathsf{b})\}$
if $\mathsf{m} \not| \mathsf{b}$
then $\{\psi_3 : \mathsf{v} \wedge \forall i(2 \leq i \leq \mathsf{m} \rightarrow i \not| \mathsf{b})\}$
$\langle \mathsf{l} ! \mathsf{b};\ \mathsf{v} := \mathsf{false} \rangle$
fi; $\{\neg \mathsf{v}\}$
$\langle ?\mathsf{b};\ \mathsf{v} := \mathsf{true} \rangle$ $\{\psi_2\}$
od; $\{\mathsf{b} \doteq \mathsf{nil}\}$
$\langle \mathsf{l} ! \mathsf{b} \rangle$ $\{\mathsf{b} \doteq \mathsf{nil}\}$
fi $\{\mathsf{b} \doteq \mathsf{nil}\}$
: $\{\mathsf{true}\}$
$\langle \mathsf{f} := \mathsf{new};\ \mathsf{c} := 2 \rangle$; $\{2 \leq \mathsf{c} \leq \max(2, n+1)\}$
while $\mathsf{c} \leq n$
do $\{2 \leq \mathsf{c} \leq n\}$
$\langle \mathsf{f} ! \mathsf{c};\ \mathsf{c} := \mathsf{c} + 1 \rangle$
od; $\{\mathsf{c} \doteq \max(2, n+1)\}$
$\langle \mathsf{f} ! \mathsf{nil} \rangle$
$\{\mathsf{c} \doteq \max(2, n+1)\}\rangle$

Figure 1: The proof outline for our example program

which is equivalent to

$$\forall p((p \not\doteq r \rightarrow p.\mathsf{m} \not\doteq \mathsf{nil} \rightarrow \mathit{Prime}(p.\mathsf{m})) \wedge (p \doteq r \rightarrow (s.\mathsf{b} \not\doteq \mathsf{nil} \rightarrow \mathit{Prime}(s.\mathsf{b}))).$$

For $p \not\doteq r$ we have that $p.\mathsf{m} \not\doteq \mathsf{nil} \rightarrow \mathit{Prime}(p.\mathsf{m})$ is implied by I_1. For $p \doteq r$ we have to show that $I \wedge \psi_0 \downarrow r \wedge \psi_3 \downarrow s \wedge s.\mathsf{l} \doteq r$ implies $\mathit{Prime}(s.\mathsf{b})$. Here the main point is that there exist no primes between $s.\mathsf{m}$ and $s.\mathsf{b}$. For suppose that i is the least prime such that $s.\mathsf{m} < i < s.\mathsf{b}$. Now by I_5 and I_6 there exists a p' such that $p'.\mathsf{m} \doteq i$ or $p'.\mathsf{b} \doteq i$. If $p'.\mathsf{m} \doteq i$ then it follows by I_2 that $s.\mathsf{l} \doteq p'$. But, as $s.\mathsf{l} \doteq r$ we have $p' \doteq r$, so $p'.\mathsf{m} \doteq \mathsf{nil}$, which contradicts $p'.\mathsf{m} \doteq i$. Suppose now that $p'.\mathsf{b} \doteq i$. We then have that $p'.\mathsf{b} < s.\mathsf{b}$, so by I_3 it follows that $p'.\mathsf{m} > s.\mathsf{m}$. But since $p'.\mathsf{m}$ is a prime by I_1, and i is the least prime greater than $s.\mathsf{m}$, it follows that $p'.\mathsf{m} \geq i \doteq p'.\mathsf{b}$, which contradicts I_8. Therefore there can be no primes between $s.\mathsf{m}$ and $s.\mathsf{b}$ and then $\psi_3 \downarrow s$ implies that $s.\mathsf{b}$ is a prime number.

As another example of an assumption for the cooperation test, we consider the following new-statement:

$$\{I \wedge (\mathsf{l} \doteq \mathsf{nil}) \downarrow z\}(z, \mathsf{l} := \mathsf{new})\{I \wedge \psi_0 \downarrow z.\mathsf{l}\}.$$

Here z is a logical variable ranging over objects of class P. By the axiom (NEW) the proof of this correctness formula amounts to proving the validity of

$$I \wedge (\mathsf{l} \doteq \mathsf{nil}) \downarrow z \rightarrow (I \wedge \psi_0 \downarrow z.\mathsf{l})[z'/z.\mathsf{l}][\mathsf{new}/z'],$$

where z' is another new logical variable of type P. We only show that $I \wedge (\mathsf{l} \doteq \mathsf{nil}) \downarrow z$ implies $I_2[z'/z.\mathsf{l}][\mathsf{new}/z']$. Now $I_2[z'/z.\mathsf{l}]$ is the formula

$$\forall p \forall p'(p.\mathsf{m} \not\doteq \mathsf{nil} \rightarrow p'.\mathsf{m} \doteq \mathit{nextpr}(p.\mathsf{m}) \rightarrow \mathsf{if}\ p \doteq z\ \mathsf{then}\ z'\ \mathsf{else}\ p.\mathsf{l}\ \mathsf{fi} \doteq p').$$

This is equivalent to

$$\forall p \forall p'(p.\mathsf{m} \not\doteq \mathsf{nil} \rightarrow p'.\mathsf{m} \doteq \mathit{nextpr}(p.\mathsf{m}) \rightarrow (p \doteq z \rightarrow z' \doteq p') \wedge (p \not\doteq z \rightarrow p.\mathsf{l} \doteq p')).$$

To this formula we apply the substitution $[\mathsf{new}/z']$, which gives us

$$\begin{aligned}
&\forall p \forall p'(p.\mathsf{m} \not\doteq \mathsf{nil} \rightarrow p'.\mathsf{m} \doteq \mathit{nextpr}(p.\mathsf{m}) \rightarrow (p \doteq z \rightarrow \mathsf{false}) \wedge (p \not\doteq z \rightarrow p.\mathsf{l} \doteq p')) \\
&\quad \wedge \forall p(p.\mathsf{m} \not\doteq \mathsf{nil} \rightarrow \mathsf{nil} \doteq \mathit{nextpr}(p.\mathsf{m}) \rightarrow (p \doteq z \rightarrow \mathsf{true}) \wedge (p \not\doteq z \rightarrow \mathsf{false})) \\
&\quad \wedge \forall p'(\mathsf{nil} \not\doteq \mathsf{nil} \rightarrow p'.\mathsf{m} \doteq \mathit{nextpr}(\mathsf{nil}) \rightarrow (\mathsf{false} \rightarrow \mathsf{false}) \wedge (\mathsf{true} \rightarrow \mathsf{nil} \doteq p')) \\
&\quad\quad \wedge (\mathsf{nil} \not\doteq \mathsf{nil} \rightarrow \mathsf{nil} \doteq \mathit{nextpr}(\mathsf{nil}) \rightarrow (\mathsf{false} \rightarrow \mathsf{true}) \wedge (\mathsf{true} \rightarrow \mathsf{false}))
\end{aligned}$$

Note that the first conjunct corresponds to interpreting p and p' as ranging over the old objects. The second conjunct results from interpreting p' as the new object, the third from interpreting p as the new object, and the last from taking both p and p' as the new object. All conjuncts but the first are trivially true, and we can get the first conjunct from I_2 if we can show that $\forall p \forall p'(p.\mathsf{m} \not\doteq \mathsf{nil} \rightarrow p'.\mathsf{m} \doteq \mathit{nextpr}(p.\mathsf{m}) \rightarrow p \not\doteq z)$.

Let $p \neq \mathsf{nil}$, $p' \neq \mathsf{nil}$, $p.\mathsf{m} \neq \mathsf{nil}$, $p'.\mathsf{m} \doteq \mathit{nextpr}(p.\mathsf{m})$, and $p \doteq z$. By I_2 we derive that $z.\mathsf{l} \doteq p'$. But this contradicts the precondition $z.\mathsf{l} \doteq \mathsf{nil}$.

The other parts of the cooperation test can be dealt with in the same way as the above ones. After that the program rule (PR) can be applied, with the following result:

$$\{\mathsf{true}\}\rho'\{I \wedge g.\mathsf{c} \doteq \max(2, n+1) \wedge \forall p\, p.\mathsf{b} \doteq \mathsf{nil}\}$$

Here ρ' is the program with the auxiliary variable v, but by applying the auxiliary variable rule (AUX) the same result can also be obtained for the original program ρ. It is easy to see that the above postcondition implies the desired assertion 5.1, so the latter can be obtained by the consequence rule (PCR).

6 Semantics

In this section we define in a formal way the semantics of the programming language and the assertion languages. First, in section 6.1, we deal with the assertion languages on their own. Then, in section 6.2, we give a formal semantics to the programming language, making use of *transition systems*. Finally, section 6.3 formally defines the notion of truth of a correctness formula.

6.1 Semantics of the assertion languages

For every type $a \in C^!$, we shall let $\mathbf{O}^a$ denote the set of objects of type a, with typical element α^a. To be precise, we define $\mathbf{O}^{\mathsf{Int}} = \mathbf{Z}$ and $\mathbf{O}^{\mathsf{Bool}} = \mathbf{B}$, 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 6.1
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 6.2
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 6.3
The set $LState^c$ of *local states* of class c, with typical element θ^c, is defined by

$$LState^c = \mathbf{O}^c \times \prod_d (IVar^c_d \to \mathbf{O}^d_\perp).$$

A local state θ^c describes in detail the situation of a single object of class c at a certain moment during program execution. The first component determines the identity of the object and the values of the instance variables are given by the second component.

It will turn out to be convenient to define the function $\bigtriangledown^c \in \prod_d IVar_d^c \to \mathbf{O}_\perp^d$ such that $\bigtriangledown^c(x) = \perp$, for every $x \in \bigcup_d IVar_d^c$. Note that this function $\bigtriangledown$ gives the values of the variables of a newly created object: these are all initialized to nil.

Definition 6.4
The set *GState* of *global states*, with typical element σ, is defined as follows:

$$GState = (\prod_d P^d) \times \prod_c \Big(\mathbf{O}^c \to \prod_d (IVar_d^c \to \mathbf{O}_\perp^d)\Big)$$

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

We introduce the following abbreviations: $\sigma_{(1)(d)}$ will be abbreviated to $\sigma^{(d)}$, and $\sigma^{(d)} \cup \{\perp\}$ to $\sigma_\perp^{(d)}$. Whenever it is clear from the context that $\alpha \in \mathbf{O}^c$, we abbreviate $\sigma_{(2)(c)}(\alpha)$ by $\sigma(\alpha)$. Furthermore, for any variable $x \in IVar_d^c$, we abbreviate $\sigma_{(2)(c,d)}(\alpha)(x)$, the value of the variable x of the object α, by $\sigma(\alpha)(x)$.

Definition 6.5
We now define the set *LEnv* of *logical environments*, with typical element ω, by

$$LEnv = \prod_a (LogVar_a \to \mathbf{O}_\perp^a).$$

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

Definition 6.6
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 $\mathcal{E}^c_d \in Exp^c_d \to LState^c \to \mathbf{O}^d_\perp$ assigns a value $\mathcal{E}[\![e]\!](\theta)$ to the expression e^c_d in the local state θ^c.

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

3. The function $\mathcal{G}_a \in GExp_a \to LEnv \to GState \to \mathbf{O}^a_\perp$ 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:

$$\mathcal{A}[\![l_{\mathsf{Bool}}]\!](\omega)(\theta) = \begin{cases} \mathsf{true} & \text{if } \mathcal{L}[\![l]\!](\omega)(\theta) = \mathsf{true} \\ \mathsf{false} & \text{if } \mathcal{L}[\![l]\!](\omega)(\theta) = \mathsf{false} \text{ or } \mathcal{L}[\![l]\!](\omega)(\theta) = \perp \end{cases}$$

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

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:

$$\mathcal{A}[\![g_{\mathsf{Bool}}]\!](\omega)(\sigma) = \begin{cases} \mathsf{true} & \text{if } \mathcal{L}[\![g]\!](\omega)(\sigma) = \mathsf{true} \\ \mathsf{false} & \text{if } \mathcal{L}[\![g]\!](\omega)(\sigma) = \mathsf{false} \text{ or } \mathcal{L}[\![g]\!](\omega)(\sigma) = \perp \end{cases}$$

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

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 $\perp$).

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

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 6.7
We define the global state σ to be *consistent*, for which we use the notation $OK(\sigma)$ iff

1. $\forall c \in C\ \forall \alpha \in \sigma^{(c)}\ \forall d \in C\ \forall x \in IVar_d^c\ \sigma(\alpha)(x) \in \sigma_\perp^{(d)}$
2. $\forall c \in C\ \forall \beta \in \mathbf{O}^c \setminus \sigma^{(c)}\ \forall d \in C^+\ \forall x \in IVar_d^c\ \sigma(\beta)(x) = \perp$

In other words, the value in σ of a variable of an existing object is either $\perp$ or an existing object itself, and the variables of non-existing objects are uninitialized.

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,

$$\forall d \in C\ \forall z \in LogVar_d\ \omega(z) \in \sigma_\perp^{(d)}$$

and

$$\forall d \in C\ \forall z \in LogVar_{d^*}\ \forall a \in \mathbf{N}\ \omega(z)(n) \in \sigma_\perp^{(d)}.$$

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

6.2 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 $Rec = \{< \alpha, \beta >, < \alpha, \beta, \gamma >:\ \alpha, \beta \in \bigcup_c \mathbf{O}_\perp^c,\ \gamma \in \bigcup_d \mathbf{O}_\perp^d\} \cup \{\epsilon\}$. A pair $< \alpha, \beta >$ is called an *activation record.* It records the information that the object α created β. A triple $< \alpha, \beta, \gamma >$ is called a *communication record.* It records the information that the object γ is sent by α to β. ϵ will denote a transition which is due to a local computation. We define for every $r \in Rec$ a transition relation $\rightarrow^r \subseteq LConf \times LConf$. To facilitate the semantics we introduce the auxiliary statement E, the empty statement, to denote termination.

Definition 6.8
We define

- $(x_d := e_d, \theta) \rightarrow^\epsilon (E, \theta')$,
  let $\theta = \langle \alpha, s \rangle$, then $\theta' = \langle \alpha, s\{\mathcal{E}[\![e_d]\!]\ (\theta)/x\} \rangle$.
- $(x_d := \mathsf{new}, \theta) \rightarrow^{<\alpha,\beta>} (E, \theta')$,
  let $\theta = \langle \alpha, s \rangle$, then $\theta' = \langle \alpha, s\{\beta/x_d\} \rangle$ and $\beta \in \mathbf{O}^d$.

- $(x_c!e_d, \theta) \rightarrow^{<\alpha,\beta,\gamma>} (E, \theta)$,
  let $\theta = \langle \alpha, s \rangle$, then $\beta = s(x) \neq \perp$ and $\gamma = \mathcal{E}[\![e_d]\!]\ (\theta)$.
- $(?y_d, \theta) \rightarrow^{<\perp,\beta,\gamma>} (E, \theta')$,
  let $\theta = \langle \beta, s \rangle$, then $\theta' = \langle \beta, s\{\gamma/y_d\} \rangle$ and $\gamma \in \mathbf{O}^d_\perp$.
- $(x_c?y_d, \theta) \rightarrow^{<\alpha,\beta,\gamma>} (E, \theta')$,
  let $\theta = \langle \beta, s \rangle$, then $\theta' = \langle \beta, s\{\gamma/y_d\} \rangle$, $\alpha = s(x_c) \neq \perp$, and $\gamma \in \mathbf{O}^d_\perp$.
- $(S, \theta) \rightarrow^{\epsilon} (S, \theta)$.
- $(E; S, \theta) \rightarrow^{\epsilon} (S, \theta)$.
- $$\frac{(S_1, \theta) \rightarrow^{r} (S_2, \theta')}{(S_1; S, \theta) \rightarrow^{r} (S_2; S, \theta')}$$
- (if e_{Bool} then S_1 else S_2 fi, θ) $\rightarrow^{\epsilon}$ (S_1, θ),
  if $\mathcal{E}[\![e_{\mathsf{Bool}}]\!]\ (\theta) = \mathsf{true}$.
- (if e_{Bool} then S_1 else S_2 fi, θ) $\rightarrow^{\epsilon}$ (S_2, θ),
  if $\mathcal{E}[\![e_{\mathsf{Bool}}]\!]\ (\theta) = \mathsf{false}$.
- (while e_{Bool} do S od, θ) $\rightarrow^{\epsilon}$ (S; while e_{Bool} do S od, θ),
  if $\mathcal{E}[\![e_{\mathsf{Bool}}]\!]\ (\theta) = \mathsf{true}$.
- (while e_{Bool} do S od, θ) $\rightarrow^{\epsilon}$ (E, θ),
  if $\mathcal{E}[\![e_{\mathsf{Bool}}]\!]\ (\theta) = \mathsf{false}$.

