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

A proof theory for a sequential version of POCL

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.
Contents

1 Introduction 4

2 The language SPOOL 6
   2.1 An informal introduction 6
   2.2 The syntax 7

3 Semantics 11
   3.1 Domain definitions 11
   3.2 The semantic functions 14
   3.3 Remarks on the semantics 21

4 The assertion language and its semantics 24
   4.1 The assertion language 24
   4.2 Semantics of assertions and correctness formulae 26

5 The proof system 31
   5.1 Simple assignments 31
   5.2 Creating new objects 36
   5.3 Sending messages 44
   5.4 Other axioms and rules 52

6 Completeness 55
   6.1 Introduction 55
   6.2 The strongest postcondition 62
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3 Freezing the initial state</td>
<td>65</td>
</tr>
<tr>
<td>6.4 Invariance</td>
<td>69</td>
</tr>
<tr>
<td>6.5 Most general correctness formulae</td>
<td>72</td>
</tr>
<tr>
<td>6.6 The context switch</td>
<td>82</td>
</tr>
<tr>
<td>7 Conclusions</td>
<td>91</td>
</tr>
<tr>
<td>References</td>
<td>93</td>
</tr>
<tr>
<td>A A generalisation of the rule (MR)</td>
<td>94</td>
</tr>
<tr>
<td>B Expressibility</td>
<td>97</td>
</tr>
<tr>
<td>B.1 Coding mappings</td>
<td>97</td>
</tr>
<tr>
<td>B.2 Arithmetizing Truth</td>
<td>98</td>
</tr>
<tr>
<td>B.3 Expressing the coding relationship</td>
<td>102</td>
</tr>
<tr>
<td>B.4 Expressing the strongest postcodition</td>
<td>103</td>
</tr>
<tr>
<td>C A closure property of the semantics</td>
<td>106</td>
</tr>
</tbody>
</table>
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 $\text{Int, Bool} \notin C$ and define the set $C^+ = C \cup \{\text{Int, Bool}\}$ with typical element $d$.

- For each $c \in C$ and $d \in C^+$ we assume a set $\text{IVar}_d^c$ 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 $\text{IVar}_d^c$ will have as a typical element $e_d^c$.

- For each $d \in C$ we assume a set $\text{TVar}_d$ of temporary variables of type $d$, with typical element $u_d$.

- We shall let the metavariable $n$ range over elements of $Z$, 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 $\text{MName}_{d_0, \ldots, d_n}^c$ of method names of class $c$ with result type $d_0$ and parameter types $d_1, \ldots, d_n$. The set $\text{MName}_{d_0, \ldots, d_n}^c$ will have $m_{d_0, \ldots, d_n}^c$ as a typical element.

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

Definition 2.2
For any $c \in C$ and $d \in C^+$ we define the set $\text{Exp}_d^c$ of expressions of type $d$ in class $c$, with typical element $e_d^c$, as follows:
\[ e^c_d ::= x^c_d \]
\[ \quad | \quad u_d \]
\[ \quad | \quad \text{nil}_d \]
\[ \quad | \quad \text{self} \quad \text{if} \ c = d \]
\[ \quad | \quad \text{true} \mid \text{false} \quad \text{if} \ d = \text{Bool} \]
\[ \quad | \quad n \quad \text{if} \ d = \text{Int} \]
\[ \quad | \quad e^1_c \equiv e^2_c \quad \text{if} \ d = \text{Bool} \]
\[ \quad | \quad e^1_{\text{int}} + e^2_{\text{int}} \quad \text{if} \ d = \text{Int} \]
\[ \quad \vdots \]
\[ \quad | \quad e^1_{\text{int}} < e^2_{\text{int}} \quad \text{if} \ d = \text{Bool} \]
\[ \quad \vdots \]

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 (\( \equiv \)) to distinguish it from the equality sign we use in the meta-language.

