1991

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

A proof theory for a sequential version of POOL

Computer Science/Department of Software Technology Report CS-R9118 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 a Sequential Version of POOL

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 theory for partial correctness of programs written in a sequential version of the parallel object-oriented language POOL. The systems described by this language give rise to dynamically evolving process structures. One of the main objectives of the proof theory is to formalize reasoning about such structures at an abstraction level at least as high as that of the programming language. We show that the proof system is sound and (relative) complete.

1980 Mathematics Subject Classification: 70A05.

CR Categories: F.3.1.

Key Words and Phrases: proof theory, pre- and post-conditions, process creation, dynamically evolving process structures, rendez-vous, soundness, completeness.

Report CS-R9118
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Contents

1 Introduction

This document explores the possibilities of giving a Hoare-style proof system for a language, called SPOOL, which is a sequential version of the language POOL [1]. SPOOL is an object-oriented language, just like POOL, but it is sequential, so that we do not have to deal with the specific problems connected with parallelism (it turns out that the other problems are already difficult enough).

The main aspect of SPOOL that is dealt with is the problem of how to reason about *pointer structures.* In SPOOL, objects can be created at arbitrary points in a program, references to them can be stored in variables and passed around as parameters in messages. This implies that complicated and dynamically evolving structures of references between objects can occur. We want to reason about these structures on an abstraction level that is *at least as high as that of the programming language.* In more detail, this means the following:

- The only operations on "pointers" (references to objects) are
  - testing for equality
  - dereferencing (looking at the value of an instance variable of the referenced object)
- In a given state of the system, it is only possible to mention the objects that exist in that state. Objects that do not (yet) exist never play a role.

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

The above restrictions have quite severe consequences for the proof system. The limited set of operations on pointers implies that first-order logic is too weak to express some interesting properties of pointer structures (for example, the fact that it is possible to go from w to z by following a finite number of x-links). It is surely too weak to apply the standard techniques in proofs of completeness of a proof system, where arbitrarily long computation sequences are coded into a finite set of variables.

Therefore we have to extend our assertion language to make it more expressive. We considered two approaches:

- Using recursively defined predicates, by which the above "interesting" properties of pointer structures can be expressed quite easily. This approach is worked out in [2].

- Allowing the assertion language to reason about finite sequences of objects. In this way the above properties can also be expressed (but not quite so elegantly). This approach is studied in this report.

In section 2 we shall present the syntax of this language SPOOL. Then, in section 3 we shall give a denotational semantics for it. In section 4 we introduce an assertion language, using quantification over finite sequences of objects, in which properties of states in a computation can be formulated, and we formally define its semantics. After that, in section 5, we present a Hoare-style proof system for SPOOL using this assertion language. This proof system is proved to be sound with respect to the denotational semantics. In section 6 we prove the completeness of the system. Finally, in section 7, some conclusions are drawn from the present work.

2 The language SPOOL

2.1 An informal introduction

The shortest description of the language SPOOL would be that it results from omitting the *body* of each class in POOL-T [1]. The most important consequence of this is that the parallelism, present in POOL-T, disappears. But let us try to give a short, independent description of SPOOL.

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

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

The sender of a message is *blocked* until the result comes back, that is, it cannot answer any message while it still has an outstanding message of its own. Therefore, when an object sends a message to itself (directly or indirectly) this will lead to abnormal termination of the program. This is an important difference with some other object-oriented languages, like Smalltalk-80 [6].

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

A program essentially consists of a number of class definitions, together with a statement to be executed by an instance of a specific class. Usually, but not necessarily, this instance is the only non-standard object that exists at the beginning of the program: the others still have to be created.

2.2 The syntax

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

Assumption 2.1
We assume the following sets to be given:

- A set C of *class names*, with typical element c (this means that metavariables like $c, c', c_1, \ldots$ range over elements of the set C). We assume that $\mathsf{Int}, \mathsf{Bool} \notin C$ and define the set $C^+ = C \cup \{\mathsf{Int}, \mathsf{Bool}\}$ with typical element d.

- For each $c \in C$ and $d \in C^+$ we assume a set $IVar_d^c$ of *instance variables* of type d in class c. By this we mean that such a variable may occur in the definition of class c and that its contents will be a reference to an object of type d. The set $IVar_d^c$ will have as a typical element x_d^c.

- For each $d \in C$ we assume a set $TVar_d$ of *temporary variables* of type d, with typical element u_d.

- We shall let the metavariable n range over elements of $\mathbf{Z}$, the set of whole numbers.

- For each $c \in C$ and $d_0, \ldots, d_n \in C^+$ $(n \geq 0)$ we assume a set $MName^c_{d_0,\ldots,d_n}$ of *method names* of class c with result type d_0 and parameter types $d_1, \ldots, d_n$. The set $MName^c_{d_0,\ldots,d_n}$ will have $m^c_{d_0,\ldots,d_n}$ as a typical element.

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

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

$$
\begin{array}{rcll}
e^c_d & ::= & x^c_d & \\
& | & u_d & \\
& | & \mathsf{nil}_d & \\
& | & \mathsf{self} & \text{if } c = d \\
& | & \mathsf{true} \mid \mathsf{false} & \text{if } d = \mathsf{Bool} \\
& | & n & \text{if } d = \mathsf{Int} \\
& | & e1^c_{d'} \doteq e2^c_{d'} & \text{if } d = \mathsf{Bool} \\
& | & e1^c_{\mathsf{Int}} + e2^c_{\mathsf{Int}} & \text{if } d = \mathsf{Int} \\
& & \vdots & \\
& | & e1^c_{\mathsf{Int}} < e2^c_{\mathsf{Int}} & \text{if } d = \mathsf{Bool} \\
& & \vdots &
\end{array}
$$

The intuitive meaning of these expressions will probably be clear from section 2.1. Note that in the programming language we put a dot over the equal sign ($\doteq$) to distinguish it from the equality sign we use in the meta-language.

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

$$
\begin{array}{rcll}
s^c_d & ::= & e^c_d & \\
& | & \mathsf{new}_d & \text{if } d \in C \text{ (i.e., } d \neq \mathsf{Int}, \mathsf{Bool}) \\
& | & e0^c_{c_0} \,!\, m^{c_0}_{d,d_1,\ldots,d_n}(e1^c_{d_1}, \ldots, en^c_{d_n}) & (n \geq 0)
\end{array}
$$

The first kind of side effect expression is a normal expression, which has no actual side effect, of course. The second kind is the creation of a new object. This new object will also be the value of the side effect expression. The third kind of side effect expression specifies that a message is to be sent to the object that results from e_0, with method name m and with arguments (the objects resulting from) $e_1, \ldots, e_n$.

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

$$
\begin{array}{rcl}
S^c & ::= & x^c_d \leftarrow s^c_d \\
& | & u_d \leftarrow s^c_d \\
& | & s^c_d \\
& | & S^c_1 \,;\, S^c_2 \\
& | & \mathsf{if}\ e^c_{\mathsf{Bool}}\ \mathsf{then}\ S^c_1\ [\mathsf{else}\ S^c_2]\ \mathsf{fi} \\
& | & \mathsf{while}\ e^c_{\mathsf{Bool}}\ \mathsf{do}\ S^c\ \mathsf{od}
\end{array}
$$

Again, the intuitive meaning of these statements will probably be clear. Note that a side effect expression s may occur in the place of a statement. This means that s is evaluated and then its value is discarded, so that only the side effect remains. If in a conditional statement the else-part is absent, the statement is interpreted as if it contained else $\mathsf{nil}_{\mathsf{Int}}$.

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

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

Here we require that the u_{id_i} are all different and that none of them occurs at the left hand side of an assignment in S^c (and that $n \geq 0$).

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

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

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

where we require that all the method names are different (and $n \geq 0$).

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

$$U^{c_1,\ldots,c_n}_{m_1,\ldots,m_k} ::= D_1{}^{c_1}_{\bar{m}_1},\ldots,D_n{}^{c_n}_{\bar{m}_n}$$

where $m_1,\ldots,m_k = \bar{m}_1,\ldots,\bar{m}_n$. We require that all the class names are different.

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

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

Here we require that c occurs in $c_1,\ldots,c_n$. (The symbol '|' is part of the syntax, not of the meta-syntax.)

The interpretation of such a program is that the statement S is executed by some object of class c (the root object) in the context of the declarations contained in the unit U. In many cases (including the following example) we shall assume that at the beginning of the execution this root object is the only existing non-standard object.

Example 2.9
The following program generates prime numbers using the sieve method of Eratosthenes. We assume the following symbols:

- The class name Sieve $\in C$ (abbreviated sometimes by c_1) with instance variables $\mathsf{p} \in IVar^{c_1}_{\mathsf{Int}}$ and next $\in IVar^{c_1}_{c_1}$, temporary variable $\mathsf{q} \in TVar_{\mathsf{Int}}$ and method name input $\in MName^{c_1}_{c_1,\mathsf{Int}}$.
- The class name Driver $\in C$ (abbreviated by c_2) with instance variables i, bound $\in$ $\mathsf{IVar}^{c_2}_{\mathsf{Int}}$ and first $\in IVar^{c_2}_{c_1}$.

Then this is the program:

```
⟨Sieve : ⟨input ⇐ (q) : if next ≐ nil
                        then next ← new;
                             p ← q
                        else if q mod p ≠ 0
                             then next ! input(q)
                             fi
                        fi
                        ↑ self ⟩,
Driver : ⟨ ⟩
|
Driver : i ← 2;
         first ← new;
         while i < bound
         do first ! input(i);
            i ← i + 1
         od
⟩
```

Figure 1 represents the system in a certain stage of the execution of the program.

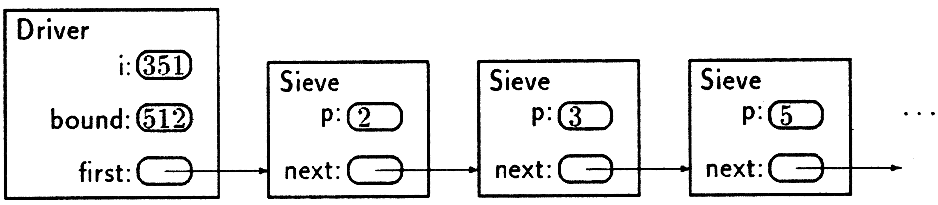

Figure 1: Objects in the sieve program in a certain stage of the execution

3 Semantics

3.1 Domain definitions

Definition 3.1
We assume for every $c \in C$ an infinite set $\mathbf{O}^c$ of *object names* of class c, with typical element β^c. We define $\mathbf{P}^c$ to be the set of all finite subsets of $\mathbf{O}^c$, with typical element π^c. Furthermore we assume a function $pick^c : \mathbf{P}^c \to \mathbf{O}^c$ that satisfies

$$\forall \pi^c \in \mathbf{P}^c : pick^c(\pi^c) \notin \pi^c. \tag{3.1}$$

This function will be used to generate the name of a new object, whenever one is created.

For the standard classes Int and Bool we define the sets of object names as follows:

$$\begin{aligned} \mathbf{O}^{\mathsf{Int}} &= \mathbf{Z} \\ \mathbf{O}^{\mathsf{Bool}} &= \mathbf{B} = \{\mathbf{t}, \mathbf{f}\} \end{aligned}$$

(We shall not need functions $pick^{\mathsf{Int}}$ or $pick^{\mathsf{Bool}}$)

Definition 3.2
For every set X we define the corresponding *flat domain* $X_\perp$ to be the set $X \cup \{\perp\}$, equipped with an ordering $\sqsubseteq$ defined by

$$x \sqsubseteq y \iff x = \perp \vee x = y.$$

Note that for every set X, $X_\perp$ is a complete partial order (cpo). Sometimes we shall only consider the underlying set of this ordering, for example in definition 3.4.

Definition 3.3
We shall often use generalized Cartesian products of the form

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

As usual, each element of this set is a function f with domain A such that $f(i) \in B(i)$ for each $i \in A$. We shall sometimes write $f_{(i)}$ for $f(i)$ if $i \in A$ and $f \in \prod_{i \in A} B(i)$, and also we sometimes write $\langle f(i) \rangle_{i \in A}$ for $\lambda(i \in A).f(i)$. Finite products are special cases: If A is of the form $\{1, \ldots, n\}$ we sometimes write $B(1) \times \cdots \times B(n)$.

Definition 3.4
We define the set Σ of *states*, with typical element σ, as follows:

$$\Sigma = \prod_c \mathbf{P}^c \times \prod_{c,d} \left(\mathbf{O}^c \to IVar_d^c \to \mathbf{O}_\perp^d\right) \times \prod_d \left(TVar_d \to \mathbf{O}_\perp^d\right)$$

A little explanation is surely required here. A state $\sigma \in \Sigma$ records the values of the variables in the whole system at a certain point in the computation:

- Its first component $\sigma_{(1)}$ gives for every class $c \in C$ a finite set of objects $\sigma_{(1)(c)} \in \mathbf{P}^c$. This set represents the objects that *exist* in this state (i.e., they have already been created).
- The second component $\sigma_{(2)}$ records the values of the instance variables. More concretely, if $c \in C$ and $d \in C^+$ are class names, $\beta^c \in \mathbf{O}^c$ is an object of class c, and $x_d^c \in IVar_d^c$ is an instance variable of type d in class c, then $\sigma_{(2)(c,d)}(\beta^c)(x_d^c) \in \mathbf{O}_\perp^d$ is the value of the instance variable x_d^c of object β^c. If this value is $\perp$, this means that the variable refers to *no* object. This is the situation for a variable that has not been initialized, but it can also be achieved by assigning nil to it.
- The third component $\sigma_{(3)}$ records the values of the temporary variables. More concretely again, if $d \in C^+$ is a class name and $u_d \in TVar_d$ is a temporary variable of type d, then $\sigma_{(3)(d)}(u_d) \in \mathbf{O}_\perp^d$ is the value of the variable u_d. Here again, it is possible that the value of the variable is nil.

For any state σ we introduce by convention that $\sigma_{(1)(\mathsf{Int})} = \mathbf{Z}$ and $\sigma_{(1)(\mathsf{Bool})} = \mathbf{B}$. Furthermore we write $\sigma^{(d)}$ for $\sigma_{(1)(d)}$.

Definition 3.5
Note that in general it is possible that in a state the variables of an existing object refer to an object that does *not* exist. If this is not the case and, additionally, the variables of the non-existing objects are not initialized, we say that the state is *consistent*. More precisely, we call a state σ consistent if

- $\forall c \in C\ \forall \beta^c \in \sigma^{(c)}\ \forall c' \in C\ \forall x_{c'}^c \in IVar_{c'}^c \quad \sigma_{(2)(c,c')}(\beta^c)(x_{c'}^c) \in \sigma_\perp^{(c')}$
- $\forall c \in C\ \forall u_c \in TVar_c \quad \sigma_{(3)(c)}(u_c) \in \sigma_\perp^{(c)}$
- $\forall c \in C\ \forall \beta^c \in \mathbf{O}^c \setminus \sigma^{(c)}\ \forall d \in C^+\ \forall x_d^c \in IVar_d^c \quad \sigma_{(2)(c,d)}(\beta^c)(x_d^c) = \perp$

(Note that it would not make sense for either c or c' to be Int or Bool.) We shall occasionally use the shorthand $OK(\sigma)$ to indicate that σ is consistent.

Definition 3.6
We say that a state σ' *extends* a state σ (notation $\sigma \preceq \sigma'$) if $\forall c \in C\ \ \sigma^{(c)} \subseteq \sigma'^{(c)}$.

Definition 3.7
We shall make flexible use of the so-called *variant notation*, especially in connection with states. The variant notation is a short way to express a new state that arises when some component of an old state is modified. For example, if we write

$$\sigma' = \sigma\{\beta_1{}^d / \beta_2{}^c, x_d^c\}$$

this means the following:

$$
\begin{array}{lll}
\sigma'_{(1)} & = & \sigma_{(1)} \\
\sigma'_{(2)(c,d)}(\beta_2)(x) & = & \beta_1 \\
\sigma'_{(2)(c,d)}(\beta_2)(x') & = & \sigma_{(2)(c,d)}(\beta_2)(x') \qquad \text{if } x' \neq x \\
\sigma'_{(2)(c,d)}(\beta') & = & \sigma_{(2)(c,d)}(\beta') \qquad \text{if } \beta' \neq \beta_2 \\
\sigma'_{(2)(c',d')} & = & \sigma_{(2)(c',d')} \qquad \text{if } c' \neq c \text{ or } d' \neq d \\
\sigma'_{(3)} = \sigma_{(3)} & &
\end{array}
$$

(This example also illustrates the usefulness of this notation.)

Definition 3.8
The set Δ^c of *contexts* of class c, with typical element δ^c, is defined as follows:

$$\Delta^c = \mathbf{O}^c \times \prod_{c'} \mathbf{P}^{c'}$$

The meaning of a context δ^c is as follows:

- The first component $\delta^c_{(1)} \in \mathbf{O}^c$ indicates the object that is currently executing (the object denoted by self).
- The second component $\delta^c_{(2)}$ represents all the objects that are waiting for the result of a message they have sent. This is because these objects become *blocked*, that is, they cannot answer any message before they have received the result of their outstanding message. If $c' \in C$ is a class name, then $\delta^c_{(2)(c')} \in \mathbf{P}^{c'}$ is the set of blocked objects of class c'.

Definition 3.9
We say that a context δ^c *agrees* with a state σ if

- $\delta^c_{(1)} \in \sigma^{(c)}$
- $\forall c' \in C \;\; \delta^c_{(2)(c')} \subseteq \sigma^{(c')}$

We shall write the shorthand $OK(\sigma, \delta)$ to indicate that σ is consistent and δ agrees with σ.

Definition 3.10
The domain Γ of *environments*, with typical element γ, is defined as follows:

$$\Gamma = \prod_{c,d_0,\ldots,d_n} \left(MName^c_{d_0,\ldots,d_n} \rightarrow \left(\prod_{i=1}^{n} \mathbf{O}_\perp^{d_i} \right) \rightarrow \Delta^c \rightarrow \Sigma_\perp \rightarrow \left(\Sigma_\perp \times \mathbf{O}_\perp^{d_0} \right) \right)$$

An environment γ records the meaning of the methods. More concretely, if $c \in C$ and $d_0, \ldots, d_n \in C^+$ are classes, $m \in MName^c_{d_0,\ldots,d_n}$ is a method name, $\bar{\beta} = \langle {\beta_1}^{d_1}, \ldots, {\beta_n}^{d_n} \rangle \in \prod_{i=1}^n \mathbf{O}^{d_i}_\perp$ a row of objects (each possibly $\perp$), $\delta \in \Delta^c$ a context, $\sigma \in \Sigma_\perp$ a state (again possibly $\perp$), then $\gamma_{(c,\bar{d})}(m)(\bar{\beta})(\delta)(\sigma)$ is a pair $\langle \sigma', \beta_0 \rangle \in \Sigma_\perp \times \mathbf{O}^{d_0}_\perp$, with the intended meaning that if the method named by m is invoked with parameters $\bar{\beta}$, in the context δ (which indicates among others the object that executes the method), and starting in the state σ, then after the execution σ' will be the new state and β_0 is the result of the message. Here $\langle \sigma', \beta_0 \rangle = \langle \perp, \perp \rangle$ indicates abnormal termination or divergence.

Definition 3.11
We call an environment γ *agreement-preserving* if for every $c, d_0, \ldots, d_n$, for every $m^c_{\bar{d}}$, for every δ^c, for every $\sigma \in \Sigma$, and for every $\bar{\beta} = \langle {\beta_1}^{d_1}, \ldots, {\beta_n}^{d_n} \rangle \in \prod_{i=1}^n \sigma^{(d_i)}_\perp$ (note that we consider only existing objects) we have that if $OK(\sigma, \delta)$ and $\langle \sigma', \beta^{d_0} \rangle = \gamma_{(c,\bar{d})}(m)(\bar{\beta})(\delta)(\sigma)$ and $\sigma' \neq \perp$ then $OK(\sigma')$, $\sigma \preceq \sigma'$, and $\beta^{d_0} \in \sigma'^{(d_0)}_\perp$.

Note that the requirement is somewhat stronger than preservation of the agreement between state and context. We require that the new state extends the old state and that it is consistent. This automatically implies that the context δ agrees with the new state.

3.2 The semantic functions

Definition 3.12
The semantics of expressions is given by a function

$$\mathcal{E}^c_d : Exp^c_d \to \Delta^c \to \Sigma_\perp \to \mathbf{O}^d_\perp,$$

which is defined as follows:

$$\begin{array}{lcll}
\mathcal{E}^c_d[\![e^c_d]\!](\delta)(\perp) & = & \perp & \text{(from now on we assume } \sigma \neq \perp) \\
\mathcal{E}^c_d[\![x^c_d]\!](\delta)(\sigma) & = & \sigma_{(2)(c,d)}(\delta_{(1)})(x^c_d) & \\
\mathcal{E}^c_d[\![u_d]\!](\delta)(\sigma) & = & \sigma_{(3)(d)}(u_d) & \\
\mathcal{E}^c_d[\![\mathsf{nil}_d]\!](\delta)(\sigma) & = & \perp & \\
\mathcal{E}^c_d[\![\mathsf{self}]\!](\delta)(\sigma) & = & \delta_{(1)} & \text{(only if } c = d) \\
\mathcal{E}^c_d[\![\mathsf{true}]\!](\delta)(\sigma) & = & \mathbf{t} & \text{(only if } d = \mathsf{Bool}) \\
\mathcal{E}^c_d[\![\mathsf{false}]\!](\delta)(\sigma) & = & \mathbf{f} & \text{(only if } d = \mathsf{Bool}) \\
\mathcal{E}^c_d[\![n]\!](\delta)(\sigma) & = & n & \text{(only if } d = \mathsf{Int})
\end{array}$$

$$\begin{array}{lcll}
\mathcal{E}^c_d[\![{e_1}^c_{d'} \doteq {e_2}^c_{d'}]\!](\delta)(\sigma) & = & \mathbf{t} & \text{if } \mathcal{E}^c_{d'}[\![e_1]\!](\delta)(\sigma) = \mathcal{E}^c_{d'}[\![e_1]\!](\delta)(\sigma) \\
& = & \mathbf{f} & \text{otherwise}
\end{array}$$

(only if $d = \mathsf{Bool}$)

$$
\begin{aligned}
\mathcal{E}^c_d[\![{e_1}^c_d + {e_2}^c_d]\!](\delta)(\sigma) &= \perp \qquad \text{if } \mathcal{E}^c_d[\![e_1]\!](\delta)(\sigma) = \perp \text{ or } \mathcal{E}^c_d[\![e_2]\!](\delta)(\sigma) = \perp \\
&= \mathcal{E}^c_d[\![e_1]\!](\delta)(\sigma) + \mathcal{E}^c_d[\![e_2]\!](\delta)(\sigma) \qquad \text{otherwise}
\end{aligned}
$$

(only if $d = \mathsf{Int}$)

$\vdots$

$$
\begin{aligned}
\mathcal{E}^c_d[\![{e_1}^c_{d'} < {e_2}^c_{d'}]\!](\delta)(\sigma) &= \perp \qquad \text{if } \mathcal{E}^c_{d'}[\![e_1]\!](\delta)(\sigma) = \perp \\
&\qquad\quad\;\; \text{or } \mathcal{E}^c_{d'}[\![e_2]\!](\delta)(\sigma) = \perp \\
&= \mathbf{t} \qquad \text{if } \mathcal{E}^c_{d'}[\![e_1]\!](\delta)(\sigma) < \mathcal{E}^c_{d'}[\![e_2]\!](\delta)(\sigma) \\
&= \mathbf{f} \qquad \text{otherwise}
\end{aligned}
$$

(only if $d = \mathsf{Bool}$ and $d' = \mathsf{Int}$)

$\vdots$

Although most of these equations speak for themselves, we shall give some informal explanation.

- The function $\mathcal{E}[\![e]\!](\delta)$ is *strict*, that is, it will always yield $\perp$ if it is applied to a state σ that is equal to $\perp$.
- The value of an instance variable is looked up in the second component of the state σ. The first component of the context δ indicates the currently active object.
- The value of a temporary variable is looked up in the third component of the state σ.
- The value of the expression nil is always $\perp$.
- The value of the expression self is the first component of the context δ.
- The Boolean constants true and false get the corresponding truth-values as their value.
- Integer numbers are mapped to themselves. Note that at this point we are confusing syntactic and semantic entities a little, but here this is harmless.
- The equal sign between expressions means that we test whether their values are really the same objects. Note that this is a kind of non-strict predicate, because if both sides yield $\perp$, the result of the equality is nevertheless $\mathbf{t}$.
- Addition is only defined for genuine integers: If one of the two sides yields $\perp$ the result is also $\perp$.
- The same is true for the relation $<$.

Definition 3.13
The semantics of expressions with possible side effect is given by the function

$$\mathcal{Z}_d^c : SExp_d^c \to \Gamma \to \Delta^c \to \Sigma_\perp \to \left(\Sigma_\perp \times \mathbf{O}_\perp^d\right).$$

To a side effect expression $s \in SExp_d^c$, an environment $\gamma \in \Gamma$, a context $\delta \in \Delta^c$, and a state $\sigma \in \Sigma_\perp$ (the state before evaluation of the side effect expression), this function assigns a pair $\langle\sigma', \beta\rangle \in \Sigma_\perp \times \mathbf{O}_\perp^d$, consisting of the state σ' after the evaluation and the result β of this side effect expression. Here $\langle\sigma', \beta\rangle = \langle\perp, \perp\rangle$ represents abnormal termination or divergence. The function $\mathcal{Z}_d^c$ is defined as follows:

$$\mathcal{Z}_d^c[\![s_d^c]\!](\gamma)(\delta)(\perp) = \langle\perp, \perp\rangle \qquad \text{(from now on we assume } \sigma \neq \perp\text{)}$$

$$\mathcal{Z}_d^c[\![e_d^c]\!](\gamma)(\delta)(\sigma) = \langle\sigma, \mathcal{E}_d^c[\![e]\!](\delta)(\sigma)\rangle$$

$$\begin{array}{l} \mathcal{Z}_d^c[\![\mathsf{new}_d]\!](\gamma)(\delta)(\sigma) = \langle\sigma', \beta\rangle \\ \quad \begin{array}{lll} \text{where } \beta & = & pick^d(\sigma^{(d)}) \\ \sigma' & = & \sigma\left\{\sigma^{(d)} \cup \{\beta\} \,/\, d\right\}\left\{\perp \,/\, \beta, x_{d'}^d\right\}_{d' \in C^+,\, x \in IVar_{d'}^d} \end{array} \\ \quad \text{note that } d \in C, \text{ i.e., } d \neq \mathsf{Int}, \mathsf{Bool} \end{array}$$

$$\begin{array}{l} \mathcal{Z}_d^c[\![e_{0\,c'}^{\;c} \,!\, m_{d,d_1,\ldots,d_n}^{c'}(e_{1\,d_1}^{\;c}, \ldots, e_{n\,d_n}^{\;c})]\!](\gamma)(\delta)(\sigma) = \langle\sigma', \beta^d\rangle \\ \quad \begin{array}{lll} \text{where } \beta_0{}^{c'} & = & \mathcal{E}_{c'}^c[\![e_{0\,c'}^{\;c}]\!](\delta)(\sigma) \\ \beta_i{}^{d_i} & = & \mathcal{E}_{d_i}^c[\![e_{i\,d_i}^{\;c}]\!](\delta)(\sigma) \qquad (i = 1, \ldots, n) \\ \langle\sigma', \beta^d\rangle & = & \langle\perp, \perp\rangle \qquad \text{if } \beta_0 = \perp \\ & & \text{(from now on we assume } \beta_0 \neq \perp\text{)} \\ \delta'^{c'} & = & \left\langle \beta_0{}^{c'}, \delta_{(2)}\left\{\delta_{(2)(c)} \cup \{\delta_{(1)}\}/c\right\}\right\rangle \\ \langle\sigma', \beta^d\rangle & = & \gamma_{(c',d,d_1,\ldots,d_n)}(m)(\beta_1, \ldots, \beta_n)(\delta')(\sigma) \end{array} \end{array}$$

Some explanation is appropriate here.

- Again, for any s, γ, and δ, $\mathcal{Z}[\![s]\!](\gamma)(\delta)$ is a strict function: if the starting state σ is $\perp$ it delivers $\langle\perp, \perp\rangle$.
- If an expression e_d^c occurs as a side effect expression, its result is computed using the function $\mathcal{E}_d^c$ and the state is unchanged.
- The resulting object β of a new-expression is obtained by applying the function $pick^d$ to the set $\sigma^{(d)}$ of existing objects of class d. By the property listed in equation 3.1 on page 24, we know that we really get a *new* object. The new state σ' reflects the situation after the creation of this object. In its first component $\sigma'_{(1)}$ the object β is added to the set of existing objects of class d, while

the other classes are unchanged (we use the variant notation to express this). The explicit initialization of the instance variables to nil would be unnecessary if we knew that σ is consistent.

- In order to evaluate a send-expression, first the destination object β_0 and the parameters $\beta_1, \ldots, \beta_n$ are computed (in the old state). Note that if the destination is $\perp$ (i.e., nil), then the program will fail, which is represented by setting $\langle \sigma', \beta^d \rangle$ to $\langle \perp, \perp \rangle$. Otherwise a new context is created, in which the executing object is the destination of the message, and in which the sending object is added to the set of blocked objects (of the appropriate class). Then the meaning of the method m is looked up in the environment γ and, provided with the parameters, the new context and the old state, it gives us the new state and the result of the send-expression.

Definition 3.14
The semantics of statements is given by a function

$$\mathcal{S}^c : Stat^c \to \Gamma \to \Delta^c \to \Sigma_\perp \to \Sigma_\perp,$$

which is defined as follows:

$$\mathcal{S}^c[\![S^c]\!](\gamma)(\delta)(\perp) = \perp \qquad \text{(from now on we assume } \sigma \neq \perp)$$

$$\begin{aligned} &\mathcal{S}^c[\![x^c_d \leftarrow s^c_d]\!](\gamma)(\delta)(\sigma) = \sigma'' \\ &\qquad \text{where } \langle \sigma', \beta \rangle = \mathcal{Z}^c_d[\![s^c_d]\!](\gamma)(\delta)(\sigma) \\ &\qquad \phantom{\text{where }} \sigma'' = \sigma'\{\beta/\delta_{(1)}, x\} \end{aligned}$$

$$\begin{aligned} &\mathcal{S}^c[\![u_d \leftarrow s^c_d]\!](\gamma)(\delta)(\sigma) = \sigma'' \\ &\qquad \text{where } \langle \sigma', \beta \rangle = \mathcal{Z}^c_d[\![z^c_d]\!](\gamma)(\delta)(\sigma) \\ &\qquad \phantom{\text{where }} \sigma'' = \sigma'\{\beta/u\} \end{aligned}$$

$$\mathcal{S}^c[\![s^c_d]\!](\gamma)(\delta)(\sigma) = (\mathcal{Z}^c_d[\![s^c_d]\!](\gamma)(\delta)(\sigma))_{(1)}$$

$$\mathcal{S}^c[\![S_1; S_2]\!](\gamma)(\delta)(\sigma) = \mathcal{S}^c[\![S_2]\!](\gamma)(\delta)\,(\mathcal{S}^c[\![S_1]\!](\gamma)(\delta)(\sigma))$$

$$\begin{aligned} \mathcal{S}^c[\![\textsf{if } e \textsf{ then } S_1 \textsf{ else } S_2 \textsf{ fi}]\!](\gamma)(\delta)(\sigma) &= \perp && \text{if } \beta = \perp \\ &= \mathcal{S}^c[\![S_1]\!](\gamma)(\delta)(\sigma) && \text{if } \beta = \mathbf{t} \\ &= \mathcal{S}^c[\![S_2]\!](\gamma)(\delta)(\sigma) && \text{if } \beta = \mathbf{f} \end{aligned}$$

where $\beta = \mathcal{E}[\![e]\!](\delta)(\sigma)$

$$\mathcal{S}^c[\![\text{while } e \text{ do } S \text{ od}]\!](\gamma) = \mu\Phi$$

where $\Phi : (\Delta^c \to (\Sigma_\perp \to \Sigma_\perp)) \to (\Delta^c \to (\Sigma_\perp \to \Sigma_\perp))$ is defined as follows:

$$\begin{array}{rcll} \Phi(\varphi)(\delta)(\sigma) & = & \perp & \text{if } \beta = \perp \\ & = & \varphi(\delta)(\mathcal{S}^c[\![S]\!](\gamma)(\delta)(\sigma)) & \text{if } \beta = \mathbf{t} \\ & = & \sigma & \text{if } \beta = \mathbf{f} \end{array}$$

where $\beta = \mathcal{E}[\![e]\!](\delta)(\sigma)$

Here is some informal explanation:

- For any S, γ, and δ, $\mathcal{S}[\![S]\!](\gamma)(\delta)$ is a strict function from $\Sigma_\perp$ to $\Sigma_\perp$.
- If an assignment to an instance variable x is done, first the right hand side is evaluated, resulting in a new state σ' (because of possible side effects), and an object β. Now the final state σ'' is constructed from σ' by modifying its second component in such a way that the object β becomes the value of the variable x.
- For an assignment to a temporary variable, essentially the same thing is done, except that the new value is stored away in the third component of the resulting state σ''.
- If a side effect expression occurs at the place of a statement, it is evaluated and its resulting object is ignored. Only the new state is kept (this is the first component of the result of the evaluation).
- Sequential composition of statements is modelled by letting the second statement act on the state that results from the first statement.
- For a conditional statement first the condition is evaluated. Depending on that the first or the second clause is executed (or a failure is signalled).
- A while statement is modelled by taking the least fixed point of the operator Φ. This operator takes its argument φ as an approximation of the meaning of the while statement and maps it to a better approximation, obtained by unwinding the loop one more time.

Definition 3.15
The semantics of method definitions is given by a function

$$\mathcal{M}^c_{d_0,\ldots,d_n} : \mathit{MethDef}^c_{d_0,\ldots,d_n} \to \Gamma \to \left(\prod_{i=1}^{n} \mathbf{O}^{d_i}_\perp\right) \to \Delta^c \to \Sigma_\perp \to \left(\Sigma_\perp \times \mathbf{O}^{d_0}_\perp\right)$$

which is defined by:

$$
\begin{array}{llll}
\multicolumn{4}{l}{\mathcal{M}^c_{d_0,\ldots,d_n}[\![(u_{1d_1},\ldots,u_{nd_n}) : S^c \uparrow e^c_{d_0}]\!](\gamma)(\beta_1{}^{d_1},\ldots,\beta_n{}^{d_n})(\delta^c)(\sigma) = \langle \sigma''', \beta^{d_0}\rangle} \\
\text{where } \sigma' & = & \perp & \text{if } \delta_{(1)} \in \delta_{(2)(c)} \text{ or } \sigma = \perp \\
 & = & \langle \sigma_{(1)}, \sigma_{(2)}, \sigma'_{(3)} \rangle & \text{otherwise} \\
\sigma'_{(3)(d)}(u_d) & = & \beta_i{}^{d_i} & \text{if } d = d_i \text{ and } u_d = u_{id_i} \\
 & & & \text{for } i = 1,\ldots,n \\
 & = & \perp & \text{otherwise} \\
\sigma'' & = & \mathcal{S}^c[\![S^c]\!](\gamma)(\delta^c)(\sigma') & \\
\beta^{d_0} & = & \mathcal{E}^c_{d_0}[\![e^c_{d_0}]\!](\delta^c)(\sigma'') & \\
\sigma''' & = & \perp & \text{if } \sigma'' = \perp \\
 & = & \langle \sigma''_{(1)}, \sigma''_{(2)}, \sigma_{(3)} \rangle & \text{if } \sigma'' \neq \perp
\end{array}
$$

Again we give an informal explanation: The first thing to be checked when a method is to be executed is whether the executing object is *blocked*, that is whether $\delta_{(1)} \in \delta_{(2)(c)}$ or whether the starting state σ is $\perp$. If this is the case the result of the method will be the pair $\langle \perp, \perp \rangle$ (this will come out automatically if we set σ' to $\perp$). Next we construct a state σ' by initializing all temporary variables to $\perp$, except the formal parameters, which are bound to the corresponding actual ones (that is, the variable u_{id_i} is set to $\beta_i{}^{d_i}$). In this modified state σ' we execute the statement S^c of the method, which results in a new state σ''. In this state we can evaluate the result expression $e^c_{d_0}$, which gives us the object β^{d_0}. The state σ''' after the method execution is obtained by restoring the temporary variables to their values before the method execution.

Definition 3.16
The semantics of class definitions is given by a function

$$\mathcal{C}^c_{m_1,\ldots,m_n} : \mathit{ClassDef}^c_{m_1,\ldots,m_n} \to \Gamma \to \Gamma$$

which is defined as follows:

$$
\begin{aligned}
&\mathcal{C}^c_{m_1,\ldots,m_n}[\![c : \langle m_1{}^c_{\bar{d}_1} \Leftarrow \mu_1{}^c_{\bar{d}_1}, \ldots, m_n{}^c_{\bar{d}_n} \Leftarrow \mu_n{}^c_{\bar{d}_n} \rangle]\!](\gamma) \\
&\quad = \gamma\Big\{\mathcal{M}[\![\mu_1{}^c_{\bar{d}_1}]\!](\gamma) \,\Big/\, m_1{}^c_{\bar{d}_1}\Big\} \ldots \Big\{\mathcal{M}[\![\mu_n{}^c_{\bar{d}_n}]\!](\gamma) \,\Big/\, m_n{}^c_{\bar{d}_n}\Big\}
\end{aligned}
$$

This means that in the environment γ the value associated with each method m in the class definition is replaced by the value obtained from the corresponding method definition. However, this method definition is still evaluated with respect to the old environment γ. Note that the order of the replacements does not matter, because it is required that all method names must be different.

Definition 3.17
The semantics of units is given by a function

$$\mathcal{U}^{c_1,\ldots,c_n}_{m_1,\ldots,m_k} : \mathit{Unit}^{c_1,\ldots,c_n}_{m_1,\ldots,m_k} \to \Gamma \to \Gamma$$

which is defined by:

$$
\begin{aligned}
&\mathcal{U}^{c_1,\ldots,c_n}_{m_1,\ldots,m_k}[\![{D_1}^{c_1}_{\bar{m}_1}, \ldots, {D_n}^{c_n}_{\bar{m}_n}]\!](\gamma) = \gamma' \\
&\qquad \text{where } \gamma' && = && \gamma\{\zeta_j/m_j\}_{j=1}^{k} \\
&\qquad \langle \zeta_1, \ldots, \zeta_k \rangle && = && \mu\Psi \\
&\qquad \Psi(\zeta'_1, \ldots, \zeta'_k) && = && \langle \gamma''(c_{i_j}, \bar{d}_j)(m_j) \rangle_{j=1}^{k} \\
&\qquad \gamma'' && = && \mathcal{C}[\![D_1]\!] \circ \cdots \circ \mathcal{C}[\![D_n]\!](\gamma\{\zeta'_j/m_j\}_{j=1}^{k})
\end{aligned}
$$

(we suppose that $m_j = {m_j}^{c_{i_j}}_{\bar{d}_j}$).

The main point in this definition is the construction of an environment γ' from the least fixed point of the operator Ψ. This operator Ψ takes as its argument a row $\zeta'_1, \ldots, \zeta'_k$ of possible meanings of the methods defined in the unit. Assuming these meanings for the corresponding methods, a new environment γ'' is determined from the class definitions in the unit and from this environment the new meanings for the previous methods are extracted, yielding the output of Ψ. The least fixed point of Ψ therefore consists of the meanings of the methods defined in the unit, where for the other methods the meanings recorded in γ are assumed.