Note that locally the value which is received when executing an input statement is chosen arbitrarily, the same holds for the identity of the object created by the execution of a new-statement. We define $\bigcup_h \rightarrow^h = TC(\bigcup_{r \in Rec} \rightarrow^r)$. Here the operation TC denotes the transitive closure which composes additionally the communication records and activation records into a *history* h, a sequence of communication records and activation records.

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.

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 $\rightarrow^r \subseteq IConf \times IConf$ as follows (note that we use the same notation as for the local transition relation, however this will cause no harm):

Definition 6.9
We define

- $$\frac{(S_j, \theta_1) \rightarrow^{\epsilon} (S_j', \theta_2)}{(\sigma, (\alpha_i, S_i)_i) \rightarrow^{\epsilon} (\sigma', (\alpha_i, S_i')_i)}$$
  where $\theta_1 = \langle \alpha_j, \sigma(\alpha_j) \rangle$,
  $S_i' = S_i \quad i \neq j$
  $= S_i' \quad$ otherwise,
  and $\sigma'_{(1)} = \sigma_{(1)}$, $\sigma'_{(2)} = \sigma_{(2)}\{s/\alpha_j\}$, with $\theta_2 = \langle \alpha_j, s \rangle$.

- $$\frac{(S_j, \theta_1) \rightarrow^{<\alpha_j, \beta>} (S_j', \theta_2)}{(\sigma, (\alpha_i, S_i)_i) \rightarrow^{<\alpha_j, \beta>} (\sigma', (\alpha_i, S_i')_i)}$$
  where $\theta_1 = \langle \alpha_j, \sigma(\alpha_j) \rangle$,
  $S_i' = S_i \quad i \neq j$
  $= S_i' \quad$ otherwise,
  and $\beta \in \mathbf{O}^d \backslash \sigma^{(d)}$, $\sigma'_{(1)} = \sigma_{(1)}\{\sigma^{(d)} \cup \{\beta\}/d\}$, and $\sigma'_{(2)} = \sigma_{(2)}\{s/\alpha_j\}\{\bigtriangledown/\beta\}$, with $\theta_2 = \langle \alpha_j, s \rangle$.

- $$\frac{(S_j, \theta_1) \rightarrow^{r} (S_j', \theta_2), (S_k, \theta_1') \rightarrow^{r'} (S_k', \theta_2')}{(\sigma, (\alpha_i, S_i)_i) \rightarrow^{r} (\sigma', (\alpha_i, S_i')_i)}$$
  where $\theta_1 = \langle \alpha_j, \sigma(\alpha_j) \rangle$, $\theta_1' = \langle \alpha_k, \sigma(\alpha_k) \rangle$, $j \neq k$, and $r =< \alpha_j, \alpha_k, \gamma >$, $r' =< \alpha_j', \alpha_k, \gamma >$, with $\gamma \in \bigcup_d \mathbf{O}_{\perp}^d$ and $\alpha_j' = \alpha_j, \perp$, furthermore we have
  $S_i' = S_i \quad i \neq j, k$
  $= S_i' \quad i = j, k,$
  and $\sigma'_{(1)} = \sigma_{(1)}$, $\sigma'_{(2)} = \sigma_{(2)}\{s_1/\alpha_j\}\{s_2/\alpha_k\}$, with $\theta_2 = \langle \alpha_j, s_1 \rangle$ and $\theta_2' = \langle \alpha_k, s_2 \rangle$.

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 rule selects an object which is about to create a new object. The variables of this new object are initialized to nil. The local state of the creator is modified according to the local transition system. Note that knowing the set of existing objects we can now indeed select a new object. The third rule finally selects two objects which are ready to communicate with each other. Note that $r' =< \perp, \alpha_k, \gamma >$ models the possibility that α_k executes an input statement of the form $?y$.

We define $\bigcup_h \rightarrow^h = TC(\bigcup_{r \in Rec} \rightarrow^r)$, again using the same notation as for the transitive closure of the local transition relation, from the context however it should be clear which one is meant.

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

$\rightarrow^r \subseteq GConf \times GConf$. 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 6.10
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 $\bigcup_h \rightarrow^h = TC(\bigcup_{r \in Rec} \rightarrow^r)$. We proceed with the following definition which characterizes the set of *initial* and *final* global configurations of a given program ρ:

Definition 6.11
Let $\rho = \langle U : S_n^{c_n} \rangle$, with $U = c_1 \leftarrow S_1^{c_1}, \ldots, c_{n-1} \leftarrow S_{n-1}^{c_{n-1}}$. Furthermore let $X \in \prod_c \mathbf{O}^c \rightarrow Stat^c$. We define

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

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

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 $Init_\rho(\sigma)$. Finally, 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 θ. Note that the value of a variable $x_c^{c_n}$ 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), $(z_{c_i}, S_1^{c_i}) \parallel (z_{c_j}, S_2^{c_j})$ and ρ.

Definition 6.12
We define

$$\mathcal{S}[\![S^c]\!](\theta_1^c) = \{\theta_2^c : \text{for some } h\ (S^c, \theta_1^c) \rightarrow^h (E, \theta_2^c)\}$$

Definition 6.13
We define $\mathcal{I}[\![(z_c, S^c)]\!](\omega)(\sigma_1) = \emptyset$ if not $OK(\omega, \sigma_1)$, and $\mathcal{I}[\![(z_{c_i}, S_1^{c_i}) \parallel (z_{c_j}, S_2^{c_j})]\!](\omega)(\sigma_1) = \emptyset$ if not $OK(\omega, \sigma_1)$ or $\omega(z_{c_i}) = \omega(z_{c_j})$. So assume from now on that $OK(\omega, \sigma_1)$, furthermore that $\omega(z_c) = \alpha$, $\omega(z_{c_i}) = \alpha_i$, and $\omega(z'_{c_j}) = \alpha'_j$.

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

Assuming furthermore that $\alpha_i \neq \alpha'_j$:

$$\mathcal{I}[\![(z_{c_i}, S_1^{c_i}) \parallel (z'_{c_j}, S_2^{c_j})]\!](\omega)(\sigma_1) =$$
$$\{\sigma_2 : \text{for some } h\ (\sigma_1, (\alpha_i, S_1^{c_i}), (\alpha'_j, S_2^{c_j})) \rightarrow^h (\sigma_2, (\alpha_i, E), (\alpha'_j, E))\}$$

Definition 6.14
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 $Init_\theta((X_1, \sigma_1))$ and $Final_\theta((X_2, \sigma_2))$.

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

6.3 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 6.15
We define

$$\models \{p^c\}S^c\{q^c\} \text{ iff } \forall \omega, \theta_1, \theta_2 \in \mathcal{S}[\![S^c]\!](\theta_1) : \ \theta_1, \omega \models p^c \ \Rightarrow \ \theta_2, \omega \models q^c.$$

Next we define the truth of intermediate correctness formulas.

Definition 6.16
We define

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

And

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

Finally, we define the truth of global correctness formulas.

Definition 6.17
We define

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

7 Soundness

In this section we prove the soundness of the proof system as presented in the previous section. The soundness of the local proof system is proved by a straightforward induction on the length of the derivation (see, for example, [Ap1]). In the following subsection we discuss the soundness of the intermediate proof system.

7.1 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 7.1
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 $\sigma'_{(1)} = \sigma_{(1)}$ and $\sigma'_{(2)} = \sigma_{(2)}\{\mathcal{G}[\![g'_d]\!](\omega)(\sigma)/\omega(z_c), x_d\}$.

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:

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

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:

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

□

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

Lemma 7.2
We have

$$\models \left\{P[e \downarrow z/z.x]\right\}(z, x := e)\left\{P\right\},$$

where we assume $x := e \in Stat^c$ and $z \in LogVar_c$.

Proof
Let σ, ω, with $OK(\omega, \sigma)$, such that $\sigma, \omega \models P[e \downarrow z/z.x]$ and $\sigma' \in \mathcal{I}[\![(z, x := e)]\!](\omega)(\sigma)$. It follows that $\sigma'_{(1)} = \sigma_{(1)}$ and $\sigma'_{(2)} = \sigma_{(2)}\{\mathcal{G}[\![e \downarrow z]\!](\omega)(\sigma)/\omega(z), x\}$ (note that $\mathcal{G}[\![e \downarrow z]\!](\omega)(\sigma) = \mathcal{E}[\![e]\!](\langle\alpha, \sigma(\alpha)\rangle)$, 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 [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 7.3
For an arbitrary $\omega, \omega', \sigma, \sigma', \beta \in \mathbf{O}^c \backslash \sigma^{(c)}$ such that $OK(\omega, \sigma)$ and

$$\begin{aligned}
\sigma'_{(1)} &= \sigma_{(1)}\{\sigma^{(c)} \cup \{\beta\}/c\} \\
\sigma'_{(2)} &= \sigma_{(2)}\{\bigtriangledown/\beta\} \\
\omega' &= \omega\{\beta/z_c\},
\end{aligned}$$

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 7.4
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 $\sigma'_{(1)} = \sigma_{(1)}\{\sigma^{(c)} \cup \{\beta\}/c\}$, $\sigma'_{(2)} = \sigma_{(2)}\{\bigtriangledown/\beta\}$, 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 7.5
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 $z_{\mathsf{Bool}^\bullet}$ does not occur in it:

$$\mathcal{A}[\![P[z_{\mathsf{Bool}^\bullet}, z_c/z_{c^\bullet}]]\!](\omega)(\sigma) = \mathcal{A}[\![P]\!](\omega')(\sigma)$$

Proof
Induction on the structure of g and P. □

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

Lemma 7.6
We have

$$\models \left\{P[z'/z.x][\mathsf{new}/z']\right\}(z, x := \mathsf{new})\left\{P\right\},$$

where $x := \mathsf{new} \in \mathit{Stat}^c$, $z \in \mathit{LogVar}_c$, 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 7.3 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 $\sigma''_{(1)} = \sigma_{(1)}\{\sigma^{(d)} \cup \{\beta\}/d\}$, $\sigma''_{(2)} = \sigma_{(2)}\{\nabla/\beta\}$. Now by lemma 7.1 it follows that $\sigma', \omega' \models P$. Finally, as z' does not occur in P we have $\sigma', \omega \models P$. □

7.2 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. The problem with proving the soundness of the rule (PR) is how to interpret the premise $A \vdash \{p\}S\{q\}$. We solve this problem by showing that a proof $A \vdash \{p\}S\{q\}$ essentially boils down to a finite conjunction of local assertions. We start with the following two definitions.

Definition 7.7
Given a bracketed program ρ, R a substatement of ρ, we define R to be *normal* iff every bracketed section of U occurs inside or outside of R.

Next we define $\mathit{After}(R, S)$, where R is a substatement of S, to be the statement to be executed when the execution of R has just terminated. On the other hand we will define $\mathit{Before}(R, S)$ to be the statement to be executed when the execution of R is about to start.

Definition 7.8
Let R be a substatement of the statement S. We define $After(R, S)$ as follows:

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

  then $After(R, S) = After(R, S_1); S_2$ if R occurs in S_1

  $After(R, S) = After(R, S_2)$ otherwise
- If $S =$ if e then S_1 else S_2 fi

  then $After(R, S) = After(R, S_1)$ if R occurs in S_1

  $After(R, S) = After(R, S_2)$ otherwise
- If $S =$ while e do S_1 od then $After(R, S) = After(R, S_1); S$

Next we define $Before(R, S)$ as follows: $Before(R, S) = R; S'$, where $After(R, S) = E; S'$.

Note that for the above definition to be formally correct we have to assume some mechanism which enables one to distinguish between different occurrences of an arbitrary statement. We will simply assume such a mechanism to exist. So in the sequel when referring to a statement we in fact will sometimes mean a particular occurrence of that statement.

We have the following lemma which can be seen as a particular formulation of the soundness of the local proof system.

Lemma 7.9
Let $S \in Stat^c$ be a statement such that every I/O statement and new statement occurring in it is contained in a bracketed section. Furthermore let A be a set of assumptions about the bracketed sections occurring in S. Then: $A \vdash \{p\}S\{q\}$ iff there exist for every normal substatement R of S local assertions $Pre(R)$ and $Post(R)$ such that:

- $\{Pre(R)\}R\{Post(R)\} \in A$, for R a bracketed section
- $p \rightarrow Pre(S)$, $Post(S) \rightarrow q$
- $Pre(R) \rightarrow Post(R)[e/x]$, $R = x := e$
- $Pre(R) \rightarrow Pre(R_1)$, $Post(R_1) \rightarrow Pre(R_2)$, and $Post(R_2) \rightarrow Post(R)$, $R = R_1; R_2$

- $Pre(R) \wedge e \rightarrow Pre(R_1)$, $Pre(R) \wedge \neg e \rightarrow Pre(R_2)$, $Post(R_1) \rightarrow Post(R)$, and $Post(R_2) \rightarrow Post(R)$, $R =$ **if** e **then** R_1 **else** R_2 **fi**
- $Pre(R) \wedge e \rightarrow Pre(R_1)$, $Post(R_1) \rightarrow Pre(R)$, and $Pre(R) \wedge \neg e \rightarrow Post(R)$, $R =$ **while** e **do** R_1 **od**

Proof
Straightforward induction on the structure of S. □

Given a derivation $A \vdash \{p\}S\{q\}$ we define $VC_A(\{p\}S\{q\})$ to be the set of local assertions corresponding to the last five clauses of the above mentioned lemma.

Given a bracketed program $\rho = \langle U : S_n^{c_n} \rangle$, with $U = c_1 \leftarrow S_1^{c_1}, \ldots, c_{n-1} \leftarrow S_{n-1}^{c_{n-1}}$, a global assertion I, and the derivations $A_k \vdash \{p_k^{c_k}\}S_k^{c_k}\{q_k^{c_k}\}$, the set of formulas and assertions which have to be verified in the cooperation test will be denoted by $CP(A_1, \ldots, A_n, I, z_{c_n})$.

Now we can phrase the soundness of the rule (PR) as follows: If all the local assertions of $VC_{A_k}(\{p_k^{c_k}\}S_k^{c_k}\{q_k^{c_k}\})$, $1 \leq k \leq n$, all the formulas and assertions of $CP(A_1, \ldots, A_n, I, z_{c_n})$ are true, and finally the global assertion I satisfies the syntactic restriction mentioned in the cooperation test then the conclusion of the rule (PR) is valid.

It is easy to see that this formulation of the soundness of the rule (PR) is implied by the following theorem.

Theorem 7.10
Let $\rho = \langle U : S_n^{c_n} \rangle$, with $U = c_1 \leftarrow S_1^{c_1}, \ldots, c_{n-1} \leftarrow S_{n-1}^{c_{n-1}}$, be bracketed.

Suppose that there exist for every normal statement R of ρ local assertions $Pre(R)$ and $Post(R)$ such that for $A_k = \{\{Pre(R)\}R\{Post(R)\} : R$ a bracketed section occurring in $S^{c_k}\}$ all the local assertions of $VC_{A_k}(\{p_k^{c_k}\}S_k^{c_k}\{q_k^{c_k}\})$ are true. Assume furthermore that we are given that all the formulas and assertions of $CP(A_1, \ldots, A_n, I, z_{c_n})$ are true, and finally that I satisfies the syntactic restriction mentioned in the first clause of the cooperation test.