**Definition 2.3**

The set \( S\text{Exp}_d^c \) of expressions with possible side effect of type \( d \) in class \( c \), with typical element \( s^c_d \), is defined as follows:

\[ s^c_d ::= e^c_d \]
\[ \quad | \quad \text{new}_d \quad \text{if} \ d \in C \ (\text{i.e.,} \ d \neq \text{Int,Bool}) \]
\[ \quad | \quad e^{c_0}_d \mid m^c_{d_1 \ldots d_n}(e^{c_1}_{d_1}, \ldots, e^{c_n}_{d_n}) \quad (n \geq 0) \]

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 \( S\text{t}_c \) of statements in class \( c \), with typical element \( S^c \), is defined by:

\[ S^c ::= s^c_d \leftarrow s^c_d \]
\[ \quad | \quad u_d \leftarrow s^c_d \]
\[ \quad | \quad s^c_d \]
\[ \quad | \quad S_1^c ; S_2^c \]
\[ \quad | \quad \text{if} \ c_{\text{Bool}} \ \text{then} \ S_1^c \ [\text{else} \ S_2^c] \ \text{fi} \]
\[ \quad | \quad \text{while} \ c_{\text{Bool}} \ \text{do} \ S^c \ \text{od} \]

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 nil\textit{int}.

**Definition 2.5**

The set \( \text{MethDef}^c_{d_0, \ldots, d_n} \) 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 \( \text{ClassDef}^c_{m_1, \ldots, m_n} \) 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:

\[
D^c_{m_1, \ldots, m_n} := c : (m_1^c \leftarrow \mu^c_{d_1}, \ldots, m_n^c \leftarrow \mu^c_{d_n})
\]

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

**Definition 2.7**

The set \( \text{Unit}^c_{m_1, \ldots, m_k} \) of units with classes \( c_1, \ldots, c_n \) defining methods \( m_1, \ldots, m_k \), with typical element \( U^c_{m_1, \ldots, m_k} \), is defined by:

\[
U^c_{m_1, \ldots, m_k} := D^c_{m_1, \ldots, m_k} : D^c_{m_1, \ldots, m_k}
\]

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

**Definition 2.8**

Finally, the set \( \text{Prog}^c \) of programs in class \( c \), with typical element \( \rho^c \), is defined by:

\[
\rho^c := \langle U^c_{m_1, \ldots, m_k} | c : S^c \rangle
\]

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 ∈ C (abbreviated sometimes by c₁) with instance variables p ∈ IVar^{c₁}_{\text{Int}} and next ∈ IVar^{c₁}_{\text{Int}}· temporary variable q ∈ TVar^{c₁}_{\text{Int}}· and method name input ∈ MName^{c₁}_{\text{Int}}·
- The class name Driver ∈ C (abbreviated by c₂) with instance variables i, bound ∈ IVar^{c₂}_{\text{Int}}· and first ∈ IVar^{c₂}_{\text{Int}}·

Then this is the program:

\begin{align*}
\langle \text{Sieve} & : (\text{input} = (q) : \text{if} \ \text{next} \neq \text{nil} \\
& \quad \text{then} \ \text{next} \leftarrow \text{new}; \\
& \quad \quad \quad p \leftarrow q \\
& \quad \text{else} \quad \text{if} \ q \ \text{mod} \ p \neq 0 \\
& \quad \quad \text{then} \ \text{next} \leftarrow \text{input}(q) \\
& \quad \quad \quad \text{fi} \\
& \quad \text{fi} \\
& \quad \uparrow \text{self} \rangle, \\
\langle \text{Driver} & : () \\
& | \\
& \text{Driver} : i \leftarrow 2; \\
& \quad \text{first} \leftarrow \text{new}; \\
& \quad \text{while} \ i < \text{bound} \\
& \quad \text{do} \ \text{first} \leftarrow \text{input}(i); \\
& \quad \quad i \leftarrow i + 1 \\
& \quad \text{od}
\end{align*}

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

![Diagram of objects in the sieve program in a certain stage of the execution](image)