Because we require that all the class names (the c_i) are different, each $\mathcal{C}[\![{D_i}^{c_i}]\!]$ modifies a different part of the environment $\gamma\{\zeta'_j/m_j\}_{j=1}^{k}$. Therefore the order in which they are composed does not matter. We cannot simply take the least fixed point of $\mathcal{C}[\![D_1]\!] \circ \ldots \circ \mathcal{C}[\![D_n]\!]$ because we want to preserve the meanings of methods not defined in $D_1, \ldots, D_n$. This is important in the soundess of the recursion rule.

Definition 3.18
Finally we give the semantics of programs by defining a function

$$\mathcal{P}^c : \mathit{Prog}^c \to \Gamma \to \Delta^c \to \Sigma_\perp \to \Sigma_\perp$$

as follows:

$$
\begin{aligned}
&\mathcal{P}[\![\langle U^{c_1,\ldots,c_n}_{m_1,\ldots,m_k} | c : S^c \rangle]\!](\gamma) = \mathcal{S}^c[\![S]\!](\gamma') \\
&\qquad \text{where } \gamma' = \mathcal{U}[\![U]\!](\gamma)
\end{aligned}
$$

If every method used in the program is defined in the unit then the meaning is independent of the environment γ. One could take the "empty" environment γ_0, defined by

$$\gamma_0(c, \bar{d})(m^c_{\bar{d}})(\bar{\beta})(\delta^c)(\sigma) = \langle \perp, \perp \rangle$$

(this is certainly agreement-preserving).

3.3 Remarks on the semantics

In the foregoing definition of the semantic functions that play a role in our language, we have omitted some details. One of these details is the fact that all the functions of which we need the least fixed point are indeed continuous.

Lemma 3.19
The function Φ, used in the semantics of while statements in definition 3.14, is continuous.

Proof
First of all it is easy to see that

- For every expression e_d^c and for every context δ^c, the function $\mathcal{E}[\![e]\!](\delta)$ is *strict*, i.e., that $\mathcal{E}[\![e]\!](\delta)(\perp) = \perp$.
- For every statement S^c, for every environment γ, and for every context δ^c, the function $\mathcal{S}[\![S]\!](\gamma)(\delta)$ is also strict, i.e., $\mathcal{S}[\![S]\!](\gamma)(\delta)(\perp) = \perp$.

Now after a little calculation it becomes clear that this is all we need to ensure the continuity of Φ, which moreover maps strict functions in $\Delta \to (\Sigma_\perp \to \Sigma_\perp)$ again to strict functions (so its least fixed point will also be a strict function). $\square$

Lemma 3.20
The function Ψ, used in definition 3.17 to define the semantics of units, is continuous.

Proof
The proof of this lemma is somewhat more involved. It would proceed in the following steps:

- For any side effect expression s_d^c, $\mathcal{Z}[\![s]\!]$ is a continuous function from Γ to $\Delta^c \to \Sigma_\perp \to \left(\Sigma_\perp \times \mathbf{O}_\perp^d\right)$.
- For any statement S^c, $\mathcal{S}[\![S]\!]$ is a continuous function from Γ to $\Delta^c \to \Sigma_\perp \to \Sigma_\perp$.
- For any method definition $\mu_{d_0,\ldots,d_n}^c$, $\mathcal{M}[\![\mu]\!]$ is a continuous function from Γ to $\left(\prod_{i=1}^n \mathbf{O}_\perp^{d_i}\right) \to \Delta^c \to \Sigma_\perp \to \left(\Sigma_\perp \times \mathbf{O}_\perp^{d_0}\right)$.
- For any class definition $D_{\bar{m}}^c$, $\mathcal{C}[\![D]\!]$ is a continuous function from Γ to Γ.
- Now we can prove that Ψ is a continuous function.

□

In retrospect we can change the domain assignments of several entities as follows (where $\xrightarrow{c}$ stands for continuous functions and $\xrightarrow{s}$ for strict functions):

$$\Gamma = \prod_{c,d_0,\ldots,d_n} \left(MName^c_{d_0,\ldots,d_n} \to \left(\prod_{i=1}^{n} \mathbf{O}^{d_i}_{\perp}\right) \to \Delta^c \to \Sigma_\perp \xrightarrow{s} \left(\Sigma_\perp \times \mathbf{O}^{d_0}_{\perp}\right)\right)$$

$$\mathcal{E}^c_d : Exp^c_d \to \Delta^c \to \Sigma_\perp \xrightarrow{s} \mathbf{O}^d_\perp$$

$$\mathcal{Z}^c_d : SExp^c_d \to \Gamma \xrightarrow{c} \Delta^c \to \Sigma_\perp \xrightarrow{s} \left(\Sigma_\perp \times \mathbf{O}^d_\perp\right)$$

$$\mathcal{S}^c : Stat^c \to \Gamma \xrightarrow{c} \Delta^c \to \Sigma_\perp \xrightarrow{s} \Sigma_\perp,$$

$$\mathcal{M}^c_{d_0,\ldots,d_n} : MethDef^c_{d_0,\ldots,d_n} \to \Gamma \xrightarrow{c} \left(\prod_{i=1}^{n} \mathbf{O}^{d_i}_{\perp}\right) \to \Delta^c \to \Sigma_\perp \xrightarrow{s} \left(\Sigma_\perp \times \mathbf{O}^{d_0}_{\perp}\right)$$

$$\mathcal{C}^c_{m_1,\ldots,m_n} : ClassDef^c_{m_1,\ldots,m_n} \to \Gamma \xrightarrow{c} \Gamma$$

$$\mathcal{U}^{c_1,\ldots,c_n}_{m_1,\ldots,m_k} : Unit^{c_1,\ldots,c_n}_{m_1,\ldots,m_k} \to \Gamma \xrightarrow{c} \Gamma$$

$$\mathcal{P}^c : Prog^c \to \Gamma \xrightarrow{c} \Delta^c \to \Sigma_\perp \xrightarrow{s} \Sigma_\perp$$

Lemma 3.21
Now we come to the issues of consistent states and agreement between context and state. We can make the following observations:

- For any expression e^c_d, for any state $\sigma \in \Sigma$, and for any context $\delta \in \Delta^c$ such that $OK(\sigma, \delta)$, we have that $\mathcal{E}[\![e]\!](\delta)(\sigma) \in \sigma^{(d)}_{\perp}$.
- For any side effect expression s^c_d, for any agreement-preserving environment γ, for any state $\sigma \in \Sigma$, and for any context $\delta \in \Delta^c$ such that $OK(\sigma, \delta)$, we have that if $\langle \sigma', \beta^d \rangle = \mathcal{Z}[\![s]\!](\gamma)(\delta)(\sigma)$ and $\sigma' \neq \perp$ then $OK(\sigma')$, $\sigma \preceq \sigma'$ (therefore also $OK(\sigma', \delta)$), and $\beta^d \in \sigma'^{(d)}_{\perp}$.
- For any statement S^c, for any agreement-preserving environment γ, for any state $\sigma \in \Sigma$, and for any context $\delta \in \Delta^c$ such that $OK(\sigma, \delta)$, if $\sigma' = \mathcal{S}[\![S]\!](\gamma)(\delta)(\sigma)$ and $\sigma' \neq \perp$ then $OK(\sigma')$ and $\sigma \preceq \sigma'$ (therefore also $OK(\sigma', \delta)$).
- For any method definition $\mu^c_{d_0,\ldots,d_n}$, for any agreement-preserving environment γ, for any state $\sigma \in \Sigma$, for any context $\delta \in \Delta^c$ such that $OK(\sigma, \delta)$, and for any row of (existing) objects $\bar{\beta} = \langle {\beta_1}^{d_1}, \ldots, {\beta_n}^{d_n} \rangle \in \prod_{i=1}^{n} \sigma^{(d_i)}_{\perp}$ we have if $\langle \sigma', \beta^{d_0} \rangle = \mathcal{M}[\![\mu]\!](\gamma)(\bar{\beta})(\delta)(\sigma)$ and $\sigma' \neq \perp$ then $OK(\sigma')$, $\sigma \preceq \sigma'$ (so also $OK(\sigma', \delta)$), and $\beta^{d_0} \in \sigma'^{(d_0)}_{\perp}$.
- For any class definition D and for any agreement-preserving environment γ we have that $\mathcal{C}[\![D]\!](\gamma)$ is again an agreement-preserving environment.

- For any unit U and for any agreement-preserving environment γ we have that $\mathcal{U}[\![U]\!](\gamma)$ is again an agreement-preserving environment.

- For any program ρ^c, for any agreement-preserving environment γ, for any state $\sigma \in \Sigma$, and for any context δ^c such that $OK(\sigma, \delta)$, if $\sigma' = \mathcal{P}[\![\rho]\!](\gamma)(\delta)(\sigma)$ and $\sigma' \neq \bot$ then $OK(\sigma')$ and $\sigma \preceq \sigma'$ (therefore also $OK(\sigma', \delta)$).

Proof

The proof consists of an easy induction on the structure of the syntactical object under consideration. $\square$

4 The assertion language and its semantics

In this section we shall develop a formalism for expressing certain properties of states, and we shall give a semantics for it.

One element of this assertion language will be the introduction of *logical variables.* These variables may not occur in the program, but only in the assertion language. Therefore we are always sure that the value of a logical variable can never be changed by a statement. Apart from a certain degree of cleanliness, this has the additional advantage that we can use logical variables to express the constancy of certain expressions (for example in the proof rule (MI) for message passing in definition 5.24). Logical variables also serve as bound variables for quantifiers.

The set of expressions in the assertion language is larger than the set of programming language expressions not only because it contains logical variables, but also because it is allowed to refer to instance variables of other objects. Furthermore we include conditional expressions in the assertion language because they are very convenient (e.g., in the axiom (SAI), see definitions 5.6 and 5.7).

In two respects our assertion language differs from the usual first-order predicate logic: Firstly, the range of quantifiers is limited to the *existing*, non-nil objects in the current state. With respect to the classes Int and Bool this only means that the range does *not* include $\perp$. This does not affect essentially the expressive power of the assertion language, but in most practical cases one wants to exclude $\perp$ from the quantification, so in these cases the assertions become shorter. For the other classes this restriction means that we cannot talk about objects that have not yet been created, even if they could be created in the future. This is done in order to satisfy the requirements on the proof system stated in the introduction. Because of this the range of the quantifiers can be different for different states. More in particular, a statement can change the truth of an assertion even if none of the program variables accessed by the statement occurs in the assertion, simply by creating an object and thereby changing the range of a quantifier. (The idea of restricting the range of quantifiers was inspired by [8].)

Secondly, in order to strengthen the expressiveness of the logic, it is augmented with quantification over finite sequences of objects. It is quite clear that this is necessary, because simple first-order logic is not able to express certain interesting properties.

4.1 The assertion language

Definition 4.1
For each $d \in C^+$ we introduce the symbol d^* for the type of all finite sequences with

elements from d, we let C^* stand for the set $\{d^* | d \in C^+\}$, and we use $C^\dagger$, with typical element a, for the union $C^+ \cup C^*$.

We define $\mathbf{O}^{d^*}$ to be the set of finite sequences with elements from $\mathbf{O}^d_\perp$ (note that the elements can also be $\perp$). The empty sequence ϵ^d is also included in $\mathbf{O}^{d^*}$. The elements in a sequence are always numbered starting from 1. In order to simplify some formulae we define $\mathbf{O}^{d^*}_\perp$ to be the same as $\mathbf{O}^{d^*}$, in deviation from definition 3.2. In addition to β^{d^*}, we shall sometimes use α^{d^*} to range over elements of $\mathbf{O}^{d^*}$.

We have the following functions:

- $len^d : \mathbf{O}^{d^*} \to \mathbf{Z}$ returns the number of elements in the sequence.
- $elt^d : \mathbf{O}^{d^*} \times \mathbf{Z} \to \mathbf{O}^d_\perp$ extracts from the first argument the element numbered by the second argument. If the second argument is "out of bounds" (less than 1 or greater that the length of the first argument) then the result is $\perp$.

Assumption 4.2
We assume that for every a in $C^\dagger$ we have a set $LVar_a$ of logical variables of type a, with typical element z_a.

Definition 4.3
We the set $LExp^c_a$ of logical expressions of type a in class c, with typical element l^c_a, as follows:

$$
\begin{array}{lll}
l^c_a & ::= & e^c_a \qquad\qquad\qquad\qquad \text{if } a \in C^+ \\
& | & z_a \\
& | & l^c_{c'} \,.\, x^{c'}_a \qquad\qquad\qquad\; \text{if } a \in C^+ \\
& | & \text{if } l_0{}^c_{\mathsf{Bool}} \text{ then } l_1{}^c_a \text{ else } l_2{}^c_a \text{ fi} \quad \text{if } a \in C^+ \\
& | & l_1{}^c_d \doteq l_2{}^c_d \qquad\qquad\quad\; \text{if } a = \mathsf{Bool} \\
& | & l_1{}^c_{\mathsf{Int}} + l_2{}^c_{\mathsf{Int}} \qquad\qquad \text{if } a = \mathsf{Int} \\
& & \vdots \\
& | & l_1{}^c_{\mathsf{Int}} < l_2{}^c_{\mathsf{Int}} \qquad\qquad \text{if } a = \mathsf{Bool} \\
& & \vdots \\
& | & |l^c_{d^*}| \qquad\qquad\qquad\quad\;\; \text{if } a = \mathsf{Int} \text{ and } d \in C^+ \\
& | & l_1{}^c_{a^*} \cdot l_2{}^c_{\mathsf{Int}} \qquad\qquad\quad \text{if } a \in C^+
\end{array}
$$

Note that the difference with the set Exp^c_d of expressions in the programming language is that in logical expressions we can use logical variables, refer to the instance variables of other objects, and write conditional expressions. Furthermore, we extended the domain of discourse by means of logical variables ranging over sequences and notations to express the length of a sequence and the selection of an element from a sequence. The selection operator '·' can be distinguished from the dereferencing operator '.' by its higher vertical position on the line and by the type of its first argument.

Definition 4.4
The set Ass^c of assertions in class c, with typical elements P^c and Q^c, is defined by:

$$\begin{array}{lcl} P^c & ::= & l^c_{\mathsf{Bool}} \\ & | & P^c \rightarrow Q^c \\ & | & \neg P^c \\ & | & \forall z_a\ P^c \\ & | & \exists z_a\ P^c \end{array}$$

This definition is rather conventional.

Definition 4.5
Of course, we shall freely use the logical connectives $\wedge$, $\vee$, and $\leftrightarrow$, which we consider as abbreviations of appropriate constructions with $\rightarrow$ and $\neg$. Furthermore we shall use $l^c_d\uparrow$ as an abbreviation for $l^c_d \doteq \mathsf{nil}_d$ and $l^c_d\downarrow$ for $\neg\ l^c_d \doteq \mathsf{nil}_d$.

Definition 4.6
Finally we define the set $CorrF^c$ of *correctness formulae* in class c, with typical element F^c, as follows:

$$\begin{array}{lcl} F^c & ::= & P^c \\ & | & \{P^c\}\rho^c\{Q^c\} \end{array}$$

4.2 Semantics of assertions and correctness formulae

Definition 4.7
In order to be able to assign a semantics to logical expressions we first define the set Ω of *valuations*, with typical element ω, as follows:

$$\Omega = \prod_a (LVar_a \rightarrow \mathbf{O}^a_\perp).$$

(Remember that $\mathbf{O}^a_\perp = \mathbf{O}^a$ if $a \in C'^*$.) A valuation assigns a value to each logical variable.

Definition 4.8
We call a valuation ω *compatible* with a state σ if

- $\forall c \in C\ \forall z_c \in LVar_c \quad \omega_{(c)}(z) \in \sigma^{(c)}_\perp$

- $\forall c \in C\ \forall z_{c^*} \in LVar_{c^*}\ \forall n \in \mathbf{Z} \quad elt^c(\omega_{(c^*)}(z), n) \in \sigma_\perp^{(c)}$

Again an abbreviation is useful: we shall write $OK(\sigma, \delta, \omega)$ meaning that σ is consistent, δ agrees with σ, and ω is compatible with σ.

Lemma 4.9
Concerning the preservation of compatibility by statements and programs we have the following properties:

- For any statement S^c, for any agreement-preserving environment γ, for any state $\sigma \in \Sigma$, for any context δ^c and for any valuation ω such that $OK(\sigma, \delta, \omega)$ we have if $\sigma' = \mathcal{S}[\![S]\!](\gamma)(\delta)(\sigma)$ and $\sigma' \neq \perp$ then $OK(\sigma', \delta, \omega)$.
- For any program ρ^c, for any agreement-preserving environment γ, for any state $\sigma \in \Sigma$, for any context δ^c and for any valuation ω such that $OK(\sigma, \delta, \omega)$ we have if $\sigma' = \mathcal{P}[\![\rho]\!](\gamma)(\delta)(\sigma)$ and $\sigma' \neq \perp$ then $OK(\sigma', \delta, \omega)$.

Proof
This is an easy consequence of lemma 3.21. □

Definition 4.10
We define the semantics of logical expressions by specifying the function

$$\mathcal{L}_a^c : LExpr_a^c \to \Omega \to \Delta^c \to \Sigma \to \mathbf{O}_\perp^a$$

as follows:

$$\begin{aligned}
\mathcal{L}_d^c[\![e_d^c]\!](\omega)(\delta)(\sigma) &= \mathcal{E}_d^c[\![e]\!](\delta)(\sigma) \\
\mathcal{L}_d^c[\![z_d]\!](\omega)(\delta)(\sigma) &= \omega_{(d)}(z)
\end{aligned}$$

$$\begin{aligned}
\mathcal{L}_d^c[\![l_{c'}^c \,.\, x_d^{c'}]\!](\omega)(\delta)(\sigma) &= \perp && \text{if } \beta = \perp \\
&= \sigma_{(2)(c',d)}(\beta)(x_d^{c'}) && \text{otherwise}
\end{aligned}$$

where $\beta^{c'} = \mathcal{L}_{c'}^c[\![l]\!](\omega)(\delta)(\sigma)$

$$\begin{aligned}
\mathcal{L}_d^c[\![\textsf{if } l_{0\,\textsf{Bool}}^c \textsf{ then } l_{1d}^c \textsf{ else } l_{2d}^c \textsf{ fi}]\!](\omega)(\delta)(\sigma) &= \perp && \text{if } \beta = \perp \\
&= \mathcal{L}_d^c[\![l_1]\!](\omega)(\delta)(\sigma) && \text{if } \beta = \mathbf{t} \\
&= \mathcal{L}_d^c[\![l_2]\!](\omega)(\delta)(\sigma) && \text{if } \beta = \mathbf{f}
\end{aligned}$$

where $\beta = \mathcal{L}_d^c[\![l_0]\!](\omega)(\delta)(\sigma)$

$$\begin{aligned}
\mathcal{L}_d^c[\![l_{1d'}^c \doteq l_{2d'}^c]\!](\omega)(\delta)(\sigma) &= \mathbf{t} && \text{if } \mathcal{L}_{d'}^c[\![l_1]\!](\omega)(\delta)(\sigma) = \mathcal{L}_{d'}^c[\![l_2]\!](\omega)(\delta)(\sigma) \\
&= \mathbf{f} && \text{otherwise}
\end{aligned}$$

(only if $d = \textsf{Bool}$)

$$\begin{aligned}
\mathcal{L}^c_d[\![l_1{}^c_d + l_2{}^c_d]\!](\omega)(\delta)(\sigma) &= \bot && \text{if } \mathcal{L}^c_d[\![l_1]\!](\omega)(\delta)(\sigma) = \bot \\
& && \text{or } \mathcal{L}^c_d[\![l_2]\!](\omega)(\delta)(\sigma) = \bot \\
&= \mathcal{L}^c_d[\![l_1]\!](\omega)(\delta)(\sigma) + \mathcal{L}^c_d[\![l_2]\!](\omega)(\delta)(\sigma) \\
& && \text{otherwise}
\end{aligned}$$

(only if $d = \mathsf{Int}$)

$\vdots$

$$\begin{aligned}
\mathcal{L}^c_d[\![l_1{}^c_{d'} < l_2{}^c_{d'}]\!](\omega)(\delta)(\sigma) &= \bot && \text{if } \mathcal{L}^c_{d'}[\![l_1]\!](\omega)(\delta)(\sigma) = \bot \\
& && \text{or } \mathcal{L}^c_{d'}[\![l_2]\!](\omega)(\delta)(\sigma) = \bot \\
&= \mathbf{t} && \text{if } \mathcal{L}^c_{d'}[\![l_1]\!](\omega)(\delta)(\sigma) < \mathcal{L}^c_{d'}[\![l_2]\!](\omega)(\delta)(\sigma) \\
&= \mathbf{f} && \text{otherwise}
\end{aligned}$$

(only if $d = \mathsf{Bool}$ and $d' = \mathsf{Int}$)

$$\begin{aligned}
\mathcal{L}^c_{\mathsf{Int}}[\![|l^c_{d^*}|]\!](\omega)(\delta)(\sigma) &= len^d(\mathcal{L}^c_{d^*}[\![l]\!](\omega)(\delta)(\sigma)) \\
\mathcal{L}^c_d[\![l_1{}^c_{d^*} \cdot l_2{}^c_{\mathsf{Int}}]\!](\omega)(\delta)(\sigma) &= elt^d(\mathcal{L}^c_{d^*}[\![l_1]\!](\omega)(\delta)(\sigma), \mathcal{L}^c_{\mathsf{Int}}[\![l_2]\!](\omega)(\delta)(\sigma))
\end{aligned}$$

These equations are just what one would expect, especially after having seen definition 3.12.

Lemma 4.11
If $\sigma \in \Sigma$, $\delta \in \Delta^c$, and $\omega \in \Omega$ are such that $OK(\sigma, \delta, \omega)$, then for every logical expression $l \in LExpr^c_d$ we have $\mathcal{L}[\![l]\!](\omega)(\delta)(\sigma) \in \sigma^{(d)}_\bot$, and for every expression $l \in LExpr^c_{d^*}$ we have $elt^d(\mathcal{L}[\![l]\!](\omega)(\delta)(\sigma), n) \in \sigma^{(d)}_\bot$, for every n.

Proof
Induction on the complexity of l. □

Definition 4.12
Now we can define the semantics of assertions in terms of the function

$$\mathcal{A}^c : Ass^c \to \Omega \to \Delta^c \to \Sigma \to \mathbf{B}$$

as follows:

$$\begin{aligned}
\mathcal{A}^c[\![l^c_{\mathsf{Bool}}]\!](v)(\omega)(\delta)(\sigma) &= \mathbf{t} && \text{if } \mathcal{L}^c_{\mathsf{Bool}}[\![l]\!](\omega)(\delta)(\sigma) = \mathbf{t} \\
&= \mathbf{f} && \text{otherwise}
\end{aligned}$$

$$\begin{aligned}
\mathcal{A}^c[\![\neg P^c]\!](v)(\omega)(\delta)(\sigma) &= \mathbf{f} && \text{if } \mathcal{A}^c[\![P]\!](v)(\omega)(\delta)(\sigma) = \mathbf{t} \\
&= \mathbf{t} && \text{otherwise}
\end{aligned}$$

$$\begin{aligned}
\mathcal{A}^c[\![\forall z_d\ P^c]\!](v)(\omega)(\delta)(\sigma) &= \mathbf{t} && \text{if for all } \beta \in \sigma^{(d)} \text{ we have} \\
& && \quad \mathcal{A}^c[\![P^c]\!](v)(\omega\{\beta/z\})(\delta)(\sigma) = \mathbf{t} \\
&= \mathbf{f} && \text{otherwise}
\end{aligned}$$

$$
\begin{array}{lcll}
\mathcal{A}^c[\![\exists z_d\ P^c]\!](v)(\omega)(\delta)(\sigma) & = & \mathbf{t} & \text{if there is a } \beta \in \sigma^{(d)} \text{ such that} \\
 & & & \quad \mathcal{A}^c[\![P^c]\!](v)(\omega\{\beta/z\})(\delta)(\sigma) = \mathbf{t} \\
 & = & \mathbf{f} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
\mathcal{A}^c[\![\forall z_{d^*}\ P^c]\!](\omega)(\delta)(\sigma) & = & \mathbf{t} & \text{if for all } \alpha \in \mathbf{O}^{d^*} \text{ such that} \\
 & & & \forall n \in \mathbf{Z}\ elt(\alpha, n) \in \sigma_{\perp}^{(d)} \text{ we have} \\
 & & & \mathcal{A}^c[\![P^c]\!](\omega\{\alpha/z\})(\delta)(\sigma) = \mathbf{t} \\
 & = & \mathbf{f} & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lcll}
\mathcal{A}^c[\![\exists z_{d^*}\ P^c]\!](\omega)(\delta)(\sigma) & = & \mathbf{t} & \text{if there is an } \alpha \in \mathbf{O}^{d^*} \text{ such that} \\
 & & & \forall n \in \mathbf{Z}\ elt(\alpha, n) \in \sigma_{\perp}^{(d)} \text{ and} \\
 & & & \mathcal{A}^c[\![P^c]\!](\omega\{\alpha/z\})(\delta)(\sigma) = \mathbf{t} \\
 & = & \mathbf{f} & \text{otherwise}
\end{array}
$$

A few remarks should be made here.

- Note that the possible values of a boolean logical expression are $\mathbf{t}$, $\mathbf{f}$, and $\perp$. If such an expression is viewed as an assertion, only $\mathbf{t}$ and $\mathbf{f}$ remain. If viewed as an expression it yields $\perp$, as an assertion it delivers $\mathbf{f}$.
- It is very important to note that in assertions of the form $\forall z_d\ P$ and $\exists z_d\ P$ the quantification ranges only over the *existing*, non-nil objects of the appropriate type. In assertions of the form $\forall z_a\ P$ and $\exists z_a\ P$ (where $a = d^*$, for some $d \in C$) the quantification ranges over sequences of existing objects, possibly nil.

Example 4.13
The formula

$$v \xrightarrow{x} w$$

from [7] can be expressed in our new assertion language in the following way:

$$\exists z_{d^*} \Big(z \cdot 1 \doteq v \wedge z \cdot |z| \doteq w \wedge \forall n\ (0 < n \wedge n < |z|) \rightarrow (z \cdot n).x \doteq z \cdot (n+1) \Big)$$

Here n denotes a logical variable ranging over integers. This formula expresses that the object w can be reached from v by a "x-path".

Example 4.14
There are no logical expressions in the language to construct a sequence with one specific element (a singleton). However, if we want to say that property P holds for

the singleton whose element is given by the logical expression l, we can do this as follows:

$$\exists z_{d^*} \; |z| \doteq 1 \wedge z \cdot 1 \doteq l \wedge P(z)$$

or equivalently:

$$\forall z_{d^*} \; (|z| \doteq 1 \wedge z \cdot 1 \doteq l) \rightarrow P(z).$$

A similar procedure is possible for the empty sequence and for the concatenation of two sequences. Furthermore we can see whether two sequences are equal by checking if they have the same lengths and whether their corresponding elements are equal. (Direct ways of expressing the above things could be included in the assertion language, but they would make the substitution operation $[\mathsf{new}/u]$ in definitions 5.13 and 5.15 much more complicated.)

Definition 4.15
Finally we define the notion of *truth* and *validity* of correctness formulae.

- We say that a correctness formula of the form P^c is *true* with respect to a valuation ω, a context δ^c, and a state σ, written as $\omega, \delta, \sigma \models P$, if $OK(\sigma, \delta, \omega)$ and
$$\mathcal{A}^c[\![P]\!](\omega)(\delta)(\sigma) = \mathbf{t}$$
- We call a correctness formula of the form P^c *valid*, written as $\models P$, if it is true with respect to every ω, δ^c, and σ such that $OK(\sigma, \delta, \omega)$.
- A correctness formula of the form $\{P^c\}\rho^c\{Q^c\}$ is called true with respect to an environment γ, a valuation ω, a context δ^c, and a state σ, written as $\gamma, \omega, \delta, \sigma \models \{P\}\rho\{Q\}$, if $\omega, \delta, \sigma \models P$ implies that for the state $\sigma' = \mathcal{P}^c[\![\rho]\!](\gamma)(\delta)(\sigma)$ we have
$$\sigma' \neq \perp \Rightarrow \omega, \delta, \sigma' \models Q.$$
- We define a correctness formula of the form $\{P^c\}\rho^c\{Q^c\}$ to be valid with respect to an environment γ, written as $\gamma \models \{P\}\rho\{Q\}$, if we have $\gamma, \omega, \delta, \sigma \models \{P\}\rho\{Q\}$ for every ω, δ^c, and σ. We call such a correctness formula simply valid if it is valid with respect to every environment.

5 The proof system

In this section we shall present a number of axioms and rules that can be used to derive correctness formulae. For each axiom and rule we shall give its justification by proving that it is valid. Note that axioms are correctness formulae so we have already defined what validity means for them. We call a proof rule valid if for every environment γ the validity of the premisses of the rule with respect to γ implies the validity of the conclusion with respect to γ. The consequence of the validity of all the axioms and rules will be that our proof system is *sound*, i.e., that if we can derive a correctness formula (without any further assumptions), this correctness formula will be valid. (There is one rule in the system that cannot be proved valid in isolation: the recursion rule (MR) in definition 5.33. It will get a special treatment in the soundness proof of the whole proof system (see theorem 5.40).)

5.1 Simple assignments

Definition 5.1
We shall call a statement a *simple assignment* if it is of the form $x \leftarrow e$ or $u \leftarrow e$ (that is, it uses the first form of a side effect expression: the one without a side effect).

5.1.1 Simple assignment to a temporary variable

Definition 5.2
Our first axiom deals with the case that the target variable is a temporary variable:

$$\left\{P^c[e^c_d/u_d]\right\} \langle U|c : u_d \leftarrow e^c_d\rangle \left\{P^c\right\} \tag{SAT}$$

Here the notation $P[e/u]$ means: P in which e is substituted for x. We shall formalize that notion in the next definition. (Note that this definition merely asserts that the name (SAT) refers to the class of formulae of the form listed above.)

Definition 5.3
We shall define the substitution operation $[e/u]$ first for logical expressions:

$$\begin{array}{lcll}
x\ [e/u] & = & x & \\
u\ [e/u] & = & e & \\
u'\ [e/u] & = & u' & \text{if } u' \neq u \\
z\ [e/u] & = & z & \\
l\ [e/u] & = & l & \text{if } l = \mathsf{nil}, \mathsf{self}, \mathsf{true}, \mathsf{false} \\
n\ [e/u] & = & n & \\
l\,.\,x\,[e/u] & = & (l[e/u])\,.\,x &
\end{array}$$

$$\begin{array}{lcl}
\textsf{if } l_0 \textsf{ then } l_1 \textsf{ else } l_2 \textsf{ fi}[e/u] & = & \textsf{if } l_0[e/u] \textsf{ then } l_1[e/u] \textsf{ else } l_2[e/u] \textsf{ fi}
\end{array}$$

$$\begin{array}{lcl}
(l_1 \doteq l_2)[e/u] & = & (l_1[e/u]) \doteq (l_2[e/u]) \\
(l_1 + l_2)[e/u] & = & (l_1[e/u]) + (l_2[e/u]) \\
\vdots & & \\
(l_1 < l_2)[e/u] & = & (l_1[e/u]) < (l_2[e/u]) \\
\vdots & &
\end{array}$$

$$\begin{array}{lcl}
|l| \quad [e/u] & = & |l[e/u]| \\
(l_1 \cdot l_2)[e/u] & = & (l_1[e/u] \cdot l_2[e/u])
\end{array}$$

Now we define this substitution for assertions other than logical expressions:

$$\begin{array}{lcl}
(P \rightarrow Q)[e/u] & = & (P[e/u]) \rightarrow (Q[e/u]) \\
(\neg P) \quad [e/u] & = & \neg(P[e/u]) \\
(\forall z\ P) \quad [e/u] & = & \forall z\ (P[e/u]) \\
(\exists z\ P) \quad [e/u] & = & \exists z\ (P[e/u])
\end{array}$$

This definition can be summarized by saying that we can perform the substitution $[e/u]$ by replacing u by e everywhere in the expression or assertion at hand. However, this will not be true for the notions of substitution that we will define in the sequel, despite the fact that we use a very similar notation to indicate those substitutions.

In the following lemma we express the most important characteristic of the substitution $[e/u]$. Informally spoken, for any logical expression or assertion, the substituted form has the same value in the state *before* the assignment as the unsubstituted form has in the state *after* the assignment.

Lemma 5.4
Consider the assignment statement $u_d \leftarrow e^c_d$. Let $\gamma \in \Gamma$, $\sigma \in \Sigma$ and $\delta \in \Delta^c$ be arbitrary, and let

$$\sigma' = \mathcal{S}[\![u \leftarrow e]\!](\gamma)(\delta)(\sigma).$$

Then we have the following facts:

1. For every logical expression $l^c_{d'}$ and every valuation ω

$$\mathcal{L}[\![l[e/u]]\!](\omega)(\delta)(\sigma) = \mathcal{L}[\![l]\!](\omega)(\delta)(\sigma').$$

2. For every assertion P^c, every valuation ω

$$\mathcal{A}[\![P[e/u]]\!](\omega)(\delta)(\sigma) = \mathcal{A}[\![P]\!](\omega)(\delta)(\sigma').$$

Proof
First we observe that $\sigma' = \mathcal{S}[\![u \leftarrow e]\!](\gamma)(\delta)(\sigma)$ means that $\sigma' = \sigma\{\beta/u\}$, where $\beta = \mathcal{E}[\![e]\!](\delta)(\sigma)$.

Now we can prove the first part of the lemma by induction with respect to the structure of l. The only interesting case occurs when $l = u$ so that $l[e/u] = e$:

$$\begin{aligned}\mathcal{L}[\![e]\!](\omega)(\delta)(\sigma) &= \mathcal{E}[\![e]\!](\delta)(\sigma)\\ &= \beta\\ &= \sigma'_{(3)(d)}(u)\\ &= \mathcal{E}[\![u]\!](\delta)(\sigma')\\ &= \mathcal{L}[\![u]\!](\omega)(\delta)(\sigma')\end{aligned}$$

After that we can prove the second part of the lemma by a straightforward induction on the structure of P.

Of course, this lemma is easily extended to the case where instead of an assignment *statement* we take a *program* in which the statement is a simple assignment to a temporary variable:

Corollary 5.5
The axiom (SAT) is valid, that is, for every environment γ we have

$$\gamma \models \left\{P[e/u]\right\} \langle U|c : u \leftarrow e\rangle \left\{P\right\}.$$

□

Note that the corollary uses only one direction of the lemma. The two directions together say that $P[e/u]$ is the *weakest precondition* of the statement $u \leftarrow e$ with respect to the postcondition P.

5.1.2 Simple assignment to an instance variable

Definition 5.6
In the case that the target variable of an assignment statement is an instance variable, we use the following axiom:

$$\left\{P^c[e^c_d/x^c_d]\right\} \langle U|c : x^c_d \leftarrow e^c_d\rangle \left\{P^c\right\} \tag{SAI}$$

This looks very similar to our first axiom (SAT), but note that we have not yet defined what substitution means if we substitute an expression for an instance variable instead of a temporary variable. We shall do that now, and the difference will become clear immediately:

Definition 5.7
The substitution operation $[e/x]$ is defined as follows on logical expressions:

$$
\begin{array}{lcll}
x\ [e/x] & = & e & \\
x'\ [e/x] & = & x' & \text{if } x' \neq x \\
u\ [e/x] & = & u & \\
z\ [e/x] & = & z & \\
l\ [e/x] & = & l & \text{if } l = \mathsf{nil}, \mathsf{self}, \mathsf{true}, \mathsf{false} \\
n\ [e/x] & = & n & \\
l\,.\,x\ [e/x] & = & \multicolumn{2}{l}{\mathsf{if}\ (l[e/x]) \doteq \mathsf{self}\ \mathsf{then}\ e\ \mathsf{else}\ (l[e/x])\,.\,x\ \mathsf{fi}} \\
l\,.\,x'[e/x] & = & (l[e/x])\,.\,x' & \text{if } x' \neq x
\end{array}
$$

$$\mathsf{if}\ l_0\ \mathsf{then}\ l_1\ \mathsf{else}\ l_2\ \mathsf{fi}[e/x] \quad = \quad \mathsf{if}\ l_0[e/x]\ \mathsf{then}\ l_1[e/x]\ \mathsf{else}\ l_2[e/x]\ \mathsf{fi}$$

$$
\begin{array}{lcl}
(l_1 \doteq l_2)[e/x] & = & (l_1[e/x]) \doteq (l_2[e/x]) \\
(l_1 + l_2)[e/x] & = & (l_1[e/x]) + (l_2[e/x]) \\
\vdots & & \\
(l_1 < l_2)[e/x] & = & (l_1[e/x]) < (l_2[e/x]) \\
\vdots & & \\
|l|\ [e/x] & = & |l[e/x]| \\
(l_1 \cdot l_2)[e/x] & = & (l_1[e/x] \cdot l_2[e/x])
\end{array}
$$

The definition is extended to assertions other than logical expressions in the same way as before:

$$
\begin{array}{lcl}
(P \rightarrow Q)[e/x] & = & (P[e/x]) \rightarrow (Q[e/x]) \\
(\neg P)\ [e/x] & = & \neg(P[e/x]) \\
(\forall z\ P)\ [e/x] & = & \forall z\ (P[e/x]) \\
(\exists z\ P)\ [e/x] & = & \exists z\ (P[e/x])
\end{array}
$$

The most important aspect of this definition is certainly the conditional expression that turns up when we are dealing with a logical expression of the form $l\,.\,x$. This is necessary because a certain form of form of *aliasing* is possible: the situation that different expressions refer to the same variable. In the case of $l\,.\,x$, it is possible that, after substitution, l refers to the currently active object, so that $l\,.\,x$ is the same variable as x and should be substituted by e. It is also possible that, after substitution, l does not refer to the currently executing object, and in this case no substitution should take place. Since we cannot decide between these possibilities by the form of the expression only, a conditional expression is constructed which decides "dynamically".

Lemma 5.8
Consider the assignment statement $x_d^c \leftarrow e_d^c$. Let $\gamma \in \Gamma$, $\sigma \in \Sigma$, and $\delta \in \Delta^c$ be arbitrary, and let

$$\sigma' = \mathcal{S}[\![x \leftarrow e]\!](\gamma)(\delta)(\sigma).$$

Then we have the following facts:

1. For every logical expression $l_{d'}^c$ and every valuation ω

$$\mathcal{L}[\![l[e/x]]\!](\omega)(\delta)(\sigma) = \mathcal{L}[\![l]\!](\omega)(\delta)(\sigma').$$

2. For every assertion P^c and every valuation ω

$$\mathcal{A}[\![P[e/x]]\!](\omega)(\delta)(\sigma) = \mathcal{A}[\![P]\!](\omega)(\delta)(\sigma').$$

Proof
Like in lemma 5.4 we first note that $\sigma' = \sigma\{\beta/\delta_{(1)}, x\}$ where $\beta = \mathcal{E}[\![e]\!](\delta)(\sigma)$. The first part of the lemma is now proved by induction on the complexity of l. We shall only deal with the most interesting case: $l = l' \,.\, x$. The induction hypothesis tells us that $\mathcal{L}[\![l'[e/x]]\!](\omega)(\delta)(\sigma) = \mathcal{L}[\![l']\!](\omega)(\delta)(\sigma')$. Let us call this object β_0. Now if $\beta_0 = \delta_{(1)}$ then $\mathcal{L}[\![l' \,.\, x]\!](\omega)(\delta)(\sigma') = \sigma'_{(2)}(\delta_{(1)})(x) = \beta = \mathcal{L}[\![e]\!](\omega)(\delta)(\sigma)$. Otherwise we have $\mathcal{L}[\![l' \,.\, x]\!](\omega)(\delta)(\sigma') = \sigma'_{(2)}(\beta_0)(x) = \mathcal{L}[\![(l'[e/x]) \,.\, x]\!](\omega)(\delta)(\sigma)$. So $\mathcal{L}[\![\text{if } l'[e/x] \doteq \text{self then } e \text{ else } (l'[e/x]) \,.\, x \text{ fi}]\!](\omega)(\delta)(\sigma) = \mathcal{L}[\![l' \,.\, x]\!](\omega)(\delta)(\sigma')$.