Let $\tau = (X_i, \sigma_i)_{i=0,\ldots,m}$ be a sequence of global configurations such that for every $0 \leq j < m$ we have $(X_j, \sigma_j) \rightarrow^{\tau_j} (X_{j+1}, \sigma_{j+1})$, where $INIT_\rho((X_0, \sigma_0))$, and for every $c \in \{c_1, \ldots, c_n\}$, $\alpha \in \sigma_m^{(c)}$ we have $X_m(\alpha) = E$, or $X_m(\alpha) = Before(R, S^c), After(R, S^c)$, R a bracketed section. Additionally we have to impose the following restriction on the computation τ: For an arbitrary c, $\alpha \in \sigma_m^{(c)}$ we have if $X_i(\alpha) = Before(R, S^c)$, for some $0 \leq i < m$, then for some $i < j < m$ we have $X_j(\alpha) = After(R, S^c)$. This additional condition is necessary to ensure that in the configuration (X_m, σ_m) every existing

object is about to enter a bracketed section or has just finished executing one or has finished executing its local process. We have to exclude situations like: $X_m(\alpha) = \mathit{Before}(R, S^c)$, with $R =<$ while e do R_1 od;R' $>$, and $X_{m-1}(\alpha) = \mathit{After}(R_1, S^c)$. Note that in the configuration (X_m, σ_m) the object α is in fact executing *inside* a bracketed section.

Finally, suppose that: $\sigma_0, \omega \models p_n^{c_n} \downarrow z_{c_n}$. Then:

1. $\sigma_m, \omega \models I$
2. For $c_i \in \{c_1, \ldots, c_n\}$, $\alpha \in \sigma_m^{(c_i)}$:
   - If $X_m(\alpha) = \mathit{Before}(R, S_i^{c_i})$ then $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models \mathit{Pre}(R)$.
   - If $X_m(\alpha) = \mathit{After}(R, S_i^{c_i})$, and for some $1 \leq k < m$, $X_k(\alpha) = \mathit{Before}(R, S_i^{c_i})$ such that for every $k < l < m$, $X_l(\alpha) = \mathit{Before}(R', S_i^{c_i})$ implies that R' is a proper normal substatement of R, then $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models \mathit{Post}(R)$.
   - If $X_m(\alpha) = E$ then $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models q_i^{c_i}$.

   The additional condition in this second clause above can be shown to be necessary as follows: Suppose the statement if t then R_1 else R_2 fi occurs in S^c. We then have that $\mathit{After}(R_1, S) = \mathit{After}(R_2, S)$. So we need some additional information about which of the statements R_1, R_2 has actually been executed.

Proof

The proof proceeds by induction on $|h|$, the length of the history of the computation τ.

$|h| = 0$:
Because $\sigma_0, \omega \models p_n^{c_n} \downarrow z_{c_n}$, $\mathit{INIT}_\rho((X_0, \sigma_0))$, and the implication of the last clause of the cooperation test holds, we have that $\sigma_0, \omega \models I$. Now σ_m agrees with σ_0 with respect to the variables occurring in I (the computation τ consists solely of a local computation of the root-object), so $\sigma_m, \omega \models I$. Next we will prove by induction on m, the length of the computation τ (let $\omega(z_{c_n}) = \alpha$):

- If $X_m(\alpha) = \mathit{Before}(R, S_n^{c_n})$, and R is minimal with respect to the relation "is a proper normal substatement of", then $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models \mathit{Pre}(R)$.
- If $X_m(\alpha) = \mathit{After}(R, S_n^{c_n})$, and for some $1 \leq k < m$, $X_k(\alpha) = \mathit{Before}(R, S_n^{c_n})$, such that for every $k < l < m$, $X_l(\alpha) = \mathit{Before}(R', S_n^{c_n})$ implies that R' is a substatement of R (this property of R will be abbreviated to "*After*(R) is τ-*reachable*"), then $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models \mathit{Post}(R)$.
- If $X_m(\alpha) = E$ then $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models q_n^{c_n}$.

The additional condition in the first clause above can be shown to be necessary as follows: Let $R = R_1; R_2$ occur in $S_n^{c_n}$, where $R_1 =$ while $0 \leq x$ do $x := x - 1$ od. Note that $Before(R, S_n^{c_n}) = Before(R_1, S_n^{c_n})$. Suppose that $Pre(R) = 0 < x$ and $Pre(R_1) = 0 \leq x$. We have $\models 0 < x \rightarrow 0 \leq x$, but of course not the other way around, so when $X_m(\alpha) = Before(R, S_n^{c_n})$ then $0 < x$ does not hold necessarily. Here we go.

$m = 0$: $X_0(\alpha) = S_n^{c_n}$. We are given that $\models p_n^{c_n} \rightarrow Pre(S_n^{c_n})$ and $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models p_n^{c_n}$. So we have that $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models Pre(S_n^{c_n})$. Now let R be minimal such that $Before(R, S_n^{c_n}) = S_n^{c_n}$. Then $S_n^{c_n} = R$ or $S_n^{c_n} = R; R'$, for some R'. In the latter case we are given that $\models Pre(S_n^{c_n}) \rightarrow Pre(R)$. So $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models Pre(R)$.

$m > 0$: We distinguish the following cases:

1. $X_m(\alpha) = E; R, E$.
   (a) $X_{m-1}(\alpha) = Before(x := e, S_n^{c_n})$:
   By the induction hypothesis we have $\langle \alpha, \sigma_{m-1}(\alpha) \rangle, \omega \models Pre(x := e)$. We are given that $\models Pre(x := e) \rightarrow Post(x := e)[e/x]$. Furthermore we have that $\langle \alpha, \sigma_m(\alpha) \rangle \in \mathcal{S}[\![x := e]\!](\langle \alpha, \sigma_{m-1}(\alpha) \rangle)$. From which by the soundness of the local assignment axiom we infer $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models Post(x := e)$. Let $After(R', S_n^{c_n}) = E; R, E$, such that $After(R')$ is τ-*reachable*. An easy induction on the complexity of R' establishes that $\models Post(x := e) \rightarrow Post(R')$, implying that for $X_m(\alpha) = E$ we have that $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models q^{c_n}$ because we are given that $\models Post(S_n^{c_n}) \rightarrow q_n^{c_n}$.
   (b) $X_{m-1}(\alpha) = Before(R', S_n^{c_n})$, $R' =$ while e do R_1 od, and $\langle \alpha, \sigma_{m-1}(\alpha) \rangle \models \neg e$:
   By the induction hypothesis we have that $\langle \alpha, \sigma_{m-1}(\alpha) \rangle, \omega \models Pre(R')$. We are given that $\models Pre(R') \wedge \neg e \rightarrow Post(R')$. Furthermore we have that $\sigma_{m-1} = \sigma_m$. Thus $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models Post(R')$. For R'' such that $After(R'', S^{c_n}) = E; R, E$, and $After(R'')$ is τ-*reachable*, one can prove by induction on the complexity of R'' that $\models Post(R') \rightarrow Post(R'')$. Furthermore, as in the previous case, we have for $X_m(\alpha) = E$ that $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models q_n^{c_n}$.
2. $X_m(\alpha) = R(\neq E)$:
   (a) $X_{m-1}(\alpha) = Before(R', S_n^{c_n})$, $R' =$ if e then R_1 else R_2 fi, and $Before(R_1, S^{c_n}) = R$:
   By the induction hypothesis we have that $\langle \alpha, \sigma_{m-1}(\alpha) \rangle, \omega \models Pre(R')$. Furthermore we know that $\langle \alpha, \sigma_{m-1}(\alpha) \rangle \models e$. We are given that $\models Pre(R') \wedge e \rightarrow Pre(R_1)$. So $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models Pre(R_1)$. Let R'' be minimal such that $Before(R'', S_n^{c_n}) = R$, it follows that $R_1 = R''$, or $R_1 = R''; R'''$, for some R'''. In the latter case we are given that

$Pre(R_1) \rightarrow Pre(R'')$, so $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models Pre(R'')$. The case that $R = R_2$ is treated in the same way.

(b) $X_{m-1}(\alpha) = Before(R', S_n^{c_n})$, R' = while e do R_1 od, and $Before(R_1, S_n^{c_n}) = R$: Analogously to the previous case.

(c) $X_{m-1}(\alpha) = E; R$:
Let R_1 be the minimal normal statement such that $R = Before(R_1, S_n^{c_n})$, and R_2 be maximal such that $After(R_2, S_n^{c_n}) = E; R$, $After(R_2)$ being τ-*reachable*. Note that by the additional restriction on the computation τ, which ensures that in the configuration (X_m, σ_m) every existing object is executing outside a bracketed section, we have that R_2 is a *normal* statement. By the induction hypothesis we have that $\langle \alpha, \sigma_{m-1}(\alpha) \rangle, \omega \models Post(R_2)$. There are the following two cases to consider:

i. $R_2; R_1$ is a substatement of $S_n^{c_n}$:
We are given that $\models Post(R_2) \rightarrow Pre(R_1)$. So we have that $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models Pre(R_1)$.

ii. R_1 = while e do R_2 od:
We have that $\models Post(R_2) \rightarrow Pre(R_1)$, from which follows the desired result.

Next we consider the case $|h| > 0$:
$h = h_1 \circ < \alpha, \beta >$, where, say, $\alpha \in O^{c_i}, \beta \in O^{c_j}$:

It is not difficult to see that we may assume that

$$\tau = (X_0, \sigma_0), \ldots, (X_k, \sigma_k), \ldots, (X_l, \sigma_l), \ldots, (X_{l'}, \sigma_{l'}), \ldots (X_m, \sigma_m)$$

where,

1. $X_k(\alpha) = Before(R, S_i^{c_i})$, $X_l(\alpha) = After(R, S_i^{c_i})$, R the bracketed section execution of which by α consists of the creation of β
2. From (X_k, σ_k) to $(X_{l'}, \sigma_{l'})$ only α is performing
3. From $(X_{l'}, \sigma_{l'})$ to (X_m, σ_m) only β is performing

Of course this can be proved formally. However a formal proof being straightforward but rather tedious notationally we think we are justified in giving only the following informal explanation: Consider the moment of the computation τ that the object α is about to enter the bracketed section execution of which consists of the creation of

β. From that moment on all objects execute independently from each other, which implies that from here on we can sequentialize the local computations in an arbitrary way.

Let for $c_k \in \{c_1, \ldots, c_n\}$, $\gamma (\neq \alpha, \beta) \in O^{c_k}$, $X_k(\gamma) = X_m(\gamma) = \mathit{Before}(R', S_k^{c_k})$, R' a bracketed section. By the induction hypothesis we have that $\langle \gamma, \sigma_k(\gamma) \rangle, \omega \models \mathit{Pre}(R')$. But $\sigma_k(\gamma) = \sigma_m(\gamma)$, so $\langle \gamma, \sigma_m(\gamma) \rangle, \omega \models \mathit{Pre}(R')$. Analogously for $X_k(\gamma) = X_m(\gamma) = \mathit{After}(R', S_k^{c_k})$, E (R' a bracketed section).

Furthermore it follows by the induction hypothesis that

$$\sigma_k, \omega\{\alpha / z_{c_i}\} \models I \wedge \mathit{Pre}(R) \downarrow z_{c_i},$$

where z_{c_i} does not occur in I. We are given that

$$\models \{I \wedge \mathit{Pre}(R) \downarrow z_{c_i}\}(z_{c_i}, R)\{I \wedge \mathit{Post}(R) \downarrow z_{c_i} \wedge p^{c_j} \downarrow z_{c_j}[z_{c_i}.x / z_{c_j}]\},$$

assuming that $x :=$ new occurs in R. We have that

$$\sigma_l \in \mathcal{I}[\![(z_{c_i}, R)]\!](\omega\{\alpha / z_{c_i}\})(\sigma_k).$$

So we may infer that

$$\sigma_l, \omega\{\alpha / z_{c_i}\} \models (I \wedge \mathit{Post}(R) \downarrow z_{c_i} \wedge p^{c_j} \downarrow z_{c_j}[z_{c_i}.x / z_{c_j}]).$$

As z_{c_i} does not occur in I and σ_l agrees with σ_m with respect to the variables occurring in I we have that $\sigma_m, \omega \models I$.

From $\langle \alpha, \sigma_l(\alpha) \rangle, \omega \models \mathit{Post}(R)$ we infer by an argument similar to the one applied to the case $|h| = 0$ that $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models \mathit{Pre}(R')$, $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models \mathit{Post}(R')$, $\langle \alpha, \sigma_m(\alpha) \rangle, \omega \models q_i^{c_i}$, in case $X_m(\alpha) = \mathit{Before}(R', S_i^{c_i})$, $X_m(\alpha) = \mathit{After}(R', S_i^{c_i})$, R' a bracketed section, and $X_m(\alpha) = E$, respectively. From $\langle \beta, \sigma_{l'}(\beta) \rangle, \omega \models p_j^{c_j}$ we derive in a similar way an analogous result for β. (Note that by the definition of a bracketed section we have $\sigma_l(\alpha)(x) = \beta$.)

Finally, let $h = h_1 \circ < \alpha, \beta, \gamma >$, where, say, $\alpha \in O^{c_i}, \beta \in O^{c_j}$:

Again it is not difficult to see that we may assume that

$$\tau = (X_0, \sigma_0), \ldots, (X_k, \sigma_k), \ldots, (X_l, \sigma_l), \ldots, (X_{l'}, \sigma_{l'}), \ldots, (X_m, \sigma_m),$$

where

1. $X_k(\alpha) = \mathit{Before}(R, S_i^{c_i})$, $X_k(\beta) = \mathit{Before}(R', S_j^{c_j})$, $X_l(\alpha) = \mathit{After}(R, S_i^{c_i})$, and $X_l(\beta) = \mathit{After}(R', S_j^{c_j})$, where R, R' are the bracketed sections execution of which by α and β consists of the transfer of γ from α to β

2. From (X_k, σ_k) to (X_m, σ_m) only α and β are performing
3. From (X_l, σ_l) to $(X_{l'}, \sigma_{l'})$ only α is performing
4. From $(X_{l'}, \sigma_{l'})$ to (X_m, σ_m) only β is performing

For $\gamma' \neq \alpha, \beta$ the desired result follows from the induction hypothesis as in the previous case. Furthermore we have by the induction hypothesis that

$$\begin{aligned} &\sigma_k, \omega \models I, \\ &\langle \alpha, \sigma_k(\alpha) \rangle, \omega \models Pre(R), \text{ and} \\ &\langle \beta, \sigma_k(\beta) \rangle, \omega \models Pre(R'). \end{aligned}$$

Let $z \in LVar_{c_i}$, $z' \in LVar_{c_j}$ do not occur in I. It then follows that

$$\sigma_k, \omega\{\alpha, \beta / z, z'\} \models (I \wedge Pre(R) \downarrow z \wedge Pre(R') \downarrow z').$$

Furthermore we are given the truth of the formula

$$\{I \wedge Pre(R) \downarrow z \wedge Pre(R') \downarrow z'\}(z, R) \parallel (z', R')\{I \wedge Post(R) \downarrow z \wedge Post(R') \downarrow z'\},$$

and

$$\sigma_l \in \mathcal{I}[\![(z_{c_i}, R) \parallel (z'_{c_j}, R')]\!](\omega\{\alpha, \beta / z_{c_i}, z'_{c_j}\})(\sigma_k).$$

From which we derive that $\sigma_l, \omega \models I$ (so $\sigma_m, \omega \models I$), $\langle \alpha, \sigma_l(\alpha) \rangle, \omega \models Post(R)$, and $\langle \beta, \sigma_l(\beta) \rangle, \omega \models Post(R')$. Finally, reasoning in a similar way as in the case $|h| = 0$ will give us the desired result. □

8 Completeness

In this section we prove the completeness of the proof system, that is, we prove that an arbitrary valid global correctness formula is derivable. Without loss of generality we may assume the sets C and $IVar$ to be finite.