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 $O^c$ of object names of class $c$, with typical element $B^c$. We define $P^c$ to be the set of all finite subsets of $O^c$, with typical element $\pi^c$. Furthermore we assume a function $pick^c : P^c \rightarrow O^c$ that satisfies

$$\forall \pi^c \in P^c : pick^c(\pi^c) \notin \pi^c.$$  \hspace{1cm} (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:

$$O^{\text{Int}} = \mathbb{Z}$$
$$O^{\text{Bool}} = B = \{t, f\}$$

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

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

$$x \sqsubseteq y \iff x = \bot \lor x = y.$$ 

Note that for every set $X$, $X_{\bot}$ 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 $\Sigma$ of states, with typical element $\sigma$, as follows:

$$\Sigma = \prod_{c} P^c \times \prod_{c,d} \left( O^c \rightarrow IVar_d^c \rightarrow O^d \right) \times \prod_{d} \left( TVar_d \rightarrow O^d \right)$$

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 \( \sigma_{(1)(c)} \in P^c \). 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 \( z \in C \) and \( d \in C^+ \) are class names, \( \beta^c \in O^c \) is an object of class \( c \), and \( x^c_d \in IVar^c_d \) is an instance variable of type \( d \) in class \( c \), then \( \sigma_{(2)(c,d)}(\beta^c)(x^c_d) \in O^c_d \) is the value of the instance variable \( x^c_d \) of object \( \beta^c \). If this value is \( \bot \), 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 \( \sigma_{(3)(d)}(u_d) \in O^c_d \) is the value of the variable \( u_d \). Here again, it is possible that the value of the variable is \( \text{nil} \).

For any state \( \sigma \) we introduce by convention that \( \sigma_{(1)(\text{Int})} = \mathbb{Z} \) and \( \sigma_{(1)(\text{Bool})} = \mathbb{B} \). 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 \( \sigma \) consistent if

- \( \forall c \in C \forall \beta^c \in \sigma^{(c)} \forall c' \in C' \forall x^c_{d'} \in IVar^c_{d'} \quad \sigma_{(2)(c,c')}(\beta^c)(x^c_{d'}) \in \sigma^{(c')} \)
- \( \forall c \in C \forall u_e \in TVar_c \quad \sigma_{(3)(c)}(u_e) \in \sigma^{(c)} \)
- \( \forall c \in C \forall \beta^c \in O^c \setminus \sigma^{(c)} \forall d \in C^+ \forall x^c_d \in IVar^c_d \quad \sigma_{(2)(c,d)}(\beta^c)(x^c_d) = \bot \)

(Note that it would not make sense for either \( c \) or \( c' \) to be \( \text{Int} \) or \( \text{Bool} \).) We shall occasionally use the shorthand \( \mathcal{O}(\sigma) \) to indicate that \( \sigma \) is consistent.

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

**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

\[
\sigma' = \sigma_{\{\beta^d_1/\beta^c_d, x^c_d\}}
\]
this means the following:

\[
\begin{align*}
\sigma'_{(1)} & = \sigma_{(1)} \\
\sigma'_{(2)(c,d)}(\beta_1(x)) & = \beta_1 \\
\sigma'_{(2)(c,d)}(\beta_1(x')) & = \sigma_{(2)(c,d)}(\beta_2(x')) \quad\text{if } x' \neq x \\
\sigma'_{(2)(c,d)}(\beta_2) & = \sigma_{(2)(c,d)}(\beta_2) \quad\text{if } \beta' \neq \beta_2 \\
\sigma'_{(2)(c',d')} & = \sigma_{(2)(c',d')} \quad\text{if } c' \neq c \text{ or } d' \neq d \\
\sigma'_{(3)} & = \sigma_{(3)}
\end{align*}
\]

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

**Definition 3.8**

The set \(\Delta^c\) of contexts of class \(c\), with typical element \(\delta^c\), is defined as follows:

\[\Delta^c = O^c \times \prod_{c'} P^{c'}\]

The meaning of a context \(\delta^c\) is as follows:

- The first component \(\delta^c_{(1)} \in O^c\) 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 \(\delta^c_{(2)(c')} \in P^{c'}\) is the set of blocked objects of class \(c'\).