The rest of the lemma is proved in a way similar to lemma 5.4. □

Again we can extend this to programs instead of statements:

Corollary 5.9
The axiom (SAI) is valid, that is, for every environment γ we have

$$\gamma \models \left\{P[e/x]\right\} \langle U|c : x \leftarrow e\rangle \left\{P\right\}.$$

□

Note that this corollary also uses only one direction of the corresponding lemma. Again the two directions together say that $P[e/x]$ is the weakest precondition of the statement $x \leftarrow e$ with respect to the postcondition P.

5.2 Creating new objects

5.2.1 Assigning a new object to a temporary variable

Definition 5.10
For an assignment of the form $u \leftarrow \mathsf{new}$ we have a axiom similar to the previous two:

$$\left\{P^c[\mathsf{new}_{c'}/u_{c'}]\right\} \langle U|c : u_{c'} \leftarrow \mathsf{new}_{c'}\rangle \left\{P^c\right\} \tag{NT}$$

Again we still have to define what this notion of substitution looks like, but first we shall define the substitution of an expression for a *logical* variable, because we shall need that later:

Definition 5.11
We define the substitution operation $[e/z]$ on logical expressions by:

$$\begin{array}{lcll}
x \; [e/z] & = & x & \\
u \; [e/z] & = & u & \\
z \; [e/z] & = & e & \\
z' \; [e/z] & = & z' & \text{if } z' \neq z \\
l' \; [e/z] & = & l' & \text{if } l' = \mathsf{nil}, \mathsf{self}, \mathsf{true}, \mathsf{false} \\
n \; [e/z] & = & n & \\
l'.x\,[e/z] & = & (l'[e/z]).x &
\end{array}$$

$$\mathsf{if}\ l_0\ \mathsf{then}\ l_1\ \mathsf{else}\ l_2\ \mathsf{fi}[e/z] \;=\; \mathsf{if}\ l_0[e/z]\ \mathsf{then}\ l_1[e/z]\ \mathsf{else}\ l_2[e/z]\ \mathsf{fi}$$

$$\begin{array}{lcl}
(l_1 \doteq l_2)[e/z] & = & (l_1[e/z]) \doteq (l_2[e/z]) \\
(l_1 + l_2)[e/z] & = & (l_1[e/z]) + (l_2[e/z]) \\
\quad\vdots & & \\
(l_1 < l_2)[e/z] & = & (l_1[e/z]) < (l_2[e/z]) \\
\quad\vdots & &
\end{array}$$

$$\begin{array}{lcl}
|l| \quad [e/z] & = & |l[e/z]| \\
(l_1 \cdot l_2)[e/z] & = & (l_1[e/z] \cdot l_2[e/z])
\end{array}$$

We extend this definition to assertions other than logical expressions as follows:

$$
\begin{array}{lcll}
(P \rightarrow Q)[e/z] & = & (P[e/z]) \rightarrow (Q[e/z]) & \\
(\neg P) \quad [e/z] & = & \neg(P[e/z]) & \\
(\forall z\ P) \ \ [e/z] & = & \forall z\ P & \\
(\forall z'\ P) \ \ [e/z] & = & \forall z'\ (P[e/z]) & \text{if } z' \neq z \\
(\exists z\ P) \ \ [e/z] & = & \exists z\ P & \\
(\exists z'\ P) \ \ [e/z] & = & \exists z'\ (P[e/z]) & \text{if } z' \neq z
\end{array}
$$

This definition can be summarized by observing that the substitution can be carried out by replacing z by e everywhere except in the scope of a quantifier where z is bound.

Lemma 5.12
Let $\sigma \in \Sigma$, $\delta \in \Delta^c$, $e \in Exp^c_d$, and $z \in LVar_d$ be arbitrary, and let $\beta = \mathcal{E}[\![e]\!](\delta)(\sigma)$. Then we have

1. For all $l \in LExp^c_{d'}$ and for all $\omega \in \Omega$:

$$\mathcal{L}[\![l[e/z]]\!](\omega)(\delta)(\sigma) = \mathcal{L}[\![l]\!](\omega\{\beta/z\})(\delta)(\sigma).$$

2. For all $P \in Ass^c$, for all $\omega \in \Omega$:

$$\mathcal{A}[\![P[e/z]]\!](\omega)(\delta)(\sigma) = \mathcal{A}[\![P]\!](\omega\{\beta/z\})(\delta)(\sigma).$$

Proof
A rather trivial induction on the complexity of l and P. □

Now we can define the substitution $[\mathsf{new}_c/u_c]$. We shall do this first for logical expressions. As with the notions of substitution used in the axioms for simple assignments, we want the expression after substitution to have the same meaning in a state before the assignment as the unsubstituted expression has in the state after the assignment. However, in the case of a new-assignment, there are expressions for which this is not possible, because they refer to the new object (in the new state) and there is no expression that could refer to that object in the old state, because it does not exist yet. Therefore the result of the substitution must be left undefined in some cases.

However we will show that we *are* able to carry out the substitution. The idea behind this is that in such an assertion the variable u referring to the new object can essentially occur only in a context where either one of its instance variables is referenced, or it is compared for equality with another expression. In both of these cases we can predict the outcome without having to refer to the new object.

Definition 5.13
Here comes the formal definition of the substitution $[\mathsf{new}/u]$ for logical expressions:

$$
\begin{array}{lll}
x\,[\mathsf{new}/u] & = & x \\
u\,[\mathsf{new}/u] & \multicolumn{2}{l}{\text{is undefined}} \\
u'[\mathsf{new}/u] & = & u \qquad \text{if } u' \neq u \\
z\,[\mathsf{new}/u] & = & z \\
l\,[\mathsf{new}/u] & = & l \qquad \text{if } l = \mathsf{nil}, \mathsf{self}, \mathsf{true}, \mathsf{false} \\
n\,[\mathsf{new}/u] & = & n
\end{array}
$$

$$
\begin{array}{llll}
x' \;\; .\, x[\mathsf{new}/u] & = & x' . x & \\
u \;\; .\, x[\mathsf{new}/u] & = & \mathsf{nil} & \\
u' \;\; .\, x[\mathsf{new}/u] & = & u' . x & \text{if } u' \neq u \\
z \;\; .\, x[\mathsf{new}/u] & = & z . x & \\
l \;\; .\, x[\mathsf{new}/u] & = & l . x & \text{if } l = \mathsf{nil}, \mathsf{self} \\
l . x' . x[\mathsf{new}/u] & = & (l . x'[\mathsf{new}/u]) . x &
\end{array}
$$

$$\left(\mathsf{if}\ l_0\ \mathsf{then}\ l_1\ \mathsf{else}\ l_2\ \mathsf{fi} . x\right)[\mathsf{new}/u]$$
$$= \mathsf{if}\ l_0[\mathsf{new}/u]\ \mathsf{then}\ (l_1 . x)[\mathsf{new}/u]\ \mathsf{else}\ (l_2 . x)[\mathsf{new}/u]\ \mathsf{fi}$$

$$\mathsf{if}\ l_0\ \mathsf{then}\ l_1\ \mathsf{else}\ l_2\ \mathsf{fi}[\mathsf{new}/u]$$
$$= \mathsf{if}\ l_0[\mathsf{new}/u]\ \mathsf{then}\ l_1[\mathsf{new}/u]\ \mathsf{else}\ l_2[\mathsf{new}/u]\ \mathsf{fi}$$

if the substitutions of the subexpressions are all defined,
otherwise undefined

$$\left(l_1 \doteq l_2\right)[\mathsf{new}/u] \;=\; \left(l_1[\mathsf{new}/u]\right) \doteq \left(l_2[\mathsf{new}/u]\right)$$

if neither l_1 nor l_2 is u or of the form $\mathsf{if}\ \ldots\ \mathsf{fi}$

$$\left(l_1 \doteq l_2\right)[\mathsf{new}/u] \;=\; \mathsf{false}$$

if either $l_1 = u$ and l_2 is not u or of the form $\mathsf{if}\ \ldots\ \mathsf{fi}$
or $l_2 = u$ and l_1 is not u or of the form $\mathsf{if}\ \ldots\ \mathsf{fi}$

$$\left(l_1 \doteq l_2\right)[\mathsf{new}/u] \;=\; \mathsf{true}$$

if $l_1 = l_2 = u$

$$\Big(\text{if } l_0 \text{ then } l_1 \text{ else } l_2 \text{ fi} \doteq l_3\Big)[\text{new}/u]$$

$$\begin{array}{rl} = & \text{if } l_0[\text{new}/u]\uparrow \\ & \text{then } \Big(l_3\uparrow\Big)[\text{new}/u] \\ & \text{else } \ \text{if } l_0[\text{new}/u] \\ & \qquad \text{then } \Big(l_1 \doteq l_3\Big)[\text{new}/u] \\ & \qquad \text{else } \Big(l_2 \doteq l_3\Big)[\text{new}/u] \\ & \qquad \text{fi} \\ & \text{fi} \end{array}$$

$$\Big(l_1 \doteq \text{if } l_0 \text{ then } l_2 \text{ else } l_3 \text{ fi}\Big)[\text{new}/u]$$

$$\begin{array}{rl} = & \text{if } l_0[\text{new}/u]\uparrow \\ & \text{then } \Big(l_1\uparrow\Big)[\text{new}/u] \\ & \text{else } \ \text{if } l_0[\text{new}/u] \\ & \qquad \text{then } \Big(l_1 \doteq l_2\Big)[\text{new}/u] \\ & \qquad \text{else } \Big(l_1 \doteq l_3\Big)[\text{new}/u] \\ & \qquad \text{fi} \\ & \text{fi} \end{array}$$

if l_1 is not of the form if ... fi

$$(l_1 + l_2)[\text{new}/u] = (l_1[\text{new}/u]) + (l_2[\text{new}/u])$$

$$\vdots$$

$$(l_1 < l_2)[\text{new}/u] = (l_1[\text{new}/u]) < (l_2[\text{new}/u])$$

$$\vdots$$

$$|l| \quad [\text{new}/u] = |l[\text{new}/u]|$$

$$(l_1 \cdot l_2)[\text{new}/u] = (l_1[\text{new}/u]) \cdot (l_2[\text{new}/u])$$

Lemma 5.14
Let $u \in TVar_d$ with $d \in C$ (i.e., $d \neq \text{Int}, \text{Bool}$).

1. For every logical expression l we have that $l[\text{new}/u]$ is defined if and only if l is *not* of the form indicated by the following BNF definition:

$$\begin{array}{rcl} lu & ::= & u \\ & | & \text{if } l_0 \text{ then } lu \text{ else } l_1 \text{ fi} \\ & | & \text{if } l_0 \text{ then } l_1 \text{ else } lu \text{ fi} \end{array}$$

2. If $\sigma \in \Sigma$, $\delta \in \Delta^c$, $\omega \in \Omega$, and $\gamma \in \Gamma$ are such that $OK(\sigma, \delta, \omega)$, and if $\sigma' = \mathcal{S}[\![u \leftarrow \mathsf{new}]\!](\gamma)(\delta)(\sigma)$ then for every logical expression l such that $l[\mathsf{new}/u]$ is defined we have

$$\mathcal{L}[\![l[\mathsf{new}/u]]\!](\omega)(\delta)(\sigma) = \mathcal{L}[\![l]\!](\omega)(\delta)(\sigma').$$

Proof
The first part is easily proved by induction on the complexity of l. For the second part we first observe that

$$\sigma' = \sigma\left\{\sigma_{(1)(d)} \cup \{\beta\} \;/\; d\right\} \left\{\beta \;/\; u\right\}$$

where $\beta = pick^d(\sigma_{(1)(d)})$, so $\beta \notin \sigma_{(1)(d)} \cup \{\perp\}$ (see definitions 3.13 and 3.14).

Now we can prove our lemma by induction on the complexity of l. In several places we need the information that $OK(\sigma, \delta, \omega)$ together with lemma 4.11 in order to prove that the result of an intermediate logical expression is not equal to β. Let us deal with one representative case: $l = x' \,.\, x$. Then $l[\mathsf{new}/u] = l = x' \,.\, x$. Now the induction hypothesis tells us that $\mathcal{L}[\![x']\!](\omega)(\delta)(\sigma) = \mathcal{L}[\![x']\!](\omega)(\delta)(\sigma')$. If we put this equal to β' then we know $\beta' \neq \beta$ because lemma 4.11 tells us that $\beta' \in \sigma_{(1)(d)} \cup \{\perp\}$. Therefore we have $\mathcal{L}[\![x' \,.\, x]\!](\omega)(\delta)(\sigma) = \sigma_{(2)}(\beta')(x) = \sigma'_{(2)}(\beta')(x) = \mathcal{L}[\![x' \,.\, x]\!](\omega)(\delta)(\sigma')$. □

Definition 5.15
We extend the substitution operation $[\mathsf{new}/u]$ to assertions other than logical expressions as follows (we assume that the type of u is $d \in C$):

$$\begin{aligned}
(P \to Q)[\mathsf{new}/u] &= (P[\mathsf{new}/u]) \to (Q[\mathsf{new}/u]) \\
(\neg P) \quad [\mathsf{new}/u] &= \neg(P[\mathsf{new}/u]) \\
(\forall z_d\ P)\ [\mathsf{new}/u] &= (\forall z(P[\mathsf{new}/u])) \wedge (P[u/z][\mathsf{new}/u]) \\
(\forall z_{d^*}\ P)\, [\mathsf{new}/u] &= (\forall z\ \forall z'_{\mathsf{Bool}^*}\ |z| \doteq |z'| \to (P[z', u/z][\mathsf{new}/u])) \\
(\forall z_a\ P)\ [\mathsf{new}/u] &= (\forall z(P[\mathsf{new}/u])) \qquad \text{if } a \neq d, d^* \\
(\exists z_d\ P)\ [\mathsf{new}/u] &= (\exists z(P[\mathsf{new}/u])) \vee (P[u/z][\mathsf{new}/u]) \\
(\exists z_{d^*}\ P)\, [\mathsf{new}/u] &= (\exists z\ \exists z'_{\mathsf{Bool}^*}\ |z| \doteq |z'| \wedge (P[z', u/z][\mathsf{new}/u])) \\
(\exists z_a\ P)\ [\mathsf{new}/u] &= (\exists z(P[\mathsf{new}/u])) \qquad \text{if } a \neq d, d^*
\end{aligned}$$

Here we choose for z' the first variable from $LVar_{\mathsf{Bool}^*}$ that does not occur in P. The idea is that z and z' together code a sequence of objects in the state after the new-statement. At the places where z' yields **t** the value of the coded sequence is the newly created object. Where z' yields **f** the value of the coded sequence is the same as the value of z and where z' delivers $\perp$ the coded sequences also yields $\perp$.

We still have to define the substitution operation $[z', u/z]$ and we shall do that now:

Definition 5.16
Let $d \in C$, $u \in TVar_d$, $z \in LVar_{d^*}$, and $z' \in LVar_{\mathsf{Bool}^*}$. For logical expressions we define the operation $[z', u/z]$ as follows:

$$\begin{array}{llll}
e & [z',u/z] & = & e \\
z & [z',u/z] & \multicolumn{2}{l}{\text{is undefined}} \\
z'' & [z',u/z] & = & z'' \qquad\qquad\qquad \text{if } z'' \neq z \\
l\,.\,x & [z',u/z] & = & (l[z',u/z])\,.\,x \\
|z| & [z',u/z] & = & |z| \\
|l| & [z',u/z] & = & |l[z',u/z]| \qquad\qquad \text{if } l \neq z \\
(z \cdot l_2) & [z',u/z] & = & \mathsf{if}\ z' \cdot (l_2[z',u/z])\ \mathsf{then}\ u\ \mathsf{else}\ z \cdot (l_2[z',u/z])\ \mathsf{fi} \\
(l_1 \cdot l_2) & [z',u/z] & = & (l_1[z',u/z]) \cdot (l_2[z',u/z]) \qquad \text{if } l_1 \neq z
\end{array}$$

$$\mathsf{if}\ l_0\ \mathsf{then}\ l_1\ \mathsf{else}\ l_2\ \mathsf{fi}\ [z',u/z] =$$
$$\mathsf{if}\ (l_0[z',u/z])\ \mathsf{then}\ (l_1[z',u/z])\ \mathsf{else}\ (l_2[z',u/z])\ \mathsf{fi}$$

$$\begin{array}{lll}
(l_1 \doteq l_2)[z',u/z] & = & (l_1[z',u/z]) \doteq (l_2[z',u/z]) \\
(l_1 + l_2)[z',u/z] & = & (l_1[z',u/z]) + (l_2[z',u/z]) \\
\vdots & & \\
(l_1 < l_2)[z',u/z] & = & (l_1[z',u/z]) < (l_2[z',u/z]) \\
\vdots & & \\
|l|[z',u/z] & = & |l[z',u/z]| \\
(l_1 \cdot l_2)[z',u/z] & = & (l_1[z',u/z] \cdot l_2[z',u/z])
\end{array}$$

We extend this definition to assertions other than logical expressions as follows:

$$\begin{array}{ll}
(P \rightarrow Q)[z',u/z] = (P[z',u/z]) \rightarrow (Q[z',u/z]) & \\
(\neg P) \quad [z',u/z] = \neg(P[z',u/z]) & \\
(\forall z\ P) \quad [z',u/z] = (\forall z\ P) & \\
(\forall z''\ P) \quad [z',u/z] = (\forall z''\ (P[z',u/z])) & \text{if } z'' \neq z \\
(\exists z\ P) \quad [z',u/z] = (\exists z\ P) & \\
(\exists z''\ P) \quad [z',u/z] = (\exists z''\ (P[z',u/z])) & \text{if } z'' \neq z
\end{array}$$

Lemma 5.17
Let u, z, z' be as in definition 5.16. Let $\sigma \in \Sigma$, $\delta \in \Delta^c$, $\omega \in \Omega$, and take $\alpha = \omega_{(d^*)}(z)$, $\alpha' = \omega_{(\mathsf{Bool}^*)}(z')$, $\beta = \sigma_{(3)(d)}(u)$. Suppose that $len^d(\alpha) = len^{\mathsf{Bool}}(\alpha')$. Define $\alpha'' \in \mathbf{O}^{d^*}$ to be the sequence that satisfies (for all $n \in \mathbf{Z}$):

$$\begin{array}{llll} len(\alpha'') & = & len(\alpha) & \\ elt(\alpha'', n) & = & \beta & \text{if } elt(\alpha', n) = \mathbf{t} \\ elt(\alpha'', n) & = & elt(\alpha, n) & \text{if } elt(\alpha', n) = \mathbf{f} \\ elt(\alpha'', n) & = & \bot & \text{if } elt(\alpha', n) = \bot \end{array}$$

and take $\omega' = \omega\{\alpha''/z\}$.

Then we have:

1. For every $l \in LExpr_a^c$ such that $l \not\equiv z$:
$$\mathcal{L}_a^c[\![l[z', u/z]]\!](\omega)(\delta)(\sigma) = \mathcal{L}_a^c[\![l]\!](\omega')(\delta)(\sigma).$$
2. For every $P \in Ass^c$ such that z' does not occur in P:
$$\mathcal{A}^c[\![P[z', u/z]]\!](\omega)(\delta)(\sigma) = \mathcal{A}^c[\![P]\!](\omega')(\delta)(\sigma).$$

Proof
The proof consists of a quite easy induction on the complexity of l and P respectively. Of course, the only interesting case is when l is of the form $z \cdot l_2$. Note that the condition on z' is necessary to exclude assertions of the form $\forall z'\ P$ or $\exists z'\ P$. □

Lemma 5.18
Let $\sigma \in \Sigma$, $\delta \in \Delta^c$, $\omega \in \Omega$ such that $OK(\sigma, \delta, \omega)$. Let $d \in C$, $u \in TVar_d$, $\gamma \in \Gamma$ and define $\sigma' = \mathcal{S}^c[\![u \leftarrow \mathsf{new}]\!](\gamma)(\delta)(\sigma)$. Then for every assertion $P \in Ass^c$ we have
$$\mathcal{A}^c[\![P[\mathsf{new}/u]]\!](\omega)(\delta)(\sigma) = \mathcal{A}^c[\![P]\!](\omega)(\delta)(\sigma').$$

Proof
Again we use induction on the complexity of P. The only case which is not yet clear from the first approach is quantification over sequences, so let us consider the case where $P = \forall z_{d^*}\ Q$. Take $\beta = f^d(\sigma^{(d)})$, so that $\sigma'^{(d)} = \sigma^{(d)} \cup \{\beta\}$ and $\beta = \sigma'_{(3)(d)}(u)$, and let z' be the first variable from $LVar_{\mathsf{Bool}^*}$ that does not occur in Q.

Now suppose that
$$\mathcal{A}[\![(\forall z_{d^*}\ Q)[\mathsf{new}/u]]\!](\omega)(\delta)(\sigma) = \mathbf{t}.$$
We shall prove that
$$\mathcal{A}[\![\forall z_{d^*}\ Q]\!](\omega)(\delta)(\sigma') = \mathbf{t}$$
so we have to show that for every $\alpha'' \in \mathbf{O}^{d^*}$ such that $elt(\alpha'', n) \in \sigma'^{(d)}_{\bot}$ for all $n \in \mathbf{Z}$, it is the case that $\mathcal{A}[\![Q]\!](\omega\{\alpha''/z\})(\delta)(\sigma') = \mathbf{t}$. If we have such an α'', we can define $\alpha \in \mathbf{O}^{d^*}$ and $\alpha' \in \mathbf{O}^{\mathsf{Bool}^*}$ as follows:

$$
\begin{array}{lllllll}
len(\alpha) = len(\alpha') = len(\alpha'') & & & & & & \\
elt(\alpha, n) & = & \bot, & elt(\alpha', n) & = & \mathbf{t} & \text{if } elt(\alpha'', n) = \beta \\
elt(\alpha, n) & = & elt(\alpha'', n), & elt(\alpha', n) & = & \mathbf{f} & \text{if } 1 \le n \le len(\alpha'') \\
 & & & & & & \text{and } elt(\alpha'', n) \neq \beta
\end{array}
$$

Now because

$$\mathcal{A}[\![\forall z_d \cdot \forall z'_{\mathsf{Bool}} \cdot |z| \doteq |z'| \to Q[z', u/z][\mathsf{new}/u]]\!](\omega)(\delta)(\sigma) = \mathbf{t}$$

and because α and α' have equal length and do not have elements outside $\sigma_\bot^{(d)}$ and $\sigma_\bot^{(\mathsf{Bool})}$ respectively, we know that

$$\mathcal{A}[\![Q[z', u/z][\mathsf{new}/u]]\!](\omega\{\alpha/z\}\{\alpha'/z'\})(\delta)(\sigma) = \mathbf{t}.$$

The induction hypothesis then tells us that

$$\mathcal{A}[\![Q[z', u/z]]\!](\omega\{\alpha/z\}\{\alpha'/z'\})(\delta)(\sigma') = \mathbf{t}.$$

Finally we can apply lemma 5.17 and use the fact that z' does not occur in Q, to see that

$$\mathcal{A}[\![Q]\!](\omega\{\alpha''/z\})(\delta)(\sigma') = \mathbf{t}.$$

To prove that $\mathcal{A}[\![\forall z_d \cdot\ Q]\!](\omega)(\delta)(\sigma') = \mathbf{t}$ implies $\mathcal{A}[\![(\forall z_d \cdot\ Q)[\mathsf{new}/u]]\!](\omega)(\delta)(\sigma) = \mathbf{t}$ involves reasoning in the other direction, in particular to find a suitable α'' for each pair α, α' that satisfies certain conditions. We omit further details. □

Again we extend this result to the case of programs:

Corollary 5.19
The axiom (NT) is valid, that is, for every environment γ we have

$$\gamma \models \left\{P[\mathsf{new}/u]\right\} \langle U|c : u \leftarrow \mathsf{new}\rangle \left\{P\right\}.$$

□

5.2.2 Assigning a new object to an instance variable

Definition 5.20
If our assignment is of the form $x \leftarrow$ new we have the following axiom:

$$\left\{P^c[\mathsf{new}_{c'}/x^c_{c'}]\right\} \langle U|c : x^c_{c'} \leftarrow \mathsf{new}_{c'}\rangle \left\{P^c\right\} \qquad \text{(NI)}$$

Fortunately, after having worked through the previous subsection, this new axiom is simple to define and to prove valid.

Definition 5.21
The substitution operation $[\mathsf{new}_{c'}/x^c_{c'}]$ is defined by:

$$P[\mathsf{new}_{c'}/x^c_{c'}] = P[u_{c'}/x^c_{c'}]\,[\mathsf{new}_{c'}/u_{c'}]$$

where $u_{c'}$ is a temporary variable that does not occur in P. (It is easy to see that this definition does not depend on the actual u used.)

Lemma 5.22
Let $\sigma \in \Sigma$, $\delta \in \Delta^c$ and $\omega \in \Omega$ be such that $OK(\sigma, \delta, \omega)$. Let $\gamma \in \Gamma$, $d \in C$, $x \in IVar^c_d$, and define $\sigma' = \mathcal{S}[\![x \leftarrow \mathsf{new}]\!](\gamma)(\delta)(\sigma)$. Then for every assertion P^c we have

$$\omega, \delta, \sigma \models P[\mathsf{new}/x] \iff \omega, \delta, \sigma' \models P.$$

Proof
Choose some $u \in TVar_d$ which does not occur in P, so that we have $P[\mathsf{new}/x] = P[u/x][\mathsf{new}/u]$. Let $\sigma'' = \mathcal{S}[\![u \leftarrow \mathsf{new}; x \leftarrow u]\!](\gamma)(\delta)(\sigma)$. We have by lemma 5.8 and lemma 5.18 that $\omega, \delta, \sigma \models P[u/x][\mathsf{new}/u] \iff \omega, \delta, \sigma'' \models P$.

Now if $\beta = pick^d(\sigma^{(d)})$ then we have $\sigma' = \sigma\{\beta/\delta_{(1)}, x\}$ and $\sigma'' = \sigma\{\beta/u\}\{\beta/\delta_{(1)}, x\}$, so that $\sigma'' = \sigma'\{\beta/u\}$. Because u does not occur in P we have $\omega, \delta, \sigma' \models P \iff \omega, \delta, \sigma'' \models P$, and the result of the lemma follows. □

Corollary 5.23
The axiom (NI) is valid, that is, for every environment γ we have

$$\gamma \models \Big\{P[\mathsf{new}/x]\Big\}\,\langle U|c : x \leftarrow \mathsf{new}\rangle\,\Big\{P\Big\}.$$

□

5.3 Sending messages

In this subsection we present some proof rules for verifying the third kind of assignments: the ones where a message is sent and the result stored in the variable on the left hand side. We start with a rule for a non-recursive method and later on we show how to deal with recursion.

Definition 5.24
For the statement $x \leftarrow e_0!m(e_1, \ldots, e_n)$, where $x \in IVar^c_{d_0}$, $m \in MName^{c'}_{d_0,\ldots,d_n}$, $e_0 \in Exp^c_{c'}$ and $e_i \in Exp^c_{d_i}$ for $i = 1, \ldots, n$, we have the following proof rule:

$$\frac{\Big\{P^{c'} \wedge \bigwedge_{i=1}^{k} v_i \doteq \mathsf{nil}\Big\}\langle U|c' : S\rangle\Big\{Q^{c'}[e/r]\Big\},\quad Q[\bar{e}/\mathsf{self}, \bar{u}][\bar{f}/\bar{z}] \rightarrow R^c[r/x]}{\Big\{P[\bar{e}/\mathsf{self}, \bar{u}][\bar{f}/\bar{z}]\Big\}\langle U|c : x \leftarrow e_0!m(e_1, \ldots, e_n)\rangle\Big\{R\Big\}} \qquad \text{(MI)}$$

where $S \in Stat^{c'}$ and $e \in Exp^{c'}_{d_0}$ are the statement and expression occurring in the definition of the method m in the unit U, $u_1, \ldots, u_n$ are its formal parameters, $v_1, \ldots, v_k$ is a row of temporary variables that are *not* formal parameters ($k \geq 0$), r is a logical variable of type d_0 that does not occur in R, $\bar{f}$ is an arbitrary row of expressions (*not* logical expressions) in class c, and $\bar{z}$ is a row of logical variables, mutually different and different from r, such that the type of each z_i is the same as the type of the corresponding f_i. Furthermore, $[\bar{e}/\mathsf{self}, \bar{u}]$ stands for a *simultaneous* substitution having the "components" $[e_0/\mathsf{self}], [e_1/u_1], \ldots, [e_n/u_n]$ (a formal definition will follow). We require that no temporary variables other than the formal parameters $u_1, \ldots, u_n$ occur in P or Q.

We still have to define precisely what $[\bar{e}/\mathsf{self}, \bar{u}]$ means, but before doing that let us give some informal explanation of the above rule. When a statement as above is executed, several things happen. First, control is transferred from the sender of the message to the receiver (context switching). The formal parameters of the receiver are initialized with the values of the expressions that form the actual parameters of the message and the other temporary variables are initialized to nil. Then the body S of the method is executed. After that the result expression e is evaluated, control is returned to the sender, the temporary variables are restored, and the result object is assigned to the variable x.

The first thing, the context switching, is represented by the substitution $[e_0/\mathsf{self}]$. A little more precisely, an assertion P as seen from the receiver's viewpoint is equivalent to $P[e_0/\mathsf{self}]$ from the viewpoint of the sender. Note that this substitution also changes the class of the assertion: $P[e_0/\mathsf{self}] \in Ass^c$ whereas $P \in Ass^{c'}$. Now the passing of the parameters is simply represented by the substitution $[e_1, \ldots, e_n/u_1, \ldots, u_n]$. Therefore after the parameters have been transferred to the receiver, P from the receiver's viewpoint corresponds to $P[\bar{e}/\mathsf{self}, \bar{u}]$ as seen by the sender. (Note that we really need *simultaneous* substitution here, because u_i might occur in an e_j with $j < i$, but it should not be substituted again.) In reasoning about the body of the method we may also use the information that temporary variables that are not parameters are initialized to nil.

The second thing to note is the way the result is passed back. Here the logical variable r plays an important role. This is best understood by imagining after the body S of the method the statement $r \leftarrow e$ (which is syntactically illegal, however, because r is a *logical* variable). In the sending object one could imagine the (equally illegal) statement $x \leftarrow r$. Now if the body S terminates in a state where $Q[e/r]$ holds (a premiss of the rule) then after this "virtual" statement $r \leftarrow e$ we would have a situation in which Q holds. Otherwise stated, the assertion Q describes the situation after executing the method body, in which the result is represented by the logical variable r, everything seen from the viewpoint of the receiver. Now if we context-switch this Q to the sender's side, and if it implies $R[r/x]$, then we know that after assigning the result to the variable x (our second imaginary assignment $x \leftarrow r$), the assertion R will hold.

Now we come to the role of $\bar{f}$ and $\bar{z}$. We know that during the evaluation of the method the sending object becomes blocked, that is, it cannot answer any incoming messages. Therefore its instance variables will not change in the meantime. The temporary variables will be restored after the method is executed, so these will also be unchanged and finally the symbol self will retain its meaning over the call. All the expressions in class c (and in particular the f_i) are built from these expressions plus some inherently constant expressions and therefore their value will not change during the call. However, the method can change the variables of other objects and new objects can be created, so that the properties of these unchanged expressions *can* change. In order to be able to make use of the fact that the expressions $\bar{f}$ are constant during the call, the rule offers the possibility to replace them temporarily by the logical variables $\bar{z}$, which are automatically constant. So, in reasoning from the receiver's viewpoint (in the rule this applies to the assertions P and Q) the value of the expression f_i is represented by z_i, and in context switching f_i comes in again by the substitution $[\bar{f}/\bar{z}]$. Note that the constancy of $\bar{f}$ is guaranteed up to the point where the result of the method is assigned to x, and that x may occur in f_i, so that it is possible to make use of the fact that x remains unchanged right up to the assignment of the result.

Definition 5.25
Now we define formally the substitution operation $[e/\mathsf{self}]$. First we do this for logical expressions:

$$\begin{aligned}
x\ &[e/\mathsf{self}] = e\,.\,x \\
u\ &[e/\mathsf{self}] = u \\
z\ &[e/\mathsf{self}] = z \\
\mathsf{self}&[e/\mathsf{self}] = e \\
l\ &[e/\mathsf{self}] = l \qquad\qquad \text{if } l = \mathsf{nil}, \mathsf{true}, \mathsf{false}, n \\
l\,.\,x&[e/\mathsf{self}] = (l[e/\mathsf{self}])\,.\,x
\end{aligned}$$

$$\mathsf{if}\ l_0\ \mathsf{then}\ l_1\ \mathsf{else}\ l_2\ \mathsf{fi}[e/\mathsf{self}] = \mathsf{if}\ l_0[e/\mathsf{self}]\ \mathsf{then}\ l_1[e/\mathsf{self}]\ \mathsf{else}\ l_2[e/\mathsf{self}]\ \mathsf{fi}$$

$$\begin{aligned}
(l_1 \doteq l_2)[e/\mathsf{self}] &= (l_1[e/\mathsf{self}]) \doteq (l_2[e/\mathsf{self}]) \\
(l_1 + l_2)[e/\mathsf{self}] &= (l_1[e/\mathsf{self}]) + (l_2[e/\mathsf{self}]) \\
&\vdots \\
(l_1 < l_2)[e/\mathsf{self}] &= (l_1[e/\mathsf{self}]) < (l_2[e/\mathsf{self}]) \\
&\vdots
\end{aligned}$$

$$\begin{aligned}
|l|[e/\mathsf{self}] &= |l[e/\mathsf{self}]| \\
(l_1 \cdot l_2)[e/\mathsf{self}] &= (l_1[e/\mathsf{self}] \cdot l_2[e/\mathsf{self}])
\end{aligned}$$

Now we extend this to assertions other than logical expressions:

$$
\begin{aligned}
(P \rightarrow Q)[e/\mathsf{self}] &= (P[e/\mathsf{self}]) \rightarrow (Q[e/\mathsf{self}]) \\
(\neg P) \quad [e/\mathsf{self}] &= \neg(P[e/\mathsf{self}]) \\
(\forall z P) \quad [e/\mathsf{self}] &= \forall z(P[e/\mathsf{self}]) \\
(\exists z P) \quad [e/\mathsf{self}] &= \exists z(P[e/\mathsf{self}])
\end{aligned}
$$

Lemma 5.26

Let $\sigma \in \Sigma$, $\delta \in \Delta^c$, $e \in Exp^c_{c'}$, and define $\beta^{c'} = \mathcal{E}[\![e]\!](\delta)(\sigma)$. Let $\delta' \in \Delta^{c'}$ be such that $\delta'_{(1)} = \beta$. Then we have

1. For every logical expression $l^{c'}_d$ and every valuation ω
$$\mathcal{L}[\![l]\!](\omega)(\delta')(\sigma) = \mathcal{L}[\![l[e/\mathsf{self}]]\!](\omega)(\delta)(\sigma).$$
2. For every assertion $P^{c'}$ and every valuation ω
$$\mathcal{A}[\![P]\!](\omega)(\delta')(\sigma) = \mathcal{A}[\![P[e/\mathsf{self}]]\!](\omega)(\delta)(\sigma).$$

Proof

An easy induction on the complexity of l and P. □

Definition 5.27

Although the intention of simultaneous substitution is probably clear to the reader, we give its definition for the case in which we really need it here, for completeness' sake. Let $\bar{e} = e_0, \ldots, e_n$ and $\bar{u} = u_1, \ldots, u_n$. Then we define:

$$
\begin{aligned}
x \quad [\bar{e}/\mathsf{self}, \bar{u}] &= e_0 \,.\, x \\
u_i \quad [\bar{e}/\mathsf{self}, \bar{u}] &= e_i && \text{for } i = 1, \ldots, n \\
u \quad [\bar{e}/\mathsf{self}, \bar{u}] &= u && \text{if } u \notin \{u_1, \ldots, u_n\} \\
z \quad [\bar{e}/\mathsf{self}, \bar{u}] &= z \\
\mathsf{self}\,[\bar{e}/\mathsf{self}, \bar{u}] &= e_0 \\
l \quad [\bar{e}/\mathsf{self}, \bar{u}] &= l && \text{if } l = \mathsf{nil}, \mathsf{true}, \mathsf{false}, n \\
l \,.\, x[\bar{e}/\mathsf{self}, \bar{u}] &= (l[\bar{e}/\mathsf{self}, \bar{u}]) \,.\, x
\end{aligned}
$$

$$\mathsf{if}\ l_0\ \mathsf{then}\ l_1\ \mathsf{else}\ l_2\ \mathsf{fi}[\bar{e}/\mathsf{self}, \bar{u}] = \mathsf{if}\ l_0[\bar{e}/\mathsf{self}, \bar{u}]\ \mathsf{then}\ l_1[\bar{e}/\mathsf{self}, \bar{u}]\ \mathsf{else}\ l_2[\bar{e}/\mathsf{self}, \bar{u}]\ \mathsf{fi}$$

$$
\begin{aligned}
(l_1 \doteq l_2)[\bar{e}/\mathsf{self}, \bar{u}] &= (l_1[\bar{e}/\mathsf{self}, \bar{u}]) \doteq (l_2[\bar{e}/\mathsf{self}, \bar{u}]) \\
(l_1 + l_2)[\bar{e}/\mathsf{self}, \bar{u}] &= (l_1[\bar{e}/\mathsf{self}, \bar{u}]) + (l_2[\bar{e}/\mathsf{self}, \bar{u}]) \\
&\vdots \\
(l_1 < l_2)[\bar{e}/\mathsf{self}, \bar{u}] &= (l_1[\bar{e}/\mathsf{self}, \bar{u}]) < (l_2[\bar{e}/\mathsf{self}, \bar{u}]) \\
&\vdots
\end{aligned}
$$

$$\begin{array}{lcl} |l|[\bar{e}/\mathsf{self},\bar{u}] & = & |l[\bar{e}/\mathsf{self},\bar{u}]| \\ (l_1 \cdot l_2)[\bar{e}/\mathsf{self},\bar{u}] & = & (l_1[\bar{e}/\mathsf{self},\bar{u}] \cdot l_2[\bar{e}/\mathsf{self},\bar{u}]) \end{array}$$

Now we extend this to assertions other than logical expressions:

$$\begin{array}{lll} (P \to Q)[\bar{e}/\mathsf{self},\bar{u}] & = (P[\bar{e}/\mathsf{self},\bar{u}]) \to (Q[\bar{e}/\mathsf{self},\bar{u}]) \\ (\neg P) \quad [\bar{e}/\mathsf{self},\bar{u}] & = \neg(P[\bar{e}/\mathsf{self},\bar{u}]) \\ (\forall z P) \quad [\bar{e}/\mathsf{self},\bar{u}] & = \forall z(P[\bar{e}/\mathsf{self},\bar{u}]) \\ (\exists z P) \quad [\bar{e}/\mathsf{self},\bar{u}] & = \exists z(P[\bar{e}/\mathsf{self},\bar{u}]) \end{array}$$

Of course we also have a corresponding lemma:

Lemma 5.28
Let $\sigma \in \Sigma$, $\delta \in \Delta^c$ and $e_i \in Exp^c_{d_i}$ for $i = 0,\ldots,n$ (with $d_0 \in C$). Define $\beta_i = \mathcal{E}[\![e_i]\!](\delta)(\sigma)$. Let $\delta' \in \Delta^{d_0}$ be such that $\delta'_{(1)} = \beta_0$ and let $\sigma' = \sigma\{\beta_i/u_i\}_{i=1}^k$. Then we have

1. For every logical expression $l_d^{d_0}$ and every valuation ω
$$\mathcal{L}[\![l]\!](\omega)(\delta')(\sigma') = \mathcal{L}[\![l[\bar{e}/\mathsf{self},\bar{u}]]\!](\omega)(\delta)(\sigma).$$
2. For every assertion P^{d_0} and every valuation ω
$$\mathcal{A}[\![P]\!](\omega)(\delta')(\sigma') = \mathcal{A}[\![P[\bar{e}/\mathsf{self},\bar{u}]]\!](\omega)(\delta)(\sigma).$$

Proof
Again a quite simple induction on the complexity of l and P. □

Example 5.29
Let us illustrate the use of the rule (MI) by a small example. Consider the unit $U = c : \langle m \Leftarrow (u_0) : x_1 \leftarrow u_0 \uparrow x_2 \rangle$ and the program $\rho = \langle U | c : x_1 \leftarrow u_1!m(x_2)\rangle$. We want to show

$$\Big\{u_1 \,.\, x_1 \doteq x_1 \wedge \neg u_1 \doteq \mathsf{self}\Big\}\rho\Big\{u_1 \,.\, x_1 \doteq x_2 \wedge x_1 \doteq u_1 \,.\, x_2\Big\}.$$

So let us apply the rule (MI) with the following choices:

$$\begin{array}{lcll} P & = & x_1 \doteq z_1 \wedge \neg\mathsf{self} \doteq z_2 & \\ Q & = & x_1 \doteq u_0 \wedge r \doteq x_2 \wedge \neg\mathsf{self} \doteq z_2 & \\ R & = & u_1 \,.\, x_1 \doteq x_2 \wedge x_1 \doteq u_1 \,.\, x_2 & \\ k & = & 0 & \text{(we shall use no } v_i\text{)} \\ f_1 & = & x_1 & \text{(represented by } z_1 \text{ in } P \text{ and } Q\text{)} \\ f_2 & = & \mathsf{self} & \text{(represented by } z_2 \text{ in } P \text{ and } Q\text{)} \end{array}$$

Figure 2: The situation before and after sending the message (example 5.29)

First notice that $P[u_1, x_2/\mathsf{self}, u_0][x_1, \mathsf{self}/z_1, z_2] = u_1 . x_1 \doteq x_1 \wedge \neg u_1 \doteq \mathsf{self}$ so that the result of the rule is precisely what we want.