8.1 Histories

First we want to modify a program 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 [Ap2]. 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. We define now the set of expressions Exp^c_a, with typical element e^c_a, as follows:

Definition 8.1

$$
\begin{array}{lcll}
e^c_a & ::= & x^c_a & \\
 & | & \mathsf{self} & a = c \\
 & | & \mathsf{nil} & \\
 & | & n & a = \mathsf{Int} \\
 & | & \mathsf{true} \mid \mathsf{false} & a = \mathsf{Bool} \\
 & | & e_1{}^c_{\mathsf{Int}} + e_2{}^c_{\mathsf{Int}} & e = \mathsf{Int} \\
 & & \vdots & \\
 & | & |e^c_{d^*}| & a = \mathsf{Int} \\
 & | & \langle e^c_d \rangle & a = d^* \\
 & | & e_1{}^c_a \circ e_2{}^c_a & a = d^* \\
 & | & e_1{}^c_d \doteq e_2{}^c_d & a = \mathsf{Bool}
\end{array}
$$

The value of the expression $|e^c_{d^*}|$ is the length of the sequence denoted by the expression $e^c_{d^*}$. The value of the expression $e_{1^c_a} \circ e_{2^c_a}$ is the result of concatenating the two sequences denoted by the subexpressions $e_{1^c_a}, e_{2^c_a}$. Finally, the expression $\langle e^c_d \rangle$ denotes the sequence consisting of one element, i.e., the value of the expression e^c_d.

The set of statements $Stat^c$ is defined as before, but for the assignment statement. Assignment statements now have the form: $x^c_a := e^c_a$.

Programs are defined as before. Note that we introduced only some very restricted repetoire of operations on sequences in our programming language. The main reason being that more elaborate operations will complicate our substitution operations. However to prove completeness the operations introduced above suffice.

The set of local expressions $LExp^c_a$ is extended as follows:

Definition 8.2

$$
\begin{array}{lcll}
l^c_a & ::= & z_a & a = d, d^*,\ d \in \{\mathsf{Int}, \mathsf{Bool}\} \\
 & | & x^c_a & \\
 & | & \mathsf{self} & a = c \\
 & | & \mathsf{nil} & \\
 & | & n & a = \mathsf{Int} \\
 & | & \mathsf{true} \mid \mathsf{false} & a = \mathsf{Bool} \\
 & | & l_{1^c_{d^*}} : l_{2\mathsf{Int}} & a = d \\
 & | & |l^c_{d^*}| & a = \mathsf{Int} \\
 & | & \langle l^c_d \rangle & a = d^* \\
 & | & l_{1^c_a} \circ l_{2^c_a} & e = d^* \\
 & | & l_{1^c_{\mathsf{Int}}} + l_{2^c_{\mathsf{Int}}} & a = \mathsf{Int} \\
 & & \vdots & \\
 & | & l_{1^c_d} \doteq l_{2^c_d} & a = \mathsf{Bool}
\end{array}
$$

The added operations on sequences are interpreted as described above. Local assertions are defined as before.

As we do not want to redefine our substitution operations we do *not* change our global assertion language but for allowing instance variables of type d^*, for $d \in C^+$. As a consequence we have to redefine the definition of the transformation of a local

expression, local assertion to a global expression, global assertion, respectively. We do so by viewing the global expressions $\langle g \rangle$, $g_1 \circ g_2$ as abbreviations in the following sense: Suppose the expression $\langle g \rangle$ occurs in the assertion P. Let P' be such that $P'[\langle g \rangle / z] = P$. Then we can view P as an abbreviation of the assertion

$$\exists z(|z| \doteq 1 \wedge z : 1 \doteq g \wedge P').$$

In case an expression of the form $g_1 \circ g_2$ occurs in P, let P' now be such that $P'[g_1 \circ g_2 / z] = P$. Then we can view P as an abbreviation of the assertion

$$\begin{array}{ll} \exists z & (\\ & \quad |z| \doteq |g_1| + |g_2| \wedge \\ & \quad \forall i(1 \leq i \leq |g_1| \rightarrow z : i \doteq g_1 : i) \wedge \\ & \quad \forall i(|g_1|| < i \leq |z| \rightarrow z : i \doteq g_2 : i) \wedge \\ & \quad P' \\ &) \end{array}$$

Definition 8.3
We can define now $l \downarrow g$ simply as follows:

$$\begin{array}{lcl} z \downarrow g & = & z \\ x \downarrow g & = & g.x \\ \mathsf{self} \downarrow g & = & g \\ \langle l \rangle \downarrow g & = & \langle l \downarrow g \rangle \\ l_1 \circ l_2 \downarrow g & = & l_1 \downarrow g \circ l_2 \downarrow g \\ & \ldots & \end{array}$$

The transformation $p \downarrow g$ is defined accordingly.

Note that the resulting assertion, in the case that p contains these newly introduced expressions, really is an abbreviation of an assertion.

Next we describe how to code histories in the extended version of our programming language. First we order the set $\{c_1, \ldots, c_n, \mathsf{Int}, \mathsf{Bool}\}$, assuming $C = \{c_1, \ldots, c_n\}$, as follows:

$$\begin{array}{lcl} Ord(c_i) & = & i \\ Ord(\mathsf{Int}) & = & n+1 \\ Ord(\mathsf{Bool}) & = & n+2 \end{array}$$

Given this ordering we assume the set of pairs $< c, d >$ to be ordered lexicographically. Now we introduce variables which will be used to code the history of an arbitrary object.

Definition 8.4
For an arbitrary c we associate with each pair $< c_i, d >$ the variables:

$$in1_m \in IVar^c_{c_i^*},\ in2_m \in IVar^c_{d^*},\ in3_m \in IVar^c_{\mathsf{Int}^*},$$
$$out1_m \in IVar^c_{c_i^*},\ out2_m \in IVar^c_{d^*},\ out3_m \in IVar^c_{\mathsf{Int}^*},$$

where $m = Ord(< c_i, d >)$, and with each d the variables

$$in2_m \in IVar^c_{d^*},\ in3_m \in IVar^c_{\mathsf{Int}^*},$$

where $m = n^2 + 2n + Ord(d)$, and, finally, with c_i the variables

$$act1_m \in IVar^c_{c_i^*},\ act2_m \in IVar^c_{\mathsf{Int}^*},$$

where $m = Ord(c_i)$.

Finally, for each c we introduce a fresh variable $count \in IVar^c_{\mathsf{Int}}$.

Definition 8.5
For $1 \leq i \leq n$ the set of variables, as introduced above, belonging to $IVar^{c_i}$, we denote by H^{c_i}.

Let $m = Ord(< c_i, d >)$. Then for an arbitrary object of class c the variable $in1_m$ will record the order in which objects of class c_i communicated an object of class d to it. These communicated objects are stored in the order of their arrival in the variable $in2_m$. The variable $in3_m$ will record for each of the above communications the *local* time it occurs.

Analogously we have that for an arbitrary object of class c the variable $out1_m$ will record the order in which objects of class c_i received an object of class d from it. The communicated objects are stored again in the order of their arrival in the variable $out2_m$. The variable $out3_m$ will record the local time of each of these communications.

Let for $d \in C'^+$ $m = n^2 + 2n + Ord(d)$. The variable $in2_m$ will record for each object of class c the sequence of objects of class d communicated by an unknown object. The variable $in3_m$ again will record the local time of each of these communications.

Let $m = Ord(c_i)$. Then for an arbitrary object of class c the variable $act1_m$ will record the order of objects of class c_i which have been created by it. The variable $act2_m$ will record for each of these activations its local time.

Finally, for each object of class c the integer variable *count* will record its local time, i.e, this variable simply counts the number of communications and activations of this object.

As we introduce instance variables ranging over sequences only to be able to code histories we assume that $IVar^c_{d^*} = H^c_{d^*}$, for $d \in C^+$.

According to the above informal explanation we transform a program ρ, assuming $IVar(\rho) \cap \bigcup_c H^c = \emptyset$, to ρ' as follows:

Definition 8.6
First we transform an arbitrary I/O statement and new statement:

$$
\begin{array}{lcl}
x^c_{c_i}?y^c_d & \Rightarrow & x^c_{c_i}?y^c_d; count := count + 1; \\
 & & in1_m, in2_m, in3_m := in1_m \circ < x^c_{c_i} >, in2_m \circ < y^c_d >, in3_m \circ < count >, \\
 & & \text{where } m = Ord(< c_i, d >) \\
x^c_{c_i}!e^c_d & \Rightarrow & x^c_{c_i}!e^c_d; count := count + 1; \\
 & & out1_m, out2_m, out3_m := out1_m \circ < x^c_{c_i} >, out2_m \circ < e^c_d >, out3_m \circ < count > \\
 & & \text{where } m = Ord(< c_i, d >) \\
?y^c_d & \Rightarrow & ?y^c_d; count := count + 1; \\
 & & in2_m, in3_m := in2_m \circ < y^c_d >, in3_m \circ < count >, \\
 & & \text{where } m = n^2 + 2n + Ord(d) \\
x^c_d := \mathsf{new} & \Rightarrow & x^c_d := \mathsf{new}; count := count + 1; \\
 & & act1_m, act2_m := act1_m \circ < x^c_d >, act2_m \circ < count >, \\
 & & \text{where } m = Ord(d)
\end{array}
$$

Here $x_1, \ldots, x_k := e_1, \ldots, e_k$ stands for $x_1 := e_1; \ldots; x_k := e_k$. Let for an arbitrary statement S^c the result of applying the above transformation to S, defined by induction on the structure of S, be denoted by S'. Given a program $\rho = \langle U : S^{c_n}_n \rangle$, with $U = c_1 \leftarrow S^{c_1}_1, \ldots, c_{n-1} \leftarrow S^{c_{n-1}}_{n-1}$, we define $\rho' = \langle U' : S'^{c_n}_n \rangle$, with $U' = c_1 \leftarrow S'^{c_1}_1, \ldots, c_{n-1} \leftarrow S'^{c_{n-1}}_{n-1}$.

We introduce the following notation to refer to the history of an object of class c encoded by the values of the variables of H^c.

Definition 8.7
Let σ and α such that $\alpha \in \sigma^{(c)}$. We define $h_{\sigma,\alpha}$ to be the history encoded by the values of the variables of H^c of α, if such a history exists, otherwise $h_{\sigma,\alpha}$ is undefined.

8.2 A most general proof outline

Let $\rho = \langle U : S^{c_n}_n \rangle$, with $U = c_1 \leftarrow S^{c_1}_1, \ldots, c_{n-1} \leftarrow S^{c_{n-1}}_{n-1}$, be a program such that $IVar(\rho) \cap \bigcup_c H^c = \emptyset$, and for which the formula $\{p \downarrow z_{c_n}\}\rho\{Q\}$ is valid, where

$IVar(p,Q) \cap \bigcup_c H^c = \emptyset$. We define a most general proof outline for ρ', that is, we will define a global invariant I and for every occurrence of an arbitrary normal substatement S of ρ' local assertions $Pre(S)$ and $Post(S)$ such that for

$$A_k = \{\{Pre(S)\}S\{Post(S)\} : S \text{ an occurrence of a bracketed section of } S'^{c_k}\}$$

we have $A_k \vdash \{Pre(S'^{c_k})\}S'^{c_k}\{Post(S'^{c_k})\}$ and $Coop(A_1,\ldots,A_n,I,z_{c_n})$. By an application of the rules (PR), (PC), (S1), (S2), (INIT) and (Aux) the derivability of $\{p \downarrow z_{c_n}\}\rho\{Q\}$ then can be shown to follow. First we modify the precondition p^{c_n} as follows:

Definition 8.8
We define

$$p'^{c_n} = p^{c_n} \wedge \bigwedge_{x \in W} (x \doteq z_x) \wedge Ihist,$$

where $W = IVar^{c_n}_{\mathsf{Int}} \cup IVar^{c_n}_{\mathsf{Bool}}$, 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. Furthermore *Ihist* stands for the assertion

$$count \doteq 0 \wedge \bigwedge_d \bigwedge_{x \in H^{c_n}_{d\bullet}} |x| \doteq 0.$$

The assertion *Ihist* initializes the history of the root-object.

We start with the definition of the global invariant. This global invariant describes all states σ for which there exists an intermediate configuration (X',σ') of a computation of ρ' such that σ and σ' agree with respect to the existing objects and with respect to the histories of these objects. Furthermore, in the configuration (X',σ') every existing object is executing *outside* a bracketed section, so no object is involved in a communication or the creation of some object.

Lemma 8.9
There exists a global assertion I, $IVar(I) \subseteq \bigcup_c H^c$, such that:

$\sigma,\omega \models I$ iff there exist configurations $(X_0,\sigma_0),(X_1,\sigma_1)$ such that:

- For some history h : $(X_0,\sigma_0) \rightarrow^h (X_1,\sigma_1)$, where $Init_{\rho'}((X_0,\sigma_0))$, and for $\alpha \in \sigma_0^{(c_n)}$ we have $\langle \alpha, \sigma_0(\alpha)\rangle, \omega \models p'^{c_n}$.
- For an arbitrary c and $\alpha \in \sigma_1^{(c)}$ we have that $X_1(\alpha)$ is normal.
- For an arbitrary c we have $\sigma^{(c)} = \sigma_1^{(c)}$, and for every $\alpha \in \sigma^{(c)}$ we have $h_{\sigma,\alpha} = h_{\sigma_1,\alpha}$.

Proof
See section 9. □

The next lemma states the existence of a local assertion $Pre(R)$, R a normal substatement of S'^{c_i}. This local assertion describes all the local states $\theta = \langle \alpha, s \rangle$ for which there exists an intermediate configuration (X, σ) of a computation of ρ' such that α exists in σ and s equals $\sigma(\alpha)$, furthermore, in the configuration (X, σ) the object α is about to execute R.

Lemma 8.10
Let R be a normal substatement of $S'^{c_i}_i$. There exists a local assertion $Pre(R)$ such that:

$\theta^{c_i}, \omega \models Pre(S)$ iff there exist configurations $(X_0, \sigma_0), (X_1, \sigma_1)$ such that:

- For some history h we have $(X_0, \sigma_0) \rightarrow^h (X_1, \sigma_1)$, where $Init_{\rho'}((X_0, \sigma_0))$, and for $\alpha \in \sigma_0^{(c_n)}$ we have $\langle \alpha, \sigma_0(\alpha) \rangle, \omega \models p'^{c_n}$.
- For some $\alpha \in \sigma_1^{(c_i)}$ we have $\langle \alpha, \sigma_1(\alpha) \rangle = \theta^{c_i}$ and $X_1(\alpha) = Before(R, S'^{c_i}_i)$.

Proof
See section 9. □

Analogously we have a lemma asserting the existence of a local assertion $Post(R)$, R a normal subprogram of, say, $S'^{c_i}_i$: Just substitute $After(R, S'^{c_i}_i)$ for the phrase $Before(R, S'^{c_i}_i)$.

Now we define our sets of assumptions.

Definition 8.11
Let

$$A_k = \{\{Pre(R)\}R\{Post(R)\} : R \text{ a bracketed section occurring in } S'^{c_k}_k\}.$$

We have the following lemma stating that from this set of assumptions A_k we can derive the local correctness formula $\{Pre(S'^{c_k}_k)\}S'^{c_k}_k\{Post(S'^{c_k}_k)\}$.

Lemma 8.12
For the set of local correctness formulas A_k as defined above we have

$$A_k \vdash \{Pre(S'^{c_k}_k)\}S'^{c_k}_k\{Post(S'^{c_k}_k)\}.$$

Proof
Follows in a straightforward manner from the semantics of the assertions $Pre(R)$ and $Post(R)$, R a normal statement of S', as described above, using lemma 7.9. □

To show that these assumptions cooperate we need the following definition and lemma:

Definition 8.13
Let, for an arbitrary c, $B^c \subseteq \sigma^{(c)}$, and, for $\alpha \in B^c$, R_α be a bracketed section of S'^c or be the empty statement E. We call the set $\bigcup_c\{< \alpha, R_\alpha >: \alpha \in B^c\}$ (σ,ω)-*reachable* iff there exist configurations $(X_0, \sigma_0), (X_1, \sigma_1)$ such that:

- For some history h: $(X_0, \sigma_0) \rightarrow^h (X_1, \sigma_1)$, where $INIT_{p'}((X_0, \sigma_0))$, and for $\alpha \in \sigma_0^{(c_n)}$ we have $\langle \alpha, \sigma_0(\alpha)\rangle, \omega \models p'^{c_n}$.
- For an arbitrary c we have $B^c \subseteq \sigma_1^{(c)}$, for $\alpha \in B^c$ we have $Before(R_\alpha, S'^c) = X_1(\alpha)$ if $R_\alpha \neq E$, $X_1(\alpha) = E$, otherwise, and $\sigma(\alpha) = \sigma_1(\alpha)$.