**Definition 3.9**

We say that a context \(\delta^c\) agrees with a state \(\sigma\) if

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

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

**Definition 3.10**

The domain \(\Gamma\) of environments, with typical element \(\gamma\), is defined as follows:

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

**Definition 3.11**

We call an environment \( \gamma \) agreement-preserving if for every \( c, d_0, \ldots, d_n \), for every \( m \), for every \( \delta \), for every \( \sigma \in \Sigma \), and for every \( \beta = (\beta_1^{d_1}, \ldots, \beta_n^{d_n}) \in \prod_{i=1}^n \sigma^{(d_i)} \) (note that we consider only existing objects) we have that if \( OK(\sigma, \delta) \) and \( \beta^{d_0} = \gamma_{(c, \delta)}(m)(\beta)(\delta)(\sigma) \) and \( \sigma' \neq \bot \) then \( OK(\sigma'), \sigma \preceq \sigma' \), and \( \beta_0 \in \sigma^{(d_0)} \).

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 \( \delta \) agrees with the new state.

### 3.2 The semantic functions

**Definition 3.12**

The semantics of expressions is given by a function

\[
\mathcal{E}^c_d : \text{Exp}_d^c \rightarrow \Delta^c \rightarrow \Sigma_{\bot} \rightarrow O_{\bot}^d,
\]

which is defined as follows:

\[
\begin{align*}
\mathcal{E}^c_d[\mathcal{E}^c_d](\delta)(\bot) &= \bot \quad \text{(from now on we assume } \sigma \neq \bot) \\
\mathcal{E}^c_d[z^c_d](\delta)(\sigma) &= \sigma_{(z)\langle c, d \rangle}(\delta_{(1)})(z^c_d) \\
\mathcal{E}^c_d[u_d](\delta)(\sigma) &= \sigma_{(3)\langle d \rangle}(u_d) \\
\mathcal{E}^c_d[\text{nil}_d](\delta)(\sigma) &= \bot \\
\mathcal{E}^c_d[\text{self}\langle c \rangle](\delta)(\sigma) &= \delta_{(1)} \quad \text{(only if } c = d) \\
\mathcal{E}^c_d[\text{true}\langle c \rangle](\delta)(\sigma) &= t \quad \text{(only if } d = \text{Bool}) \\
\mathcal{E}^c_d[\text{false}\langle c \rangle](\delta)(\sigma) &= f \quad \text{(only if } d = \text{Bool}) \\
\mathcal{E}^c_d[n\langle c \rangle](\delta)(\sigma) &= n \quad \text{(only if } d = \text{Int}) \\
\mathcal{E}^c_d[e_1^c_d \div e_2^c_d](\delta)(\sigma) &= t \quad \text{if } \mathcal{E}^c_d[\mathcal{E}^c_d](\delta)(\sigma) = \mathcal{E}^c_d[\mathcal{E}^c_d](\delta)(\sigma) \\
&= f \quad \text{otherwise} \\
&\quad \text{(only if } d = \text{Bcol})
\end{align*}
\]
\[ E_\delta^\circ[e_1^\circ + e_2^\circ](\delta)(\sigma) = \bot \quad \text{if } E_\delta^\circ[e_1^\circ](\delta)(\sigma) = \bot \text{ or } E_\delta^\circ[e_2^\circ](\delta)(\sigma) = \bot \]
\[ = E_\delta^\circ[e_1^\circ](\delta)(\sigma) + E_\delta^\circ[e_2^\circ](\delta)(\sigma) \quad \text{otherwise} \]
(only if \( d = \text{Int} \))

\[ E_\delta^\circ[e_1^\circ < e_2^\circ](\delta)(\sigma) = \bot \quad \text{if } E_\delta^\circ[e_1^\circ](\delta)(\sigma) = \bot \]
\[ \quad \text{or } E_\delta^\circ[e_2^\circ](\delta)(\sigma) = \bot \]
\[ = t \quad \text{if } E_\delta^\circ[e_1^\circ](\delta)(\sigma) < E_\delta^\circ[e_2^\circ](\delta)(\sigma) \]
\[ = f \quad \text{otherwise} \]
(only if \( d = \text{Bool} \) and \( d' = \text{Int} \))

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

- The function \( E[\cdot](\delta) \) is strict, that is, it will always yield \( \bot \) if it is applied to a state \( \sigma \) that is equal to \( \bot \).