For the first premiss we have to prove

$$\Big\{x_1 \doteq z_1 \wedge \neg\mathsf{self} \doteq z_2\Big\}\langle U|c : x_1 \leftarrow u_0\rangle\Big\{x_1 \doteq u_0 \wedge x_2 \doteq x_2 \wedge \neg\mathsf{self} \doteq z_2\Big\}.$$

This is easily done with the axiom (SAI) and the rule of consequence (which will be introduced in definition 5.39).

With respect to the second premiss, we have

$$\begin{array}{lcl} Q[u_1, x_2/\mathsf{self}, u_0][x_1, \mathsf{self}/z_1, z_2] & = & u_1 . x_1 \doteq x_2 \wedge r \doteq u_1 . x_2 \wedge \neg u_1 \doteq \mathsf{self} \\ R[r/x_1] & = & \mathsf{if}\ u_1 \doteq \mathsf{self}\ \mathsf{then}\ r\ \mathsf{else}\ u_1 . x_1\ \mathsf{fi} \doteq x_2 \wedge r \doteq u_1 . x_2 \end{array}$$

It is quite clear that the first implies the second, and we can use this implication as an axiom (see definition 5.38).

Lemma 5.30
The proof rule (MI) is valid.

Proof
Consider the rule as listed in definition 5.24. Let $\gamma \in \Gamma$ and suppose that the premisses are valid with respect to γ. We shall prove that the conclusion is valid with respect to γ. So let $\sigma \in \Sigma$, $\delta \in \Delta^c$, and $\omega \in \Omega$ be such that $\sigma, \delta, \omega \models P[\bar{e}/\mathsf{self}, \bar{u}][\bar{f}/\bar{z}]$. Let $\gamma' = \mathcal{U}[\![U]\!](\gamma)$ and let $\sigma' = \mathcal{P}[\![\langle U|c : x \leftarrow e_0!m(e_1, \ldots, e_n)\rangle]\!](\gamma)(\delta)(\sigma)$. So $\sigma' = \mathcal{S}[\![x \leftarrow e_0!m(e_1, \ldots, e_n)]\!](\gamma')(\delta)(\sigma)$. We have to prove $\sigma', \delta, \omega \models R$.

Let $\omega' = \omega\{\mathcal{E}[\![f_i]\!](\delta)(\sigma)/z_i\}_{i=1}^{|\bar{f}|}$. Then lemma 5.12 gives us $\sigma, \delta, \omega' \models P[\bar{e}/\mathsf{self}, \bar{u}]$. Let $\beta_i = \mathcal{E}[\![e_i]\!](\delta)(\sigma)$ for $i = 0, \ldots, n$ and suppose that $\beta_0 \neq \perp$ and $\beta_0 \notin \delta_{(2)(c)}$ (otherwise we would have that $\sigma' = \perp$ and the result would be trivial). Define $\delta' = \Big\langle\beta_0, \delta_{(2)}\Big\{\delta_{(2)(c)} \cup \{\delta_{(1)}\}\Big/c\Big\}\Big\rangle$ and $\sigma_1 = \langle\sigma_{(1)}, \sigma_{(2)}, \sigma_{1(3)}\rangle$ where $\sigma_{1(3)(d_i)}(u_i) = \beta_i$ and $\sigma_{1(3)(d)}(u_d) = \perp$ if $u \notin \{u_1, \ldots, u_n\}$. Now because of lemma 5.28 and the fact

$$
\begin{array}{ccccc}
\sigma,\delta,\omega \models P[\bar{e}/\mathsf{self},\bar{u}][\bar{f}/\bar{z}] & \Rightarrow & \sigma,\delta,\omega' \models P[\bar{e}/\mathsf{self},\bar{u}] & \Rightarrow & \sigma_1,\delta',\omega' \models P \\
 & & & & \Downarrow \\
 & & & & \sigma_2,\delta',\omega' \models Q[e/r] \\
 & & & & \Downarrow \\
\sigma'',\delta,\omega_1 \models Q[\bar{e}/\mathsf{self},\bar{u}][\bar{f}/\bar{z}] & \Leftarrow & \sigma'',\delta,\omega_1' \models Q[\bar{e}/\mathsf{self},\bar{u}] & \Leftarrow & \sigma_2,\delta',\omega_1' \models Q \\
\Downarrow & & & & \\
\sigma'',\delta,\omega_1 \models R[r/x] & & & & \\
\Downarrow & & & & \\
\sigma',\delta,\omega_1 \models R & & & & \\
\Downarrow & & & & \\
\sigma',\delta,\omega \models R & & & &
\end{array}
$$

Figure 3: The structure of the proof of lemma 5.30.

that temporary variables other than the u_i may not occur in P, we have $\sigma_1, \delta', \omega' \models P$. We also know that $\sigma_1, \delta', \omega' \models v_i \doteq \mathsf{nil}$ for $i = 1, \ldots, k$ so $\sigma_1, \delta', \omega' \models P \wedge \bigwedge_{i=1}^{k} v_i \doteq \mathsf{nil}$.

Now because of the construction of γ' in definition 3.17 we know that $\gamma'(c', \bar{d})(m) = \mathcal{M}[\![(\bar{u}) : S \uparrow e]\!](\gamma')$ so we can refer directly to the method definition of m in U to see what $\gamma'(c', \bar{d})(m)$ does. So let us take $\sigma_2 = \mathcal{P}[\![\langle U | c' : S \rangle]\!](\gamma)(\delta')(\sigma_1)$, then $\sigma_2 = \mathcal{S}[\![S]\!](\gamma')(\delta')(\sigma_1)$. Assume that $\sigma_2 \neq \perp$, otherwise we have $\sigma' = \perp$ and we are ready. The validity of the first premiss with respect to γ tells us that $\sigma_2, \delta', \omega' \models Q[e/r]$. Let $\beta = \mathcal{E}[\![e]\!](\delta')(\sigma_2)$, $\omega_1 = \omega\{\beta/r\}$, and $\omega_1' = \omega'\{\beta/r\}$. Then because of lemma 5.12 we have $\sigma_2, \delta', \omega_1' \models Q$.

Let $\sigma'' = \langle \sigma_{2\,(1)}, \sigma_{2(2)}, \sigma_{(3)} \rangle$ (we restore the temporary variables). Now we appeal to the reader's understanding of the semantics of the language to see that the method destination e_0, the actual parameters $e_1, \ldots, e_n$ and the expressions $\bar{f}$ are unchanged in σ'' in comparison with σ. Otherwise stated, $\mathcal{E}[\![e_i]\!](\delta)(\sigma) = \mathcal{E}[\![e_i]\!](\delta)(\sigma'')$ and the same for f_i. (Of course, this can also be proved formally.) Then we know from lemma 5.28 that $\sigma'', \delta, \omega_1' \models Q[\bar{e}/\mathsf{self}, \bar{u}]$ and from lemma 5.12 together with the observation that $\omega_1' = \omega_1\{\mathcal{E}[\![f_i]\!](\delta)(\sigma)/z_i\}_{i=1}^{|\bar{f}|}$ we get $\sigma'', \delta, \omega_1 \models Q[\bar{e}/\mathsf{self}, \bar{u}][\bar{f}/\bar{z}]$.

From the second premiss we can conclude that $\sigma'', \delta, \omega_1 \models R[r/x]$. Now for the final state σ' we know that $\sigma' = \sigma''\{\beta/\delta_{(1)}, x\}$, so lemma 5.8 tells us that $\sigma', \delta, \omega_1 \models R$. Finally, because r does not occur in R, we have $\sigma', \delta, \omega \models R$. □

Definition 5.31
For the statement $u \leftarrow e_0!m(e_1, \ldots, e_n)$, where $u \in TVar_{d_0}$, $m \in MName^{c'}_{d_0,\ldots,d_n}$,

$e_0 \in Exp^c_{c'}$ and $e_i \in Exp^c_{d_i}$ for $i = 1, \ldots, n$, we have the proof rule (MT) which is identical to the rule (MI) introduced in definition 5.24, except that the instance variable x is replaced everywhere by the temporary variable u.

Lemma 5.32
The proof rule (MT) is valid.

Proof
This can be proved by a slight adaptation of the proof of lemma 5.30. □

Now we come to the issues of how to handle recursive and even mutually recursive methods. For this we use an adapted version of the classical recursion rule (see for example [3]). The classical rule goes as follows (in the notation of [3]):

$$\frac{\{p\}P\{q\} \vdash \{p\}S_0\{q\}}{\{p\}P\{q\}}$$

The idea is to prove (the operator $\vdash$ expresses provability) the correctness of the body (S_0) from the assumption that the procedure call (P) itself satisfies its specification. If that has been done we can conclude the correctness of the procedure call without assumptions. The validity of this rule can be proved as follows: the meaning of the procedure call is the limit of a increasing sequence starting with $\perp$, in which every element is obtained from the previous one by assuming the previous as the meaning of the procedure call and calculating the meaning of the body from that. From the premiss of the rule we can prove that every element in the sequence satisfies the specification and by a continuity argument we conclude that the procedure call itself satisfies the specification.

There are several remarks to be made. One is that in proving the premiss of the rule we may not make use of the declaration of P, because otherwise we are not sure that the implication also holds for the intermediate elements in the approximating sequence. The second remark is that if we have a non-recursive rule like our rules (MI) and (MT), then we could change the conclusion of the recursion rule into $\{p\}S\{q\}$, from which we could infer $\{p\}P\{q\}$ by the non-recursive rule. We do that in our proof system to be able to use the outcome of the recursion rule for different values of the parameters. Finally it is clear how to extend the rule to several, mutually recursive procedures.

Definition 5.33
For mutually recursive methods $m_1, \ldots, m_n$ we have the following rule:

$$\frac{\tilde{F}_1, \ldots, \tilde{F}_n \qquad F_1, \ldots, F_n \vdash F'_1, \ldots, F'_n}{F} \tag{MR}$$

where

$$
\begin{array}{lcl}
F_i & = & \left\{P_i^{c'_i}[\bar{e}^i/\mathsf{self},\bar{u}^i][\bar{f}^i/\bar{z}^i]\right\}\langle U^-|c_i : x_i \leftarrow e_0^i!m_i(e_1^i,\ldots,e_{n_i}^i)\rangle\left\{R_i^{c_i}\right\} \\
F'_i & = & \left\{P_i^{c'_i} \wedge \bigwedge_{j=1}^{k_i} v_j^i \doteq \mathsf{nil}\right\}\langle U^-|c'_i : S_i\rangle\left\{Q_i^{c'_i}[e_i/r_i]\right\} \\
\tilde{F}_i & = & Q_i[\bar{e}^i/\mathsf{self},\bar{u}^i][\bar{f}^i/\bar{z}^i] \rightarrow R_i[r_i/x_i] \\
F & = & \left\{P_1 \wedge \bigwedge_{j=1}^{k_1} v_j^1 \doteq \mathsf{nil}\right\}\langle U|c'_1 : S_1\rangle\left\{Q_1^{c'_1}[e_1/r_1]\right\}
\end{array}
$$

$\bar{u}^i$, S_i, and e_i are as they occur in the definition of m_i in U

x_i are instance *or* temporary variables

U^- results from U by deleting the definitions of $m_1,\ldots,m_n$

P_i, Q_i, R_i, $\bar{f}^i$, $\bar{z}^i$, $\bar{v}^i$, k_i, and r_i are just like in definition 5.24

We cannot prove the validity of this proof rule on its own, because it depends on what the other rules can prove (the operator $\vdash$ occurs in the premiss).

5.4 Other axioms and rules

Finally in this subsection we shall list the remaining axioms and rules of our proof system. They will deal with the more ordinary statements and therefore they are not very new (most of them can already be found in [4]).

Definition 5.34
For a side effect expression s_d^c functioning as a statement we have the following rule:

$$
\frac{\left\{P^c\right\}\langle U|c : u_d \leftarrow s_d^c\rangle\left\{Q^c\right\}}{\left\{P\right\}\langle U|c : s\rangle\left\{Q\right\}} \qquad \text{(ES)}
$$

where u_d is a temporary variable not occurring in P or Q.

Definition 5.35
For the sequential composition of statements we have the following proof rule:

$$
\frac{\left\{P^c\right\}\langle U|S_1^c\rangle\left\{Q^c\right\} \qquad \left\{Q^c\right\}\langle U|S_2^c\rangle\left\{R^c\right\}}{\left\{P\right\}\langle U|S_1; S_2\rangle\left\{R\right\}} \qquad \text{(SC)}
$$

Definition 5.36
For the conditional statement we have this rule:

$$
\frac{\left\{P^c \wedge e_{\mathsf{Bool}}^c\right\}\langle U|S_1^c\rangle\left\{Q^c\right\} \qquad \left\{P^c \wedge \neg e\right\}\langle U|S_2^c\rangle\left\{Q^c\right\}}{\left\{P\right\}\langle U|\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}\rangle\left\{Q\right\}} \qquad \text{(C)}
$$

Definition 5.37
For the while loop we have the following rule:

$$\frac{\left\{P^c \wedge e^c_{\mathsf{Bool}}\right\}\langle U|S^c\rangle\left\{P^c\right\}}{\left\{P\right\}\langle U|\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}\rangle\left\{P \wedge \neg e\right\}} \tag{W}$$

Definition 5.38
For every *valid* (see definition 4.15) assertion P^c we have the axiom:

$$P \tag{TR}$$

Definition 5.39
Finally, we have the so-called rule of consequence:

$$\frac{P_1^c \to P_2^c \qquad \left\{P_2\right\}\langle U|c : S\rangle\left\{Q_2\right\} \qquad Q_2^c \to Q_1^c}{\left\{P_1\right\}\langle U|c : S\rangle\left\{Q_1\right\}} \tag{RC}$$

Theorem 5.40
The proof system consisting of the axioms (SAT), (SAI), (NT), (NI), and (TR), plus the rules (MI), (MT), (MR), (ES), (SC), (C), (W), and (RC) is sound, that is, for every row of correctness formulae $F_0, \ldots, F_n$ and for every environment γ we have if $F_1, \ldots, F_n \vdash F_0$ and $\gamma \models F_i$ for $i = 1, \ldots, n$ then $\gamma \models F_0$.

Proof
For all rules except (MR) the validity can be proved individually. For some we have already done that, for the others it is very easy. The rest of the proof runs by induction on the length of the proof of F_0 from $F_1, \ldots, F_n$. The only interesting case occurs if the last rule applied is (MR). From now on let us use the notation of definition 5.33 and forget about the old $F_0, \ldots, F_n$.

In the premiss of the rule (MR) we first have $\tilde{F}_1, \ldots, \tilde{F}_n$, and these are valid because the only way to get them is by using the axiom (TR). The second premiss says that $F_1, \ldots, F_n \vdash F_1', \ldots, F_n'$. This must be provable by a shorter proof than our current one so the induction hypothesis says that for every environment γ such that $\gamma \models F_1, \ldots, F_n$ we also have that $\gamma \models F_1', \ldots, F_n'$. Let us take a particular γ and define $\gamma' = \mathcal{U}[\![U]\!](\gamma)$. Now γ' is the limit of an increasing sequence $\gamma_0', \gamma_1', \ldots$ where $\gamma_0' = \gamma'\left\{\lambda\bar{\beta} \,.\, \lambda\delta \,.\, \lambda\sigma \,.\, \langle\bot, \bot\rangle \,\middle/\, m_i\right\}_{i=1}^{n}$ and γ_{i+1}' is obtained from γ_i' by calculating and filling in the meanings of the method definitions of $m_1, \ldots, m_n$. Furthermore we observe that for every i and for every $m \in \{m_1, \ldots, m_n\}$ we have that $\mathcal{U}[\![U^-]\!](\gamma_i')(m) = \gamma_i'(m)$ because m is not defined in U^-.

Now for γ_0' we have quite trivially that $\gamma_0' \models F_1', \ldots, F_n'$ (the send-expression never terminates). Furthermore from $\gamma_i' \models F_j'$ we can get to $\gamma_{i+1}' \models F_j$ by an argument analogous to that in lemma 5.30. ¿From the validity of the second premiss we can then conclude that $\gamma_{i+1}' \models F_j'$ for $j = 1, \ldots, n$. By induction we get $\gamma_i' \models F_1', \ldots, F_n'$ for every i, so by continuity we get in particular $\gamma' \models F_1'$. And this in turn implies $\gamma \models F$. □

6 Completeness

6.1 Introduction

We prove in this section that every valid correctness formula about an arbitrary closed program is derivable from the proof system based on the assertion language with quantification over finite sequences of objects. To this end we use enhanced versions of the standard techniques for proving completeness. These techniques are based on the expressibility of the *strongest postcondition*, or, alternatively, the *weakest precondition*. Using the assertion language with quantification over finite sequences of objects we know how to express the strongest postcondition. However, we conjecture that we cannot in general express the strongest postcondition or the weakest precondition within the assertion language with recursive predicates. We think this is due to the inexpressibility within this assertion language of the notion of *finiteness*.

In order to get a complete proof system, however, we have to modify the rules (MI), (MT), and (MR) so that we can reason about *deadlock behaviour*. Regardless of the assertion language we use these rules are incomplete. Consider the following example:

Example 6.1
Let $\rho = \langle U|c : x \leftarrow \mathsf{self}!m()\rangle$ be closed and $m() \Leftarrow \mathsf{nil} \uparrow \mathsf{nil}$ occur in U. We obviously have $\models \{\mathsf{true}\}\rho\{\mathsf{false}\}$. But we do *not* have the derivability of this correctness formula. For otherwise there would exist assertions P, Q and R such that:

1. $\vdash \{P \wedge \bigwedge_i v_i \doteq \mathsf{nil}\}\langle U|c : \mathsf{nil}\rangle\{Q[\mathsf{nil}/r]\}$
2. $\models Q[\mathsf{self}/\mathsf{self}][\bar{f}/\bar{z}] \rightarrow R[r/x]$
3. $\models \mathsf{true} \rightarrow P[\mathsf{self}/\mathsf{self}][\bar{f}/\bar{z}]$ and $\models R \rightarrow \mathsf{false}$

for some sequence of expressions $\bar{f}$, sequence of corresponding logical variables $\bar{z}$ and logical variable r of the same type as the instance variable x. Now, as $\models R \rightarrow \mathsf{false}$, we have $\models R[r/x] \rightarrow \mathsf{false}$. So from clause 2 it then follows that $\models Q[\mathsf{self}/\mathsf{self}][\bar{f}/\bar{z}] \rightarrow \mathsf{false}$. Furthermore we have $\models Q[\mathsf{self}/\mathsf{self}] \leftrightarrow Q$ so we infer $\models Q[\bar{f}/\bar{z}] \rightarrow \mathsf{false}$. From clause 1 in turn it is not difficult to deduce that $\models P \rightarrow Q[\mathsf{nil}/r]$ (use $\bar{v}_i \cap TVar(P,Q) = \emptyset$ and the truth of the correctness formula of clause 1). So we have $\models P[\bar{f}/\bar{z}] \rightarrow Q[\mathsf{nil}/r][\bar{f}/\bar{z}]$. Note next that $\models Q[\bar{f}/\bar{z}] \rightarrow \mathsf{false}$ implies $\models Q[\mathsf{nil}/r][\bar{f}/\bar{z}] \rightarrow \mathsf{false}$, from which we infer that $\models P[\bar{f}/\bar{z}] \rightarrow \mathsf{false}$, which in turn, using $\models P[\mathsf{self}/\mathsf{self}] \leftrightarrow P$, would imply by clause 3 $\models \mathsf{true} \rightarrow \mathsf{false}$. We thus have reached a contradiction. So we conclude that $\not\vdash \{\mathsf{true}\}\rho\{\mathsf{false}\}$.

Note that adding the conjunct $\neg(\mathsf{self} \doteq e_0)$ to the precondition of the conclusion of the rules (MI) and (MT) does not solve the general case of longer cycles in the calling chain.

To reason about deadlock in the proof system based on the assertion language containing quantification over finite sequences we introduce a collection of logical variables with special roles.

Definition 6.2
We fix for each class name c a logical variable $b_c \in LVar_{c^*}$. Furthermore we define $BVar = \{b_c : c \in C\}$.

We will interpret the variable b_c as denoting a sequence of all the blocked objects of class c. Formally, we redefine the notion $OK(\sigma, \delta, \omega)$ as follows:

Definition 6.3
For arbitrary σ, δ, ω we define $OK(\sigma, \delta, \omega)$ iff σ is consistent, δ agrees with σ, ω is compatible with σ and for an arbitrary c we have

$$\delta_{(2)(c)} = \{\alpha : \exists n \in \mathbf{N}(elt(b_c, n) = \alpha \neq \perp)\}.$$

So we have $OK(\sigma, \delta, \omega)$ if additionally b_c, for an arbitrary c, consists precisely of all the blocked objects of class c. Note that we have thus introduced in the assertion language a means to refer to the second component of a context. Given this fixed interpretation we do not allow the variable b_c to be quantified. It is a straightforward exercise to check that under this definition of $OK(\sigma, \delta, \omega)$ the soundness proofs given still hold.

Next we modify the rule (MI) as follows:

Definition 6.4
For the statement $x \leftarrow e_0!m(e_1, \ldots, e_n)$, where $x \in IVar^c_{d_0}$, $m \in MName^{c'}_{d_0,\ldots,d_n}$, $e_0 \in Exp^c_{c'}$ and $e_i \in Exp^c_{d_i}$ for $i = 1, \ldots, n$, we have the following proof rule:

$$\frac{\left\{P^{c'} \wedge \bigwedge_{i=1}^{k} v_i \doteq \mathsf{nil} \wedge \neg(\mathsf{self} \in b_{c'})\right\}\langle U|c' : S\rangle\left\{Q^{c'}[e/r]\right\}, \quad Q' \rightarrow R^c[r/x]}{\left\{P'\right\}\langle U|c : x \leftarrow e_0!m(e_1, \ldots, e_n)\rangle\left\{R\right\}} \qquad \text{(MI)}$$

where $P' = P[\bar{e}/\mathsf{self}, \bar{u}][\bar{f}/\bar{z}][b_c \circ \langle\mathsf{self}\rangle/b_c]$, $Q' = Q[\bar{e}/\mathsf{self}, \bar{u}][\bar{f}/\bar{z}][b_c \circ \langle\mathsf{self}\rangle/b_c]$, $S \in Stat^{c'}$ and $e \in Exp^{c'}_{d_0}$ are the statement and expression occurring in the definition of the method m in the unit U, $u_1, \ldots, u_n$ are its formal parameters, $v_1, \ldots, v_k$ is a row of temporary variables that are *not* formal parameters ($k \geq 0$), r is a logical variable of type d_0 that does not occur in R, $\bar{f}$ is an arbitrary row of expressions (*not* logical expressions) in class c, and $\bar{z}$ is a row of logical variables, mutually different and different from r, such that the type of each z_i is the same as the type of the corresponding f_i. We require that no temporary variables other than the formal parameters $u_1, \ldots, u_n$ occur in P or Q. The boolean expression $l_1 \in l_2$ abbreviates $\exists i(l_1 \doteq l_2 \cdot i)$, where i is some fresh logical integer variable. $P[b_c \circ \langle \mathsf{self} \rangle / b_c]$, for an arbitrary assertion P, equals the assertion

$$\exists z(P[z/b_c] \wedge |z| = |b_c| + 1 \wedge \forall i(i \leq |b_c| \rightarrow z \cdot i \doteq b_c \cdot i) \wedge (z \cdot |z| \doteq \mathsf{self}))$$

where $z \in LVar_{c^*}, i \in LVar_{\mathsf{Int}}$ are some fresh variables.

The idea of this substitution $[b_c \circ \langle \mathsf{self} \rangle / b_c]$ can be explained roughly as follows: Occurrences of the variable b_c in the assertions $P^{c'}$ and $Q^{c'}$, which describe the input state and the output state of the receiver of the method call, denote the set of blocked objects of class c belonging to those states. When we want to describe the input state and the output state of the receiver from the point of view of the sender we have to take into account that this set of blocked objects can now be viewed as the set of blocked objects of class c belonging to the input state and the output state of the sender of the method call plus the sender itself.

The rules (MT) and (MR) are modified accordingly. The soundness proofs of these new versions of (MI) and (MT) are straightforward modifications of the proofs of the soundness of the original ones (in the proof of 5.30 the substitution $[b_c \circ \langle \mathsf{self} \rangle / b_c]$ can be considered simply as part of the simultaneous substitution $[\bar{f}/\bar{z}]$). The proof of the soundness of the new version of (MR), assuming the soundness of the new versions of (MI) and (MT), does not need to be modified.

We note that with respect to the proof system based on the assertion language containing recursive predicates this proof method does not apply. To incorporate some reasoning mechanism about deadlock behaviour in this system one could add to it some notion of auxiliary variables, which can be used to code the relevant control information.

It will appear to be technically convenient to introduce another modification of the rule (MR). This modification consists simply of replacing every occurrence of U^- in this rule by U itself. We denote the resulting rule by (NMR). The main difference between the rules (NMR) and (MR) is that the rule (NMR) allows nested applications to some method name. However, in appendix A it is shown that a proof using the rule (NMR) can be transformed into a proof using (MR), and vice versa.

To be able to prove completeness we have to add the following rules to the proof system (based on the assertion language containing quantification over finite sequences).

Definition 6.5

Conjunction rule:

$$\frac{\left\{P_1^c\right\}\rho^c\left\{Q_1^c\right\} \qquad \left\{P_2^c\right\}\rho^c\left\{Q_2^c\right\}}{\left\{P_1^c \wedge P_2^c\right\}\rho^c\left\{Q_1^c \wedge Q_2^c\right\}} \tag{CR}$$

Definition 6.6
Elimination rule 1:

$$\frac{\left\{P^c\right\}\rho^c\left\{Q^c\right\}}{\left\{\exists z_d P^c \vee P[\mathsf{nil}/z_d]\right\}\rho^c\left\{Q^c\right\}} \tag{ER1}$$

where $z_d \notin LVar(Q^c) \cup BVar$. Due to the interpretation of the quantifiers as ranging only over existing objects we have to express explicitly that the precondition also holds when the value of the quantified variable is undefined (nil).

Definition 6.7
Elimination rule 2:

$$\frac{\left\{P^c\right\}\rho^c\left\{Q^c\right\}}{\left\{\exists z_a P^c\right\}\rho^c\left\{Q^c\right\}} \tag{ER2}$$

where $a = d^*$, for some d, and $z_a \notin LVar(Q^c) \cup BVar$.

Definition 6.8
Initialization rule 1:

$$\frac{\left\{P^c\right\}\rho^c\left\{Q^c\right\}}{\left\{P^c[l/z]\right\}\rho^c\left\{Q^c\right\}} \tag{IR1}$$

where z and l are of the same type, and $z \notin LVar(Q^c) \cup BVar$.

Definition 6.9
Initialization rule 2:

$$\frac{\left\{P^c\right\}\rho^c\left\{Q^c\right\}}{\left\{P^c[l/u]\right\}\rho^c\left\{Q^c\right\}} \tag{IR2}$$

where u and l are of the same type and $u \notin TVar(\rho, Q)$.

Definition 6.10
Substitution rule:

$$\frac{\left\{P^c\right\}\rho^c\left\{Q^c\right\}}{\left\{P^c[z'/z]\right\}\rho^c\left\{Q^c[z'/z]\right\}} \tag{SR}$$

where z', z are logical variables of the same type, and $z \notin BVar$.

The soundness of these new rules is a straightforward exercise. We illustrate the necessity of the condition $z \notin BVar$ by the following example:

Example 6.11
Let $\rho = \langle U|c' : y \leftarrow x!m()\rangle$. By the new definition of $OK(\sigma, \delta, \omega)$ we have, assuming the type of the variable x to be c,

$$\models \{x \in b_c\}\rho\{\mathsf{false}\},$$

where $x \in b_c$ abbreviates the assertion $\exists i(x \doteq b_c \cdot i)$. If we would allow the initialization of the variable b_c, or allow it to be substituted, we could derive from this formula by an application of the rule (SR) or (IR1) the following:

$$\{x \in z\}\rho\{\mathsf{false}\}.$$

Applying next the elimination rule (ER2), assuming $z \notin BVar$, then gives us the derivability of the formula:

$$\{\exists z(x \in z)\}\rho\{\mathsf{false}\}.$$

Finally, we apply the consequence rule:

$$\{\mathsf{true}\}\rho\{\mathsf{false}\}.$$

But this last formula is not valid in general!

Finally, for technical convenience we would like to assume that the sets C, $IVar$, and $TVar$ are finite. This assumption can be justified as follows: Let C' be a finite subset of C, and $IVar'$ be a finite subset of $\bigcup_{c,d} IVar^c_d$, where c ranges over C', and d ranges over the set $C'^+ = C' \cup \{\mathsf{Int}, \mathsf{Bool}\}$. Next we fix the temporary integer variables u, u', and for every $d \in C'^+$ the temporary variables re_d, re'_d. Let $\bar{re}$ denote a sequence of these variables. Now let $TVar'$ be a finite subset of $\bigcup_d TVar_d$ (again, d ranging over C'^+), such that $\bar{re} \subseteq TVar'$. Given these sets $C', IVar'$, and $TVar'$ we have the following definition.

Definition 6.12
We define an expression l^c_a to be *restricted* iff $c \in C'$, $a = d, d^*$, with $d \in C'^+$, $IVar(l^c_a) \subseteq IVar'$, and $TVar(l^c_a) \subseteq TVar'$. We define an assertion P^c to be restricted iff $c \in C'$ and every expression occurring in P^c is restricted. We call a program $\rho = \langle U|c : S\rangle$ restricted iff $c \in C'$, every expression occurring in ρ is restricted, $u, u' \notin TVar(\rho)$, and, finally, the temporary variables re_d, re'_d are only allowed in the main statement S itself, where $S = re_d \leftarrow s_d$ or $S = re'_d \leftarrow s_d$, with $TVar(s) \cap \bar{re} = \emptyset$. A correctness formula $\{P\}\rho\{Q\}$ is called restricted iff P, Q, and ρ are restricted.

We will prove that an arbitrary valid restricted correctness formula is derivable by a derivation in which there occur only restricted correctness formulae. Such a derivation we call restricted too. The extra variables $\bar{re}$ are used in applications of the rules (W) and (ES): The variables re_d, re'_d are used to store temporarily the result of the execution of a statement s_d; the variables u, u' are needed to express the *invariant* of a while statement. However applications of the consequence rule in a restricted derivation are based on a different notion of validity of assertions and correctness formulae. This new notion of validity consists of restricting all the semantic entities to the sets C', $IVar'$, and $TVar'$. As an example of the restriction of a semantic entity we define that of a state.

Definition 6.13
We define the restriction of a state σ, which we denote by $\sigma\downarrow$, to be an element of

$$\Sigma\downarrow = \prod_{c\in C'} \mathbf{P}^c \times \prod_{c\in C', d\in C'^+} \left(\mathbf{O}^c \rightarrow IVar'^c_d \rightarrow \mathbf{O}^d_\perp\right) \times \prod_{d\in C'^+} \left(TVar'_d \rightarrow \mathbf{O}^d_\perp\right)$$

such that

- $\sigma\downarrow^{(c)} = \sigma^{(c)}$, $c \in C'$.
- $\sigma\downarrow(\alpha)(x) = \sigma(\alpha)(x)$, $\alpha \in \mathbf{O}^c$, for $c \in C'$, and $x \in IVar'$.
- $\sigma\downarrow(u) = \sigma(u)$, $u \in TVar'$.

In a similar way we have corresponding restricted versions of all our semantic entities. We have the following lemma, which states that the meaning of a restricted program depends only on those parts of a state specified by the sets C', $IVar'$, and $TVar'$.

Lemma 6.14
For an arbitrary restricted program ρ, and $\sigma, \sigma', \delta, \gamma$ such that

1. $\sigma^{(c)} = \sigma'^{(c)}$, $c \notin C'$.
2. $\sigma(\alpha) = \sigma'(\alpha)$, for $\alpha \in \mathbf{O}^c$, $c \notin C'$.
3. $\sigma(\alpha) = \sigma'(\alpha)$, for $\alpha \in \mathbf{O}^c \setminus \sigma'^{(c)}$, $c \in C'$.
4. $\sigma(\alpha)(x) = \sigma'(\alpha)(x)$, for $\alpha \in \sigma^{(c)}$, $c \in C'$, $x \notin IVar'$.
5. $\sigma'(\alpha)(x) = \perp$, for $\alpha \in \sigma'^{(c)} \setminus \sigma^{(c)}$, $c \in C'$, $x \notin IVar'$.
6. $\sigma(u) = \sigma'(u)$, $u \notin TVar'$.

we have

$$\sigma' = \mathcal{P}[\![\rho]\!](\gamma)(\delta)(\sigma) \text{ iff } \sigma'\downarrow = \mathcal{P}'[\![\rho]\!](\gamma\downarrow)(\delta\downarrow)(\sigma\downarrow),$$

where $\mathcal{P}'$, $\gamma\downarrow$, and $\delta\downarrow$ denote the restricted versions of $\mathcal{P}, \gamma$, and δ, respectively. (Here $\sigma(\alpha)$ denotes the local state of α and $\sigma(\alpha)(x)$, x an instance variable, denotes the value of the variable x of the object α, finally, $\sigma(u)$, u a temporary variable, denotes the value of u in state σ.)

The first condition above states that σ and σ' agree with respect to the existing objects of class c, $c \notin C'$. The second condition states that σ and σ' agree with respect to the local states of objects belonging to a class c, $c \notin C'$. That the states σ and σ' agree with respect to the local states of objects belonging to a class c, $c \in C'$, which do not exist in σ', is expressed by the third clause. The fourth clause states that σ and σ' agree with respect to the variables not belonging to $IVar'$ of objects of a class c, $c \in C'$, which exist in σ. The fifth clause then states that the value of a variable not belonging to $IVar'$ of an object of a class c, $c \in C'$, which exist in σ' but does not exist in σ, is undefined in the state σ'. The last clause states that σ and σ' agree with respect to the temporary variables not belonging to $TVar'$. These conditions are necessary to prove that if $\sigma' \downarrow = \mathcal{P}'[\![\rho]\!](\gamma\downarrow)(\delta\downarrow)(\sigma\downarrow)$ then $\sigma' = \mathcal{P}[\![\rho]\!](\gamma)(\delta)(\sigma)$.

Proof
Induction on the structure of the program ρ. □

By the following two lemmas we have that applications of the consequence rule occurring in a restricted derivation also apply with respect to the original notion of validity, thus justifying our assumption of the finiteness of the sets C, $IVar$, and $TVar$. These lemmas state that the truth of a restricted assertion and that of a correctness formula only depend on those parts of a state specified by the sets C', $IVar'$, and $TVar'$.

Lemma 6.15
For an arbitrary restricted assertion P^c, and σ, δ, ω such that $OK(\sigma, \delta, \omega)$ we have

$$\sigma, \delta, \omega \models P^c \text{ iff } \sigma\downarrow, \delta\downarrow, \omega\downarrow \models P^c,$$

where $\omega\downarrow \in \prod_a LVar_a \rightarrow \mathbf{O}^a_\perp$, with a ranging over the set $\{d, d^* : d \in C'^+\}$, and $\omega\downarrow(z) = \omega(z)$.

Proof
Straightforward induction on the structure of P^c. □

Furthermore we have

Lemma 6.16
Let σ, δ, ω such that $OK(\sigma, \delta, \omega)$. We have for an arbitrary restricted correctness formula $\{P\}\rho\{Q\}$

$$\sigma, \delta, \omega \models \{P\}\rho\{Q\} \text{ iff } \sigma\downarrow, \delta\downarrow, \omega\downarrow \models \{P\}\rho\{Q\}.$$

Proof
Straightforward, using lemmas 6.14 and 6.15. □

So in the sequel we may assume the sets $C, IVar$, and $TVar$ to be finite. Further, we assume given a set of temporary variables $\bar{re}$ as defined above. A program ρ from now on will denote, when not stated otherwise, a program such that the temporary variables re, re' are allowed to occur in it only in assignments $re \leftarrow s, re' \leftarrow s$, with $re, re' \notin TVar(s)$, and $u, u' \notin TVar(\rho)$. This concludes our discussion concerning the justification of the assumption of the finiteness of the sets $C, IVar$, and $TVar$.