If, additionally, we have for an arbitrary c that $\sigma^{(c)} = \sigma_1^{(c)}$ and for all $\alpha \in \sigma^{(c)}$ we have that $X_1(\alpha)$ is normal and $h_{\sigma,\alpha} = h_{\sigma_1,\alpha}$ we speak of $(I, \sigma, \omega) - reachability$.

Note that $\sigma, \omega \models Pre(R) \downarrow z$, assuming R to be a bracketed section of S'^c and $z \in LogVar_c$, implies the $(\sigma, \omega) - reachability$ of $< \alpha, R >$, where $\alpha = \omega(z)$. We have the following lemma, which is called "the merging lemma", about merging different computations into one computation.

Lemma 8.14
Let, for an arbitrary c, $B^c \subseteq \sigma^{(c)}$ be such that for $\alpha \in B^c$ and R_α (a bracketed section of S'^c or the empty statement E) we have that $< \alpha, R_\alpha >$ is $(\sigma, \omega) - reachable$ and $\sigma, \omega \models I$. It then follows that $\bigcup_c\{< \alpha, R_\alpha >: \alpha \in B^c\}$ is $(I, \sigma, \omega) - reachable$.

Proof
By $\sigma, \omega \models I$ we have for some configurations $(X_0, \sigma_0), (X_1, \sigma_1)$:

- For some history h : $(X_0, \sigma_0) \rightarrow^h (X_1, \sigma_1)$, where $Init_{p'}((X_0, \sigma_0))$, and for $\alpha \in \sigma_0^{(c_n)}$ we have $\langle \alpha, \sigma_0(\alpha)\rangle, \omega \models p'^{c_n}$.

- For an arbitrary c and $\alpha \in \sigma_1^{(c)}$ we have that $X_1(\alpha)$ is normal.
- For an arbitrary c we have $\sigma^{(c)} = \sigma_1^{(c)}$, and for every $\alpha \in \sigma^{(c)}$ we have $h_{\sigma,\alpha} = h_{\sigma_1,\alpha}$.

By the $(\sigma,\omega)-$*reachability* of $< \alpha, R_\alpha >$ we have for some configurations $(X_\alpha, \sigma_\alpha), (X'_\alpha, \sigma'_\alpha)$:

- For some history h_α:$(X_\alpha, \sigma_\alpha) \rightarrow^{h_\alpha} (X'_\alpha, \sigma'_\alpha)$, where $Init_{\rho'}((X_\alpha, \sigma_\alpha >)$, and for $\beta \in \sigma_\alpha^{(c_n)}$ we have $\langle \beta, \sigma_\alpha(\beta)\rangle, \omega \models p'^{c_n}$.
- For $\alpha \in \sigma'^{(c)}_\alpha$ we have $\sigma(\alpha) = \sigma'_\alpha(\alpha)$ and $X'_\alpha(\alpha) = Before(R_\alpha, S'^c)$.

As a local computation of an object which is executing outside a bracketed section does not affect the history variables, i.e., the variables used to encode the local history, we may assume without loss of generality that there exists no configuration (X, σ) such that $(X_1, \sigma_1) \rightarrow^\epsilon (X, \sigma)$, so every existing object in (X_1, σ_1) is about to enter a bracketed section or has terminated. (Just execute the local process of an object until a bracketed section has reached or it has terminated.) It suffices to prove that for $\alpha \in B^c$ we have $(X_1(\alpha), \sigma_1(\alpha)) = (X'_\alpha(\alpha), \sigma'_\alpha(\alpha))$.

Now let $\alpha \in B^c$, with $c \neq c_n$: From the definition of the global transition system and the construction of ρ' it follows that $(S'^c, \bigtriangledown) \rightarrow^{h_{\sigma'_\alpha,\alpha}} (X'_\alpha(\alpha), \sigma'_\alpha(\alpha))$ and $(S'^c, \bigtriangledown) \rightarrow^{h_{\sigma,\alpha}} (X_1(\alpha), \sigma_1(\alpha))$. Now by $h_{\sigma'_\alpha,\alpha} = h_{\sigma,\alpha} = h_{\sigma_1,\alpha}$ (the first identity follows from $\sigma(\alpha) = \sigma'_\alpha(\alpha)$, the second from $\sigma, \omega \models I$), and the fact that the environment completely determines the local behaviour of an object we have $(X_1(\alpha), \sigma_1(\alpha)) = (X'_\alpha(\alpha), \sigma'_\alpha(\alpha))$.

Let $\alpha \in B^{c_n}$: From $\langle \alpha, \sigma_0(\alpha)\rangle, \omega \models p'$, $\langle \alpha, \sigma_\alpha(\alpha)\rangle, \omega \models p'$, and $Init_{\rho'}((X_0, \sigma_0), (X_\alpha, \sigma_\alpha))$ it follows that $\sigma_0(\alpha) = \sigma_\alpha(\alpha)$ (the assertion $\bigwedge_{x \in W}(x \doteq z_x)$ freezes the part of the initial state specified by the variables ranging over the standard objects, the predicate $Init_{\rho'}$ fixes the part of the initial state specified by the variables ranging over the non-standard objects, and, finally, the assertion *Ihist* fixes the variables introduced to code the local history). So we can apply an argument similar to the one given above. □

Given the merging lemma we are now ready to prove that the assumptions introduced above cooperate. First we show how to discharge assumptions about bracketed sections containing a new-statement.

Lemma 8.15
Let $R = x := \text{new}; R_1$ be a bracketed section occurring in $S'^{c_i}_i$, assuming the type of the variable x to be c_j, then:

$$\vdash \{I \wedge Pre(R) \downarrow z\}(z, R)\{I \wedge Post(R) \downarrow z \wedge Pre(S'^{c_j}_j) \downarrow z')[z.x/z']\}.$$

where $z \in LVar_{c_i}$ and $z' \in LVar_{c_j}$ are two distinct new variables.

Proof
Let $[act]$ abbreviate the substitution corresponding to the updating of the local history as described by R_1:

$$[z.act1_j \circ < z.x >, z.act2_j \circ < z.count + 1 >, z.count + 1/z.act1_j, z.act2_j, z.count].$$

By the axioms (IASS) and (NEW) it suffices to show that:

$$\begin{aligned} &\models I \wedge Pre(R) \downarrow z \rightarrow \\ &(I \wedge Post(R) \downarrow z \wedge Pre(S'^{c_j}_j) \downarrow z'[z.x/z'])[act][z''/z.x][\text{new}/z''] \end{aligned}$$

where $z'' \in LVar_{c_j}$ is a new variable: Let $\sigma, \omega \models I \wedge Pre(R) \downarrow z$. Now $\sigma, \omega \models Pre(R) \downarrow z$ implies the (σ, ω) – *reachability* of $< \omega(z), R >$. An application of the merging lemma gives us the (I, σ, ω) – *reachability* of $< \omega(z), R >$. So there exist configurations $(X_1, \sigma_1), (X_2, \sigma_2)$ such that

- For some history h: $(X_1, \sigma_1) \rightarrow^h (X_2, \sigma_2)$, where $Init_{\rho'}((X_1, \sigma_1))$, and for $\alpha \in \sigma_1^{(c_n)}$ we have $\langle \alpha, \sigma(\alpha) \rangle, \omega \models p'^{c_n}$.
- For an arbitrary c we have $\sigma^{(c)} = \sigma_2^{(c)}$, and for $\alpha \in \sigma^{(c)}$ we have that $X_1(\alpha)$ is normal and $h_{\sigma,\alpha} = h_{\sigma_2,\alpha}$.
- For $\alpha = \omega(z)$ we have $\sigma(\alpha) = \sigma_2(\alpha)$ and $Before(R, S'^{c_i}) = X_2(\alpha)$.

Let $\sigma' \in \mathcal{I}[\![(z, R)]\!](\omega)(\sigma)$. It then follows that for σ_3 such that

$$\begin{aligned} \sigma_3^{(c)} &= \sigma_2^{(c)} \cup \{\beta\} \quad c = c_j \\ &= \sigma_2^{(c)} \qquad \text{otherwise,} \end{aligned}$$

with $\beta = \sigma'(\alpha)(x)$, and $\sigma_{3(2)} = \sigma_{2(2)}\{\sigma'(\alpha)/\alpha\}\{\nabla/\beta\}$, with $\alpha = \omega(z)$, we have $(X_1, \sigma_1) \rightarrow^{h'} (X_3, \sigma_3)$, where $h' = h \circ < \alpha, \beta >$ and $X_3 = X_2\{After(R, S'^{c_i})/\alpha\}$.

By $(X_1, \sigma_1) \rightarrow^{h'} (X_3, \sigma_3)$ we have $\sigma', \omega \models I$. Furthermore it follows that $\sigma', \omega \models Post(R) \downarrow z$ and $\sigma', \omega\{\beta/z'\} \models Pre(S'^{c_j}_j) \downarrow z'$, or, equivalently, $\sigma', \omega \models Pre(S'^{c_j}) \downarrow z'[z.x/z']$.

Summerizing we have

$$\sigma', \omega \models (I \wedge Post(R) \downarrow z \wedge Pre(S'^{c_j}) \downarrow z'[z.x/z']).$$

From which in turn it is not difficult to derive by an application of the lemmas 7.1 and 7.3 that:

$$\sigma, \omega \models (I \wedge Post(R) \downarrow z \wedge Pre(S'^{c_j}) \downarrow z'[z.x/z'])[act][z''/z.x][\mathsf{new}/z''].$$

□

Next we show how to discharge assumptions about matching bracketed sections.

Lemma 8.16
For two arbitrary matching bracketed sections R_1 and R_2, occurring in, say, $S'^{c_i}_i, S'^{c_j}_j$, $1 \leq i < n$, $1 \leq j \leq n$, we have:

$$\begin{array}{rc} & \{I \wedge Pre(R_1) \downarrow z \wedge Pre(R_2) \downarrow z'\} \\ \vdash & (z, R_1) \parallel (z', R_2) \\ & \{I \wedge Post(R_1) \downarrow z \wedge Post(R_2) \downarrow z'\} \end{array}$$

where $z \in LVar_{c_i}$ and $z' \in LVar_{c_j}$ are two new distinct variables.

Proof
We prove the following case, the other ones are treated in a similar way: Let $R_1 = x!e; R'_1$, $R_2 = ?y; R'_2$, and $i \neq j$. Let $[in]$ abbreviate the substitution corresponding to the update of the local history as given by R'_1:

$$\left[\begin{array}{l} z'.in1_k \circ < z'.y >, z'.in2_k \circ < z'.count + 1 >, z'.count + 1/ \\ z'.in1_k, z'.in2_k, z'.count. \end{array}\right].$$

And $[out]$ abbreviate the substitution corresponding to the update of the local history as given by R'_2:

$$\left[\begin{array}{l} z.out1_l \circ < z.x >, z.out2_l \circ < e \mid z >, z.out3_l \circ < z.count + 1 >, z.count + 1/ \\ z.out1_l, z.out2_l, z.out3_l, z.count. \end{array}\right].$$

By the axioms (COMM) and (IASS), the rules (SR2), (PAR1) and (PAR2) it suffices to show that:

$$\begin{array}{l} \models I \wedge Pre(R_1) \downarrow z \wedge Pre(R_2) \downarrow z' \wedge z.x = z' \wedge z.x \neq \mathsf{nil} \rightarrow \\ (I \wedge Post(R_1) \downarrow z \wedge Post(R_2) \downarrow z')[in][out][e \downarrow z/z'.y]. \end{array}$$

Let $\sigma, \omega \models (I \wedge Pre(R_1) \downarrow z \wedge Pre(R_2) \downarrow z' \wedge z.x = z' \wedge z.x \neq \mathsf{nil})$. It follows that $< \alpha, R_1 >$ and $< \beta, R_2 >$ are (σ, ω) – *reachable*, where $\alpha = \omega(z)$ and $\beta = \omega(z')$. From lemma 8.14 then we infer the (I, σ, ω) – *reachability* of $\{< \alpha, R_1 >, < \beta, R_2 >\}$. So there exist configurations $(X_1, \sigma_1), (X_2, \sigma_2)$ such that

- For some history h: $(X_1, \sigma_1) \rightarrow^h (X_2, \sigma_2)$, $Init_{\rho'}((X_1, \sigma_1))$, and for $\alpha \in \sigma_1^{(c_n)}$ we have $\langle \alpha, \sigma(\alpha) \rangle, \omega \models p'^{c_n}$.
- For an arbitrary c we have $\sigma^{(c)} = \sigma_2^{(c)}$ and for $\alpha \in \sigma^{(c)}$ we have that $X_2(\alpha)$ is normal and $h_{\sigma,\alpha} = h_{\sigma_2,\alpha}$.
- We have $Before(R_1, S'^{c_i}_i) = X_2(\alpha)$, $Before(R_2, S'^{c_j}_j) = X_2(\beta)$, and $\sigma(\alpha) = \sigma_2(\alpha)$, $\sigma(\beta) = \sigma_2(\beta)$.

Let $\sigma' \in \mathcal{I}[\![(z, R_1) \parallel (z', R_2)]\!](\omega)(\sigma)$. Note that such a σ' exists because we have $\sigma, \omega \models z.x \doteq z' \wedge z.x \neq \mathsf{nil}$.

By the definition of the global transition system it follows that for σ_3, with $\sigma_3^{(c)} = \sigma_2^{(c)}$, c arbitrary, and $\sigma_{3(2)} = \sigma_{2(2)}\{\sigma'(\alpha)/\alpha\}\{\sigma'(\beta)/\beta\}$ we have $(X_1, \sigma_1) \rightarrow^{h'} (X_3, \sigma_3)$, where $h' = h \circ < \alpha, \beta, \gamma >$, for some γ, and $X_3 = X_2\{After(R_1, S'^{c_i}_i)/\alpha\}\{After(R_2, S'^{c_j}_j)/\beta\}$.

From $(X_1, \sigma_1) \rightarrow^{h'} (X_3, \sigma_3)$ it then follows that $\sigma', \omega \models I$. Furthermore we have $\sigma', \omega \models Post(R_1) \downarrow z$ and $\sigma', \omega \models Post(R_2) \downarrow z'$. Summerizing we have:

$$\sigma', \omega \models (I \wedge Post(R_1) \downarrow z \wedge Post(R_2) \downarrow z').$$

From which in turn we derive by applying lemma 7.1 that

$$\sigma, \omega \models (I \wedge Post(R_1) \downarrow z \wedge Post(R_2) \downarrow z')[in][out][e \downarrow z/z'.y].$$

□

We summerize the above by the following theorem.

Theorem 8.17
The following formula about ρ', which is called the most general correctness formula about ρ', is derivable:

$$\{Pre(S'^{c_n}_n) \downarrow z_{c_n}\}$$
$$\rho'$$
$$\{I \wedge \textstyle\bigwedge_{i \neq n} \forall z_{c_i} Post(S'^{c_i}_i) \downarrow z_{c_i} \wedge Post(S'^{c_n}_n) \downarrow z_{c_n}\}.$$

Proof
By lemma 8.12 we have

$$A_k \vdash \{Pre(S'^{c_k}_k)\} S'^{c_k}_k \{Post(S'^{c_k}_k)\}.$$

Furthermore it is not difficult to prove that

$$\models Pre(S'^{c_n}_n) \downarrow z_{c_n} \wedge \forall z'_{c_n}(z'_{c_n} = z_{c_n}) \wedge \bigwedge_{1 \leq i < n} (\forall z \ \mathsf{false}) \rightarrow I.$$

From this and the lemmas 8.15 and 8.16 it follows that $Coop(A_1, \ldots, A_n, I, z_{c_n})$. Applying the rule (PR) then finishes the proof. □

We are now ready for the completeness theorem.