- The value of an instance variable is looked up in the second component of the state \( \sigma \). The first component of the context \( \delta \) indicates the currently active object.

- The value of a temporary variable is looked up in the third component of the state \( \sigma \).

- The value of the expression \( \text{nil} \) is always \( \bot \).

- The value of the expression \( \text{self} \) is the first component of the context \( \delta \).

- The Boolean constants \( \text{true} \) and \( \text{false} \) 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 \( \bot \), the result of the equality is nevertheless \( t \).

- Addition is only defined for genuine integers: If one of the two sides yields \( \bot \) the result is also \( \bot \).

- The same is true for the relation \( < \).
Definition 3.13
The semantics of expressions with possible side effect is given by the function

\[ Z^\varepsilon_d : SExp^\varepsilon_d \rightarrow \Gamma \rightarrow \Delta^\varepsilon \rightarrow \Sigma_\perp \rightarrow (\Sigma_\perp \times O^d) \, . \]

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

\[ Z^\varepsilon_d[s^\varepsilon_d](\gamma)(\delta)(\perp) = (\perp, \perp) \quad \text{(from now on we assume } \sigma \neq \perp) \]

\[ Z^\varepsilon_d[e^\varepsilon_d]^c(\gamma)(\delta)(\sigma) = (\sigma, E^\varepsilon_d[e^\varepsilon_d](\delta)(\sigma)) \]

\[ Z^\varepsilon_d[new_i](\gamma)(\delta)(\sigma) = (\sigma', \beta) \]

where \( \beta = \text{pick}^d(\sigma^{(d)}) \)

\[ \sigma' = \sigma\{\sigma^{(d)} \cup \{\beta\} / d\}\{\perp / \beta, x^d\}_{d \in C'} \in \varset^d \]

note that \( d \in C' \), i.e., \( d \neq \text{Int, Bool} \)

\[ Z^\varepsilon_d[e^\varepsilon_d]! m_{d_1, \ldots, d_n}(e_{d_1}, \ldots, e_{d_n})(\gamma)(\delta)(\sigma) = (\sigma', \beta^{(d)}) \]

where \( \beta_0 = E^\varepsilon_d[e^\varepsilon_d](\delta)(\sigma) \)

\[ \beta^{(d)} = E^\varepsilon_d[e^\varepsilon_d](\delta)(\sigma) \quad \{i = 1, \ldots, n\} \]

\[ (\sigma', \beta^{(d)}) = (\perp, \perp) \quad \text{if } \beta_0 = \perp \]

(from now on we assume \( \beta_0 \neq \perp \))

\[ \delta^{(c')} = \left\langle \beta_0^{(c')}, \delta(1) \{\delta(1) \cup \{\delta(1)\} / c\} \right\rangle \]

\[ (\sigma', \beta^{(d)}) = \gamma(c', d, d_1, \ldots, d_n)(m)(\beta_1, \ldots, \beta_n)(\delta')(\sigma) \]

Some explanation is appropriate here.

- Again, for any \( s, \gamma, \) and \( \delta, Z[s]^\varepsilon_d(\gamma)(\delta) \) is a strict function: if the starting state \( \sigma \) is \( \perp \) it delivers \( (\perp, \perp) \).

- If an expression \( e^\varepsilon_d \) occurs as a side effect expression, its result is computed using the function \( E^\varepsilon_d \) and the state is unchanged.

- The resulting object \( \beta \) of a new-expression is obtained by applying the function \( \text{pick}^d \) to the se: \( \sigma^{(d)} \) 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 \( \sigma' \) reflects the situation after the creation of this object. In its first component \( \sigma'_{(1)} \) the object \( \beta \) 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 $\sigma$ is consistent.

- In order to evaluate a send-expression, first the destination object $\beta_0$ and the parameters $\beta_1, \ldots, \beta_n$ are computed (in the old state). Note that if the destination is $\bot$ (i.e., nil), then the program will fail, which is represented by setting $\langle \sigma', \beta_0\rangle$ to $\langle \bot, \bot \rangle$. 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 $\gamma$ 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.