6.2 The strongest postcondition

To be able to prove completeness we first have to analyze the notion of a *strongest postcondition* and its expressibility in the assertion language. As noted already in the introduction, the expressibility of the strongest postcondition in the assertion language with recursive predicates is still an open problem and so is the completeness of the proof system based on this assertion language.

For the analysis of the notion of a strongest postcondition we need some definitions and a theorem. We start with the following definition:

Definition 6.17
An *object-space isomorphism* (*osi*) is a family of functions $f = \langle f^d \rangle_{d \in C^+}$, where $f^d \in \mathbf{O}^d_\perp \rightarrow \mathbf{O}^d_\perp$ is a bijection, $f^d(\perp) = \perp$ and f^d, for $d = \mathsf{Int}, \mathsf{Bool}$, is the identity mapping.

Given an *osi* f we next define the *isomorphic* image of an arbitrary state.

Definition 6.18
Given an *osi* f we define for an arbitrary state σ the state $f(\sigma)$ as follows:

- For every c: $f(\sigma)^{(c)} = f^c(\sigma^{(c)})$.
- For every c, d, α^c, x^c_d: $f(\sigma)(\alpha)(x^c_d) = f^d(\sigma(f^{-1^c}(\alpha))(x^c_d))$, where the *osi* f^{-1} denotes the *inverse* of f: $f^{-1} = \langle (f^d)^{-1} \rangle_d$.
- For every d, u_d: $f(\sigma)(u_d) = f^d(\sigma(u_d))$.

Here $f^c(X)$, for some $X \subseteq O^c$, denotes the set $\{f^c(\alpha) : \alpha \in X\}$.

The following theorem essentially expresses that states which are isomorphic cannot be distinguished by the assertion language.

Theorem 6.19
Let f be an *osi* and σ, δ, ω be such that $OK(\sigma, \delta, \omega)$. Then for every logical expression l^c_a and assertion P^c we have:

- $f^a(\mathcal{L}[\![l_a^c]\!](\omega)(\delta)(\sigma)) = \mathcal{L}[\![l_a^c]\!](f(\omega))(f(\delta))(f(\sigma))$,
- $\mathcal{A}[\![P^c]\!](\omega)(\delta)(\sigma) = \mathcal{A}[\![P^c]\!](f(\omega))(f(\delta))(f(\sigma))$.

where $f(\delta)_{(1)} = f^c(\delta_{(1)})$,
$f(\delta)_{(2)(c')} = f^{c'}(\delta_{(2)(c')})$, for an arbitrary c', and
$f(\omega)(z_d) = f^d(\omega(z_d))$, $(f^{d^*}(\langle\alpha_1,\ldots,\alpha_n\rangle) = \langle f^d(\alpha_1),\ldots,f^d(\alpha_n)\rangle)$.

Proof
Straightforward induction on the structure of l_a^c, P^c. We only treat the case $l = x_d^c$:

$$\mathcal{L}[\![x]\!](f(\omega))(f(\delta))(f(\sigma)) = f(\sigma)(f^c(\delta_{(1)}))(x) = f^d(\sigma(\delta_{(1)})(x)) = f^d(\mathcal{L}[\![x]\!](\omega)(\delta)(\sigma))$$

□

We are now sufficiently prepared to analyze the notion of a strongest postcondition. Given a program ρ^c and an assertion P^c, we denote by $sp(\rho^c, P^c)$ the set of final states of executions of ρ^c starting from a state satisfying P^c. An assertion, defining this set of states $sp(\rho^c, P^c)$ is called the strongest postcondition of P^c with respect to ρ^c. As established by the previous theorem, the set of states defined by an arbitrary assertion is closed under isomorphism. However, in general, given a program ρ^c and an assertion P^c, the set of states $sp(\rho^c, P^c)$ is not closed under isomorphism. Consider the following example:

Example 6.20
Take $\rho^c = \langle U|c : x \leftarrow \mathsf{new}\rangle$, with ρ^c closed, and σ, σ', δ such that $\sigma'^{(c)} = \{\alpha, \beta\}$, $\sigma^{(c)} = \{\alpha\}$, $\delta_{(1)} = \alpha$ and $\sigma' = \mathcal{P}[\![\rho^c]\!](\gamma)(\delta)(\sigma)$. Let $P^c = \mathsf{true}$. So we have that $pick^c(\{\alpha\}) = \beta$. Let f be an arbitrary *osi* such that $pick^c(\{f^c(\alpha)\}) \neq f^c(\beta)$ and $pick^c(\{f^c(\beta)\}) \neq f^c(\alpha)$. So we have that $f(\sigma')^{(c)} = \{f^c(\alpha), f^c(\beta)\}$. Now suppose that there is a σ_0 such that $f(\sigma') = \mathcal{P}[\![\rho^c]\!](\gamma)(f(\delta))(f(\sigma_0))$. Then we would have $\sigma_0^{(c)} = \{f^c(\alpha)\}$ or $\sigma_0^{(c)} = \{f^c(\beta)\}$, but both cases lead to a contradiction. Therefore such a σ_0 does not exist and $f(\sigma') \notin sp(\rho^c, \mathsf{true})$.

This discrepancy between the assertion language and the semantics of the programming language is solved by closing this set $sp(\rho^c, P^c)$ under isomorphism. Of course it is not immediately clear that this will work! We will see later that we indeed encounter some difficulties in the completeness proof due to this. These difficulties require some additional reasoning not present in the standard completeness proofs. The following theorem states the existence of an assertion defining the closure under isomorphism of the set $sp(\rho^c, P^c)$.

Theorem 6.21
Let ρ^c be closed (*not* necessarily restricted), $BVar \subseteq L \subseteq LVar$ (L finite), P^c

such that $LVar(P^c) \subseteq L$. Then there exists an assertion $SP^c_L(\rho, P^c)$ such that $LVar(SP^c_L(\rho, P^c)) \subseteq L$ and for σ, δ, ω such that $OK(\sigma, \delta, \omega)$ we have:

$$\sigma, \delta, \omega \models SP^c_L(\rho, P^c)$$

iff there exist an *osi* f and a state σ_0 such that:

- $f(\sigma) = \mathcal{P}[\![\rho]\!](\gamma)(\delta')(\sigma_0)$, γ arbitrary,
- $\sigma_0, \delta', \omega' \models P^c$,

where $\delta' = f(\delta)$ and $\omega' = f(\omega) \downarrow L$. Here we define

$$\begin{array}{rcll} (f(\omega) \downarrow L)(z) & = & f(\omega(z)) & z \in L \\ & = & \perp & z \in (LVar \cap \bigcup_d LVar_d) \setminus L \\ & = & \epsilon & z \in (LVar \cap \bigcup_d LVar_{d^*}) \setminus L. \end{array}$$

Note that in the above theorem we cannot take $f(\omega)$, where $f(\omega)(z) = f(\omega(z))$, for ω'. This would require that $f(\omega)$ and σ_0 are compatible, which cannot be expressed by our assertion language. For suppose there exists an $\alpha \in \sigma^{(c')}$, for some c', such that $f^{c'}(\alpha) \notin \sigma_0^{(c')}$. Let $z_{c'} \notin L$, it then follows that $\sigma, \delta, \omega\{\alpha/z_{c'}\} \models SP^c_L(\rho, P^c)$, but on the other hand it is not the case that $f(\omega\{\alpha/z_{c'}\})$ and σ_0 are compatible, so we do not have $\sigma_0, \delta', f(\omega\{\alpha/z_{c'}\}) \models P^c$. Note that the above argument essentially boils down to the fact that we cannot describe by one assertion the values of infinitely many logical variables. Thus we have to specify a finite set of logical variables L such that the restriction of $f(\omega)$ to this set L is compatible with σ_0.

Proof
See appendix B. $\square$

The following two lemmas together state the correctness of our definition of the notion of strongest postcondition.

Lemma 6.22
For an arbitrary $BVar \subseteq L \subseteq LVar$ (L finite), closed program ρ^c and assertion P^c such that $LVar(P^c) \subseteq L$, we have

$$\models \Big\{P^c\Big\}\rho\Big\{SP^c_L(\rho, P^c)\Big\}.$$

Proof
Let $\sigma, \sigma', \delta, \omega$ ($\sigma, \sigma' \neq \perp$) be such that $OK(\sigma, \delta, \omega)$, $\sigma' = \mathcal{P}^c[\![\rho^c]\!](\gamma)(\delta)(\sigma)$ (γ arbitrary),

and $\sigma, \delta, \omega \models P^c$. We have that $\sigma', \delta, \omega \models SP^c_L(\rho, P^c)$, for take for the *osi* f the family of identity mappings, for σ_0 the state σ, and note that because $LVar(P^c) \subseteq L$ we have $\sigma, \delta, \omega' \models P^c$, where $\omega' = \omega \downarrow L$. □

Lemma 6.23
For an arbitrary closed program ρ^c, assertions P^c, Q^c, $BVar \subseteq L \subseteq LVar$ (L finite) such that $LVar(P^c, Q^c) \subseteq L$ we have

$$\models \left\{P^c\right\}\rho^c\left\{Q^c\right\} \text{ implies } \models SP^c_L(\rho^c, P^c) \rightarrow Q^c.$$

Proof
Assume $\models \left\{P\right\}\rho\left\{Q\right\}$ and let σ, δ, ω such that $OK(\sigma, \delta, \omega)$ and $\sigma, \delta, \omega \models SP^c_L(\rho^c, P^c)$. So there exist an *osi* f and a state σ_0 such that:

- $f(\sigma) = \mathcal{P}[\![\rho]\!](\gamma)(\delta')(\sigma_0)$, γ arbitrary.
- $\sigma_0, \delta', \omega' \models P^c$.

where $\delta' = f(\delta)$ and $\omega' = f(\omega) \downarrow L$. From $\models \left\{P^c\right\}\rho^c\left\{Q^c\right\}$ we then infer that $f(\sigma), \delta', \omega' \models Q^c$. By $LVar(Q^c) \subseteq L$ we have $f(\sigma), \delta', f(\omega) \models Q^c$. So by theorem 6.19 we conclude $\sigma, \delta, \omega \models Q^c$. □

6.3 Freezing the initial state

An essential notion of the standard technique for proving completeness consists of what is called freezing the initial state. To explain this notion, let, only in this paragraph, ρ denote a program of some simple procedural language (like the ones treated in [3] or [10]) and σ, σ' denote some simple functions assigning values to program variables. Let $\bar{x}$ denote the set of program variables occurring in ρ, $\bar{z}$ denote a corresponding sequence of logical variables and $\bar{x} \doteq \bar{z}$ abbreviate $\bigwedge_i (x_i \doteq z_i)$. Furthermore let $SP(\rho, \bar{x} \doteq \bar{z})$ be an assertion describing the set of final states resulting from executions of ρ starting in a state satisfying $\bar{x} \doteq \bar{z}$. In the standard completeness proof an important consequence of the definition of the notion of strongest postcondition is that the assertion $SP(\rho, \bar{x} \doteq \bar{z})$ in the following sense describes the *graph* of ρ:

- If the execution of ρ starting from the state σ results in the state σ' then $SP(\rho, \bar{x} \doteq \bar{z})$ holds in σ' when the logical variable z_i is interpreted as $\sigma(x_i)$, the value of x_i in σ.

- If $SP(\rho, \bar{x} \doteq \bar{z})$ holds in a state σ', assuming the logical variable z_i to be interpreted as some value d_i, then there exists an execution of ρ starting from the state $\sigma'\{d_i/x_i\}_i$ which results in σ'.

Note that the logical variables $\bar{z}$ are used to "freeze" the initial state.

Now one of the problems in applying the standard techniques for proving completeness to our proof system consists of how to store a state in a *finite* set of logical variables. A simple assertion like $\bar{x} = \bar{z}$ does not make sense, because a variable x can be evaluated only with respect to some object. To be able to construct an assertion which expresses how a state is stored in the logical environment we introduce some special logical variables. First we fix for each class name c the logical variables $cr_c, bl_c \in LVar_{c^*}$. Every existing object belonging to class c is supposed to be a member of the sequence denoted by cr_c. For convenience, we also include nil in cr_c. The sequence denoted by bl_c on the other hand is supposed to contain all the blocked objects belonging to class c. Furthermore for each instance variable x_d we fix a logical variable $iv_x \in LVar_{d^*}$ and, finally, for each temporary variable u_d we fix a logical variable $tv_u \in LVar_d$. The sequence denoted by iv_x, $x \in IVar^c$, will store the value of the variable x for every existing object belonging to class c in the following way: Every existing object of class c occurs at least once in the sequence denoted by cr_c. Now the ith element of the sequence iv_x is the value of the variable x in the object that is the ith element of the sequence cr_c. The value of tv_u, $u \in TVar$, just equals that of u.

All these newly introduced logical variables we assume to be distinct. We let $\bar{st}$ denote a particular sequence (without repetitions) of these logical variables. Now we are ready to define formally the assertion *init*, which expresses that the current state is represented by $\bar{st}$. In other words, *init* is our analogue of the assertion $\bar{x} = \bar{z}$.

Definition 6.24
We define the assertion *init* as follows:

$$\begin{array}{rll} init = & \bigwedge_c cr_c \cdot 1 \doteq \mathsf{nil} \wedge \forall z_c \exists i (z_c \doteq cr_c \cdot i) & \wedge \\ & \bigwedge_c \forall i (\bigwedge_{x \in IV^c}((cr_c \cdot i) \,.\, x \doteq iv_x \cdot i)) & \wedge \\ & \bigwedge_{u \in TV}(u \doteq tv_u) & \wedge \\ & \bigwedge_c (b_c \doteq bl_c) & \end{array}$$

where $IV^c = \bigcup_d IVar^c_d$, $TV = \bigcup_d TVar_d$, and the logical variable i is supposed to range over the integers. Note that in our assertion language we do *not* have equality between logical expressions of type d^*, for an arbitrary d. However, these equalities can easily be expressed in the assertion language: If l_1 and l_2 are two logical expressions ranging over sequences, then $l_1 \doteq l_2$ can be expressed as $\forall i(l_1 \cdot i \doteq l_2 \cdot i)$, where i is some logical integer variable. Furthermore we remark that for every class name c we have $init \in Ass^c$.

In the following two definitions we define a transformation of a logical expression and an assertion such that the transformed versions only refer to the logical environment. Expressions referring to the state will be translated into expressions which refer to the corresponding part of the logical environment $\bar{st}$ used to reflect the state. The problem such a transformation poses can be best explained by the following example:

Example 6.25
Suppose we want to transform the expression consisting of the instance variable x. This expression denotes the value of x with respect to the object denoted by the expression self. But to look up this value in the logical environment one has to know where the object denoted by self occurs in the sequence denoted by cr_c, assuming $x \in IVar^c_d$ for some d. However, this cannot be determined statically! Note also that we cannot force the existing objects of a class, say class c, to occur in a particular order in the sequence denoted by cr_c. Our solution to this problem consists essentially of using a second logical expression, of type Bool, to describe under which conditions the first expression correctly translates the original one. We will also need a number of logical variables that range over integers, more precisely, over indices in the sequences cr_c. In our example above, the expression x is then translated into the triple $(\langle i \rangle, \mathsf{self} \doteq cr_c \cdot i, iv_x \cdot i)$, where i is some logical integer variable. This is interpreted as follows: Whenever the variable i takes such a value that the Boolean expression $\mathsf{self} \doteq cr_c \cdot i$ is true, then the expression $iv_x \cdot i$ takes the desired value.

The analogue of these transformations in the standard completeness proof is the substitution $[\bar{z}/\bar{x}]$, where $\bar{z}$ is the part of the logical environment which is used to store the part of the state as specified by $\bar{x}$.

Definition 6.26
We define $l^c_a\lceil \bar{st} \rceil = (\bar{i}, l_1{}^c_{\mathsf{Bool}}, l_2{}^c_a)$ for an arbitrary logical expression l^c_a by induction on the structure of l^c_a. Let ϵ denote the empty sequence. We treat the following cases:

- $x^c_d\lceil \bar{st} \rceil = (\langle i \rangle, \mathsf{self} \doteq cr_c \cdot i, iv_x \cdot i)$
  where i is some fresh logical integer variable (it does not occur in $\bar{st}$).
- $u_d\lceil \bar{st} \rceil = (\epsilon, \mathsf{true}, tv_u)$
- $l\lceil \bar{st} \rceil = (\epsilon, \mathsf{true}, l)$
  where $l = \mathsf{nil}, \mathsf{self}, \mathsf{true}, \mathsf{false}, n$, or z.
- $(l_c \,.\, x^c_d)\lceil \bar{st} \rceil = (\bar{i} \circ \langle j \rangle, l_1 \wedge l_2 \doteq cr_c \cdot j, iv_x \cdot j)$
  where $l_c\lceil \bar{st} \rceil = (\bar{i}, l_1, l_2)$ and $j \notin \bar{i}$.
- $(l_1 + l_2)\lceil \bar{st} \rceil = (\bar{i}, l_{1_1} \wedge l'_{2_1}, l_{1_2} + l'_{2_2})$
  where $l_1\lceil \bar{st} \rceil = (\bar{i}_1, l_{1_1}, l_{1_2})$, $l_2\lceil \bar{st} \rceil = (\bar{i}_2, l_{2_1}, l_{2_2})$, $\bar{i} = \bar{i}_1 \circ \bar{j}$, $\bar{j}$ is some sequence of fresh logical integer variables of the same length as $\bar{i}_2$, $l'_{2_1} = l_{2_1}[\bar{j}/\bar{i}_2]$, and $l'_{2_2} = l_{2_2}[\bar{j}/\bar{i}_2]$.

- (if l_1 then l_2 else l_3 fi)$\lceil \bar{st} \rceil = (\bar{i}, l_{1_1} \wedge l'_{2_1} \wedge l'_{3_1}$, if l_{1_2} then l'_{2_2} else l'_{3_2} fi)
  where $l_1\lceil \bar{st} \rceil = (\bar{i}_1, l_{1_1}, l_{1_2})$, $l_2\lceil \bar{st} \rceil = (\bar{i}_2, l_{2_1}, l_{2_2})$, $l_3\lceil \bar{st} \rceil = (\bar{i}_3, l_{3_1}, l_{3_2})$, $\bar{i} = \bar{i}_1 \circ \bar{j}_2 \circ \bar{j}_3$, $\bar{j}_2$ and $\bar{j}_3$ are mutually disjoint sequences of fresh logical variables of the same length as $\bar{i}_2$ and $\bar{i}_3$, respectively, $l'_{2_1} = l_{2_1}[\bar{j}_2/\bar{i}_2]$, $l'_{2_2} = l_{2_2}[\bar{j}_2/\bar{i}_2]$, $l'_{3_1} = l_{3_1}[\bar{j}_3/\bar{i}_3]$, and $l'_{3_2} = l_{3_2}[\bar{j}_3/\bar{i}_3]$.

- $(l_1 \cdot l_2)\lceil \bar{st} \rceil = (\bar{i}, l_{1_1} \wedge l'_{2_1}, l_{1_2} \cdot l'_{2_2})$
  where $l_1\lceil \bar{st} \rceil = (\bar{i}_1, l_{1_1}, l_{1_2})$, $l_2\lceil \bar{st} \rceil = (\bar{i}_2, l_{2_1}, l_{2_2})$, $\bar{i} = \bar{i}_1 \circ \bar{j}$, $\bar{j}$ is some sequence of fresh logical integer variables of the same length as $\bar{i}_2$, $l'_{2_1} = l_{2_1}[\bar{j}/\bar{i}_2]$, and $l'_{2_2} = l_{2_2}[\bar{j}/\bar{i}_2]$.

- $(l_1 \doteq l_2)\lceil \bar{st} \rceil = (\bar{i}, l_{1_1} \wedge l'_{2_1}, l_{1_2} \doteq l'_{2_2})$
  where $l_1\lceil \bar{st} \rceil = (\bar{i}_1, l_{1_1}, l_{1_2})$, $l_2\lceil \bar{st} \rceil = (\bar{i}_2, l_{2_1}, l_{2_2})$, $\bar{i} = \bar{i}_1 \circ \bar{j}$, $\bar{j}$ is some sequence of fresh logical integer variables of the same length as $\bar{i}_2$, $l'_{2_1} = l_{2_1}[\bar{j}/\bar{i}_2]$, and $l'_{2_2} = l_{2_2}[\bar{j}/\bar{i}_2]$.

Note that in $l^c_a\lceil \bar{st} \rceil = (\bar{i}, l_1, l_2)$, the expression l_1 describes where the relevant existing objects, with respect to the evaluation of l^c_a, are stored in that part of the logical environment as specified by cr_c, $c \in C$. An object is said to be of relevance with respect to the evaluation of an expression if it requires the values of some variables of this object. The expression l_2 then uses this information to select the relevant values in that part of the logical environment where the values of the variables of the existing objects are stored.

Example 6.27
Consider the expression $z.x.y$, where $z \in LVar_c$, $x \in IVar^c_{c'}$. We have

$$(z.x.y)\lceil \bar{st} \rceil = (\langle i, j \rangle, z \doteq cr_c \cdot i \wedge iv_x \cdot i \doteq cr_{c'} \cdot j, iv_y \cdot j)$$

where i and j are distinct logical integer variables.

Definition 6.28
Next we define the transformation $P^c\lfloor \bar{st} \rfloor$ for an arbitrary assertion P^c by induction on the structure of P^c. We treat the following cases:

- $l^c_{\mathsf{Bool}}\lfloor \bar{st} \rfloor = \exists \bar{i}(l_1 \wedge l_2)$
  where $l^c_{\mathsf{Bool}}\lceil \bar{st} \rceil = (\bar{i}, l_1, l_2)$.

- $(P_1 \wedge P_2)\lfloor \bar{st} \rfloor = P_1\lfloor \bar{st} \rfloor \wedge P_2\lfloor \bar{st} \rfloor, \ldots$

- $(\forall z_a P)\lfloor \bar{st} \rfloor = \forall z_a P\lfloor \bar{st} \rfloor$,
  where $a = d, d^*$, $d = \mathsf{Int}, \mathsf{Bool}$.

- $(\forall z_c P)\lfloor \bar{st} \rfloor = \forall z_c(z_c \in cr_c \rightarrow P\lfloor \bar{st} \rfloor)$.

- $(\forall z_a P)\lfloor \bar{st} \rfloor = \forall z_a(z_a \subseteq cr_c \rightarrow P\lfloor \bar{st} \rfloor)$,
  where $a = c^*$.
- $(\exists z_a P)\lfloor \bar{st} \rfloor = \exists z_a P\lfloor \bar{st} \rfloor$,
  where $a = d, d^*$, $d = \mathsf{Int}, \mathsf{Bool}$.
- $(\exists z_c P)\lfloor \bar{st} \rfloor = \exists z_c(z_c \in cr_c \wedge P\lfloor \bar{st} \rfloor)$.
- $(\exists z_a P)\lfloor \bar{st} \rfloor = \exists z_a(z_a \subseteq cr_c \wedge P\lfloor \bar{st} \rfloor)$,
  where $a = c^*$.

Here $l_1 \in l_2$ abbreviates $\exists i(l_1 \doteq l_2 \cdot i)$ and $l_1 \subseteq l_2$ abbreviates $\forall i(l_1 \cdot i \in l_2)$. Note that, although $\mathsf{nil} \in cr_c$, the quantification in $(\forall z_c P)\lfloor \bar{st} \rfloor$ and $(\exists z_c P)\lfloor \bar{st} \rfloor$ excludes nil, because quantification always excludes nil.

The following theorem states that the above transformation as applied to assertions preserves truth. It can be seen as an analogue of the substitution lemma of first-order predicate logic.

Theorem 6.29
Let P^c be an arbitrary assertion. Furthermore let σ, δ, ω such that $OK(\sigma, \delta, \omega)$ and $\sigma, \delta, \omega \models init$. Then:

$$\sigma, \delta, \omega \models P^c \text{ iff } \sigma, \delta, \omega \models P^c\lfloor \bar{st} \rfloor.$$

Proof
The proof proceeds by induction on the structure of P^c. The case that P^c equals l^c_{Bool} is treated as follows: We prove that for every logical expression l^c_a there exists a sequence of integers $\bar{n}$ such that $\sigma, \delta, \omega\{\bar{n}/\bar{i}\} \models l_1$ and that for all such $\bar{n}$ we have $\mathcal{L}[\![l^c_a]\!](\omega)(\delta)(\sigma) = \mathcal{L}[\![l_2]\!](\omega\{\bar{n}/\bar{i}\})(\sigma)$, where $l^c_a\lceil \bar{st} \rceil = (\bar{i}, l_1, l_2)$. This is proved by induction on the structure of l^c_a. $\square$

6.4 Invariance

In this section we formulate a syntactic criterion for an assertion to be invariant over the execution of an arbitrary program. First we note that not allowing program variables to occur in an assertion does *not* guarantee this invariance property! This is due to the restriction of the range of the quantifiers to existing objects. Consider the following example:

Example 6.30
Let P denote the assertion $\exists z \forall z'(z \doteq z')$, where $z, z' \in LVar_c$ for some class name c.

This assertion P expresses that there exists precisely one object of class c. Let $\rho^c = \langle U|x \leftarrow \mathsf{new}\rangle$, U arbitrary and $x \in IVar^c_c$. Then it is not the case that $\models \big\{P\big\}\rho^c\big\{P\big\}$, because there exist two objects of class c in the output state.

However, the standard technique to prove completeness relies heavily on the invariance of assertions in which no program variables occur. To be able to apply this technique we define the notion of *quantification-restricted* assertions.

Definition 6.31
We define an assertion P^c to be *quantification-restricted* if

$$\begin{array}{rcl} P^c & ::= & l^c_{\mathsf{Bool}} \\ & & \vdots \\ & | & \exists z_a P \mid \forall z_a P \\ & & \text{where } a = d, d^*,\ d = \mathsf{Int}, \mathsf{Bool} \\ & | & \exists z_c(z_c \in z_{c^*} \wedge P^c) \\ & | & \exists z_{c^*}(z_{c^*} \subseteq z'_{c^*} \wedge P^c) \\ & | & \forall z_c(z_c \in z_{c^*} \rightarrow P^c) \\ & | & \forall z_{c^*}(z_{c^*} \subseteq z'_{c^*} \rightarrow P^c) \end{array}$$

Here we assume the variables z_{c^*} and z'_{c^*} to be distinct and the assertion P at the right-hand side of the symbol ::= to be quantification-restricted.

An important property of such a quantification-restricted assertion is that its truth is not affected by the creation of new objects:

Lemma 6.32
For every quantification-restricted assertion P and every variable v such that $v \notin IVar(P) \cup TVar(P)$ we have $\models P \leftrightarrow P[\mathsf{new}/v]$.

Proof
Induction on the complexity of P. We treat the representative case of $P = \exists z_c(z_c \in z_{c^*} \wedge Q)$, assuming the type of the variable v to be c: Now $P[\mathsf{new}/v] = \exists z_c(z_c \in z_{c^*} \wedge Q[\mathsf{new}/v]) \vee (v \in z_{c^*} \wedge Q[v/z_c])[\mathsf{new}/v]$. But as $(v \in z_{c^*})[\mathsf{new}/v]$ can be easily seen to be equivalent to false the second disjunct will be equivalent to false too. Furthermore we have by the induction hypothesis that $Q[\mathsf{new}/v]$ is equivalent to Q. Putting these observations together gives us the equivalence of P and $P[\mathsf{new}/v]$. The case $P = \forall z_c(z_c \in z'_{c^*} \rightarrow Q)$ is treated analogously. The cases of $P = \exists z_{c^*}(z_{c^*} \subseteq z'_{c^*} \wedge Q), \forall z_{c^*}(z_{c^*} \subseteq z'_{c^*} \rightarrow Q)$ are slightly more complex due to the complexity of the substitution operations involved, but the reasoning pattern is basically the same. □

A consequence of this lemma is the following *invariance* property of quantification-restricted assertions:

Theorem 6.33
Let $\rho^c = \langle U|c : S\rangle$ be closed and P^c be a quantification-restricted assertion such that $IVar(P^c) \cap IVar(\rho^c) = \emptyset$ and $TVar(P^c) \cap TVar(\rho^c) = \emptyset$. Then: $\vdash \{P^c\}\rho^c\{P^c\}$.

Proof
The proof proceeds by induction on the complexity of S. We consider the case of $S = v \leftarrow e_0!m(e_1, \ldots, e_n)$: Let M be the smallest set such that

- $\rho \in M$,
- if $\rho' = \langle U|c' : v' \leftarrow e'_0!m'(e'_1, \ldots, e'_k)\rangle \in M$
  then $\rho_i = \langle U|c_i : v_i \leftarrow e^i_0!m_i(e^i_1, \ldots, e^i_{n_i})\rangle \in M$,
  where $v_i \leftarrow e^i_0!m_i(e^i_1, \ldots, e^i_{n_i})$ or $e^i_0!m_i(e^i_1, \ldots, e^i_{n_i})$ occurs in S', S' being the body of the method m'. In the latter case we have $v_i = re_{d_i}$, assuming d_i to be the type of the result expression of m_i.

Let $M = \{\rho_1, \ldots, \rho_k\}$, $\rho = \rho_1$, assuming the following notational conventions: $\rho_i = \langle U|c_i : v_i \leftarrow e^i_0!m_i(e^i_1, \ldots, e^i_{n_i})\rangle \in M$ and $m_i(u^i_1, \ldots, u^i_{n_i}) \Leftarrow S_i \uparrow e_i$ occurs in U, $i = 1, \ldots, k$. Furthermore, $\bar{e}^i$ denotes the sequence $e^i_1, \ldots, e^i_{n_i}$ and $\bar{u}^i$ the sequence $u^i_1, \ldots, u^i_{n_i}$. Next we introduce for every class name c a new variable z_{c^*}. We let $\bar{z}$ denote a sequence (without repetitions) of these variables and $\bar{b}$ denote the corresponding sequence of the variables $b_c \in BVar$. Finally we put for $i = 1, \ldots, k$: $F_i = \{P'\}\rho_i\{P'\}$, where $P' = P^c[\bar{z}/\bar{b}][z_c/\mathsf{self}]$, z_c being a new variable.

Now we have that
$$F_1, \ldots, F_k \vdash \{P'\}\langle U|c'_i : S_i\rangle\{P'\}$$
(c'_i being the type of e^i_0). This is established by induction on the complexity of S_i. The only slightly less straightforward case of $S_i = v \leftarrow \mathsf{new}$ is taken care by the previous lemma.

Putting $P_i, Q_i, R_i = P'$ and introducing some logical variable $r_i \notin LVar(P')$ (of the same type as the variable v_i), $i = 1, \ldots, k$, and observing that $P'[\bar{e}^i/\mathsf{self}, \bar{u}^i][b_{c_i} \circ \langle\mathsf{self}\rangle/b_{c_i}] = P'$ we infer by (NMR) that:
$$\vdash \{P'\}\langle U|c'_1 : S_1\rangle\{P'\}.$$

Next we put $P_1, Q_1 = P'$ and $R_1 = P^c[\bar{z}/\bar{b}]$. We have that:
$$\models P_1[\bar{e}^1/\mathsf{self}, \bar{u}^1][\mathsf{self}/z_{c_1}][b_{c_1} \circ \langle\mathsf{self}\rangle/b_{c_1}] \rightarrow P^c[\bar{z}/\bar{b}]$$

and

$$\models Q_1[\bar{e}^1/\mathsf{self}, \bar{u}^1][\mathsf{self}/z_{c_1}][b_{c_1} \circ \langle \mathsf{self} \rangle / b_{c_1}] \rightarrow R_1[r_1/v_1].$$

Thus applying (MI) (or (MT)) gives us that:

$$\vdash \Big\{P^c[\bar{z}/\bar{b}]\Big\}\rho^c\Big\{P^c[\bar{z}/\bar{b}]\Big\}.$$

Finally an application of the substitution rule gives us the derivability of the correctness formula $\Big\{P^c\Big\}\rho^c\Big\{P^c\Big\}$. □

6.5 Most general correctness formulae

Now we are able to prove that for an arbitrary $\rho^c = \langle U|c : v \leftarrow e_0!m(e_1, \ldots, m_n)\rangle$ the correctness formula $\Big\{init\Big\}\rho^c\Big\{SP^c_L(\rho^c, init)\Big\}$, for some $L \subseteq LVar$, is a most general one in the sense that an arbitrary valid correctness formula can be derived from the proof system which results from adding these correctness formulae as additional axioms. Completeness then follows by establishing the derivability of $\Big\{init\Big\}\rho^c\Big\{SP^c_L(\rho^c, init)\Big\}$, for an arbitrary $\rho^c = \langle U|c : v \leftarrow e_0!m(e_1, \ldots, m_n)\rangle$.

But first we need to introduce some new logical variables corresponding to those of $\bar{st}$. This is necessary because the variables of $\bar{st}$ have a fixed interpretation as specified by the assertion *init*. But every valid correctness formula in which variables of $\bar{st}$ occur, implicitly provides these variables with some possibly different interpretation. To avoid a clash between these different interpretations we must temporarily substitute in the correctness formula, of which we want to establish its derivability, every variable of $\bar{st}$ by some corresponding new variable.

So we introduce for each c fresh logical variables $cr1_c, bl1_c \in LVar_{c^*}$. For each instance variable $x \in IVar_d$ we introduce the fresh logical variable $iv1_x \in LVar_{d^*}$, and with each temporary variable $u \in TVar_d$ we associate the fresh logical variable $tv1_u \in LVar_d$. We assume again that all these newly introduced logical variables are distinct. We let $\bar{st1}$ denote a sequence (without repetitions) of these variables. We can thus assume that $\bar{st} \cap \bar{st1} = \emptyset$.

Furthermore we introduce for every temporary variable re_d (defined in the introduction to justify the assumption of the finiteness of the sets C, $IVar$, and $TVar$) a fresh logical variable lre_d. Let $\bar{lre}$ denote a sequence of these logical variables. We will use the variable lre when applying the rule (ES): Applications of this rule will make use of the variable re to store temporarily the result of the expression s. Therefore we have to substitute occurrences of re in the precondition and the postcondition by the corresponding variable lre. We will see later how to restore the original precondition and postcondition after such an application of the rule (ES).

We start with the following lemma stating the derivability of valid correctness formulae about simple assignments.

Lemma 6.34
For an arbitrary program $\rho = \langle U|c : v \leftarrow e\rangle$ we have

$$\models \{P^c\}\rho\{Q^c\} \text{ implies } \vdash \{P^c\}\rho\{Q^c\}.$$

Proof
Let $v = u$, u some temporary variable. (The case of v being an instance variable is treated similarly.) By lemma 5.4 (note that we actually mean here the corresponding lemma for the proof system based on the assertion language with quantification over sequences) and the assumption that $\models \{P^c\}\rho\{Q\}$ it follows that $\models P^c \rightarrow Q^c[e/u]$. So an application of the axiom (SAT) and the consequence rule gives us the derivability of the correctness formula $\{P^c\}\rho\{Q\}$. □

We have a similar lemma for the creation of new objects:

Lemma 6.35
For an arbitrary program $\rho = \langle U|c : v \leftarrow \mathsf{new}\rangle$ we have

$$\models \{P^c\}\rho\{Q^c\} \text{ implies } \vdash \{P^c\}\rho\{Q^c\}.$$

Proof
Let $v = u$, u some temporary variable. (The case of v being an instance variable is treated similarly.) By lemma 5.18 and the assumption that $\models \{P^c\}\rho\{Q\}$ it follows that $\models P^c \rightarrow Q^c[\mathsf{new}/v]$. So an application of the axiom (NT) and the consequence rule gives us the derivability of the correctness formula $\{P^c\}\rho\{Q\}$. □

Next we have the following lemma stating the derivability of an arbitrary valid correctness formula about sending messages:

Lemma 6.36
Let $\rho = \langle U|c : v \leftarrow e_0!m(e_1, \ldots, e_n)\rangle$ be a closed program. Furthermore let P^c, Q^c and $BVar \subseteq L \subseteq LVar$ (L finite) such that $LVar(P,Q) \subseteq L \setminus \bar{st1}$, and $\bar{st} \cup \bar{st1} \subseteq L$. Then:

$$\models \{P^c\}\rho\{Q^c\} \text{ implies } \{init\}\rho\{SP^c_L(\rho, init)\} \vdash \{P^c\}\rho\{Q^c\}.$$

Proof
Let $P' = P[\bar{st1}/\bar{st}]$ and $Q' = Q[\bar{st1}/\bar{st}]$. Furthermore we introduce the following abbreviation: $P'' = P'\lfloor\bar{st}\rfloor$. We start with the assumption:

$$\Big\{init\Big\}\rho\Big\{SP^c_L(\rho, init)\Big\}.$$

By theorem 6.33 (note that P'' is quantification-restricted, $IVar(P'') = \emptyset$, and $TVar(P'') = \emptyset$) we have the derivability of the following formula:

$$\Big\{P''\Big\}\rho\Big\{P''\Big\}.$$

Applying the conjunction rule gives us:

$$\Big\{P'' \wedge init\Big\}\rho\Big\{P'' \wedge SP^c_L(\rho, init)\Big\}.$$

We next prove that $\models P'' \wedge SP^c_L(\rho, init) \rightarrow Q'$:
Let $\sigma, \delta, \omega \models P'' \wedge SP^c_L(\rho, init)$. So there exist a state σ_0 and an *osi* f such that

- $f(\sigma) = \mathcal{P}[\![\rho]\!](\gamma)(\delta')(\sigma_0)$, γ arbitrary,
- $\sigma_0, \delta', \omega' \models init$,

where $\delta' = f(\delta)$ and $\omega' = f(\omega) \downarrow L$.

By theorem 6.19 we have that $f(\sigma), f(\delta), f(\omega) \models P''$. It is not difficult to check that $LVar(P'') \subseteq L$, so we have $f(\sigma), \delta', \omega' \models P''$. Furthermore we have that $\models \Big\{\neg P''\Big\}\rho\Big\{\neg P''\Big\}$ (by theorem 6.33 we have $\vdash \Big\{\neg P''\Big\}\rho\Big\{\neg P''\Big\}$, so the truth of the above correctness formula follows from the soundness of the proof system). It follows that $\sigma_0, \delta', \omega' \models P''$. By theorem 6.29, note that $\sigma_0, \delta', \omega' \models init$, we then infer $\sigma_0, \delta', \omega' \models P'$. By the soundness of the substitution rule (SR) we have that $\models \Big\{P\Big\}\rho\Big\{Q\Big\}$ implies the truth of the correctness formula $\Big\{P'\Big\}\rho\Big\{Q'\Big\}$. So we infer that $f(\sigma), \delta', \omega' \models Q'$. But as $LVar(Q') \subseteq L$ we have $f(\sigma), \delta', f(\omega) \models Q'$. Finally an application of theorem 6.19 gives us the desired result $\sigma, \delta, \omega \models Q'$.

Now we return to our main argument. By the consequence rule we thus infer:

$$\Big\{P'' \wedge init\Big\}\rho\Big\{Q'\Big\}.$$

Next we apply the initialization rule (IR1):

$$\Big\{(P'' \wedge init)[\bar{u}/\bar{tv}]\Big\}\rho\Big\{Q'\Big\},$$

where $\bar{u}$ is a sequence of all the temporary variables and $\bar{tv}$ denotes the corresponding sequence of logical variables tv_u, $u \in \bar{u}$. Now we use the elimination rule (ER2):

$$\left\{\exists \bar{z}'(P'' \wedge init)[\bar{u}/\bar{tv}]\right\}\rho\left\{Q'\right\},$$

where $\bar{z}'$ is a sequence of the logical variables $\{cr_c, bl_c : c \in C\}$ and $\{iv_x : x \in IVar\}$. Note that instead of initializing the variables $\bar{tv}$ we could also eliminate them by rule (ER1). However, applying the rule (ER1) would require some additional notational machinery in order to deal with the extra case of nil.