Theorem 8.18
The valid correctness formula $\{p \downarrow z_{c_n}\}\rho\{Q\}$ is derivable (assuming $IVar(p, Q, \rho) \cap \bigcup_c H^c = \emptyset$):

$$\vdash \{p \downarrow z_{c_n}\}\rho\{Q\}.$$

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

$$\{Pre(S'^{c_n}_n) \downarrow z_{c_n}\}$$
$$\rho'$$
$$\{I \wedge \textstyle\bigwedge_{i \neq n} \forall z_{c_i} Post(S'^{c_i}_i) \downarrow z_{c_i} \wedge Post(S'^{c_n}_n) \downarrow z_{c_n}\}.$$

It is not difficult to prove that $\models p'^{c_n} \wedge \bigwedge_{x \in W} x \doteq \mathsf{nil} \rightarrow Pre(S'^{c_n}_n)$, where $W = \bigcup_c IVar^{c_n}_c$. Next we prove

$$\models I \wedge \bigwedge_{i \neq n} \forall z_{c_i} Post(S'^{c_i}_i) \downarrow z_{c_i} \wedge Post(S'^{c_n}_n) \downarrow z_{c_n} \rightarrow Q.$$

Let the antecedent of the implication be true with respect to some logical environment ω and some global state σ. Note that for an arbitrary c, $\alpha \in \sigma^{(c)}$, we then have that $< \alpha, E >$ is (σ, ω) – *reachable*. Applying lemma 8.14 thus gives us the (I, σ, ω) – *reachability* of $\bigcup_c \{< \alpha, E >: \alpha \in \sigma^{(c)}\}$. It thus follows that $(X_1, \sigma_1) \rightarrow^h (X_2, \sigma)$ for some history h, where $Init_{\rho'}((X_1, \sigma_1))$, $Final_{\rho'}((X_2, \sigma))$, and $\sigma_1, \omega \models p \downarrow z_{c_n}$. Now the validity of the formula $\{p \downarrow z_{c_n}\}\rho\{Q\}$ implies that of $\{p \downarrow z_{c_n}\}\rho'\{Q\}$ (note that $IVar(p, Q, \rho) \cap \bigcup_c H^c = \emptyset$, and that the updating of the history variables does not affect the flow of control of ρ), from which we conclude $\sigma, \omega \models Q$.

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

$$\vdash \{p' \downarrow z_{c_n}\}\rho'\{Q\}.$$

Applying the substitution rules (S1) and (S2), substituting every logical variable z_x by the corresponding instance variable x, substituting the instance variable *count* by

0, and every variable $x \in H_{d^*}^{c_n}$, $d \in C^+$, by nil, the empty sequence, then gives us, after an trivial application of the consequence rule,

$$\vdash \{p \downarrow z_{c_n}\}\rho'\{Q\}.$$

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

9 Expressibility

In this section we show that the assertions I, $Pre(R)$ and $Post(R)$, as defined in the section on the completeness of the proof system, can indeed be expressed in our assertion languages. We assume throughout this section the sets C and $IVar$ to be finite. We start with a description of the *coding* techniques we will use to express these assertions.

9.1 Coding techniques

We will have to use some coding mechanism to *arithmetize* the semantics of programs and assertions. However we will not go into the details of the construction of the mechanism we need. We will just assume such a coding mechanism to exist, and list some of its properties we shall make use of. For the details of the coding mechanism we refer to [TZ]. Let ρ' be the program defined in the section on completeness (see definition 8.6), relative to which the assertions I, $Pre(R)$ and $Post(R)$ are defined.

For an arbitrary statement S, variable x, the code of S and x will be denoted by $[S], [x] \in N$, respectively.

For an arbitrary d we assume given an injection $[]_d \in O_\perp^d \to N$ such that $[\perp]_d = 0$. For $d = \mathsf{Int}$ we assume $[]_d$ to be surjective too.

Furthermore we assume given for an arbitrary d an injection $[]_{d^*} \in O^{d^*} \to N$ such that for $\alpha =< \beta_1, \ldots, \beta_n > \in O^{d^*}$, with $d \in C$, we have $[\alpha]_{d^*} = [< [\beta_1]_d, \ldots, [\beta_n]_d >]_{\mathsf{Int}^*}$.

The code of an arbitrary global state σ and a global configuration (X, σ) will be denoted by $[\sigma], [(X, \sigma)] \in N$, respectively.

We assume the representability in the local assertion language of $[]_{\mathsf{Int}^*}$ and of the coding functions $[]_{\mathsf{Int}}$ and $[]_{\mathsf{Bool}}$. So, for example, we assume the existence of an assertion $p(z_1, z_2)$, where $z_1 \in LVar_{d^*}$, $z_2 \in LVar_d$, $d = \mathsf{Int}$, such that for an arbitrary ω, θ we have

$$\theta, \omega \models p(z_1, z_2) \text{ iff } [\omega(z_1)]_{d^*} = \omega(z_2).$$

As we will explain in the following subsection the coding function $[]_c$ is *not* representable in the local assertion language or in the global assertion language. For example there exists no local assertion $p(x, z)$, the type of the variable x being c and that of the variable z being Int, such that for every θ and ω:

$$\theta, \omega \models p(x, z) \text{ iff } [\theta(x)]_c = \omega(z).$$

However we will see that we do not need these functions to be representable in the assertion language.

Next we present a list of relations between natural numbers which we assume to be recursive, and thus definable in the local assertion language:

Definition 9.1
We define

- $Len_d(n) = m$ iff there exists a sequence $\alpha \in \mathbf{O}^{d^*}$ such that $[\alpha]_{d^*} = n$ and the length of α equals m.
- $Conc_d(n, m) = k$ iff there exist $\alpha, \beta \in O^{d^*}$ such that $n = [\alpha]_{d^*}$, $m = [\beta]_{d^*}$, and $k = [\alpha \circ \beta]_{d^*}$.
- $Elt_d(n, m) = k$ iff there exist $\alpha \in O^{d^*}, \beta \in O_\perp^{\mathsf{Int}}$ such that $[\alpha]_{d^*} = n$, $[\beta]_{\mathsf{Int}} = m$, and $[\alpha(\beta)]_d = k$.
- $Act_c(n, m)$ iff there exist a global state σ, $\alpha \in \sigma^{(c)}$ such that $m = [\sigma]$, $[\alpha]_c = n$.
- $Val_a^c(k, l, m) = n$ iff there exist a program variable x_a^c, a global state σ, $\alpha \in \sigma^{(c)}$ such that $[\sigma] = m$, $[\alpha]_c = k$, $[x_a^c] = l$, and $[\sigma(\alpha)(x_a^c)]_a = n$.
- $Trans(n, m)$ iff there exist global configurations (X_0, σ_0), (X, σ) such that $[(X_0, \sigma_0)] = n$, $[(X, \sigma)] = m$, $Init_{\rho'}((X_0, \sigma_0))$, and for some history h $(X_0, \sigma_0) \rightarrow^h (X, \sigma)$.
- $Norm_c(n, m)$ iff there exist a global configuration (X, σ), $\alpha \in \sigma^{(c)}$ such that $[\alpha]_c = n$, $[(X, \sigma)] = m$, and $X(\alpha)$ is normal.
- $Before_c(k, l) = m$ iff there exist statements R^c and S^c such that $[R^c] = k$, $[S^c] = l$, and $[Before(R^c, S^c)] = m$.
- $After_c(k, l) = m$ iff there exist statements R^c and S^c such that $[R^c] = k$, $[S^c] = l$, and $[After(R^c, S^c)] = m$.
- $Prog_c(n, m) = k$ iff there exist a global configuration (X, σ), $\alpha \in \sigma^{(c)}$ such that $[(X, \sigma)] = m$, $[\alpha]_c = n$, and $[X(\alpha)] = k$.
- $State(n) = m$ iff there exists a global configuration (X, σ) such that $[(X, \sigma)] = n$ and $[\sigma] = m$.

In the sequel we identify the relations defined above with their representations in the local assertion language.

Next we code the truth relation $\sigma, \omega \models p^c \downarrow z_c$, p^c a local assertion. We do so by translating a local expression l^c and a local assertion p^c to an *arithmetical* one, which

we denote by $l^c\lfloor u, v\rfloor$ and $p^c\lfloor u, v\rfloor$, where u, v are fresh logical integer variables. We call an expression (assertion) arithmetical if only logical variables ranging over (sequences of) integers occur in it. (So occurrences of instance variables are not allowed.) The idea is that the translated expression (assertion) "speaks" about a state in terms of the arithmetical structure of its code number. The variable u will be interpreted as the code of the current state, and the variable v as the code of the object with respect to which the expression (assertion) is evaluated. The translation runs as follows:

Definition 9.2
We define

- $z_d\lfloor u, v\rfloor = [z_d]_d$
- $x_a^c\lfloor u, v\rfloor = Val(v, [x_a^c], u)$
- $\mathsf{nil}\lfloor u, v\rfloor = 0$
- $\mathsf{self}\lfloor u, v\rfloor = v$
- $\underline{n}\lfloor u, v\rfloor = [\underline{n}]_{\mathsf{Int}}$
- $\mathsf{true}\lfloor u, v\rfloor = [\mathsf{true}]_{\mathsf{Bool}}$
- $\mathsf{false}\lfloor u, v\rfloor = [\mathsf{false}]_{\mathsf{Bool}}$
- $(e_1^c : e_2^c)\lfloor u, v\rfloor = Elt(e_1^c\lfloor u, v\rfloor, e_2^c\lfloor u, v\rfloor)$
- $< e^c > \lfloor u, v\rfloor = [< e^c\lfloor u, v\rfloor >]_{\mathsf{Int}^*}$
- $(e_1^c \circ e_2^c)\lfloor u, v\rfloor = Conc(e_1^c\lfloor u, v\rfloor, e_2^c\lfloor u, v\rfloor)$
- $|e^c|\lfloor u, v\rfloor = [Len(e^c\lfloor u, v\rfloor)]_{\mathsf{Int}}$
- $(e_1^c \doteq e_2^c)\lfloor u, v\rfloor = e_1^c\lfloor u, v\rfloor \doteq e_2^c\lfloor u, v\rfloor$
- $(\neg p^c)\lfloor u, v\rfloor = \neg(p^c\lfloor u, v\rfloor)$
- $(p_1^c \wedge p_2^c)\lfloor u, v\rfloor = p_1^c\lfloor u, v\rfloor \wedge p_2^c)\lfloor u, v\rfloor$
- $(\exists z_a p^c)\lfloor u, v\rfloor = \exists z_a(p^c\lfloor u, v\rfloor)$

Note that by definition of the local assertion language we have in the first clause above that $d = \mathsf{Int}, \mathsf{Bool}$. With respect to the last clause we have $a = d, d^*$, $d = \mathsf{Int}, \mathsf{Bool}$. We have the following semantical property of this translation:

Lemma 9.3
For an arbitrary $\sigma, \alpha \in \sigma^{(c)}$, ω such that $OK(\sigma, \omega)$ we have

$$[\mathcal{L}[\![l_a^c \downarrow z_c]\!](\omega\{\alpha/z_c\})(\sigma)]_a = \mathcal{L}[\![l_a^c\lfloor u, v\rfloor]\!](\omega\ \{[\sigma], [\alpha]_c/u, v\})(\sigma),$$

and

$$\mathcal{A}[\![p^c \downarrow z_c]\!](\omega\{\alpha/z_c\})(\sigma) = \mathcal{A}[\![p^c\lfloor u, v\rfloor]\!](\omega\ \{[\sigma], [\alpha]_c/u, v\})(\sigma).$$

This lemma states that taking the code of the object denoted by the expression l yields the same result as when we evaluate the translation of l into an arithmetical expression, interpreting the variables u and v as the current object and the current state, respectively. With respect to assertions this lemma expresses a corresponding proposition. Note that in fact the state with respect to which we evaluate the translated expression (assertion) is irrelevant because there do not occur instance variables in the translated expression (assertion).

Proof
Straightforward induction on the structure of l_a^c and p^c. □

9.2 Object-space isomorphisms

To proceed we next introduce the notion of an object-space isomorphism, an *osi* for short.

Definition 9.4
An *object-space isomorphism* (*osi*) is a family of functions $f = \langle f^d \rangle_{d \in C^+}$, where $f^d \in \mathbf{O}_\perp^d \to \mathbf{O}_\perp^d$ is a bijection, $f^d(\perp) = \perp$ and f^d, for $d = \mathsf{Int}, \mathsf{Bool}$, is the identity mapping.

Given an *osi* we define the isomorphic image of a local state as follows:

Definition 9.5
For an arbitrary local state θ^c, *osi* f we define the local state $f(\theta^c)$ as follows:

- $f(\theta^c)_{(1)} = f^c(\theta^c_{(1)})$.
- For an arbitrary variable $x \in IVar_d^c$ we have $f(\theta^c)_{(2)}(x) = f^d(\theta^c_{(2)}(x))$.

Given an *osi* we define next the isomorphic image of a global state.

Definition 9.6
For an arbitrary global state σ, *osi* f we define the global state $f(\sigma)$ as follows:

- For an arbitrary c we have $f(\sigma)^{(c)} = f^c(\sigma^{(c)})$.
- For an arbitrary c, $\alpha \in \sigma^{(c)}$ we have $f(\sigma)(\alpha) = f^d(\sigma(g^c(\alpha)))$, where $g = f^{-1}$ and f^{-1} denotes the *inverse* of f: $f^{-1} =< (f^d)^{-1} >_d$.

Finally, given an *osi* we define the isomorphic image of a history.

Definition 9.7
Let h be a history and f be an *osi* we define $f(h)$ as follows:

$$\begin{aligned} f(h) &= \epsilon \\ &\text{if } h = \epsilon \\ &= f(h_1) \circ < f^{d_i}(\alpha), f^{d_j}(\beta) > \\ &\text{if } h = h_1 \circ < \alpha, \beta >,\ \alpha \in O_\perp^{d_i}, \beta \in O_\perp^{d_j} \\ &= f(h_1) \circ < f^{d_i}(\alpha), f^{d_j}(\beta), f^{d_k}(\gamma) > \\ &\text{if } h = h_1 \circ < \alpha, \beta, \gamma >,\ \alpha \in O_\perp^{d_i}, \beta \in O_\perp^{d_j}, \gamma \in O_\perp^{d_k} \end{aligned}$$

The following theorem states that isomorphic states cannot be distinguished by the assertion languages.

Theorem 9.8
For an arbitrary *osi* f, local state θ^c, logical environment ω and local expression l_a^c, local assertion p^c we have

- $f^a(\mathcal{L}[\![l_a^c]\!](\omega)(\theta^c)) = \mathcal{L}[\![l_a^c]\!](f(\omega))(f(\theta^c))$.
- $\mathcal{A}[\![p^c]\!](\omega)(\theta^c) = \mathcal{A}[\![p^c]\!](f(\omega))(f(\theta^c))$.

where $f(\omega)(z) = f^a(\omega(z))$, $z \in LVar_a$. Furthermore for an arbitrary global state σ, logical environment ω such that $OK(\omega, \sigma)$, global expression g_a and global assertion P we have

- $f^a(\mathcal{G}[\![g_a]\!](\omega)(\sigma)) = \mathcal{G}[\![g_a]\!](f(\omega))(f(\sigma))$.
- $\mathcal{A}[\![P]\!](\omega)(\sigma) = \mathcal{A}[\![P]\!](f(\omega))(f(\sigma))$.

Proof
Induction on the structure of l_a^c, p^c, g_a and P respectively. □

The following theorem states that our transition system is closed under isomorphic images.