Next we prove $\models P' \rightarrow \exists \bar{z}'(P'' \wedge init)[\bar{u}/\bar{tv}]$: Let σ, δ, ω be such that $OK(\sigma, \delta, \omega)$ and $\sigma, \delta, \omega \models P'$. It is not difficult to see that there exists an ω' such that ω' differs from ω only with respect to the variables of $\bar{st}$ and $\sigma, \delta, \omega' \models init$. As $LVar(P') \cap \bar{st} = \emptyset$ we have $\sigma, \delta, \omega' \models P'$. Applying theorem 6.29 then gives us $\sigma, \delta, \omega' \models P'\lfloor \bar{st} \rfloor$. For every temporary variable u we have $\sigma_{(3)}(u) = \omega'(tv_u)$, so we infer $\sigma, \delta, \omega' \models (P'' \wedge init)[\bar{u}/\overline{tv}]$. So we conclude $\sigma, \delta, \omega \models \exists \bar{z}'(P'' \wedge init)[\bar{u}/\bar{tv}]$.

We thus have by the consequence rule:

$$\left\{P'\right\}\rho\left\{Q'\right\}.$$

Finally an application of the substitution rule finishes the proof. Note that since $LVar(P^c, Q^c) \cap \bar{st1} = \emptyset$, we have that $P'[\bar{st}/\bar{st1}] = P^c$ and $Q'[\bar{st}/\bar{st1}] = Q^c$, so we get

$$\left\{P\right\}\rho\left\{Q\right\}.$$

□

We next have lemmas 6.38 and 6.39 stating the derivability of valid correctness formulae about statements $S = s$, where s is a side-effect expression. In these two lemmas we make use of the following lemma:

Lemma 6.37
Let $\rho = \langle U | c : s\rangle$ and $\rho' = \langle U | c : re \leftarrow s\rangle$ be restricted programs (see definition 6.12). We then have for arbitrary assertions P and Q that

$$\models \left\{P\right\}\rho\left\{Q\right\} \text{ implies } \models \left\{P'\right\}\rho'\{Q'\},$$

where $P' = P[lre/re]$ and $Q' = Q[lre/re]$.

Proof
Let $\sigma, \delta, \omega \models P'$ and $\sigma' = \mathcal{P}[\![\rho']\!](\gamma)(\delta)(\sigma)$. We have that $\sigma' = \sigma''\{\beta/re\}$, with $\langle \sigma'', \beta\rangle = \mathcal{Z}[\![s]\!](\gamma')(\delta)(\sigma)$, $\gamma' = \mathcal{U}[\![U]\!](\gamma)$. As $re \notin TVar(s)$ (ρ being restricted) we have $\langle \sigma_1, \beta\rangle = \mathcal{Z}[\![s]\!](\gamma')(\delta)(\sigma_0)$, with $\sigma_1 = \sigma''\{\omega(lre)/re\}$ and $\sigma_0 = \sigma\{\omega(lre)/re\}$). This being intuitively clear we feel justified in stating it without a proof. Now,

as $\sigma, \delta, \omega \models P'$ we have that $\sigma_0, \delta, \omega \models P$. So from $\models \{P\}\rho\{Q\}$ we then infer $\sigma_1, \delta, \omega \models Q$, or, equivalently, $\sigma'', \delta, \omega \models Q'$. Finally, as $re \notin TVar(Q')$, we conclude that $\sigma', \delta, \omega \models Q'$. □

Lemma 6.38
Let $\rho = \langle U|c : s\rangle$, where $s = e, \mathsf{new}$. Furthermore let P, Q such that $LVar(P,Q) \cap \bar{lre} = \emptyset$. Then:

$$\models \{P\}\rho\{Q\} \text{ implies } \vdash \{P\}\rho\{Q\}.$$

Proof
Let $P' = P[lre/re]$ and $Q' = Q[lre/re]$, where lre and re are of the same type as the expression s. By lemma 6.37 we have $\models \{P'\}\rho'\{Q'\}$, where $\rho' = \langle U|c : re \leftarrow s\rangle$. By lemma 6.34, in case $s = e$, and lemma 6.35, if $s = \mathsf{new}$, we then have

$$\vdash \{P'\}\rho'\{Q'\}.$$

So by rule (ES) it follows that

$$\vdash \{P'\}\rho\{Q'\}.$$

Furthermore we have $\models \{lre \doteq re\}\langle U|c : re' \leftarrow s\rangle\{lre \doteq re\}$. So again by lemmas 6.34 and 6.35 we have

$$\vdash \{lre \doteq re\}\langle U|c : re' \leftarrow s\rangle\{lre \doteq re\}.$$

Applying again the rule (ES) then gives

$$\vdash \{lre \doteq re\}\rho\{lre \doteq re\}.$$

Next we apply the conjunction rule

$$\vdash \{lre \doteq re \wedge P'\}\rho\{lre \doteq re \wedge Q'\}.$$

Now $\models (lre \doteq re \wedge Q') \rightarrow Q$ and $\models P \rightarrow (\exists lre\, P'' \vee P''[\mathsf{nil}/lre])$, where $P'' = lre \doteq re \wedge P'$. (Note that $lre \notin LVar(P)$.) So applying first the consequence rule for Q, then the elimination rule (ER1) (note that $lre \notin LVar(Q)$), and finally the consequence rule for P, gives us the derivability of

$$\vdash \{P\}\rho\{Q\}.$$

□

We have a similar lemma for valid correctness formulae about a program ρ of the form $\langle U|c : e_0!m(e_1, \ldots, e_n)\rangle$.

Lemma 6.39
Let $\rho = \langle U | c : e_0!m(e_1, \ldots, e_n)\rangle$ be a closed program. Furthermore let P, Q, and $BVar \subseteq L \subseteq LVar$ (L finite) such that $LVar(P,Q) \subseteq L \setminus (\bar{st1} \cup \bar{lre})$, $\bar{st} \cup \bar{st1} \cup \bar{lre} \subseteq L$. Then we have

$$\models \Big\{P\Big\}\rho\Big\{Q\Big\} \text{ implies } \Big\{init\Big\}\rho'\Big\{SP_L(\rho', init)\Big\} \vdash \Big\{P\Big\}\rho\Big\{Q\Big\},$$

where $\rho' = \langle U | c : re_d \leftarrow e_0!m(e_1, \ldots, e_n)\rangle$, assuming the type of the result expression of m to be d.

Proof
Let $P' = P[lre_d/re_d]$ and $Q' = Q[lre_d/re_d]$. An application of lemma 6.37 gives us $\models \Big\{P'\Big\}\rho'\Big\{Q'\Big\}$ (remember that ρ is assumed to be restricted). By lemma 6.36 we have

$$\Big\{init\Big\}\rho'\Big\{SP_L(\rho', init)\Big\} \vdash \Big\{P'\Big\}\rho'\Big\{Q'\Big\}.$$

Applying next the rule (ES) gives us

$$\Big\{init\Big\}\rho'\Big\{SP_L(\rho, init)\Big\} \vdash \Big\{P'\Big\}\rho\Big\{Q'\Big\}.$$

By theorem 6.33 (observe that $re_d \notin TVar(\rho)$) we have the derivability of the formula

$$\vdash \Big\{lre_d \doteq re_d\Big\}\rho\Big\{lre_d \doteq re_d\Big\}.$$

So by an application of the conjunction rule we have

$$\Big\{init\Big\}\rho\Big\{SP_L(\rho, init)\Big\} \vdash \Big\{P' \wedge lre_d \doteq re_d\Big\}\rho\Big\{Q' \wedge lre_d \doteq re_d\Big\}.$$

Now we have $\models (Q' \wedge lre_d \doteq re_d) \rightarrow Q$. Furthermore for $P'' = P' \wedge lre_d \doteq re_d$ we have $\models P \rightarrow (\exists lre_d P'' \vee P''[\mathsf{nil}/lre_d])$ (note that $lre_d \notin LVar(P)$). So first applying the consequence rule for Q, then the elimination rule (ER1) (note that $lre_d \notin LVar(Q)$), and finally the consequence rule for P finishes the proof. □

Next we have the following main theorem of this section stating the derivability of an arbitrary valid correctness formula using as additional axioms the correctness formulae of the form $\Big\{init\Big\}\rho\Big\{SP_L(\rho, init)\Big\}$, where $\rho = \langle U | c : v \leftarrow e_0!m(e_1, \ldots, e_n)\rangle$.

Theorem 6.40
Let $\rho = \langle U | c : S\rangle$ be a closed program. Furthermore let P^c, Q^c, and $BVar \subseteq L \subseteq LVar$ (L finite) such that $LVar(P^c, Q^c) \subseteq L \setminus (\bar{st1} \cup \bar{lre})$, $\bar{st} \cup \bar{st1} \cup \bar{lre} \subseteq L$. Then:

$$\models \Big\{P^c\Big\}\rho\Big\{Q^c\Big\} \text{ implies } F_1, \ldots F_n \vdash \Big\{P^c\Big\}\rho\Big\{Q^c\Big\},$$

where $F_i = \{init\}\rho_i\{SP_L^{c_i}(\rho_i, init)\}$, $\rho_i = \langle U | c_i : v_i \leftarrow s_i\rangle$, with $s_1, \ldots, s_n$ being all the send-expressions occurring in S such that $v_i \leftarrow s_i$ occurs in S or $v_i = re_{d_i}$ and s_i occurs as a statement in S. Here d_i is assumed to be the type of s_i.

Proof
The proof proceeds by induction on the complexity of S.

$S = v \leftarrow s$: Depending on the structure of s, by one of the lemmas 6.34, 6.35, 6.36.

$S = s$: Depending on the structure of s, by one of the lemmas 6.38, 6.39.

$S = S_1; S_2$:
Let $L^- = L \setminus (\bar{st1} \cup \bar{lre})$. We have by lemma 6.22

$$\models \Big\{P^c\Big\}\rho_1\Big\{SP_{L^-}(\rho_1, P^c)\Big\},$$

and

$$\models \Big\{SP_{L^-}(\rho_1, P^c)\Big\}\rho_2\Big\{SP_{L^-}(\rho_2, SP_{L^-}(\rho_1, P^c))\Big\},$$

where $\rho_i = \langle U|c : S_i\rangle$. By the induction hypothesis we have

$$F_1, \ldots, F_n \vdash \Big\{P^c\Big\}\rho_1\Big\{SP_{L^-}(\rho_1, P^c)\Big\},$$

and

$$F_1, \ldots, F_n \vdash \Big\{SP_{L^-}(\rho_1, P^c)\Big\}\rho_2\Big\{SP_{L^-}(\rho_2, SP_{L^-}(\rho_1, P^c))\Big\}.$$

It thus suffices to prove that $\models SP_{L^-}(\rho_2, SP_{L^-}(\rho_1, P^c)) \rightarrow Q^c$: An application of the rule for sequential composition (SC) and the consequence rule then gives us the desired result.

So suppose that $\sigma, \delta, \omega \models SP_{L^-}(\rho_2, SP_{L^-}(\rho_1, P^c))$, with $OK(\sigma, \delta, \omega)$. By theorem 6.21 there exist a state σ_0 and an *osi* f such that

- $f(\sigma) = \mathcal{P}[\![\rho_2]\!](\gamma)(\delta')(\sigma_0)$, γ arbitrary,
- $\sigma_0, \delta', \omega' \models SP_{L^-}(\rho_1, P^c)$,

where $\delta' = f(\delta)$ and $\omega' = f(\omega) \downarrow L^-$.
Now $\sigma_0, \delta', \omega' \models SP_{L^-}(\rho_1, P^c)$ in turn implies that there exist a state σ_0' and an *osi* g such that

- $g(\sigma_0) = \mathcal{P}[\![\rho_1]\!](\gamma)(\delta'')(\sigma_0')$, γ arbitrary,
- $\sigma_0', \delta'', \omega'' \models P^c$,

where $\delta'' = g(\delta')$ and $\omega'' = g(\omega') \downarrow L^-$.
To relate these computations of ρ_1 and ρ_2 we apply corollary C.8 of appendix C: There exists an *osi* h such that $h^c \downarrow \sigma_0^{(c)} = g^c \downarrow \sigma_0^{(c)}$, for every c, and $h(f(\sigma)) = \mathcal{P}[\![\rho_2]\!](\gamma)(g(\delta'))(g(\sigma_0))$, where γ is arbitrary.

Since $g(\delta') = \delta''$ it follows that $h(f(\sigma)) = \mathcal{P}[\![\rho]\!](\gamma)(\delta'')(\sigma_0')$, with γ arbitrary. So by $\sigma_0', \delta'', \omega'' \models P^c$ and $\models \{P^c\}\rho\{Q^c\}$ we infer $h(f(\sigma)), \delta'', \omega'' \models Q^c$.

Now note that $OK(\sigma_0, \delta')$. So we have $h(\delta'_{(1)}) = g^c(\delta'_{(1)}) = \delta''_{(1)}$ and $h(\delta'_{(2)(c)}) = g(\delta'_{(2)(c)}) = \delta''_{(2)(c)}$, for every c. Thus we infer that $\delta'' = h(\delta') = h(f(\delta))$. Moreover for $z \in L^-$ we have $h(f(\omega(z))) = h(\omega'(z)) = g(\omega'(z)) = \omega''(z)$. Note that the second identity is justified by $OK(\sigma_0, \delta', \omega')$. So by theorem 6.19 and the fact that $LVar(Q^c) \subseteq L^-$ we conclude $\sigma, \delta, \omega \models Q^c$.

$S = \mathsf{if} \ldots \mathsf{fi}$: Straightforward.

$S = \mathsf{while}\ e\ \mathsf{do}\ S_1\ \mathsf{od}$:
In order to deal with this case we construct a loop invariant R as follows. Let $L^- = L \setminus (\overline{st1} \cup \overline{lre})$ and $L^+ = L^- \cup \{z_u, z_{u'}\}$, where z_u and $z_{u'}$ are some new logical integer variables. We define $P' = P[z_u, z_{u'}/u, u']$ and $Q' = Q[z_u, z_{u'}/u, u']$. Let $\rho' = \langle U | c : \mathsf{while}\ e \wedge u < u'\ \mathsf{do}\ S_1; u \leftarrow u + 1\ \mathsf{od}\rangle$. Furthermore let $R' = SP_{L^+}(\rho', P' \wedge u \doteq 0)$ and define $R = \exists z\, R'[z, z/u, u']$, where $z \in LVar_{\mathsf{Int}}$ is a new variable. Note that $LVar(R) \subseteq L^+$. Furthermore we have $\models \{P'\}\rho\{Q'\}$ (note that $u, u' \notin TVar(\rho)$, ρ being restricted).

We have $\models P' \rightarrow R$:
Let $\sigma, \delta, \omega \models P'$, with $OK(\sigma, \delta, \omega)$. We prove that for $\omega' = \omega\{0/z\}$ we have $\sigma, \delta, \omega' \models R'[z, z/u, u']$. Now $\sigma, \delta, \omega' \models R'[z, z/u, u']$ iff $\sigma', \delta, \omega \models R'$ by a straightforward extension of lemma 5.4 (note that $z \notin Exp$), where $\sigma' = \sigma\{0, 0/u, u'\}$ (note that $z \notin LVar(R')$). Because $u, u' \notin TVar(P')$ we have $\sigma', \delta, \omega \models P' \wedge u \doteq 0$. Furthermore it is easy to see that $\sigma' = \mathcal{P}[\![\rho']\!](\gamma)(\delta)(\sigma')$, with γ arbitrary. Finally, as $LVar(P') \subseteq L^+$ we have by theorem 6.21 $\sigma', \delta, \omega \models R'$.

Next we prove $\models R \wedge \neg e \rightarrow Q'$:
Let $\sigma, \delta, \omega \models R \wedge \neg e$. So let $\alpha \in \mathbf{N}$ such that $\sigma', \delta, \omega \models R'$, where $\sigma' = \sigma\{\alpha, \alpha/u, u'\}$. So there exist f, σ_0 such that

- $f(\sigma') = \mathcal{P}[\![\rho']\!](\gamma)(\delta')(\sigma_0)$, γ arbitrary,
- $\sigma_0, \delta', \omega' \models P' \wedge u \doteq 0$,

where $\delta' = f(\delta)$ and $\omega' = f(\omega) \downarrow L^+$. Now $u, u' \notin TVar(e)$ so $\sigma, \delta, \omega \models \neg e$ implies $\sigma', \delta, \omega \models \neg e$. By theorem 6.19 we have $f(\sigma'), \delta', f(\omega) \models \neg e$. So from $LVar(e) = \emptyset$

it follows that $f(\sigma'), \delta', \omega' \models \neg e$. From this it is not difficult to derive that $f(\sigma') = \mathcal{P}\llbracket\rho\rrbracket(\gamma)(\delta')(\sigma'_0)$, where $\sigma'_0 = \sigma_0\{\alpha, \alpha/u, u'\}$. Now as $u, u' \notin TVar(P')$ it follows that $\sigma'_0, \delta', \omega' \models P'$. So by $\models \left\{P'\right\}\rho\left\{Q'\right\}$ we have $f(\sigma'), \delta', \omega' \models Q'$. By $LVar(Q') \subseteq L^+$ and theorem 6.19 we have $\sigma', \delta, \omega \models Q'$. So that from $u, u' \notin TVar(Q')$ we finally conclude $\sigma, \delta, \omega \models Q'$.

Finally, we have $\models \left\{R \wedge e\right\}\rho_1\left\{R\right\}$, where $\rho_1 = \langle U|c : S_1\rangle$:
Let $\sigma_0, \delta, \omega \models R \wedge e$, with $OK(\sigma_0, \delta, \omega)$, and $\sigma_1 = \mathcal{P}\llbracket\rho_1\rrbracket(\gamma)(\delta)(\sigma_0)$, with γ arbitrary. Let $\alpha \in \mathbf{N}$ such that $\sigma'_0, \delta, \omega \models R'$, where $\sigma'_0 = \sigma_0\{\alpha, \alpha/u, u'\}$. So there exist f, σ such that

- $f(\sigma'_0) = \mathcal{P}\llbracket\rho'\rrbracket(\gamma)(\delta')(\sigma)$, γ arbitrary,
- $\sigma, \delta', \omega' \models P' \wedge u \doteq 0$,

where $\delta' = f(\delta)$ and $\omega' = f(\omega) \downarrow L^+$.
So we have the following situation:

$$\begin{array}{ccc} & \sigma_0, \delta & \xrightarrow{\rho_1} & \sigma_1 \\ & | & & \\ & \sigma'_0 & & \\ & | & & \\ \sigma, \delta' \xrightarrow{\rho'} & f(\sigma'_0) & & \end{array}$$

Here $\sigma, \delta \xrightarrow{\rho} \sigma'$ should be interpreted as $\sigma' = \mathcal{P}\llbracket\rho\llbracket(\gamma)(\delta)(\sigma)$, γ arbitrary. We have $\sigma'_0, \delta, \omega \models e$ because $u, u' \notin TVar(e)$. So by theorem 6.19 and $LVar(e) = \emptyset$ we infer $f(\sigma'_0), \delta', \omega' \models e$. Now let $\sigma'_1 = \sigma_1\{\alpha, \alpha/u, u'\}$. It then follows that $\sigma'_1 = \mathcal{P}\llbracket\rho_1\llbracket(\gamma)(\delta)(\sigma'_0)$. We now have the following situation:

$$\begin{array}{cccc} & \sigma_0, \delta & \xrightarrow{\rho_1} & \sigma_1 \\ & | & & | \\ & \sigma'_0, \delta & \xrightarrow{\rho_1} & \sigma'_1 \\ & | & & \\ \sigma, \delta' \xrightarrow{\rho'} & f(\sigma'_0) & & \end{array}$$

An application of corollary C.8 then gives us an *osi* g such that $g^c \downarrow {\sigma'}_0^{(c)} = f^c \downarrow {\sigma'}_0^{(c)}$ for every c, and $g(\sigma'_1) = \mathcal{P}\llbracket\rho_1\rrbracket(\gamma)(g(\delta))(f(\sigma'_0))$, with γ arbitrary. Note that from $OK(\sigma'_0, \delta)$ it then follows that $g(\delta) = f(\delta) = \delta'$. Finally, we thus have reached the following situation:

$$\begin{array}{ccccc} & & \sigma_0,\delta & \xrightarrow{\rho_1} & \sigma_1 \\ & & | & & | \\ & & \sigma_0',\delta & \xrightarrow{\rho_1} & \sigma_1' \\ & & | & & | \\ \sigma,\delta' & \xrightarrow{\rho'} & f(\sigma_0'),\delta' & \xrightarrow{\rho_1} & g(\sigma_1') \end{array}$$

Now it follows that for $\sigma_2 = \sigma\{\alpha + 1/u'\}$ and $\sigma_3 = g(\sigma_1')\{\alpha + 1, \alpha + 1/u, u'\}$ we have $\sigma_3 = \mathcal{P}[\![\rho']\!](\gamma)(\delta')(\sigma_2)$, with γ arbitrary. (Of course this can be proved formally, but as the intuition behind a formal proof is quite obvious, the main idea being simply that the temporary variable u counts the number of loops, we think we are justified in omitting such a proof.) Now $\sigma, \delta', \omega' \models P'$, $u, u' \notin TVar(P')$, so $\sigma_2, \delta', \omega' \models P'$, from which in turn it follows by lemma 6.22 that $\sigma_3, \delta', \omega' \models R'$. So we infer $g(\sigma_1'), \delta', \omega' \models R$. Now $LVar(R) \subseteq L^+$ and for $z \in L^+$ we have $g(\omega(z)) = f(\omega(z)) = \omega'(z)$ (the first identity follows from $OK(\sigma_0', \omega)$) so we have $g(\sigma_1'), \delta', g(\omega) \models R$. It follows by an application of theorem 6.19 that $\sigma_1', \delta, \omega \models R$. Finally, as we have $u, u' \notin TVar(R)$ we conclude $\sigma_1, \delta, \omega \models R$.

Now by $\models \{R \wedge e\}\rho_1\{R\}$ it follows that $\models \{R'' \wedge e\}\rho_1\{R''\}$ (note that $u, u' \notin TVar(\rho_1)$), where $R'' = R[u, u'/z_u, z_{u'}]$. As $LVar(R'') \subseteq L^-$ we can apply the induction hypothesis:

$$F_1, \ldots, F_n \vdash \{R'' \wedge e\}\rho_1\{R''\}.$$

By theorem 6.33 we have

$$\vdash \{z_u \doteq u \wedge z_{u'} \doteq u'\}\rho_1\{z_u \doteq u \wedge z_{u'} \doteq u'\}.$$

Furthermore we have $\models (R'' \wedge z_u \doteq u \wedge z_{u'} \doteq u') \to R$ and $R \to (R'' \wedge z_u \doteq u \wedge z_{u'} \doteq u')[z_u, z_{u'}/u, u']$ (note that $u, u' \notin TVar(R)$). So applying the conjunction rule, the consequence rule for the postcondition, the initialization rule (IR2), and the consequence rule for the precondition gives us

$$F_1, \ldots, F_n \vdash \{R \wedge e\}\rho_1\{R\}.$$

From an application of the rule (W) and the consequence rule, using the truth of the implications $P' \to R$ and $R \wedge \neg e \to Q'$, it then follows that:

$$F_1, \ldots, F_n \vdash \{P'\}\rho\{Q'\}.$$

Now again by an application of theorem 6.33 and the conjunction rule we have

$$F_1, \ldots, F_n \vdash \{P' \wedge z_u \doteq u \wedge z_{u'} \doteq u'\}\rho\{Q' \wedge z_u \doteq u \wedge z_{u'} \doteq u'\}.$$

We have $\models (Q' \wedge z_u \doteq u \wedge z_{u'} \doteq u') \to Q$ and $\models P \to (P' \wedge z_u \doteq u \wedge z_{u'} \doteq u')[u, u'/z_u, z_{u'}]$. So applying first the consequence rule for Q, then the initialization rule (IR1), and finally the consequence rule for P gives us the desired result. $\square$

6.6 The context switch

In this subsection we prove the derivability of the correctness formula $\{init\}\rho\{SP^c_L(\rho, init)\}$, for $\rho = \langle U|c : v \leftarrow e_0!m(e_1,\ldots,e_n)\rangle$ closed and $BVar \subseteq L \subseteq LVar$ such that $\bar{st} \cup \bar{st1} \cup \bar{lre} \subseteq L$. From now on until the end of this section unless stated otherwise we assume ρ and L to be fixed. We want to apply the rule (NMR) and theorem 6.40. To apply the rule (NMR) we need the following definition:

Definition 6.41
Let M be the smallest set such that

- $\rho \in M$,
- if $\rho' = \langle U|c' : v' \leftarrow e'_0!m'(e'_1,\ldots,e'_k)\rangle \in M$
  then $\rho_i = \langle U|c_i : v_i \leftarrow e^i_0!m_i(e^i_1,\ldots,e^i_{n_i})\rangle \in M$,
  where $v_i \leftarrow e^i_0!m_i(e^i_1,\ldots,e^i_{n_i})$ or $e^i_0!m_i(e^i_1,\ldots,e^i_{n_i})$ occurs in S' as a statement (in this latter case we have $v_i = re_{d_i}$, assuming d_i to be the type of the result expression of m_i), S' being the body of the method m'.

Let $M = \{\rho_1,\ldots,\rho_k\}$, $\rho = \rho_1$, assuming the following notational conventions: $\rho_i = \langle U|c_i : v_i \leftarrow e^i_0!m_i(e^i_1,\ldots,e^i_{n_i})\rangle \in M$, and $m_i(u^i_1,\ldots,u^i_{n_i}) \Leftarrow S_i \uparrow e_i$ occurs in U, $i = 1,\ldots,k$. We let $\bar{e}^i$ denote the sequence $e^i_0,\ldots,e^i_{n_i}$. Furthermore let $\bar{u}$ be a sequence of all the temporary variables, and let the formal parameters of the method m_i be denoted by $\bar{u}^i$.

We start with a sketch of the proof strategy. To apply theorem 6.40 and the rule (NMR) we have to define assertions P_i, Q_i, $i = 1,\ldots,k$, such that $LVar(P_i, Q_i) \subseteq L \setminus (\bar{st1} \cup \bar{lre})$, and

$$\models \Big\{P_i \wedge \bigwedge_j v^i_j \doteq \mathsf{nil} \wedge \mathsf{self} \notin b_{c'_i}\Big\}\langle U|c'_i; S_i\rangle\Big\{Q_i[e_i/r_i]\Big\}, \tag{6.1}$$

where $\bar{v}^i = \bar{u} \setminus \bar{u}^i$ and c'_i is the type of e^i_0,

$$\models init \rightarrow P_i[\bar{e}^i/\mathsf{self}, \bar{u}^i][\bar{g}^i/\bar{z}^i][b_{c_i} \circ \langle\mathsf{self}\rangle/b_{c_i}] \tag{6.2}$$

and

$$\models Q_i[\bar{e}^i/\mathsf{self}, \bar{u}^i][\bar{g}^i/\bar{z}^i][b_{c_i} \circ \langle\mathsf{self}\rangle/b_{c_i}] \rightarrow SP^{c_i}_L(\rho_i, init)[r_i/v_i], \tag{6.3}$$

for some sequence of expressions $\bar{g}^i$ and corresponding sequence of logical variables $\bar{z}^i$. Here r_i for $i = 1,\ldots,k$ is a logical variable of the same type as v_i. By 6.1 an application of theorem 6.40 then gives us

$$F'_1,\ldots,F'_k \vdash \Big\{P_i \wedge \bigwedge_j v^i_j \doteq \mathsf{nil} \wedge \mathsf{self} \notin b_{c'_i}\Big\}\langle U|c'_i : S_i\rangle\Big\{Q_i[e_i/r_i]\Big\}$$

where

$$F_i' = \Big\{init\Big\}\rho_i\Big\{SP_L^{c_i}(\rho_i, init)\Big\}.$$

Furthermore by an application of the consequence rule, using (6.2), we have $F_i \vdash F_i'$ where

$$F_i = \Big\{P_i[\bar{e}^i/\mathsf{self}, \bar{u}^i][\bar{g}^i/\bar{z}^i][b_{c_i} \circ \langle\mathsf{self}\rangle/b_{c_i}]\Big\}\rho_i\Big\{SP_L^{c_i}(\rho_i, init)\Big\}.$$

So we have

$$F_1, \ldots, F_k \vdash \Big\{P_i \wedge \bigwedge_j v_j^i \doteq \mathsf{nil} \wedge \mathsf{self} \not\in b_{c_i'}\Big\}\langle U|c_i' : S_i\rangle\Big\{Q_i[e_i/r_i]\Big\}.$$

An application of (NMR) plus (MI) or (MT) and the consequence rule, using (6.2) and (6.3), then concludes the proof.

We start with the considering equations (6.2) and (6.3): We define a substitution which neutralizes the context switch. To do so we first introduce some new logical variables.

Definition 6.42
We associate with $u \in \bar{u}$ a new logical variable $tv2_u$ of the same type and with each $c \in C$ a new logical variable id_c. We define $\overline{tv2}$ to be the sequence of logical variables $tv2_u$ corresponding to the sequence $\bar{u}$. Finally let $\bar{id}^i$, $i = 1, \ldots, k$, denote the sequence consisting of the variable id_{c_i} followed by the elements of $\overline{tv2}$.

We have the following lemma about the neutralizing capacity of the substitution $[\bar{id}^i/\mathsf{self}, \bar{u}]$ with respect to the context switch:

Lemma 6.43
For any $i \in \{1, \ldots, k\}$ and every assertion $P \in Ass^{c_i}$ we have

$$P^{c_i}[\bar{id}^i/\mathsf{self}, \bar{u}][\bar{e}^i/\mathsf{self}, \bar{u}^i] = P^{c_i}[\bar{id}^i/\mathsf{self}, \bar{u}].$$

Proof
Straightforward induction on the complexity of P^{c_i}. □

Note that the substitution $[\bar{id}^i/\mathsf{self}, \bar{u}]$ transforms the assertion P^{c_i} into an assertion in Ass^c for arbitrary c. Furthermore it is easy to see that if $LVar(P) \cap \bar{id}^i = \emptyset$ then $\models P^{c_i} \leftrightarrow P^{c_i}[\bar{id}^i/\mathsf{self}, \bar{u}][\bar{f}/\bar{id}^i]$, where $\bar{f}$ denotes the sequence consisting of the expression self followed by the elements of $\bar{u}$. Note that in general we do *not* have that P^{c_i} is syntactically equal to $P^{c_i}[\bar{id}^i/\mathsf{self}, \bar{u}][\bar{f}/\bar{id}^i]$, as is shown by the following example:

Example 6.44
Take for $P^{c_i} = x \doteq z.y$, where $z \notin \bar{id}^i$. We have $P^{c_i}[\bar{id}^i/\mathsf{self}, \bar{u}] = id_{c_i}.x \doteq z.y$ and $(id_{c_i}.x \doteq z.y)[\bar{f}/\bar{id}^i] = \mathsf{self}.x \doteq z.y$.

Next we consider the substitution $[b_{c_i} \circ \langle\mathsf{self}\rangle/b_{c_i}]$. It is not difficult to see that for every assertion P^{c_i} we have

$$\models (P^{c_i}[bl_{c_i}/b_{c_i}] \wedge b_{c_i} = bl_{c_i} \circ \langle\mathsf{self}\rangle)[b_{c_i} \circ \langle\mathsf{self}\rangle/b_{c_i}] \rightarrow P^{c_i}.$$

But note that we do *not* have the other way around! However, as $\models init \rightarrow bl_{c_i} = b_{c_i}$, we *do* have

$$\models init \rightarrow ((init[bl_{c_i}/b_{c_i}] \wedge b_{c_i} = bl_{c_i} \circ \langle\mathsf{self}\rangle)[b_{c_i} \circ \langle\mathsf{self}\rangle/b_{c_i}]).$$

To summarize the argument above we introduce the following definition:

Definition 6.45
For any $i \in \{1, \ldots, k\}$ and any assertion $P \in Ass^{c_i}$ we define its *reverse context switch* $\mathbf{R}(P^{c_i})$ as follows:

$$\mathbf{R}(P^{c_i}) = (P^{c_i}[bl_{c_i}/b_{c_i}] \wedge b_{c_i} = bl_{c_i} \circ \langle\mathsf{self}\rangle)[\bar{id}^i/\mathsf{self}, \bar{u}]$$

We have the following lemma about this reverse context switch:

Lemma 6.46
For any $i \in \{1, \ldots, k\}$ and every assertion $P \in Ass^{c_i}$ we have

$$\models \mathbf{R}(P^{c_i})[\bar{e}^i/\mathsf{self}, \bar{u}][\bar{f}/\bar{id}^i][b_{c_i} \circ \langle\mathsf{self}\rangle/b_{c_i}] \rightarrow P^{c_i}.$$

and if $\models P^{c_i} \rightarrow b_{c_i} \doteq bl_{c_i}$ then

$$\models P^{c_i} \rightarrow \mathbf{R}(P^{c_i})[\bar{e}^i/\mathsf{self}, \bar{u}][\bar{f}/\bar{id}^i][b_{c_i} \circ \langle\mathsf{self}\rangle/b_{c_i}]).$$

Here $\bar{f} = \mathsf{self}, \bar{u}$.

Proof
Clear from the above. □

So at this stage candidates for P_i, Q_i, $i = 1, \ldots, k$, satisfying equations (6.2) and (6.3) are the assertions $\mathbf{R}(init)$ and $\mathbf{R}(SP_L^{c_i}(\rho_i, init)[r_i/v_i])$, $i = 1, \ldots, k$. We now proceed by analyzing equation (6.1). Suppose we are given that for some P and Q we have $\models \{P\}\rho_i\{Q\}$. In general we do *not* have

$$\models \Big\{\mathbf{R}(P) \wedge \bigwedge_j v^i_j \doteq \mathsf{nil} \wedge \mathsf{self} \notin b_{c'_i}\Big\}\langle U|c'_i : S_i\rangle\Big\{\mathbf{R}(Q')[e_i/r_i]\Big\},$$

where $Q' = Q[r_i/v_i]$. This is because it is possible that the object executing S_i is not the object which is sent the message and furthermore nothing is said about the values of the formal parameters. So we add to $\mathbf{R}(P)$ the information $\mathsf{self} \doteq e_0^i[\bar{id}^i/\mathsf{self}, \bar{u}]$ and $u_j^i \doteq e_j^i[\bar{id}^i/\mathsf{self}, \bar{u}]$, $j = 1, \ldots, n_i$. We have the following lemma:

Lemma 6.47

$$\models (f_j^i \doteq (e_j^i[\bar{id}^i/\mathsf{self}, \bar{u}]))[\bar{e}^i/\mathsf{self}, \bar{u}][\bar{f}/\bar{id}^i]$$

where $\bar{f}^i = \mathsf{self}, \bar{u}^i$ and $\bar{f} = \mathsf{self}, \bar{u}$.

Proof
Easy. □

Note that from lemma 6.46 and lemma 6.47 it follows that for every P^{c_i} such that $\models P^{c_i} \to b_{c_i} \doteq bl_{c_i}$ we have

$$\models P \to \Big(\mathbf{R}(P) \wedge \bigwedge_j f_j^i \doteq (e_j^i[\bar{id}^i/\mathsf{self}, \bar{u}])\Big)[\bar{e}^i/\mathsf{self}, \bar{u}][\bar{f}/\bar{id}^i][b_{c_i} \circ \langle \mathsf{self} \rangle / b_{c_i}].$$

Now we are ready for the following lemma which shows how to transform a valid correctness formula about sending a message into a valid formula about the execution of the body of the message by the receiver:

Lemma 6.48
For any $i \in \{1, \ldots, k\}$ and every $P, Q \in Ass^{c_i}$ such that $\models \big\{P\big\}\rho_i\big\{Q\big\}$ we have

$$\models \Big\{P' \wedge \bigwedge_j v_j^i \doteq \mathsf{nil} \wedge \mathsf{self} \notin b_{c_i'}\Big\}\langle U|c_i' : S_i\rangle\Big\{Q'[e_i/r_i]\Big\},$$

where $P' = \mathbf{R}(P) \wedge \bigwedge_j f_j^i \doteq (e_j^i[\bar{id}^i/\mathsf{self}, \bar{u}])$ and $Q' = \mathbf{R}(Q[r_i/v_i])$, with r_i a fresh logical variable of the same type as v_i. Here $\bar{v}^i = \bar{u} \setminus \bar{u}^i$.

Proof
Let $\sigma, \delta, \omega \models P' \wedge \bigwedge_j v_j^i \doteq \mathsf{nil} \wedge \mathsf{self} \notin b_{c_i'}$, for σ, δ, ω such that $OK(\sigma, \delta, \omega)$, and $\sigma' = \mathcal{P}[\![\langle U|c_i' : S_i\rangle]\!](\gamma)(\delta)(\sigma)$, with γ arbitrary and $\sigma' \neq \perp$.

We define $\sigma_1 = \sigma\{\omega(tv2_u)/u\}_{u \in \bar{u}}$ and $\delta'_{(1)} = \omega(id_{c_i})$, $\delta'_{(2)(c)} = \delta_{(2)(c)}$ for every c.

It follows from lemma 5.28 that $\sigma_1, \delta', \omega \models P[bl_{c_i}/b_{c_i}] \wedge b_{c_i} = bl_{c_i} \circ \langle \mathsf{self} \rangle$.

Next we define δ'' as follows: $\delta''_{(1)} = \delta'_{(1)}$, $\delta''_{(2)(c_i)} = \delta'_{(2)(c_i)} \setminus \{\omega(id_{c_i})\}$, and $\delta''_{(2)(c)} = \delta'_{(2)(c)}$ for any $c \neq c_i$. Furthermore we put $\omega_1 = \omega\{\omega(bl_{c_i})/b_{c_i}\}$. It then follows that $OK(\sigma_1, \delta'', \omega_1)$ and $\sigma_1, \delta'', \omega_1 \models P$.

$$\begin{array}{ccc}
\sigma, \delta, \omega \models P' & & \sigma', \delta, \omega \models Q'[e_i/r_i] \\
\Downarrow & & \Uparrow \\
\sigma_1, \delta', \omega \models P[bl_{c_i}/b_{c_i}] \wedge b_{c_i} = bl_{c_i} \circ \langle \mathsf{self} \rangle & & \sigma'', \delta', \omega_3 \models Q[bl_{c_i}, r_i/b_{c_i}, v_i] \wedge b_{c_i} = bl_{c_i} \circ \langle \mathsf{self} \\
\Downarrow & & \Uparrow \\
\sigma_1, \delta'', \omega_1 \models P & \Rightarrow & \sigma_2, \delta'', \omega_1 \models Q
\end{array}$$

Figure 4: The structure of the proof of lemma 6.48.