Theorem 9.9
For two arbitrary global configurations $(X_0, \sigma_0), (X_1, \sigma_1)$, *osi* f and history h we have if

$$(X_0, \sigma_0) \rightarrow^h (X_1, \sigma_1)$$

then

$$(f(X_0), f(\sigma_0)) \rightarrow^{f(h)} (f(X_1), f(\sigma_1)),$$

where $f(X)(\alpha) = X(f^{-1}(\alpha))$.

Proof
It is not difficult to see that it suffices to prove the corresponding proposition for the local transition system: If

$$(S_1, \theta_1) \rightarrow^r (S_2, \theta_2)$$

then

$$(S_1, f(\theta_1)) \rightarrow^{f(r)} (S_2, f(\theta_2)).$$

This is proved by a straightforward case analysis, making use of theorem 9.8. □

9.3 Coding the global invariant

Now we are ready to show how to express the global invariant as defined in the section on completeness. One of the main difficulties with expressing the global invariant is caused by the fact that we cannot express directly for example the relation $\sigma^{(c)} = \sigma'^{(c)}$, σ and σ' arbitrary, in terms of the code for σ'. More precisely, there exists no assertion $P(z)$ such that for every state σ and environment ω, with $OK(\sigma, \omega)$ and $\omega(z) = [\sigma']$, for some state σ', we have

$$\sigma, \omega \models P(z) \text{ iff } \sigma^{(c)} = \sigma'^{(c)}.$$

This is an immediate consequence of theorem 9.8: We have

$$\sigma, \omega \models P(z) \text{ iff } f(\sigma), f(\omega) \models P(z),$$

for an arbitrary *osi* f. But it is not the case that for every *osi* f we have $f(\sigma)^{(c)} = \sigma'^{(c)}$ (note that $f(\omega)(z) = [\sigma']$). However using theorem 9.9 we will see that we do not need to express this coding relation directly.

To proceed we first introduce some new logical variables.

Definition 9.10
For an arbitrary c let $bij_c \in LVar_{c^*}$. Furthermore let $\overline{bij}$ denote a sequence of these variables.

Given a state σ and an *osi* f these variables $\overline{bij}$ will be used to code the restriction of f to the existing objects of σ in the following way:

Definition 9.11
For an arbitrary state σ, environment ω, with $OK(\omega, \sigma)$, and *osi* f we define $Code(\omega, \sigma, f)$ iff for an arbitrary c, $\alpha \in \sigma^{(c)}$ we have $\omega(bij_c)([f^c(\alpha)]_c) = \alpha$.

So every existing object of an arbitrary class c is stored in the sequence denoted by bij_c at a position which equals the code of its image under f.

Now we are ready to define an assertion $Bij(H^c, z_c, n, m)$ which expresses for two arbitrary states σ, σ_1, $\alpha \in \sigma^{(c)}$, and *osi* f the relation $f(h_{\sigma,\alpha}) = h_{\sigma_1, f^c(\alpha)}$. The idea is that the logical variable z_c is interpreted as the object α, the integer variable n as the code of $f^c(\alpha)$, and the integer variable m as the code of σ_1. In the formulation of the assertion $Bij(H^c, z_c, n, m)$ we will assume that the variables of $\overline{bij}$ code the restriction of f to the existing objects of σ.

Definition 9.12
Let n and m be two logical integer variables. We define

$$Bij(H^c, z_c, n, m) = \bigwedge_{x \in H^c} P(z_c.x, n, m),$$

where
$P(z_c.x, n, m) =$

$$\begin{aligned} \forall 1 \leq j \leq |z_c.x| \quad (\quad & z_c.x : j \doteq \mathsf{nil} \rightarrow Elt(\, Val(n, [x], m), [j]_{\mathsf{Int}}) \doteq 0 \wedge \\ & \forall k (z_c.x : j \doteq bij_{c_i} : k \neq \mathsf{nil} \rightarrow Elt(\, Val(n, [x], m), [j]_{\mathsf{Int}}) \doteq k \wedge \\ & |x| \doteq Len(\, Val(n, [x], m)) \\ ) & \end{aligned}$$

Here we assume the type of x to be c_i^*. If on the other hand the type of x is Int^* we have
$P(z_c.x, n, m) =$

$$\begin{aligned} \forall 1 \leq j \leq |z_c.x| \quad (\quad & z_c.x : j \neq \mathsf{nil} \rightarrow Elt(\, Val(n, [x], m), [j]_{\mathsf{Int}}) \doteq [z_c.x : j]_{\mathsf{Int}} \wedge \\ & z_c.x : j \doteq \mathsf{nil} \rightarrow Elt(\, Val(n, [x], m), [j]_{\mathsf{Int}}) \doteq 0 \wedge \\ & |z_c.x| \doteq Len(\, Val(n, [x], m)) \\ ) & \end{aligned}$$

The meaning of the assertion $P(z_c.x, n, m)$, with c_i^* the type of x, can be explained as follows: Suppose that the variable n is interpreted as the code of the object $\alpha = \omega(z_c)$ and the variable m as the code of the state σ such that $\alpha \in \sigma^{(c)}$. The assertion $P(z_c.x, n, m)$ then states that every element of the sequence denoted by $\sigma(\alpha)(x)$ occurs at a position in the sequence denoted by bij_{c_i} which equals its code. Formally we have the following lemma about this semantical property of the assertion defined above.

Lemma 9.13
Let $Code(\omega, \sigma, f)$, $\alpha \in \sigma^{(c)}$ and σ' such that $f(\sigma) \approx \sigma'$. (Here $\sigma_1 \approx \sigma_2$ holds iff for every c we have $\sigma_1^{(c)} = \sigma_2^{(c)}$.) We have

$$\sigma, \omega\{\alpha/z_c, [f^c(\alpha)]_c/n, [\sigma']/m\} \models Bij(H^c, z_c, n, m) \text{ iff } f(h_{\sigma,\alpha}) = h_{\sigma', f(\alpha)}.$$

Proof
It is not difficult to prove that

$$f(h_{\sigma,\alpha}) = h_{\sigma', f^c(\alpha)} \text{ iff}$$
$$\text{for every } d, x \in H^c_{d*} : \ f(\sigma(\alpha)(x)) = \sigma'(f^c(\alpha))(x).$$

Furthermore from $Code(\omega, \sigma, f)$ and $f(\sigma) \approx \sigma'$ it follows by a straightforward but slightly tedious argument that

$$f(\sigma(\alpha)(x)) = \sigma'(f^c(\alpha))(x) \text{ iff } \sigma, \omega\{\alpha/z_c, [f^c(\alpha)]_c/n, [\sigma']/m\} \models P(z_c.x, n, m).$$

□

Now we can express the global invariant as follows.

Lemma 9.14
We define

$$\begin{aligned}
&I = \\
&\exists n, m, k \quad (\quad Trans(n, m) \wedge p'\lfloor u, v \rfloor [State(n), k/u, v] \wedge Act_{c_n}(k, State(n)) \wedge \\
&\qquad \textstyle\bigwedge_c \forall i (Act_c(i, m) \rightarrow Norm_c(i, m)) \wedge \\
&\qquad \exists \overline{bij} \quad (\quad \textstyle\bigwedge_c \forall z_c \exists! 1 \leq j \leq |bij_c| (bij_c : j = z_c) \wedge \\
&\qquad\qquad \textstyle\bigwedge_c \forall i (Act(i, State(m)) \leftrightarrow bij_c : i \neq \mathsf{nil}) \wedge \\
&\qquad\qquad \textstyle\bigwedge_c \forall 1 \leq j \leq |bij_c| Bij(H^c, bij_c : j, j, State(m)) \\
&\qquad\qquad) \\
&\qquad)
\end{aligned}$$

(Here "$\exists!$" is interpreted as "there exists an unique", which can be expressed by the usual quantifiers.) Note that the quantification $\exists n, m$ corresponds to the phrase "there exist configurations $(X_0, \sigma_0), (X_1, \sigma_1)$". The assertion $Trans(n, m)$ then corresponds to "there exists an history h such that $(X_0, \sigma_0) \rightarrow^h (X_1, \sigma_1)$ and $Init_{\rho'}((X_0, \sigma_0))$". The assertion $\exists k(p'\lfloor u, v\rfloor[State(n), k/u, v] \wedge Act_{c_n}(k, State(n)))$ translates the phrase "for $\alpha \in \sigma_0^{(c_n)}$ we have $\langle \alpha, \sigma_0(\alpha)\rangle, \omega \models p'$". The assertion $\forall i(Act_c(i, m) \rightarrow Norm_c(i, m))$ corresponds to "for an arbitrary c and $\alpha \in \sigma_1^{(c)}$ we have that $X_1(\alpha)$ is normal". Finally, the assertion

$$\begin{array}{lll} \exists \overline{bij} & (& \bigwedge_c \forall z_c \exists! 1 \leq j \leq |bij_c| bij_c : j = z_c \wedge \\ & & \bigwedge_c \forall i Act(i, State(m)) \leftrightarrow bij_c : i \neq \mathsf{nil} \wedge \\ & & \bigwedge_c \forall 1 \leq j \leq |bij_c| Bij(H^c, bij_c : j, j, State(m)) \\ &) & \end{array}$$

will correspond to the phrase "for an arbitrary c we have $\sigma^{(c)} = \sigma_1^{(c)}$ and for every $\alpha \in \sigma^{(c)}$ we have $h_{\sigma,\alpha} = h_{\sigma_1,\alpha}$".

Proof

Let $\sigma, \omega \models I$. So there exist $\gamma_1, \gamma_2, \gamma_3 \in \mathbf{N}$ such that $\sigma, \omega' \models I$, where $\omega' = \omega\{\gamma_1, \gamma_2, \gamma_3/n, m, k\}$.

By $\sigma, \omega' \models Trans(n, m)$ we have $Init_{\rho'}((X_0, \sigma_0))$ and $(X_0, \sigma_0) \rightarrow^h (X_1, \sigma_1)$, for some history h, where $[(X_0, \sigma_0)] = \gamma_1$ and $[(X_1, \sigma_1)] = \gamma_2$.

From $\sigma, \omega' \models p'\lfloor u, v\rfloor[State(n), k/u, v] \wedge Act_{c_n}(k, State(n))$ by an application of lemma 9.3 it follows that $\langle \alpha, \sigma_0(\alpha)\rangle, \omega \models p'$, where $\alpha \in \sigma_0^{(c_n)}$.

By $\sigma, \omega' \models \bigwedge_c \forall i(Act_c(i, m) \rightarrow Norm_c(i, m))$ we have that $X_1(\alpha)$ is normal for every $\alpha \in \sigma_1^{(c)}$, c arbitrary.

Now let $\alpha_i \in \mathbf{O}^{c_i^*}$ such that for $\omega'' = \omega\{\alpha_i/bij_{c_i}\}$ we have

1. $\sigma, \omega'' \models \bigwedge_c \forall z_c \exists! 1 \leq j \leq |bij_c|(bij_c : j = z_c)$
2. $\sigma, \omega'' \models \bigwedge_c \forall i(Act(i, State(m)) \leftrightarrow bij_c : i \neq \mathsf{nil})$
3. $\sigma, \omega'' \models \bigwedge_c \forall 1 \leq j \leq |bij_c| Bij(H^c, bij_c : j, j, State(m))$

Let f be an *osi* such that for an arbitrary c_i, $\alpha \in \sigma^{(c_i)}$ we have $\alpha_i([\beta]_{c_i}) = \alpha$ iff $f^{c_i}(\alpha) = \beta$. Note that this is well-defined because of 1 and 2.

By 3 and an application of 9.13 we have $f(h_{\sigma,\alpha}) = h_{\sigma_1,f^c(\alpha)}$, for an arbitrary c and $\alpha \in \sigma^{(c)}$. (Note that we have $Code(\omega, \sigma, f)$ and $f(\sigma) \approx \sigma_1$.)

Finally, applying theorem 9.9 gives us

$$(f^{-1}(X_0), f^{-1}(\sigma_0)) \rightarrow^{f^{-1}(h)} (f^{-1}(X_1), f^{-1}(\sigma_1)).$$

(Note that we have $Init_\rho((f^{-1}(X_0), f^{-1}(\sigma_0)))$ and that $f^{-1}(X_1)(\alpha))$ is normal, $\alpha \in f^{-1}(\sigma_1)^{(c)}$, c arbitrary.)

It then follows that for an arbitrary c, $\alpha \in \sigma^{(c)}$: $h_{\sigma,\alpha} = f^{-1}(h_{\sigma_1,f^c(\alpha)}) = h_{f^{-1}(\sigma_1),\alpha}$. (The second equality is an instance of the general fact that for an arbitrary σ, $\alpha \in \sigma^{(c)}$, and *osi* f we have $f(h_{\sigma,\alpha}) = h_{f(\sigma),f^c(\alpha)}$.)

The other way around: Suppose for the configurations $(X_0, \sigma_0), (X_1, \sigma_1)$ we have

- For some history h : $(X_0, \sigma_0) \rightarrow^h (X_1, \sigma_1)$, where $Init_{\rho'}((X_0, \sigma_0 >)$, and for $\alpha \in \sigma_0^{(c_n)}$ we have $\langle \alpha, \sigma_0(\alpha)), \omega \models p'^{c_n}$.
- For an arbitrary c and $\alpha \in \sigma_1^{(c)}$ we have that $X_1(\alpha)$ is normal.
- For an arbitrary c we have $\sigma^{(c)} = \sigma_1^{(c)}$, and for every $\alpha \in \sigma^{(c)}$ we have $h_{\sigma,\alpha} = h_{\sigma_1,\alpha}$.

Let $\gamma_1 = [(X_0, \sigma_0)], \gamma_2 = [(X_1, \sigma_1)]$, and $\gamma_3 = [\alpha]_c$, where $\alpha \in \sigma_0^{c_n}$. Furthermore let $\omega' = \omega\{\gamma_1, \gamma_2, \gamma_3/n, m, k\}$.

By $(X_0, \sigma_0) \rightarrow^h (X_1, \sigma_1)$ and $Init_{\rho'}((X_0, \sigma_0))$ we have $\sigma, \omega' \models Trans(n, m)$.

By $\langle \alpha, \sigma_0(\alpha)), \omega \models p'^{c_n}$ and lemma 9.3 we infer $\sigma, \omega' \models p'\lfloor u, v \rfloor[State(n), k/u, v]$. Furthermore from $\alpha \in \sigma_0^{(c_n)}$ it follows that $\sigma, \omega' \models Act(k, State(n))$.

From the second clause above it immediately follows that $\sigma, \omega' \models \forall i(Act_c(i, m) \rightarrow Norm_c(i, m))$.

Finally, let α_i be a sequence of objects of class c_i such that for $\beta \in \sigma^{(c_i)}$ we have $\alpha_i([\beta]_{c_i}) = \beta$. So α_i stores every existing object of $\sigma^{(c_i)}$ at a position which equals its code number. Furthermore let $\omega'' = \omega'\{\alpha_i/bij_{c_i}\}_i$. It then follows that

$$\sigma, \omega'' \models \bigwedge_c \forall z_c \exists! 1 \leq j \leq |bij_c|(bij_c : j = z_c).$$

From $\sigma^{(c)} = \sigma_1^{(c)}$ we infer that

$$\sigma, \omega'' \models \bigwedge_c \forall i(Act(i, State(m)) \leftrightarrow bij_c : i \neq \text{nil}).$$

As $h_{\sigma,\alpha} = h_{\sigma_1,\alpha}$, for $\alpha \in \sigma^{(c)}$, c arbitrary, we have by lemma 9.13 that

$$\sigma, \omega'' \models \bigwedge_c \forall 1 \leq j \leq |bij_c| Bij(H^c, bij_c : j, j, State(m))$$

(take for the *osi* f the family of identity mappings). □

9.4 Expressing preconditions and postconditions

We next show how to express $Pre(S)$ and $Post(S)$ as defined in the section on completeness. First we define some local assertions which partly express an isomorphism between a local state θ and some local state $\langle \alpha, \sigma(\alpha) \rangle$, for some σ and α, in terms of the code numbers for α and σ.