Let on the other hand $\sigma'' = \sigma'\{\omega(tv2_u)/u\}_{u \in \bar{u}}$ and

$$\sigma_2 = \begin{cases} \sigma''\{\beta/v_i\}) & \text{if } v_i \in TVar \\ \sigma''\{\beta/\omega(id_{c_i}), v_i\} & \text{if } v_i \in IVar, \end{cases}$$

where $\beta = \mathcal{E}[\![e_i]\!](\delta)(\sigma')$ (remember that e_i is the result expression of the method m_i). Now from $\sigma, \delta, \omega \models \bigwedge_j f^i_j = (e^i_j[\bar{id}^i/\mathsf{self}, \bar{u}])$ it follows from lemma 5.28 that $\delta_{(1)} = \mathcal{L}[\![e^i_0[\bar{id}^i/\mathsf{self}, \bar{u}]]\!](\omega)(\delta)(\sigma) = \mathcal{L}[\![e^i_0]\!](\omega)(\delta'')(\sigma_1)$ and $\sigma_{(3)}(u^i_j) = \mathcal{L}[\![e^i_j]\!](\omega)(\delta'')(\sigma_1)$. Furthermore from $\sigma, \delta, \omega \models \mathsf{self} \notin b_{c'_i}$ it in turn follows that $\delta_{(1)} \notin \delta_{(2)(c'_i)}$. Now putting this together with the assumption that $\sigma' = \mathcal{P}[\![\langle U | c'_i : S_i \rangle]\!](\gamma)(\delta)(\sigma)$, using $\sigma, \delta, \omega \models \bigwedge_j v^i_j \doteq \mathsf{nil}$, enables one to infer that $\sigma_2 = \mathcal{P}[\![\rho_i]\!](\gamma)(\delta'')(\sigma_1)$.

Furthermore we are given that $\models \{P\}\rho_i\{Q\}$ so from $\sigma_1, \delta'', \omega_1 \models P$ and $\sigma_2 = \mathcal{P}[\![\rho_i]\!](\gamma)(\delta'')(\sigma_1)$ we infer that $\sigma_2, \delta'', \omega_1 \models Q$. Now let $\omega_2 = \omega_1\{\beta/r_i\}$. It then follows by lemma 5.8 that $\sigma'', \delta'', \omega_2 \models Q[r_i/v_i]$. Next we note that as $\omega_2(b_{c_i}) = \omega_1(b_{c_i}) = \omega(bl_{c_i})$ we have $\sigma'', \delta', \omega_3 \models Q[r_i, bl_{c_i}/v_i, b_{c_i}]$, where $\omega_3 = \omega\{\beta/r_i\}$.

From $\sigma, \delta, \omega \models \mathbf{R}(P)$ we infer that $\omega(b_{c_i}) = \omega(bl_{c_i}) \circ \langle \omega(id_{c_i}) \rangle$. But $\omega(id_{c_i}) = \delta'_{(1)}$ so we have $\sigma'', \delta', \omega_3 \models Q[r_i, bl_{c_i}/v_i, b_{c_i}] \wedge b_{c_i} = bl_{c_i} \circ \langle \mathsf{self} \rangle$.

Now an application of lemma 5.28 gives us $\sigma', \delta, \omega_3 \models \mathbf{R}(Q[r_i/v_i])$. From this in turn it follows that $\sigma', \delta, \omega \models Q'[e_i/r_i]$. □

Now we want to apply lemma 6.48 taking *init* for P and $SP^{c_i}_L(\rho_i, init)$ for Q. Note that by lemma 6.22 we have $\models \{init\}\rho_i\{SP^{c_i}_L(\rho_i, init)\}$. Now taking for P_i the assertion $\mathbf{R}(init) \wedge \bigwedge_j f^i_j \doteq (\bar{e}^i_j[\bar{z}/\mathsf{self}, \bar{u}])$ and for Q_i the assertion $\mathbf{R}(SP^{c_i}_L(\rho_i, init)[r_i/v_i])$ we have by lemma 6.46 and lemma 6.47 that equations (6.2) and (6.3) are satisfied. However since in the assertions P_i and Q_i new logical variables occur which are not contained in L, we must apply theorem 6.40 for $F_i = \{init\}\rho_i\{SP^{c_i}_{L^+}(\rho_i, init)\}$, where $L^+ = L \cup \{id_c : c \in C\} \cup \{tv2_u : u \in \bar{u}\}$. But to apply the rule (NMR) we then have to take for Q_i the assertion $\mathbf{R}(SP^{c_i}_{L^+}(\rho_i, init)[r_i/v_i])$. An application of (NMR) and (MI) or (MT) would then give us the derivability of the correctness formula $\{init\}\rho\{SP^c_{L^+}(\rho, init)\}$. However, as $\models SP^c_{L^+}(\rho, init) \rightarrow SP^c_L(\rho, init)$ (use $LVar(init) \subseteq L \subseteq L^+$), we have by an application of the consequence rule the derivability of $\{init\}\rho\{SP^c_L(\rho, init)\}$.

But there is one problem we did not discuss yet. As $\bar{st1} \cup \bar{lre} \subseteq LVar(SP^{c_i}_{L^+}(\rho_i, init))$ we can not apply theorem 6.40! This problem is solved as follows: First we define $L^- = L^+ \setminus (\bar{st1} \cup \bar{lre})$. Next we define the following abbreviation:

Definition 6.49
Let $Subs(\bar{lre}, \bar{st1}, \bar{cr})$ abbreviate the assertion:

$$\bigwedge_c (cr1_c \subseteq cr_c \wedge bl1_c \subseteq cr_c \wedge lre_c \in cr_c \wedge \bigwedge_{d \in C} \bigwedge_{x \in IVar^c_d} iv1_x \subseteq cr_d \wedge \bigwedge_{u \in TVar_c} tv1_u \in cr_c).$$

The assertion $Subs(\bar{lre}, \bar{st1}, \bar{cr})$ states that all the objects which are denoted by a variable of $\bar{lre}$ or $\bar{st1}$, or which occur in a sequence denoted by some variable of $\bar{st1}$, are stored in the corresponding variable of $\bar{cr}$. We have the following proposition:

Proposition 6.50
Let $P_i = \mathbf{R}(init) \wedge \bigwedge_j f^i_j \doteq (e^i_j[\bar{id}^i/\mathsf{self}, \bar{u}])$, $Q^-_i = \mathbf{R}(SP^{c_i}_{L^-}(\rho_i, init)[r_i/v_i])$ and $Q^+_i = \mathbf{R}(SP^{c_i}_{L^+}(\rho_i, init)[r_i/v_i])$. We have

$$\models P_i \wedge Subs(\bar{lre}, \bar{st1}, \bar{cr}) \leftrightarrow P_i$$

and

$$\models Q^-_i[e_i/r_i] \wedge Subs(\bar{lre}, \bar{st1}, \bar{cr}) \rightarrow Q^+_i[e_i/r_i].$$

Proof
The first assertion follows immediately from the fact that the assertion $init$ (and so the assertion $\mathbf{R}(init)$) implies the assertion $\forall z_c(z_c \in cr_c)$, for every c.

Now we prove the second assertion. Let $\sigma, \delta, \omega \models Q^-_i[e_i/r_i] \wedge Subs(\bar{lre}, \bar{st1}, \bar{cr})$. For $\omega_1 = \omega\{\mathcal{E}[\![e_i]\!](\delta)(\sigma)/r_i\}$, we then have $\sigma, \delta, \omega_1 \models Q^-_i \wedge Subs(\bar{lre}, \bar{st1}, \bar{cr})$.

Next we define $\sigma' = \sigma\{\omega_1(tv2_u)/u\}_{u \in \bar{u}}$, and $\delta'_{(1)} = \omega_1(id_{c_i})$, $\delta'_{(2)(c)} = \delta_{(2)(c)}$, for every c. It then follows by lemma 5.28 that: $\sigma', \delta', \omega_1 \models SP^{c_i}_{L^-}(\rho_i, init)[r_i, bl_{c_i}/v_i, b_{c_i}] \wedge b_{c_i} = bl_{c_i} \circ \langle \mathsf{self} \rangle \wedge Subs(\bar{lre}, \bar{st1}, \bar{cr})$.

For $\omega_2 = \omega_1\{\omega_1(bl_{c_i})/b_{c_i}\}$ and $\delta''_{(1)} = \delta'_{(1)}$, for $c \neq c_i$: $\delta''_{(2)(c)} = \delta'_{(2)(c)}$, otherwise: $\delta''_{(2)(c)} = \delta'_{(2)(c)} \setminus \delta'_{(1)}$, we have $\sigma', \delta'', \omega_2 \models SP^{c_i}_{L^-}(\rho_i, init)[r_i/v_i] \wedge Subs(\bar{lre}, \bar{st1}, \bar{cr})$.

Next, let

$$\sigma'' = \begin{cases} \sigma'\{\omega_2(r_i)/\delta'_{(1)}, v_i\} & \text{if } v_i \in IVar \\ \sigma'\{\omega_2(r_i)/v_i\} & \text{if } v_i \in TVar. \end{cases}$$

It follows that $\sigma'', \delta'', \omega_2 \models SP^{c_i}_{L^-}(\rho_i, init) \wedge Subs(\bar{lre}, \bar{st1}, \bar{cr})$.

So by theorem 6.21 there exist f and σ_0 such that:

- $f(\sigma'') = \mathcal{P}[\![\rho]\!](\gamma)(f(\delta''))(\sigma_0)$, with γ arbitrary,
- $\sigma_0, f(\delta''), \omega' \models init$,

where $\omega' = f(\omega_2) \downarrow L^-$. Let $\sigma_1 = f(\sigma'')$. Now by theorem 6.19 we have that $\sigma_1, f(\delta''), f(\omega_2) \models Subs(\bar{lre}, \bar{st1}, \bar{cr})$. So from $\{cr_c : c \in C\} \subseteq L^-$ and the compatibility of ω' and σ_0 we then infer the compatibility of $f(\omega_2) \downarrow L^+$ and σ_0. Let $\omega'' = f(\omega_2) \downarrow L^+$. We have that $\sigma_0, f(\delta''), \omega'' \models init$, so we have $\sigma'', \delta'', \omega_2 \models SP^{c_i}_{L^+}(\rho_i, init)$. From this it follows, by "reversing" the part of the above argument which led to the statement $\sigma'', \delta'', \omega_2 \models SP^{c_i}_{L^-}(\rho_i, init)$, that $\sigma, \delta, \omega \models Q^+_i[e_i/r_i]$. □

Now we are ready for the following theorem.

Theorem 6.51
Let the program $\rho = \langle U | c : v \leftarrow e_0!m(e_1, \ldots, e_n)\rangle$ be closed and let $BVar \subseteq L \subseteq LVar$ such that $\bar{st} \cup \bar{st1} \cup \bar{lre} \subseteq L$. Then we have

$$\vdash \{init\}\rho\{SP^c_L(\rho, init)\}.$$

Proof
Let $P_i = \mathbf{R}(init) \wedge \bigwedge_j f^i_j \doteq (e^i_j[\bar{id}^i/\mathsf{self}, \bar{u}])$, $Q^-_i = \mathbf{R}(SP^{c_i}_{L^-}(\rho_i, init)[r_i/v_i])$ and $Q^+_i = \mathbf{R}(SP^{c_i}_{L^+}(\rho_i, init)[r_i/v_i])$. Now by lemma 6.22 we get

$$\models \{init\}\rho_i\{SP^{c_i}_{L^-}(\rho_i, init)\}$$

So we have, by lemma 6.48,

$$\models \Big\{P_i \wedge \bigwedge_j v^i_j \doteq \mathsf{nil} \wedge \mathsf{self} \notin b_{c'_i}\Big\}\langle U | c'_i : S_i\rangle\Big\{Q^-_i[e_i/r_i]\Big\}.$$

An application of theorem 6.40 then gives us (note that the restrictions on the logical variables are satisfied)

$$F'_1, \ldots, F'_k \vdash \Big\{P_i \wedge \bigwedge_j v^i_j \doteq \mathsf{nil} \wedge \mathsf{self} \notin b_{c'_i}\Big\}\langle U | c'_i : S_i\rangle\Big\{Q^-_i[e_i/r_i]\Big\},$$

where

$$F_i' = \Big\{init\Big\}\rho_i\Big\{SP^{c_i}_{L^+}(\rho_i, init)\Big\}.$$

Now by lemma 6.46 and lemma 6.47 an application of the consequence rule gives us $F_i \vdash F_i'$ where

$$F_i = \Big\{P_i[\bar{e}^i/\mathsf{self}, \bar{u}^i][\bar{f}/\bar{id}^i][b_{c_i} \circ \langle\mathsf{self}\rangle/b_{c_i}]\Big\}\rho_i\Big\{SP^{c_i}_{L^+}(\rho_i, init)\Big\}.$$

So we have

$$F_1, \ldots, F_k \vdash \Big\{P_i \wedge \bigwedge_j v^i_j \doteq \mathsf{nil} \wedge \mathsf{self} \notin b_{c_i'}\Big\}\langle U|c_i' : S_i\rangle\Big\{Q_i^-[e_i/r_i]\Big\}.$$

By theorem 6.33 we have

$$\vdash \Big\{Subs(\bar{lre}, \bar{st1}, \bar{cr})\Big\}\langle U|c_i' : S_i\rangle\Big\{Subs(\bar{lre}, \bar{st1}, \bar{cr})\Big\}.$$

So by the conjunction rule we infer

$$F_1, \ldots, F_m \vdash \begin{array}{c} \Big\{P_i \wedge \bigwedge_j v^i_j \doteq \mathsf{nil} \wedge \mathsf{self} \notin b_{c_i'} \wedge Subs(\bar{lre}, \bar{st1}, \bar{cr})\Big\} \\ \langle U|c_i' : S_i\rangle \\ \Big\{Q_i^-[e_i/r_i] \wedge Subs(\bar{lre}, \bar{st1}, \bar{cr})\Big\}. \end{array}$$

By proposition 6.50 an application of the consequence rule gives us

$$F_1, \ldots, F_m \vdash \Big\{P_i \wedge \bigwedge_j v^i_j \doteq \mathsf{nil} \wedge \mathsf{self} \notin b_{c_i'}\Big\}\langle U|c_i' : S_i\rangle\Big\{Q_i^+[e_i/r_i]\Big\}.$$

We now can apply rule (NMR), making use of lemma 6.46, yielding the derivability of the correctness formula:

$$\Big\{P_1 \wedge \bigwedge_j v^1_j \doteq \mathsf{nil} \wedge \mathsf{self} \notin b_{c_1'}\Big\}\langle U|c_1' : S_1\rangle\Big\{Q_1^+[e_1/r_1]\Big\}.$$

Applying next (MI) or (MT) gives us the derivability of

$$\Big\{P_1[\bar{e}^1/\mathsf{self}, \bar{u}^1][\bar{f}/\bar{z}^1][b_{c_1} \circ \langle\mathsf{self}\rangle/b_{c_1}]\Big\}\rho_1\Big\{SP_{L^+}(\rho_1, init)\Big\}.$$

So an application of the consequence rule (the assertion *init* by lemma 6.46 implies the precondition, and $\models SP_{L^+}(\rho_1, init) \rightarrow SP_L(\rho_1, init)$) gives us the desired result (note that $\rho_1 = \rho$ by definition)

$$\vdash \Big\{init\Big\}\rho\Big\{SP^c_L(\rho, init)\Big\}.$$

□

We conclude with the completeness theorem:

Theorem 6.52
Let $\rho^c = \langle U|c : S\rangle$ be a closed program. We have for an arbitrary correctness formula $\left\{P^c\right\}\rho^c\left\{Q^c\right\}$:

$$\models \left\{P^c\right\}\rho^c\left\{Q^c\right\} \text{ implies } \vdash \left\{P^c\right\}\rho^c\left\{Q^c\right\}.$$

Proof
Let P' and Q' result from substituting for every variable of $\bar{st1}$ and $\bar{lre}$ a corresponding new variable (new with respect to the sets $LVar(P^c, Q^c), \bar{st}, \bar{st1}, \bar{lre}$). Let $L \subseteq LVar$ (L finite) be such that $BVar \subseteq L$, $LVar(P', Q') \subseteq L$ and $\bar{st} \cup \bar{st1} \cup \bar{lre} \subseteq L$. By the soundness of the substitution rule we have $\models \left\{P'\right\}\rho^c\left\{Q'\right\}$, so applying theorem 6.40 gives us

$$F_1, \ldots F_n \vdash \left\{P'\right\}\rho^c\left\{Q'\right\},$$

where $F_i = \{init\}\rho_i\{SP_L^{c_i}(\rho_i, init)\}$, $\rho_i = \langle U|c_i : v_i \leftarrow e_0^i!m_i(e_1^i, \ldots, e_{n_i}^i)\rangle$ and $e_0^i!m_i(e_1^i, \ldots, e_{n_i}^i$ $i = 1, \ldots, n$, are all the send-expressions occurring in S, and if such an expression $e_0^i!m_i(e_1^i, \ldots, e_{n_i}^i)$ occurs in S as a statement we have that $v_i = re_{d_i}$, assuming d_i to be the type of the result expression of m_i. By theorem 6.51 we have the derivability of F_i, so we infer that $\vdash \left\{P'\right\}\rho^c\left\{Q'\right\}$. Finally an application of the substitution rule gives us the derivability of $\left\{P^c\right\}\rho^c\left\{Q^c\right\}$. □

7 Conclusions

In the previous sections we have given a proof system for SPOOL that fulfills the requirements we have listed in the introduction:

- The only possible operations on object references (pointers) are testing for equality and dereferencing.
- In each state of the system only the existing objects play a role in assertions about that state.

In fact, we have given even *two* proof systems fulfilling these requirements: one with recursively defined predicates and one with the ability to reason about finite sequences of objects.

The technique which we have given for computing the weakest precondition for an assignment with respect to a given postcondition, a generalized version of substitution, seems very powerful. Especially the fact that is possible to do this for a new assignment, in the situation that it is not possible to mention the newly created object in the state before the statement, is a little bit surprising.

The proof rule for message passing, incorporating the passing of parameters and result, context switching, and the constancy of the variables of the sending object, is a very complex rule. It seems to work fine for our proof system, but its properties have not yet been studied extensively enough. It would be interesting to see whether the several things that are handled in one rule could be dealt with by a number of different, simpler rules.

We have proved completeness for the proof system based on the assertion language containing quantification over finite sequences using the standard techniques (see [3], for example). But how to apply these techniques to the proof system based on recursive predicates remains an open problem.

Therefore we must conclude that there is still some work to be done on these issues. In addition, in the present proof systems the protection properties of object are not reflected very well. While in the programming language it is not possible for one object to access the internal details (variables) of another one, in the assertion language this *is* allowed. In order to improve this it might be necessary to develop a system in which an object presents some abstract view of its behaviour to the outside world. Perhaps techniques developed to deal with abstract data types are useful here.

Finally it is clear that the work on SPOOL is meant as a preparation for the study of POOL, the parallel language. In the following two chapters of this thesis we show

how to combine a system like the one presented here with the known techniques for reasoning about parallel programs.

References

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

[2] Pierre America: *A proof theory for a sequential version of POOL.* ESPRIT project 415A, Doc. No. 188, Philips Research Laboratories, Eindhoven, the Netherlands.

[3] Krzysztof 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.

[4] J.W. de Bakker: *Mathematical Theory of Program Correctness.* Prentice-Hall International, Englewood Cliffs, New Jersey, 1980.

[5] Herbert B. Enderton: *A Mathematical Introduction to Logic.* Academic Press, 1972.

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

[7] Joseph M. Morris: *Assignment and linked data structures.* Manfred Broy, Gunther Schmidt (eds.): Theoretical Foundations of Programming Methodology. Reidel, 1982, pp. 35–41.

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

[9] Joseph R. Shoenfield: *Mathematical Logic.* Addison-Wesley, 1967.

[10] John V. Tucker, Jeffery I. Zucker: *Program Correctness over Abstract Data Types, with Error-State Semantics.* CWI Monograph Series, Vol. 6, Centre for Mathematics and Computer Science/North-Holland, 1988.

A A generalisation of the rule (MR)

In this section we show that in the recursion rule (MR), as introduced in definition 5.33 and adapted in definition 6.4, we can replace U^- by U itself, thus allowing nested applications of (MR) to the same methods. Let (NMR) denote the recursion rule resulting from (MR) by replacing all occurrences of U^- by U. Furthermore let $|\vdash$ denote the derivability using (NMR) ($\vdash$ denotes derivability using (MR)). We have the following theorem:

Theorem A.1
For every correctness formula F we have $|\vdash F$ iff $\vdash F$.

Proof
$\Rightarrow$: We prove that if $F_1, \ldots, F_n |\vdash F$ then $F_1, \ldots, F_n \vdash F$ by induction on the length of the derivation. We treat the case that the last rule applied is (NMR). So let the following be an instance of (NMR):

$$\frac{\bar{F}_1, \ldots, \bar{F}_n \qquad F_1, \ldots, F_n |\vdash F'_1, \ldots, F'_n}{F'_1}$$

where $F'_1 = F$. Let U be the unit occurring in this application of (NMR). We may assume without loss of generality that all the methods declared by U are specified by one of the F_i. (Otherwise, let $\{\rho_1, \ldots, \rho_k\}$, where $\rho_i = \langle U|c_i : v_i \leftarrow e_0^i!m_i(e_1^i, \ldots, e_{n_i}^i)\rangle$, be all the send statements occurring in U. Now simply add to $F_1, \ldots, F_n$ for $i = 1, \ldots, k$, $G_i = \{\mathsf{true}\}\rho_i\{\mathsf{true}\}$, and note that

$$G_1, \ldots, G_k \vdash \{\mathsf{true}\}\langle U|c'_i : S_i\rangle\{\mathsf{true}\}$$

where c'_i is the type of e_0^i and S_i denotes the body of m_i.) We shall prove by induction on the number of applications of (NMR) in the derivation $F_1, \ldots, F_n |\vdash F'_1, \ldots, F'_n$ that for some $\bar{H}_1, \ldots, \bar{H}_k$, $H_1, \ldots, H_k$, $H'_1, \ldots, H'_k$ such that for $1 \leq i \leq k$

$$\frac{H'_i \qquad \bar{H}_i}{H_i}$$

is an instance of (MI) or (MT), we have:

$$\bar{F}_1, \ldots, \bar{F}_n, \bar{H}_1, \ldots, \bar{H}_k \vdash \bar{F}'_1, \ldots, \bar{F}'_n, \bar{H}'_1, \ldots, \bar{H}'_k$$

where, for $G = \{P\}\langle U|c : S\rangle\{Q\}$, $\bar{G}$ denotes $\{P\}\langle E|c : S\rangle\{Q\}$, E being the empty unit. Having proved this we apply (MR) thus yielding $\vdash F'_1 (= F)$. Here we go:

Induction basis: Assume that no application of (NMR) occurs in the derivation $F_1, \ldots, F_n |\vdash F'_1, \ldots, F'_n$. So we have that $F_1, \ldots, F_n \vdash^- F'_1, \ldots, F'_n$, where $\vdash^-$ denotes derivability from $\vdash$ without (MR). It is not difficult to see that it suffices to prove by induction on the length of the derivation that for an arbitrary correctness formula G if $F_1, \ldots, F_n \vdash^- G$ then for some $\tilde{H}_1, \ldots, \tilde{H}_k$, $H_1, \ldots, H_k$, $H'_1, \ldots, H'_k$ we have

$$\bar{F}_1, \ldots, \bar{F}_n, \bar{H}_1, \ldots, \bar{H}_k \vdash \bar{G}, \bar{H}'_1, \ldots, \bar{H}'_k,$$

where for $i = 1, \ldots, k$

$$\frac{H'_i \qquad \tilde{H}_i}{H_i}$$

is an instance of (MI) or (MT). We treat the only interesting case that the last rule applied is an instance of (MI) or (MT). So suppose $F_1, \ldots, F_n \vdash^- G', \tilde{G}$, where

$$\frac{G' \qquad \tilde{G}}{G}$$

is an instance of (MI) or (MT). Now by the induction hypothesis we know that for some $\tilde{H}_1, \ldots, \tilde{H}_k, H_1, \ldots, H_k$, $H'_1, \ldots, H'_k$:

$$\bar{F}_1, \ldots, \bar{F}_n, \bar{H}_1, \ldots, \bar{H}_k \vdash \bar{G}', \bar{H}'_1, \ldots, \bar{H}'_k,$$

such that for $i = 1, \ldots, k$

$$\frac{H'_i \qquad \tilde{H}_i}{H_i}$$

is an instance of (MI) or (MT). Now let $\tilde{H}_{k+1} = \tilde{G}$, $H_{k+1} = G$, and $H'_{k+1} = G'$. We then have that

$$\bar{F}_1, \ldots, \bar{F}_n, \bar{H}_1, \ldots, \bar{H}_{k+1} \vdash \bar{G}, \bar{H}'_1, \ldots, \bar{H}'_{k+1}.$$

Induction step: Let for $i = 1, \ldots, m$

$$\frac{\tilde{G}^i_1, \ldots, \tilde{G}^i_{n_i} \qquad G^i_1, \ldots, G^i_{n_i} \Vdash G^{i\,\prime}_1, \ldots, G^{i\,\prime}_{n_i}}{G^{i\,\prime}_1}$$

be all the applications of (NMR) in the derivation $F_1, \ldots, F_n \Vdash F'_1, \ldots, F'_n$ such that

$$F_1, \ldots, F_n, G^{1\prime}_1, \ldots, G^{m\prime}_1 \vdash^- F'_1, \ldots, F'_n.$$

By the same induction argument as used in the basis step above we have for some $\tilde{H}_1, \ldots, \tilde{H}_k$, $H_1, \ldots, H_k$, $H'_1, \ldots, H'_k$ (such that for $i = 1, \ldots, k$

$$\frac{H'_i \qquad \tilde{H}_i}{H_i}$$

is an instance of (MI) or (MT)) that

$$F_1, \ldots, F_n, H_1, \ldots, H_k, G^{1\prime}_1, \ldots, G^{m\prime}_1 \vdash^= F'_1, \ldots, F'_n, H'_1, \ldots, H'_k,$$

where $\vdash^=$ denotes derivability from $\vdash$ minus the rules (MR), (MI), and (MT). Now, applying the induction hypothesis gives us for $i = 1, \ldots, m$: $\tilde{H}_1^i, \ldots, \tilde{H}_{k_i}^i$, $H_1^i, \ldots, H_{k_i}^i$, $H_1^{i\prime}, \ldots, H_{k_i}^{i\;\prime}$ such that

$$\bar{G}_1^i, \ldots, \bar{G}_{n_i}^i, \bar{H}_1^i, \ldots, \bar{H}_{k_i}^i \vdash \bar{G}_1^{i\prime}, \ldots, \bar{G}_{n_i}^{i\;\prime}, \bar{H}_1^{i\prime}, \ldots, \bar{H}_{k_i}^{i\;\prime}.$$

Now it follows by a straightforward induction on the length of the derivation

$$F_1, \ldots, F_n, H_1, \ldots, H_k, G_1^{1\prime}, \ldots, G_1^{m\prime} \vdash^= F_1', \ldots, F_n', H_1', \ldots, H_k'$$

that

$$\mathcal{F} \cup \bigcup_i \mathcal{G}_i \cup \bigcup_i \mathcal{H}_i \vdash \mathcal{F}' \cup \bigcup_i \mathcal{G}_i' \cup \bigcup_i \mathcal{H}_i'$$

where

- $\mathcal{F} = \{\bar{F}_1, \ldots, \bar{F}_{n+k}\}$, $F_{n+i} = H_i$, $i = 1, \ldots, k$,
- $\mathcal{F}' = \{\bar{F}', \ldots, \bar{F}_{n+k}'\}$, $F_{n+i}' = H_i'$, $i = 1, \ldots, k$,
- $\mathcal{G}_i = \{\bar{G}_1^i, \ldots, \bar{G}_{n_i}^i\}$, $1 \leq i \leq m$,
- $\mathcal{G}_i' = \{\bar{G}_1^{i\prime}, \ldots, \bar{G}_{n_i}^{i\;\prime}\}$, $1 \leq i \leq m$,
- $\mathcal{H}_i = \{\bar{H}_1^i, \ldots, \bar{H}_{k_i}^i\}$, $1 \leq i \leq m$,
- $\mathcal{H}_i' = \{\bar{H}_1^{i\prime}, \ldots, \bar{H}_{k_i}^{i\;\prime}\}$, $1 \leq i \leq m$.

$\Leftarrow$: This is proved in a similar way as the other direction. $\square$

B Expressibility

In this section we show how to formulate the assertion $SP^c_L(\rho^c, P^c)$ in our assertion language, for an arbitrary closed program ρ^c, $BVar \subseteq L \subseteq LVar$ (L finite), such that $LVar(P^c) \subseteq LVar$.

As in section 6 we assume the sets C, $IVar$, and $TVar$ to be finite.

B.1 Coding mappings

Assumption B.1
We assume the existence of the following *coding* mappings:

- For every instance variable or temporary variable $v \in ITVar$ we have $[v] \in \mathbf{N}$, and for an arbitrary program ρ we have $[\rho] \in \mathbf{N}$.
- For every $d \in C^+$, $[.]_d \in \mathbf{O}^d_\perp \to \mathbf{N}$ denotes an injection such that $[\perp]_d = 0$. In addition, we assume that the function $[.]_{\mathsf{Int}}$ is surjective.
- For every state $\sigma \in \Sigma$ such that $OK(\sigma)$, $[\sigma] \in \mathbf{N}$.
- For every context $\delta^c \in \Delta^c$: $[\delta^c] \in \mathbf{N}$.

Furthermore we assume that the mappings $[.]_{\mathsf{Int}}$ and $[.]_{\mathsf{Bool}}$ are definable in our assertion language. That is, we regard the following function symbols as abbreviations for assertions that are expressible in our assertion language:

- $Ic(n) = m$ (mnemonic: *integer coding*) iff $[n]_{\mathsf{Int}} = m$.
- $Bc(b) = m$ (mnemonic: *Boolean coding*) iff $[b]_{\mathsf{Bool}} = m$.
- $Id(n) = m$ (mnemonic: *integer decoding*) iff $[m]_{\mathsf{Int}} = n$.

To be precise, with the first assumption above we mean that there is an assertion $Ic(z_1) = z_2$, where z_1 and z_2 are integer logical variables, such that for every $\sigma \in \Sigma, \delta \in \Delta, \omega \in \Omega$ with $OK(\sigma, \delta, \omega)$ we have

$$\sigma, \delta, \omega \models Ic(z_1) = z_2 \qquad \text{iff} \qquad [\omega(z_1)]_{\mathsf{Int}} = \omega(z_2).$$

In fact from now on for every $c \in C$ and $\alpha \in \mathbf{O}^c$ we identify $[\alpha]_c$ with α. So we assume $\mathbf{O}^c \subseteq \mathbf{N}$.

In the same say we assume the following predicates and functions to be expressible in our assertion language:

- $E^c(n,m)$ (mnemomic: *exists*) iff there exist a $\sigma \in \Sigma$ and $\alpha \in \sigma^{(c)}$ such that $[\alpha]_c = n$ and $[\sigma] = m$.
- $A^c(n) = m$ (mnemonic: *active*) iff there exists a $\delta \in \Delta^c$ such that $[\delta] = n$ and $[\delta_{(1)}]_c = m$.
- $B^c(n,m)$ (mnemonic: *blocked*) iff there exist a $\delta \in \Delta$ and an $\alpha \in \delta_{(2)(c)}$ such that $[\alpha]_c = n$ and $[\delta] = m$.
- $Val^c_d(k,l,m) = n$ (mnemonic: *value*) iff there exist $\sigma \in \Sigma$, $\alpha \in \sigma^{(c)}$, and $x^c_d \in IVar^c_d$ such that $[\alpha]_c = k$, $[x^c_d] = l$, $[\sigma] = m$, and $[\sigma(\alpha)(x^c_d)]_d = n$.
- $Val_d(l,m) = n$ iff there exist a $\sigma \in \Sigma$ and a $u_d \in TVar_d$ such that $[u_d] = l$, $[\sigma] = m$, and $[\sigma(u_d)]_d = n$.
- $T^c(n,m,l,k)$ (mnemonic: *transforms*) iff there exist a closed $\rho \in Prog^c, \delta \in \Delta^c$, and $\sigma, \sigma' \in \Sigma$ such that $OK(\sigma,\delta)$, $[\rho] = n$, $[\delta] = m$, $[\sigma] = l$, $[\sigma'] = k$, and $\sigma' = \mathcal{P}^c[\![\rho]\!](\gamma)(\delta)(\sigma)$ (where γ is arbitrary).

The above assumptions may appear quite implausible at first sight, but they can be justified by Church's Thesis, which states that every function or relation that can be effectively calculated is recursive, together with the (mathematical) fact that every recursive function is representable in the standard Peano theory of natural numbers and therefore it is certainly definable in our assertion language. (For a discussion of these issues, see [5] or [9].)

B.2 Arithmetizing Truth

To express the strongest postcondition we have to arithmetize the truth of an assertion in a state. More precisely, we will define a translation which transforms an arbitrary assertion into an assertion in which no instance variables or temporary variables occur. The idea of this translation is similar to the one given in the definitions 6.26 and 6.28. But instead of transforming an assertion into an assertion referring to a sequence of logical variables used to store the state, we now transform it into an assertion referring to the *code* of a state. This is necessary to be able to use the predicates of assumption B.1, in particular the predicate T.

To get started we introduce some new variables: Let $\overline{bij}$ denote a sequence of some variables $bij^c \in LVar_{c^*}$, $c \in C$. We shall use these variables to store the essential parts of the bijections that constitute an *osi* (see definition 6.17). The way in which this is done will be made precise in definition B.3, but here we can already explain how the $\overline{bij}$ can be used as a kind of decoding tables. To that end we assume that we have a certain state σ such that for every $c \in C$ and $\alpha \in \mathbf{O}^c_\perp$

$$elt(\beta, [\alpha]_c) = \begin{cases} \alpha & \text{if } \alpha \in \sigma^{(c)} \\ \perp & \text{otherwise.} \end{cases}$$

where $\beta \in \mathbf{O}^{c^*}$ is the value of bij^c in a certain ω. So every existing object of class c occurs in the sequence denoted by bij^c at a position which equals its code number. It is important to note that we cannot express this property of the sequence denoted by bij^c in the assertion language: There exists no assertion $P(bij^c)$ such that for every σ, δ, ω with $OK(\sigma, \delta, \omega)$ we have $\sigma, \delta, \omega \models P(bij^c)$ precisely if the above property holds. This is because at the level of the assertion language objects simply are not integers. Fortunately we shall not need the expressibility of exactly this property, but only of this property modulo an *osi*. This is the subject of section B.3.

Definition B.2
Let z^α, z^σ be some logical integer variables. We assume that the value of z^σ equals the code $[\sigma]$ of some state σ, and that the value of z^α equals $[\alpha]_c$ for some $\alpha \in \sigma^{(c)}$. For every logical expression l^c_d we define $l^c_d\lceil z^\alpha, z^\sigma\rceil$ as a triple $(\bar{\imath}, l_1{}^c_{\mathsf{Bool}}, l_2{}^c_{\mathsf{Int}})$, where $\bar{\imath}$ denotes a sequence of logical integer variables, and l_1 and l_2 are logical expressions. Note that we do *not* define this transformation for logical expressions of type d^* with $d \in C^+$. The idea behind this transformation $l^c_d\lceil z^\alpha, z^\sigma\rceil = (\bar{\imath}, l_1, l_2)$ can be described as follows: The expression l_1 is constructed such that it is only true if the variables $\bar{\imath}$ contain the code numbers of certain objects that are relevant for the evaluation of l^c_d. To do this, l_1 can consult the variables $\overline{bij}$ as a translation table from code numbers to actual objects. Using this information, l_2 is a translation of l^c_d such that every operation on objects described by l^c_d is translated into a corresponding *arithmetical* operation on code numbers.

Here is the formal definition:

- $x^c_d\lceil z^\alpha, z^\sigma\rceil = (\epsilon, \mathsf{true}, Val^c_d(z^\alpha, k, z^\sigma))$,
  where $k = [x^c_d]$ and ϵ is the empty sequence.

- $u_d\lceil z^\alpha, z^\sigma\rceil = (\epsilon, \mathsf{true}, Val_d(k, z^\sigma))$,
  where $k = [u_d]$.

- $\mathsf{nil}\lceil z^\alpha, z^\sigma\rceil = (\epsilon, \mathsf{true}, 0)$.

- $\mathsf{self}\lceil z^\alpha, z^\sigma\rceil = (\epsilon, \mathsf{true}, z^\alpha)$.

- $l\lceil z^\alpha, z^\sigma\rceil = (\epsilon, \mathsf{true}, Bc(l))$,
  where $l = \mathsf{true}, \mathsf{false}$.

- $n\lceil z^\alpha, z^\sigma\rceil = (\epsilon, \mathsf{true}, [n]_{\mathsf{Int}})$.

- $z_{\mathsf{Int}}\lceil z^\alpha, z^\sigma\rceil = (\epsilon, \mathsf{true}, Ic(z_{\mathsf{Int}})$,
  $z_{\mathsf{Bool}}\lceil z^\alpha, z^\sigma\rceil = (\epsilon, \mathsf{true}, Bc(z_{\mathsf{Bool}})$.

- $z_c\lceil z^\alpha, z^\sigma\rceil = (\langle i\rangle, \mathsf{if}\ z_c \doteq \mathsf{nil}\ \mathsf{then}\ i \doteq 0\ \mathsf{else}\ bij^c \cdot i = z_c\ \mathsf{fi}, i)$.

- $(l^c_{c'} . x^{c'}_d)\lceil z^\alpha, z^\sigma\rceil = (\bar{i}, l_1, Val^{c'}_d(l_2, k, z^\sigma))$,
  where $k = [x^{c'}_d]$ and $l^c_{c'}\lceil z^\alpha, z^\sigma\rceil = (\bar{i}, l_1, l_2)$.

- $(l_{\mathsf{Int}^*} \cdot l_{\mathsf{Int}})\lceil z^\alpha, z^\sigma\rceil = (\bar{i}, l_1, Ic(l_{\mathsf{Int}^*} \cdot Id(l_2)))$,
  $(l_{\mathsf{Bool}^*} \cdot l_{\mathsf{Int}})\lceil z^\alpha, z^\sigma\rceil = (\bar{i}, l_1, Bc(l_{\mathsf{Bool}^*} \cdot Id(l_2)))$,
  where $l_{\mathsf{Int}}\lceil z^\alpha, z^\sigma\rceil = (\bar{i}, l_1, l_2)$.

- $(l_{d^*} \cdot l_{\mathsf{Int}})\lceil z^\alpha, z^\sigma\rceil = (\bar{i} \circ \langle j\rangle, l_1 \wedge \mathsf{if}\ l_{d^*} \cdot Id(l_2) \doteq \mathsf{nil}\ \mathsf{then}\ j \doteq 0\ \mathsf{else}\ bij^d \cdot j \doteq l_{d^*} \cdot Id(l_2)\ \mathsf{fi}, j)$,
  where $d \in C$, $l_{\mathsf{Int}}\lceil z^\alpha, z^\sigma\rceil = (\bar{i}, l_1, l_2)$ and j is a fresh integer logical variable.

- $(l_1 + l_2)\lceil z^\alpha, z^\sigma\rceil = (\bar{i}, l_{1_1} \wedge l'_{2_1}, Ic(Id(l_{1_2}) + Id(l'_{2_2})))$
  where $l_1\lceil z^\alpha, z^\sigma\rceil = (\bar{i}_1, l_{1_1}, l_{1_2})$, $l_2\lceil z^\alpha, z^\sigma\rceil = (\bar{i}_2, l_{2_1}, l_{2_2})$, $\bar{i} = \bar{i}_1 \circ \bar{j}$, $\bar{j}$ is some sequence of new logical integer variables of the same length as $\bar{i}_2$, $l'_{2_1} = l_{2_1}[\bar{j}/\bar{i}_2]$, and $l'_{2_2} = l_{2_2}[\bar{j}/\bar{i}_2]$.

- $\mathsf{if}\ l_1\ \mathsf{then}\ l_2\ \mathsf{else}\ l_3\ \mathsf{fi}\lceil z^\alpha, z^\sigma\rceil = (\bar{i}, l_{1_1} \wedge l'_{2_1} \wedge l'_{3_1}, \mathsf{if}\ l_{1_2}\ \mathsf{then}\ l'_{2_2}\ \mathsf{else}\ l'_{3_2}\ \mathsf{fi})$
  where $l_1\lceil z^\alpha, z^\sigma\rceil = (\bar{i}_1, l_{1_1}, l_{1_2})$, $l_2\lceil z^\alpha, z^\sigma\rceil = (\bar{i}_2, l_{2_1}, l_{2_2})$, $l_3\lceil z^\alpha, z^\sigma\rceil = (\bar{i}_3, l_{3_1}, l_{3_2})$, $\bar{i} = \bar{i}_1 \circ \bar{j}_2 \circ \bar{j}_3$, $\bar{j}_2$ and $\bar{j}_3$ are sequences of new logical variables of the same length as $\bar{i}_2$, $\bar{i}_3$, respectively, such that $\bar{i}_1$, $\bar{j}_2$ and $\bar{j}_3$ are mutually disjoint, $l'_{2_1} = l_{2_1}[\bar{j}_2/\bar{i}_2]$, $l'_{2_2} = l_{2_2}[\bar{j}_2/\bar{i}_2]$, $l'_{3_1} = l_{3_1}[\bar{j}_3/\bar{i}_3]$, and $l'_{3_2} = l_{3_2}[\bar{j}_3/\bar{i}_3]$.

- $(l_1 \doteq l_2)\lceil z^\alpha, z^\sigma\rceil = (\bar{i}, l_{1_1} \wedge l'_{2_1}, l_{1_2} \doteq l'_{2_2})$
  where $l_1\lceil z^\alpha, z^\sigma\rceil = (\bar{i}_1, l_{1_1}, l_{1_2})$, $l_2\lceil z^\alpha, z^\sigma\rceil = (\bar{i}_2, l_{2_1}, l_{2_2})$, $\bar{i} = \bar{i}_1 \circ \bar{j}$, $\bar{j}$ is some sequence of new logical integer variables of the same length as $\bar{i}_2$, $l'_{2_1} = l_{1_2}[\bar{j}/\bar{i}_2]$, and $l'_{2_2} = l_{2_2}[\bar{j}/\bar{i}_2]$.

Next we define for every assertion P^c its transformation $P^c\lfloor z^\alpha, z^\sigma\rfloor$.

- $l^c_{\mathsf{Bool}}\lfloor z^\alpha, z^\sigma\rfloor = \exists\bar{i}(l_1 \wedge l_2 \doteq Bc(\mathsf{true}))$,
  where $l^c_{\mathsf{Bool}}\lceil z^\alpha, z^\sigma\rceil = (\bar{i}, l_1, l_2)$.

- $(P_1 \wedge P_2)\lfloor z^\alpha, z^\sigma\rfloor = P_1\lfloor z^\alpha, z^\sigma\rfloor \wedge P_2\lfloor z^\alpha, z^\sigma\rfloor$.

- $(\exists z_a P)\lfloor z^\alpha, z^\sigma\rfloor = \exists z_a P\lfloor z^\alpha, z^\sigma\rfloor$
  for $a = \mathsf{Int}, \mathsf{Bool}, \mathsf{Int}^*, \mathsf{Bool}^*$.

- $(\exists z_d P)\lfloor z^\alpha, z^\sigma\rfloor = \exists z_d(z_d \in bij^d \wedge P\lfloor z^\alpha, z^\sigma\rfloor)$
  for every $d \in C$. Here $z_d \in bij^d$ abbreviates $\exists i\ z_d \doteq bij^d \cdot i$ (cf. definition 6.28).

- $(\exists z_{d^*} P)\lfloor z^\alpha, z^\sigma\rfloor = \exists z_{d^*}(z_{d^*} \subseteq bij^d \wedge P\lfloor z^\alpha, z^\sigma\rfloor)$
  for every $d \in C$. Here $z_{d^*} \subseteq bij^d$ abbreviates $\forall i\ z_{d^*} \cdot i \in bij^d$ (cf. definition 6.28).

- $(\forall z_a P)\lfloor z^\alpha, z^\sigma \rfloor = \forall z_a P\lfloor z^\alpha, z^\sigma \rfloor$
  for $a = \mathsf{Int}, \mathsf{Bool}, \mathsf{Int}^*, \mathsf{Bool}^*$.
- $(\forall z_d P)\lfloor z^\alpha, z^\sigma \rfloor = \forall z_d (z_d \in bij^d \rightarrow P\lfloor z^\alpha, z^\sigma \rfloor)$
  for every $d \in C$.
- $(\forall z_{d^*} P)\lfloor z^\alpha, z^\sigma \rfloor = \forall z_{d^*} (z_{d^*} \subseteq bij^d \rightarrow P\lfloor z^\alpha, z^\sigma \rfloor)$
  for every $d \in C$.

In this transformation we assume that the quantified variables are distinct from any of the variables of $\overline{bij}$. Note that the result of this transformation applied to an arbitrary assertion is a quantification-restricted assertion.

To describe the semantics of this transformation we need the following definition.

Definition B.3
Let $\omega \in \Omega, \sigma \in \Sigma$, and let f be an *osi* (see definition 6.17). Then we write $Code(\omega, \sigma, f)$ iff for every $c \in C$ we have

- $\sigma_\perp^{(c)} = \{elt(\beta^{c^*}, n) : n \in \mathbf{N}\}$
- for all $\alpha \in \sigma^{(c)}$ and for all $n \in \mathbf{N}$ we have

$$elt(\beta^{c^*}, n) = \alpha \qquad \text{iff} \qquad f^c(alpha) = n$$

where $\beta^{c^*} = \omega(bij^c)$.

We write $Code_L(\omega, \sigma, f)$ if $Code(\omega, \sigma, f)$ and additionally for every $c \in C$ we have

- $\omega(z) \in \omega(bij^c)$ for every $z \in L \cap LVar_c$
- $\omega(z) \subseteq \omega(bij^c)$ for every $z \in L \cap LVar_{c^*}$

In a sense $Code(\omega, \sigma, f)$ can be interpreted as saying that $\omega(bij^c)$ codes the restriction of the *osi* f to the existing objects of σ.

Now we are ready for the following semantical interpretation of the transformation described above.

Theorem B.4
Assume to be given the states $\sigma, \sigma', \sigma''$ such that $\sigma \preceq \sigma''$ and an *osi* f such that $f(\sigma^{(c)}) = \sigma'^{(c)}$ for every $c \in C$. Furthermore let $\omega \in \Omega$ and $\delta \in \Delta^c$ be such that

$OK(\omega,\delta,\sigma'')$ and $Code_L(\omega,\sigma,f)$, where $BVar \subseteq L \subseteq LVar$. Then for every assertion P^c such that $LVar(P^c) \subseteq L$ and $LVar(P^c) \cap \overline{bij} = \emptyset$ we have

$$\sigma',\delta',\omega' \models P^c \qquad \text{iff} \qquad \sigma'',\delta,\omega\{n,m/z^\alpha,z^\sigma\} \models P^c\lfloor z^\alpha,z^\sigma\rfloor,$$

where $\delta' = f(\delta)$, $\omega' = f(\omega) \downarrow L$, $n = [f(\delta_{(1)})]_c$, $m = [\sigma']$, and z^α, z^σ are new logical integer variables.

Proof

Induction on the complexity of P^c. The case $P^c = l^c_{\mathsf{Bool}}$ is treated as follows. For every logical expression l^c_d such that $LVar(l^c_d) \subseteq L$ and $LVar(l^c_d) \cap \overline{bij} = \emptyset$ we prove, by induction on the complexity of l^c_d, the following: Let $l^c_d\lceil z^\alpha,z^\sigma\rceil = (\bar{i},l_1,l_2)$ where $\bar{i} = i_1,\ldots,i_q$. Then there exists a unique sequence of natural numbers $\bar{k} = k_1,\ldots,k_q$ such that

$$\mathcal{L}[\![l_1]\!](\omega\{\bar{k},n,m/\bar{i},z^\alpha,z^\sigma\})(\delta)(\sigma'') = \mathbf{t}$$

and for this $\bar{k}$ we have

$$[\mathcal{L}[\![l^c_d]\!](\omega')(\delta')(\sigma')]_d = \mathcal{L}[\![l_2]\!](\omega\{\bar{k},n,m/\bar{i},z^\alpha,z^\sigma\})(\delta)(\sigma'').$$

□

B.3 Expressing the coding relationship

In this section we show how to express in the assertion language the relationship between a state and its code number. In definition B.6 we shall define the assertion $Bij(z^\sigma)$, which expresses, as accurately as possible, that the current state is coded by the value of z^σ and that the logical variables $\overline{bij}$ form a correct decoding table. However, it is only possible to express this up to isomorfism, as we shall see in lemma B.7.

Definition B.5

First we define the following auxiliary assertions:

- $CI^c_{\mathsf{Int}}(x^c_{\mathsf{Int}},z^\alpha,z^\sigma) = Ic((bij^c\cdot z^\alpha).x^c_{\mathsf{Int}}) \doteq Val^c_{\mathsf{Int}}(z^\alpha,k,z^\sigma)$,
  $CI^c_{\mathsf{Bool}}(x^c_{\mathsf{Bool}},z^\alpha,z^\sigma) = Ic((bij^c\cdot z^\alpha).x^c_{\mathsf{Bool}}) \doteq Val^c_{\mathsf{Bool}}(z^\alpha,k,z^\sigma)$,
  where $k = [x]$.
- $CI^c_d(x^c_d,z^\alpha,z^\sigma) =$
  $((bij^c\cdot z^\alpha).x^c_d \doteq \mathsf{nil} \to Val^c_d(z^\alpha,k,z^\sigma) \doteq 0)\wedge$
  $((bij^c\cdot z^\alpha).x^c_d \not\doteq \mathsf{nil} \to \forall p((bij^c\cdot z^\alpha).x^c_d \doteq bij^d\cdot p \to Val^c_d(z^\alpha,k,z^\sigma) \doteq p))$,
  where $d \in C$ and $k = [x^c_d]$
- $CT_{\mathsf{Int}}(u_{\mathsf{Int}},z^\sigma) = Ic(u_{\mathsf{Int}}) \doteq Val_{\mathsf{Int}}(k,z^\sigma)$,
  $CT_{\mathsf{Bool}}(u_{\mathsf{Bool}},z^\sigma) = Ic(u_{\mathsf{Bool}}) \doteq Val_{\mathsf{Bool}}(k,z^\sigma)$,
  where $k = [u]$.

- $CT_d(u_d, z^\sigma) =$
  $(u_d \doteq \mathsf{nil} \rightarrow Val_d(k, z^\sigma) \doteq 0) \wedge$
  $(u_d \not\doteq \mathsf{nil} \rightarrow \forall p(bij^d \cdot p \doteq u_d \rightarrow Val_d(k, z^\sigma) = p)),$
  where $d \in C$ and $k = [u_d]$

Definition B.6
Next we define the assertion $Bij(z^\sigma)$, where z^σ is some logical integer variable, as follows.

$$\begin{array}{rcl} Bij(z^\sigma) & = & \bigwedge_c \forall z_c \exists! i(bij^c \cdot i \doteq z_c) \wedge \\ & & \bigwedge_c \forall i(E^c(i, z^\sigma) \leftrightarrow bij^c \cdot i \not\doteq \mathsf{nil}) \wedge \\ & & \bigwedge_c \forall i\Big(bij^c \cdot i \not\doteq \mathsf{nil} \rightarrow \bigwedge_d \bigwedge_{x \in IVar_d^c} CI_d^c(x, i, z^\sigma)\Big) \wedge \\ & & \bigwedge_d \bigwedge_{u \in TVar_d} CT_d(u, z^\sigma) \end{array}$$

The first conjunct states that for every c the sequence denoted by bij^c stores each existing object of class c exactly once. The second conjunct then can be interpreted as stating that every existing object of class c occurs in the sequence denoted by bij^c at a position which equals the code of *some* object that exists in the state coded by z^σ. The third conjunct relates the local state of every existing object with the one of its corresponding code. Finally, the fourth conjunct relates the values of the temporary variables with their coded versions.

In the following lemma we show how this assertion $Bij(z)$ can be used to describe the isomorphism between two states.

Lemma B.7
Let σ, ω, f such that $OK(\omega, \sigma)$, $Code(\omega, \sigma, f)$ and $\omega(z) = [\sigma']$.
Then:

$$\sigma, \delta, \omega \models Bij(z) \text{ iff } f(\sigma) = \sigma',$$

for an arbitrary δ such that $OK(\sigma, \delta, \omega)$.

Proof
Straightforward. □

B.4 Expressing the strongest postcodition

Finally we are ready for the theorem stating the expressibility of the strongest post-condition.

Theorem B.8
Let ρ^c be closed, $BVar \subseteq L \subseteq LVar$, P^c such that $LVar(P^c) \subseteq L$ and $\overline{bij} \cap L = \emptyset$. Then: $SP^c_L(\rho^c, P^c) = \exists bij^{c_1}, \ldots, bij^{c_n}, z_1, z_2, z_3(Q)$ (assuming $C = \{c_1, \ldots, c_n\}$), where $Q = \bigwedge_{1 \leq p \leq 5} Q_p$, and

- $Q_1 = T([\rho^c], z_1, z_2, z_3)$,
- $Q_2 = Bij(z_3)$,
- $Q_3 = bij^c \cdot A^c(z_1) \doteq \mathsf{self}$,
- $Q_4 = \bigwedge_c \forall i(B^c(i, z_1) \leftrightarrow bij^c \cdot i \in b_c)$,
- $Q_5 = \exists z_{c_1^*}, \ldots, z_{c_n^*} \bigwedge_{1 \leq p \leq 4} R_p$,
  where
  $R_1 = \bigwedge_c (z_{c^*} \sqsubseteq bij^c)$
  $R_2 = \bigwedge_c \forall i(E^c(i, z_2) \leftrightarrow z_{c^*} \cdot i \not\doteq \mathsf{nil})$
  $R_3 = \bigwedge_c (\bigwedge_{z'_c \in L}(z'_c \in z_{c^*}) \wedge \bigwedge_{z'_{c^*} \in L}(z'_{c^*} \subseteq z_{c^*}))$
  $R_4 = P^c\lfloor z, z' \rfloor[\bar{z}/\overline{bij}, A^c(z_1)/z, z_2/z']$

where $l_{1a} \sqsubseteq l_{2a}$, for $a = d^*$, abbreviates the assertion $\forall i(l_{1a} \cdot i \doteq \mathsf{nil} \vee l_{1a} \cdot i \doteq l_{2a} \cdot i)$, and $\bar{z}$ denotes a sequence $z_{c_1^*}, \ldots, z_{c_n^*}$ of fresh logical variables.

The quantification $\exists bij^{c_1}, \ldots, bij^{c_n}$ will correspond to the phrase (in theorem 6.21) "there exists an osi f". The variables z_1, z_2, z_3 will correspond to δ', σ_0, and $f(\sigma)$, respectively. The conjunction $\bigwedge_{1 \leq i \leq 4} Q_i$ then expresses $f(\sigma) = \mathcal{P}[\![\rho]\!](\gamma)(\delta')(\sigma_0)$. Finally, the assertion Q_5 expresses $\sigma_0, \delta', \omega' \models P$, where $\omega' = f(\omega) \downarrow L$. Let us look into this more closely. The conjunction $R_1 \wedge R_2$ states that the variable z_{c_i}, $1 \leq i \leq n$, stores all the existing objects of σ_0 (of class c_i) at a position which equals its code. The assertion R_3 then states that ω' is compatible with σ_0. Finally, the assertion R_4 expresses that $\sigma_0, \delta', \omega' \models P$.

Proof
Let $\sigma, \delta, \omega \models SP^c_L(\rho^c, P^c)$. So there exists for $i = 1, \ldots, n$, $\alpha_i \in \mathbf{O}^{c_i^*}$, and $\beta_1, \beta_2, \beta_3 \in \mathbf{N}$ such that $\sigma, \delta, \omega' \models Q$, where $\omega' = \omega\{\alpha_i/bij^{c_i}\}_i\{\beta_1, \beta_2, \beta_3/z_1, z_2, z_3\}$.

As $\sigma, \delta, \omega' \models Q_1$ there exists $\sigma_0, \sigma_1, \delta'$ such that $\sigma_1 = \mathcal{P}[\![\rho^c]\!](\gamma)(\delta')(\sigma_0)$, γ arbitrary, and $[\delta'] = \beta_1$, $[\sigma_0] = \beta_2$, $[\sigma_1] = \beta_3$.

Now let f be an *osi* such that for $\alpha \in \sigma^{(c_i)}$ we have: $f(\alpha) = \beta$ iff $elt(\omega'(bij^{c_i}), \beta) = \alpha$. (Note that as $\sigma, \delta, \omega' \models Q_2$ we have that for $\alpha \in \sigma^{(c_i)}$ there exists some $\beta \in \mathbf{N}$ such that $elt(\alpha_i, \beta) = \alpha$, furthermore we have $E^{c_i}(\beta, \beta_3)$ so $\beta \in \sigma_1^{(c_i)}$.) So we have $Code(\omega', \sigma, f)$ and by lemma B.7 we infer $f(\sigma) = \sigma_1$.

From $\sigma, \delta, \omega' \models Q_3$ it follows that $\delta'_{(1)} = f(\delta_{(1)})$. Furthermore from $\sigma, \delta, \omega' \models Q_4$ it follows that $\delta'_{(2)(c)} = \{f^c(\alpha) : \alpha \in \omega(b_c)\}$. Note that $OK(\omega, \delta, \sigma)$ so we infer that $\delta' = f(\delta)$.

Finally, we have $\sigma, \delta, \omega' \models Q_5$. So there exists for $i = 1, \ldots, n$, $\alpha'_i \in \mathbf{O}^{c_i^*}$, $\omega'' = \omega'\{\alpha'_i / z_{c_i^*}\}_i$ such that $\sigma, \delta, \omega'' \models \bigwedge_{1 \leq j \leq 4} R_j$. Let σ' such that, for an arbitrary c, $\sigma'^{(c)} = f^{-1^c}(\sigma_0^{(c)})$. It then follows that $\sigma' \preceq \sigma$ and by $\sigma, \delta, \omega'' \models \bigwedge_{1 \leq j \leq 3} R_j$ we have $Code_L(\omega''', \sigma', f)$, where $\omega''' = \omega\{\alpha'_j / bij^{c_j}\}_j\{\delta'_{(1)}, \beta_2 / z, z'\}$. Furthermore we have $\sigma, \delta, \omega''' \models P^c\lfloor z, z' \rfloor$, so we have by theorem B.4: $\sigma_0, \delta', \bar{\omega} \models P^c$, where $\bar{\omega} = f(\omega''') \downarrow L = f(\omega) \downarrow L$. This finishes one part of the proof.

On the other hand, let $\sigma, \sigma_0, \delta, \omega, f$ such that:

- $f(\sigma) = \mathcal{P}[\![\rho^c]\!](\gamma)(\delta')(\sigma_0)$, γ arbitrary.
- $\sigma_0, \delta', \omega' \models P^c$.

where $\delta' = f(\delta)$ and $\omega' = f(\omega) \downarrow L$.

Let $\beta_1 = [\delta']$, $\beta_2 = [\sigma_0]$, $\beta_3 = [f(\sigma)]$ and $\alpha_i \in \mathbf{O}^{c_i^*}$, for $i = 1, \ldots, n$ (assuming $C = \{c_1, \ldots, c_n\}$), such that $elt(\alpha_i, m) = \alpha (\neq \perp)$ iff $\alpha \in \sigma^{(c_i)}$ and $f^{c_i}(\alpha) = m$. Furthermore let $\omega'' = \omega\{\alpha_i / bij^{c_i}\}_i\{\beta_1, \beta_2, \beta_3 / z_1, z_2, z_3\}$.

Now $f(\sigma) = \mathcal{P}[\![\rho^c]\!](\gamma)(\delta')(\sigma_0)$ so we have $\sigma, \delta, \omega'' \models Q_1$.

We have $Code(\omega'', \sigma, f)$, and $\omega''(z_3) = [f(\sigma)]$, so by lemma B.7 we have $\sigma, \delta, \omega'' \models Q_2$.

From $\delta' = f(\delta)$, $OK(\sigma, \delta, \omega)$ and $OK(\sigma_0, \delta')$ it easily follows that $\sigma, \delta, \omega'' \models Q_3 \wedge Q_4$.

Let, for $i = 1, \ldots, n$, α'_i be a subsequence of α_i, such that $\sigma_0^{(c_i)} = \{\alpha : elt(\alpha'_i, \alpha) \neq \perp\}$. Furthermore let $\omega''' = \omega''\{\alpha'_p / z_{c_p^*}\}_p$. Now from α'_i being a subsequence of α_i it immediately follows that $\sigma, \delta, \omega''' \models R_1$.

From $\sigma_0^{(c_i)} = \{\alpha \in \mathbf{O}^{c_i} : elt(\alpha'_i, \alpha) \neq \perp\}$ it in turn follows that $\sigma, \delta, \omega''' \models R_2$. Furthermore we have that σ_0 and ω' are compatible, and $\omega' = f(\omega) \downarrow L = f(\omega''') \downarrow L$, from which it follows that: $\sigma, \delta, \omega''' \models R_3$.

Finally, let σ' be such that for an arbitrary c we have $\sigma'^{(c)} = f^{-1^c}(\sigma_0^{(c)})$ and $\bar{\omega} = \omega'''\{\alpha'_i / bij^{c_i}\}_i\{\delta'_{(1)}, \beta_2 / z, z'\}$. We then have that $Code_L(\bar{\omega}, \sigma', f)$ and $\sigma' \preceq \sigma$. So from $\sigma_0, \delta', \omega' \models P^c$ and $\omega' = f(\bar{\omega}) \downarrow L$ applying theorem B.4 it follows that $\sigma, \delta, \bar{\omega} \models P^c\lfloor z, z' \rfloor$. So we infer that $\sigma, \delta, \omega''' \models R_4$.

Summerizing we conclude that: $\sigma, \delta, \omega \models SP^c_L(\rho^c, P^c)$. □

C A closure property of the semantics

In this appendix we prove a closure property of the semantics with respect to object-space isomorphisms. To get started it it turns out to be convenient to have the following definition.

Definition C.1
Let $\beta_1^{d_1}, \ldots, \beta_n^{d_n}$ be some sequence of objects. We define $OK(\beta_1^{d_1}, \ldots, \beta_n^{d_n}, \delta, \sigma)$ iff $OK(\delta, \sigma)$ and additionally $\beta_i \in \sigma^{(d_i)}$, $i = 1, \ldots, n$.

Definition C.2
For

- $\mathcal{F} \in \left(\prod_{i=1}^{n} \mathbf{O}_\perp^{d_i}\right) \to \Delta^c \to \Sigma_\perp \to \left(\Sigma_\perp \times \mathbf{O}_\perp^{d_0}\right)$, for some $c, n, d_0, \ldots, d_n$,
- $\mathcal{G} \in \Delta^c \to \Sigma_\perp \to \left(\Sigma_\perp \times \mathbf{O}_\perp^{d}\right)$, for some c, d,
- $\mathcal{H} \in \Delta^c \to \Sigma_\perp \to \Sigma_\perp$, for some c,

we define

- $Cl(\mathcal{F})$ iff for an arbitrary $\beta_0^{d_0}, \ldots, \beta_n^{d_n}, \delta, \sigma, \sigma', f$ such that $OK(\beta_1, \ldots, \beta_n, \delta, \sigma)$:
  if $\mathcal{F}(\beta_1, \ldots, \beta_n)(\delta)(\sigma) = \langle \sigma', \beta_0 \rangle$
  then there exists an *osi* g such that $f^c \downarrow \sigma^{(c)} = g^c \downarrow \sigma^{(c)}$, for an arbitrary c, and
  $\mathcal{F}(f^{d_1}(\beta_1), \ldots, f^{d_n}(\beta_n))(f(\delta))(f(\sigma)) = \langle g(\sigma'), g^{d_0}(\beta_0) \rangle$,
- $Cl(\mathcal{G})$ iff for an arbitrary $\beta, \delta, \sigma, \sigma', f$ such that $OK(\delta, \sigma)$:
  if $\mathcal{G}(\delta)(\sigma) = \langle \sigma', \beta \rangle$
  then there exists an *osi* g such that $f^c \downarrow \sigma^{(c)} = g^c \downarrow \sigma^{(c)}$, for an arbitrary c, and
  $\mathcal{G}(f(\delta))(f(\sigma)) = \langle g(\sigma'), g^{d_0}(\beta) \rangle$,
- $Cl(\mathcal{H})$ iff for an arbitrary $\delta, \sigma, \sigma', f$ such that $OK(\delta, \sigma)$:
  if $\mathcal{H}((\delta)(\sigma) = \sigma'$
  then there exists an *osi* g such that $f^c \downarrow \sigma^{(c)} = g^c \downarrow \sigma^{(c)}$, for an arbitrary c, and
  $\mathcal{H}(f(\delta))(f(\sigma)) = g(\sigma')$.

(Here $\downarrow$ denotes the restriction operator.)

Now we are ready to analyse this closure property denoted by Cl. We start with the following lemma which states that the meaning of an arbitrary expression $s \in SExp$ satisfies this property assuming it holds for the meaning assigned to an arbitrary method:

Lemma C.3
Let γ be an environment such that for an arbitrary method name m we have $Cl(\gamma(m))$. Then for every expression $s \in SExp$ we have $Cl(\mathcal{Z}[\![s]\!](\gamma))$.

Proof
The proof proceeds by induction on the complexity of s:

$s = e$: Note that we have by theorem 6.21 $\mathcal{E}[\![e]\!](\delta)(\sigma) = \mathcal{E}[\![e]\!](f(\delta))(f(\sigma))$ for an arbitrary δ, σ such that $OK(\delta, \sigma)$.

$s = \mathsf{new}_d$: Let $\mathcal{Z}[\![\mathsf{new}_d]\!](\gamma)(\delta)(\sigma) = \langle \sigma', \beta \rangle$. So we have $pick^d(\sigma^{(d)}) = \beta$. Let $\beta' = pick^{(d)}(f(\sigma)^{(d)})$ and g be an *osi* such that $f^c \downarrow \sigma^{(c)} = g^c \downarrow \sigma^{(c)}$, for an arbitrary c, and $g^d(\beta) = \beta'$. It follows that $\mathcal{Z}[\![\mathsf{new}_d]\!](\gamma)(f(\delta))(f(\sigma)) = \langle g(\sigma'), \beta' \rangle$.

$s = e_0!m(e_1, \ldots, e_n)$: Let for $i = 0, \ldots, n$ $\mathcal{E}[\![e_i]\!](\delta)(\sigma) = \beta_i$ $(OK(\delta, \sigma))$ and $\gamma(m)(\beta_1, \ldots, \beta_n)(\delta')(\sigma) = \langle \sigma', \beta \rangle$, where

$$\begin{array}{ll} \delta'_{(1)} = \beta_0 & \\ \delta'_{(2)(c')} = \delta_{(2)(c')}\{\delta_{(2)(c')} \cup \delta_{(1)}/c'\} & c' = c \\ \delta'_{(2)(c')} = \delta_{(2)(c')} & c' \neq c, \end{array}$$

assuming $s \in SExp^c_d$, for some d.
As we have $Cl(\gamma(m))$ it follows that $\gamma(m)(f(\beta_1), \ldots, f(\beta_n)(f(\delta'))(f(\sigma)) = \langle g(\sigma'), g(\beta) \rangle$, for some *osi* g such that $g^c \downarrow \sigma^{(c)} = f^c \downarrow \sigma^{(c)}$, c arbitrary. (Note that by lemma 3.21 and $OK(\delta, \sigma)$ we have $OK(\beta_1, \ldots, \beta_n, \delta', \sigma)$.) By theorem 6.21 we have $\mathcal{E}[\![e_i]\!](f(\delta))(f(\sigma)) = f(\beta_i)$. Furthermore we have

$$\begin{array}{ll} f(\delta')_{(1)} = f(\beta_0) & \\ f(\delta')_{(2)(c')} = f^{c'}(\delta_{(2)(c')})\{f^{c'}(\delta_{(2)(c')}) \cup f^{c'}(\delta_{(1)})/c'\} & c' = c \\ f(\delta')_{(2)(c')} = f^{c'}(\delta_{(2)(c')}) & c \neq c'. \end{array}$$

So we conclude $\mathcal{E}[\![s]\!](\gamma)(f(\delta))(f(\sigma)) = \langle g(\sigma'), g(\beta) \rangle$. □

Next we prove the closure property Cl for the meaning assigned to statements assuming it holds for the one assigned to expressions.

Lemma C.4
Let γ be an agreement-preserving environment such that for an arbitrary $s \in SExp$ we have $Cl(\mathcal{Z}[\![s]\!](\gamma))$. Then we have $Cl(\mathcal{S}[\![S]\!](\gamma))$ for an arbitrary $S \in Stat$.

Proof
The proof proceeds by induction on the complexity of S. We treat the following cases:

$S = x^c_d \leftarrow s^c_d$: Let $\mathcal{S}[\![S]\!](\gamma)(\delta)(\sigma) = \sigma''$ $(OK(\delta, \sigma))$ and f be some *osi*. So we have $\mathcal{Z}[\![s]\!](\gamma)(\delta)(\sigma) = \langle \sigma', \beta \rangle$ such that $\sigma'' = \sigma'\{\beta/\delta_{(1)}, x\}$. By $Cl(\mathcal{Z}[\![s]\!](\gamma))$ it then follows that there exists an *osi* g such that $g^c \downarrow \sigma^{(c)} = f^c \downarrow \sigma^{(c)}$, for an arbitrary c, and $\mathcal{Z}[\![s]\!](\gamma)(f(\delta))(f(\sigma)) = \langle g(\sigma'), g(\beta) \rangle$. Now $g(\sigma'') = g(\sigma')\{g^d(\beta)/g^c(\delta_{(1)}, x\}$, so we conclude $\mathcal{S}[\![S]\!](\gamma)(f(\delta))(f(\sigma)) = g(\sigma'')$.

$S = S_1; S_2$: Let $\mathcal{S}[\![S]\!](\gamma)(\delta)(\sigma) = \sigma'$ $(OK(\delta, \sigma))$ and f be some *osi*. So there exists a σ'' such that $\mathcal{S}[\![S_1]\!](\gamma)(\delta)(\sigma) = \sigma''$ and $\mathcal{S}[\![S_2]\!](\gamma)(\delta)(\sigma'') = \sigma'$. By the induction hypothesis we have for some *osi* g such that $g^c \downarrow \sigma^{(c)} = f^c \downarrow \sigma^{(c)}$ for an arbitrary c and $\mathcal{S}[\![S_1]\!](\gamma)(f(\delta))(f(\sigma)) = g(\sigma'')$. Another application of the induction hypothesis gives us an *osi* h such that $h^c \downarrow \sigma''^{(c)} = g^c \downarrow \sigma''^{(c)}$, for an arbitrary c, and $\mathcal{S}[\![S_2]\!](\gamma)(g(\delta))(g(\sigma'')) = h(\sigma')$. Putting these applications of the induction hypothesis together gives us $h^c \downarrow \sigma^{(c)} = f^c \downarrow \sigma^{(c)}$, for an arbitrary c, and $\mathcal{S}[\![S]\!](\gamma)(f(\delta))(f(\sigma)) = h(\sigma')$. (Note that by lemma 3.21 $\sigma \preceq \sigma''$ and, as $OK(\delta, \sigma)$, $f(\delta) = g(\delta)$.)

$S =$ while e do S_1 od: Let $\mathcal{S}[\![S]\!](\gamma)(\delta)(\sigma) = \sigma'$. So we have $\mu\Phi(\delta)(\sigma) = \sigma'$, where Φ is as defined in definition 3.14. Now it suffices to prove that for an arbitrary $\varphi \in \Delta^c \to (\Sigma_\perp \to \Sigma_\perp)$ such that $Cl(\varphi)$ we have $Cl(\Phi(\varphi))$. So assume for some φ we have $Cl(\varphi)$. Let $\Phi(\varphi)(\delta)(\sigma) = \sigma'$ $(OK(\delta, \sigma)$ and f be some *osi*. We consider the case that $\mathcal{E}[\![e]\!](\delta)(\sigma) = \mathbf{t}$. By theorem 6.21 we then have $\mathcal{E}[\![e]\!](f(\delta))(f(\sigma)) = \mathbf{t}$. Furthermore we have $\varphi(\delta, \mathcal{S}[\![S_1]\!](\gamma)(\delta)(\sigma)) = \sigma'$. Let $\mathcal{S}[\![S_1]\!](\gamma)(\delta)(\sigma) = \sigma''$, by the induction hypothesis it then follows that for some *osi* g we have $g^c \downarrow \sigma^{(c)} = f^c \downarrow \sigma^{(c)}$, for an arbitrary c, and $\mathcal{S}[\![S_1]\!](\gamma)(f(\delta))(f(\sigma)) = g(\sigma'')$. By assumption there exists also an *osi* h such that $h^c \downarrow \sigma''^{(c)} = g^c \downarrow \sigma''^{(c)}$, for an arbitrary c, and $\varphi(g(\delta), g(\sigma'')) = h(\sigma')$. Putting this together gives us $h^c(\sigma^{(c)}) = f^c(\sigma^{(c)})$ for an arbitrary c and $\Phi(\varphi)(f(\delta))(f(\sigma)) = h(\sigma')$. (Note that by lemma 3.21 $\sigma \preceq \sigma''$ and, as $OK(\delta, \sigma)$, $f(\delta) = g(\delta)$.) □

We proceed with the following lemma which states the closure property of the meaning of class definitions assuming it holds for the meaning of statements:

Lemma C.5
Let γ be an agreement-preserving environment such that $Cl(\mathcal{S}[\![S]\!](\gamma))$ for an arbitrary statement S. Then we have for every method name m defined by D $Cl(\mathcal{C}[\![D]\!](\gamma)(m))$ for an arbitrary class definition D.

Proof
Let $\gamma' = \mathcal{C}[\![D]\!](\gamma)$ and f be some *osi*. Now let the method name m be defined by D, say m is declared as $\mu^c_{d_0,\ldots,d_k}$. We have $\gamma'(m) = \mathcal{M}[\![\mu^c_{d_0,\ldots,d_k}]\!](\gamma)$. Let $\mu^c_{d_0,\ldots,d_k} = (u_1, \ldots, u_k) : S \uparrow e$. Moreover let

$$
\begin{array}{llll}
\multicolumn{4}{l}{\mathcal{M}[\![(u_1,\ldots,u_k) : S \uparrow e]\!](\gamma)(\beta_1,\ldots,\beta_k)(\delta)(\sigma) = \langle \sigma''', \beta\rangle} \\
\textbf{where } & \sigma' & = & \langle \sigma_{(1)}, \sigma_{(2)}, \sigma'_{(3)} \rangle \\
& \sigma'_{(3)}(u) & = & \beta_i \qquad\qquad \textbf{if } u = u_i \\
& & = & \perp \qquad\qquad \textbf{otherwise} \\
& \sigma'' & = & \mathcal{S}^c[\![S]\!](\gamma)(\delta)(\sigma') \\
& \beta & = & \mathcal{E}[\![e]\!](\delta)(\sigma'') \\
& \sigma''' & = & \langle \sigma''_{(1)}, \sigma''_{(2)}, \sigma_{(3)} \rangle
\end{array}
$$

Note that we assume $\sigma \neq \perp$ and $\delta_{(1)}$ not to be blocked. If one of these do hold we have $\sigma' = \perp$ from which follows that $\sigma''', \beta = \perp$. By the assumption about γ we have for some *osi* g $g^c \downarrow \sigma'^{(c)} = f^c \downarrow \sigma'^{(c)}$, for every arbitrary c, and $\mathcal{S}^c[\![S]\!](\gamma)(f(\delta))(f(\sigma')) = g(\sigma'')$. (Note that $OK(\beta_1,\ldots,\beta_k,\delta,\sigma)$ implies $OK(\delta,\sigma')$.) As $\sigma'^{(c)} = \sigma^{(c)}$ for an arbitrary c we have $g^c(\sigma^{(c)}) = f^c(\sigma^{(c)})$ for an arbitrary c. By theorem 6.21 we have $g(\beta) = \mathcal{E}[\![e]\!](g(\delta))(g(\sigma''))$. (Note that as γ is agreement-preserving we have by lemma 3.21 $\sigma \preceq \sigma''$, and so $OK(\delta,\sigma'')$.) Putting this together gives us

$$\mathcal{M}[\![(u_1,\ldots,u_k) : S \uparrow e]\!](\gamma)(f(\beta_1),\ldots,f(\beta_k))(f(\delta))(f(\sigma)) = \langle g(\sigma'''), g(\beta)\rangle.$$

□

In the next lemma we prove that the meaning of units satisfies the closure property *Cl*.

Lemma C.6
Let $U = D_1,\ldots,D_n$ be an unit such that every method occurring in it is defined by it. Then for every method name m we have $Cl(\gamma'(m))$, where $\mathcal{U}[\![U]\!](\gamma_0) = \gamma'$ and γ_0 is the "empty" environment defined by

$$\gamma_0(\bar{\beta})(\delta)(\sigma) = \langle \perp, \perp\rangle.$$

Proof
We have $\gamma' = \bigsqcup_i \gamma_i$, γ_0 being the "empty" environment and $\mathcal{C}[\![D_1]\!] \circ \cdots \circ \mathcal{C}[\![D_n]\!](\gamma_i) = \gamma_{i+1}$. We prove by induction that $Cl(\gamma_i(m))$, m arbitrary. From this it is not difficult to prove that $Cl(\gamma'(m))$.

$i = 0$: Evident.

$i = j+1$: By the induction hypothesis we have $Cl(\gamma_i(m))$. Furthermore by lemma 3.21 we know that γ_i is agreement-preserving. From this follows by applying the lemmas C.3, C.4 and C.5 that $Cl(\gamma_{i+1}(m))$. (Note that lemma C.5 can be applied only for method names defined by U, but as we have $\gamma_i(m) = \gamma_0(m)$, for $i \in \mathbf{N}$ and m not defined by U this suffices.) □

We conclude this appendix with the following theorem which states the closure property of the meaning assigned to closed programs:

Theorem C.7
For an arbitrary closed program $\rho = \langle U | c : S \rangle$, environment γ we have $Cl(\mathcal{P}[\![\rho]\!](\gamma))$.

Proof
First note that as ρ is a closed program we have $\mathcal{P}[\![\rho]\!](\gamma) = \mathcal{P}[\![\rho]\!](\gamma_0)$. We have by definition 3.18 that $\mathcal{P}[\![\rho]\!](\gamma_0) = \mathcal{S}[\![S]\!](\gamma')$, where $\gamma' = \mathcal{U}[\![U]\!](\gamma_0)$. By lemma C.6 we have $Cl(\gamma'(m))$ for every method name m. So applying the lemmas C.3 and C.4 gives us $Cl(\mathcal{S}[\![S]\!](\gamma')$. (Note that by lemma 3.21 γ' is agreement- preserving.) □

Corollary C.8
For an arbitrary closed program ρ, $\sigma, \sigma', \delta, f$ such that $\sigma' = [\![\rho]\!](\gamma)(\delta)(\sigma)$ there exists an *osi* g such that $g^c \downarrow \sigma^{(c)} = f^c \downarrow \sigma^{(c)}$ and $g(\sigma') = [\![\rho]\!](\gamma)(f(\delta))(f(\sigma))$.