Definition 9.15
We define

- $LCode^c_{c_i}(x^c_{c_i}, y^c_{c_i}, n, m) =$
  $x^c_{c_i} \doteq y^c_{c_i} \leftrightarrow Val(n, [x^c_{c_i}], m) \doteq Val(n, [y^c_{c_i}], m)$

- $LCode^c_c(x^c_c, \mathsf{self}, n, m) =$
  $x^c_c \doteq \mathsf{self} \leftrightarrow Val(n, [x^c_c], m) \doteq n$

- $LCode^c_{\mathsf{Int}}(x^c_{\mathsf{Int}}, n, m) =$
  $x^c_{\mathsf{Int}} \doteq \mathsf{nil} \rightarrow Val(n, [x^c_{\mathsf{Int}}], m) \doteq 0 \wedge$
  $x^c_{\mathsf{Int}} \not\doteq \mathsf{nil} \rightarrow Val(n, [x^c_{\mathsf{Int}}], m) \doteq [x^c_{\mathsf{Int}}]_{\mathsf{Int}}$

- $LCode^c_{\mathsf{Int}^*}(x^c_{\mathsf{Int}^*}, n, m) =$
  $\forall 1 \leq k \leq |x^c_{\mathsf{Int}^*}|$
  $(x^c_{\mathsf{Int}^*} : k \doteq \mathsf{nil} \rightarrow Elt(\, Val(n, [x^c_{\mathsf{Int}^*}], m), [k]_{\mathsf{Int}}) \doteq 0 \wedge$
  $x^c_{\mathsf{Int}^*} : k \not\doteq \mathsf{nil} \rightarrow Elt(\, Val(n, [x^c_{\mathsf{Int}^*}], m), [k]_{\mathsf{Int}}) \doteq [x^c_{\mathsf{Int}^*} : k]_{\mathsf{Int}} \wedge$
  $|x^c_{\mathsf{Int}^*}| \doteq Len(\, Val(n, [x^c_{\mathsf{Int}^*}], m)))$

- $LCode^c_{c^*}(x^c_{c^*}, \mathsf{self}, n, m) =$
  $\forall 1 \leq k \leq |x^c_{c^*}|$
  $(x^c_{c^*} : k \doteq \mathsf{self} \leftrightarrow Elt(\, Val(n, [x^c_{c^*}], m), [k]_{\mathsf{Int}}) \doteq n)$

- $LCode^c_{c_i, c_i^*}(x^c_{c_i}, y^c_{c_i^*}, n, m) =$
  $\forall 1 \leq k \leq |y^c_{c_i^*}|$
  $(y^c_{c_i^*} : k \doteq x^c_{c_i} \leftrightarrow Elt(\, Val(n, [y^c_{c_i^*}], m), [k]_{\mathsf{Int}}) \doteq Val(n, [x^c_{c_i}], m))$

- $LCode^c_{c_i^*}(x^c_{c_i^*}, y^c_{c_i^*}, n, m) =$
  $\forall 1 \leq k \leq |x^c_{c_i^*}| \forall 1 \leq l \leq |y^c_{c_i^*}|$
  $(x^c_{c_i^*} : k \doteq y^c_{c_i^*} : l \leftrightarrow Elt(Val(n, [x^c_{c_i^*}], m), [k]_{\mathsf{Int}}) \doteq Elt(Val(n, [y^c_{c_i^*}], m), [l]_{\mathsf{Int}})) \wedge$
  $|x^c_{c_i^*}| \doteq Len(Val(n, [x^c_{c_i^*}], m)) \wedge |y^c_{c_i^*}| \doteq Len(Val(n, [y^c_{c_i^*}], m))$

To explain the interpretation of these assertions consider the assertion $LCode(x, y, n, m)$, where x and y are of a type $c \in C$. The idea is that the variable n is interpreted as the code of some object α and the variable m as the code of some state σ such that α exists in σ. The assertion $LCode(x, y, n, m)$ then states that in the current local state the variables x and y denote the same object iff they denote the same object in $\sigma(\alpha)$, i.e., $\sigma(\alpha)(x) = \sigma(\alpha)(y)$.

Given the above definition we now define a local assertion which describes an isomorphism between a local state θ and a local state $\langle \alpha, \sigma(\alpha) \rangle$ in terms of the codes for α and σ.

Definition 9.16
Let n and m be two distinct logical integer variables. We define $LCode(n, m)$ to be the following local assertion:

$$
\begin{array}{l}
Act_c(n, m) \wedge \\
\bigwedge_{c'} \bigwedge_{x,y \in IVar^c_{c'}} LCode_{c'}(x, y, n, m) \wedge \bigwedge_{x \in IVar^c_c} LCode_c(x, \mathsf{self}, n, m) \wedge \\
\bigwedge_{x \in IVar^c_{\mathsf{Int}}} LCode_{\mathsf{Int}}(x, n, m) \wedge \bigwedge_{x \in IVar^c_{\mathsf{Int}^*}} LCode_{\mathsf{Int}}(x, n, m) \wedge \\
\bigwedge_{c'} \bigwedge_{x,y \in IVar^c_{c'^*}} LCode_{c'^*}(x, y, n, m) \wedge \bigwedge_{x \in IVar^c_{c^*}} LCode_{c^*}(x, \mathsf{self}, n, m) \wedge \\
\bigwedge_{c'} \bigwedge_{x \in IVar^c_{c'}, y \in IVar^c_{c'^*}} LCode_{c', c'^*}(x, y, n, m)
\end{array}
$$

We have the following lemma about the meaning of the local assertion $LCode(n, m)$.

Lemma 9.17
For $\sigma, \alpha \in \sigma^{(c)}$, ω and θ^c we have

$$\theta, \omega\{[\alpha]_c, [\sigma]/n, m\} \models LCode^c(n, m) \text{ iff}$$
$$\text{there exists an } osi\ f \text{ such that } f(\theta^c) = \langle \alpha, \sigma(\alpha) \rangle.$$

Proof
Let f be such that $f(\theta^c) = \langle \alpha, \sigma(\alpha) \rangle$. Now let x and y be two variables of type, say, c'. The other cases are treated similarly. We have

$$\theta \models x \doteq y \text{ iff } \langle \alpha, \sigma(\alpha) \rangle \models x \doteq y \text{ iff } Val([\alpha]_c, [x], [\sigma]) = Val([\alpha]_c, [y], [\sigma]).$$

Note that the first equivalence follows from theorem 9.8. On the other hand using $\theta^c, \omega\{[\alpha]_c, [\sigma]/n, m\} \models LCode^c(n, m)$ it is not difficult to construct an *osi* f such that for $\beta \in \mathbf{O}^{c'}$, c' arbitrary, we have: If β is referred to by some variable in θ we put $f^{c'}(\beta) = \gamma$, where γ is the object referred to by the same variable in $\langle \alpha, \sigma(\alpha) \rangle$. It then follows from the construction that $f(\theta) = \langle \alpha, \sigma(\alpha) \rangle$. □

Finally we are ready to express $Pre(R)$ and $Post(R)$ in our assertion language.

Lemma 9.18
Let n, m, k be three distinct logical integer variables. We define

$$\begin{array}{ll} Pre(R) & = \\ \exists n, m, k & (Trans(n, m) \wedge p'\lfloor u, v \rfloor [State(n), k/u, v] \wedge Act_{c_n}(k, State(n)) \wedge \\ & \exists i(LCode(i, State(m)) \wedge Prog_c(i, m) = Before([R], [S'^{(c)}])) \\ &) \end{array}$$

$Post(R)$ is defined analogously.

The assertion $\exists i(LCode(i, State(m)) \wedge Prog_c(i, m) = Before([R], [S'^{(c)}]))$ will correspond to the phrase "for $\alpha \in \sigma_1^{(c_i)}$ we have $\langle \alpha, \sigma_1(\alpha) \rangle = \theta^{c_i}$ and $X_1(\alpha) = Before(R, S'^{c_i})$.

Proof
Let $\theta, \omega \models Pre(R)$. So there exist $\gamma_1, \gamma_2, \gamma_3$ such that $\theta, \omega' \models Pre(R)$, where $\omega' = \omega\{\gamma_1, \gamma_2, \gamma_3/n, m, k\}$.

By $\theta, \omega' \models Trans(n, m)$ we have that there exist configurations $(X_0, \sigma_0), (X_1, \sigma_1)$ such that $(X_0, \sigma_0) \rightarrow^h (X_1, \sigma_1)$ for some history h, $Init_{\rho'}((X_0, \sigma_0))$, and $[(X_0, \sigma_0)] = \gamma_1$, $[(X_1, \sigma_1)] = \gamma_2$.

From $\theta, \omega' \models p'\lfloor u, v \rfloor [State(n), k/u, v] \wedge Act_{c_n}(k, State(n))$ by an application of lemma 9.3 it follows that $\langle \alpha, \sigma_0(\alpha) \rangle, \omega \models p'$, where $\alpha \in \sigma_0^{(c_n)}$.

Next let γ and $\omega'' = \omega'\{\gamma/i\}$ be such that $\theta, \omega'' \models LCode(i, State(m)) \wedge Prog_c(i, m) = Before([R], [S'^{(c)}])$. It then follows, by an application of the previous lemma, that for $\alpha \in \sigma_1^{(c)}$ such that $[\alpha] = \gamma$ we have $X_1(\alpha) = Before(R, S'^{(c)})$ and $f(\theta) = \langle \alpha, \sigma_1(\alpha) \rangle$ for some *osi* f.

Finally, an application of theorem 9.9 gives us

$$(f^{-1}(X_0), f^{-1}(\sigma_0)) \rightarrow^{f^{-1}(h)} (f^{-1}(X_1), f^{-1}(\sigma_1)).$$

Note that we now have $f^{-1}(X_1)(\theta(1)) = X_1(f^c(\theta(1))) = X_1(\alpha) = Before(R, S'^{(c)})$ and $f^{-1}(\sigma_1)(\theta(1)) = f^{-1}(\sigma_1(f^c(\theta(1)))) = f^{-1}(\sigma_1(\alpha)) = \theta(2)$.

The other way around we invite the reader to check. □

10 Conclusion

We have developed a proof system for the partial correctness of programs of a parallel language with dynamic process creation. The basic ingredients for dealing with parallelism in this proof system are the same as in a proof system for CSP [AFR], but they have been enhanced considerably to deal with the present, much more powerful programming language. One of the main problems we solved is how to reason about the dynamically evolving pointer structures that can arise during the execution of a program. A previous proof system [Bo] did this by considering pointers simply as numbers, which is of course not very abstract. Our present proof system allows reasoning at an abstraction level which is as high as that of the programming language, using a technique developed for a sequential language [Am1].

We have proved that the system is sound and complete. Again, the techniques developed for CSP [Ap2] could be used only after drastic modifications. Special care was necessary to be able to represent, in a finite number of variables, complete computations involving an unbounded number of objects. Also the programming language had to be extended to allow instance variables of sequence types. This extension, however, does not seem to be necessary in concrete proofs.

We have already mentioned that our language and our proof techniques can be considered as very powerful extensions to CSP [Ho2, AFR, Ap2]. A different kind of extension, where processes can split themselves recursively into subprocesses, is dealt with in [ZREB], for example. Here, the reasoning is not only based on *states*, as in our case, but also on *traces* (communication histories). The limitation of the possible process interconnection structures to trees allows a nice form of compositionality. In [Mel], a language similar to ours is tackled with trace-based reasoning. The problem of dynamic pointer structures is not dealt with explicitly.

We do not have the illusion that the proof system presented in this paper is suitable for proving 'practical' programs correct. One of the problems is that too much information is centralized in the global invariant. This seems unavoidable with the completely dynamic process structures allowed by our language. We would not like to dispense with this flexibility altogether, but we hope that by a judicious combination of our techniques with the compositional techniques developed elsewhere, future systems will allow us to have the best of both worlds.

Acknowledgements

We thank Jaco de Bakker, Arie de Bruin, Joost Kok, John-Jules Meyer, Jan Rutten and Erik de Vink, members of the Amsterdam Concurrency Group, for participating in discussions of preliminary versions of the proof system presented in this paper.

References

[Am1] P.H.M. America: *A Proof Theory for a Sequential Version of POOL.* ESPRIT project 415A, Doc. No. 188, Philips Research Laboratories, Eindhoven, the Netherlands, October 1986.

[Am2] P.H.M. America: *Issues in the Design of a Parallel Object-Oriented Language.* ESPRIT project 415A, Doc. No. 452, Philips Research Laboratories, Eindhoven, the Netherlands, November 1988. To appear in Formal Aspects of Computing.

[Am3] P.H.M America: *A Behavioural Approach to Subtyping in Object-Oriented Programming Languages.* Workshop on Inheritance Hierarchies in Knowledge Representation and Programming Languages, Viareggio, Italy, February 6–8, 1989 Also appeared in Philips Journal of Research, Vol. 44, No. 2/3, July 1989, pp. 365–383.

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

[Ap1] K.R. Apt: *Ten years of Hoare logic: a survey — part I.* ACM Transactions on Programming Languages and Systems, Vol. 3, No. 4, October 1981, pp. 431–483.

[Ap2] K.R. Apt: *Formal justification of a proof system for Communicating Sequential Processes.* Journal of the ACM, Vol. 30, No. 1, January 1983, pp. 197–216.

[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 rule for process creation.* M. Wirsing (ed.): *Formal Description of Programming Concepts.* Proceedings of the third IFIP WG 2.2 working conference, Gl. Avernæs, Ebberup, Denmark, August 25–28, 1986, North-Holland.

[GR] A. Goldberg and D. Robson: *Smalltalk-80, The Language and its Implementation.* Addison-Wesley, 1984.

[Ho1] C.A.R. Hoare: *An axiomatic basis for computer programming.* Communications of the ACM, Vol. 12, No. 10, 1969, pp. 567–580,583.

[Ho2] C.A.R. Hoare: *Communicating Sequential Processes.* Communications of the ACM, Vol. 21, No. 8, 1978, pp. 666–677.

[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, Springer LNCS 224, 1986, pp. 343–395.

[Mel] S. Meldal: *Axiomatic Semantics of Access Type Tasks in Ada.* Report No. 100, Institute of Informatics, University of Oslo, Norway, May 1986. To appear in Distributed Computing.

[Mey] B. Meyer: *Object-Oriented Software Construction.* Prentice-Hall, 1988.

[TZ] J.V. Tucker, J.I. Zucker: *Program Correctness over Abstract Data Types, with Error-State Semantics,* CWI Monographs 6, North-Holland, 1988.

[ZREB] J. Zwiers, W.P. de Roever, P. van Emde Boas: *Compositionality and concurrent networks: soundness and completeness of a proof system.* In Proceedings of the 12th ICALP, Nafplion, Greece, July 15–19, 1985, Springer LNCS 194, pp. 509–519.

A Index of notation

A.1 Sets and their typical elements

Typ. elt.	Set	Description	Where defined
i, j, k, n	$\mathbf{Z}$	integers	-
c	C	class names	above definition 2.1
d	C^+	data types	above definition 2.1
x_d^c	$IVar_d^c$	instance variables of type d in class c	above definition 2.1
e_d^c	Exp_d^c	expressions of type d in class c	definition 2.1
S^c	$Stat^c$	statements in class c	definition 2.2
ρ	$Prog$	programs	definition 2.3
l_d^c	$LExp_d^c$	local expressions of type d in class c	definition 3.1
p^c	$LAss^c$	local assertions in class c	definition 3.2
d^*	C^*	sequence types	above definition 3.3
a	$C^\dagger$	data and sequence types	above definition 3.3
z_a	$LogVar_a$	logical variables of type a	above definition 3.3
g_a	$GExp_a$	global expressions of type a	definition 3.3
P	$GAss$	global assertions	definition 3.4

A.2 Syntactic transformations

Notation	Where defined
$l_d^c \downarrow g_c$	definition 3.5
$p^c \downarrow g_c$	definition 3.5
$[e/x]$	standard
$[g/z.x]$	definition 4.3
$[\mathsf{new}/z]$	definitions 4.5 and 4.6
$[z_{\mathsf{Bool}^*}, z_c/z_{c^*}]$	definition 4.7