\textbf{Definition 3.14}

The semantics of statements is given by a function

$$S'^c : \text{Stat}^c \rightarrow \Delta^c \rightarrow \Sigma_\bot \rightarrow \Sigma_\bot,$$

which is defined as follows:

$$S'^c[S][\gamma](\delta)(\bot) = \bot \quad \text{(from now on we assume } \sigma \not= \bot)$$

$$S'^c[u_0 \leftarrow s_0][\gamma](\delta)(\sigma) = \sigma''$$

where $\langle \sigma', \beta \rangle = Z_d[s_0][\gamma](\delta)(\sigma)$

$$\sigma'' = \sigma' / \beta_{(1)}, \varepsilon$$

$$S'^c[u_d \leftarrow s_d][\gamma](\delta)(\sigma) = \sigma''$$

where $\langle \sigma', \beta \rangle = Z_d['s_d'][\gamma](\delta)(\sigma)$

$$\sigma'' = \sigma' / \beta / u$$

$$S'^c[s_0][\gamma](\delta)(\sigma) = (Z_d[s_0][\gamma](\delta)(\sigma))_{(1)}$$

$$S'^c[S_1; S_2][\gamma](\delta)(\sigma) = S'^c[S_2][\gamma](\delta)(S'^c[S_1][\gamma](\delta)(\sigma))$$

$$S'^c[\text{if } e \text{ then } S_1 \text{ else } S_2][\gamma](\delta)(\sigma) = \bot \quad \text{if } \beta = \bot$$

$$= S'[S_1][\gamma](\delta)(\sigma) \quad \text{if } \beta = t$$

$$= S'[S_2][\gamma](\delta)(\sigma) \quad \text{if } \beta = f$$

where $\beta = \mathcal{E}[e][\gamma](\delta)(\sigma)$
\[ S^c[\text{while } e \text{ do } S \text{ od}](\gamma) = \mu \Phi \]

where \( \Phi : (\Delta^c \rightarrow (\Sigma_\bot \rightarrow \Sigma_\bot)) \rightarrow (\Delta^c \rightarrow (\Sigma_\bot \rightarrow \Sigma_\bot)) \) is defined as follows:

\[
\Phi(\varphi)(\delta)(\sigma) = \begin{cases} 
\bot & \text{if } \beta = \bot \\
\varphi(\delta)(S^c[S](\gamma)(\delta)(\sigma)) & \text{if } \beta = t \\
\sigma & \text{if } \beta = f
\end{cases}
\]

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

Here is some informal explanation:

- For any \( S, \gamma, \) and \( \delta, S[S](\gamma)(\delta) \) is a strict function from \( \Sigma_\bot \) to \( \Sigma_\bot \).

- If an assignment to an instance variable \( x \) is done, first the right hand side is evaluated, resulting in a new state \( \sigma' \) (because of possible side effects), and an object \( \beta \). Now the final state \( \sigma'' \) is constructed from \( \sigma' \) by modifying its second component in such a way that the object \( \beta \) 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 \( \sigma'' \).

- 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 \( \Phi \). This operator takes its argument \( \varphi \) 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

\[
\mathcal{M}^d_{\delta_0, \ldots, \delta_n} : \text{MethDef}^d_{\delta_0, \ldots, \delta_n} \rightarrow \Gamma \rightarrow \left( \prod_{i=1}^{n} \text{O}^d_i \right) \rightarrow \Delta^c \rightarrow \Sigma_\bot \rightarrow (\Sigma_\bot \times \text{O}^d)
\]

which is defined by:
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 $\delta_{1(1)} \in \delta_{2(2)}(\gamma)$ or whether the starting state $\sigma$ is $\bot$. If this is the case the result of the method will be the pair $(\bot, \bot)$ (this will come out automatically if we set $\sigma'$ to $\bot$). Next we construct a state $\sigma'$ by initializing all temporary variables to $\bot$, 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 $\sigma'$ we execute the statement $S^c$ of the method, which results in a new state $\sigma''$. In this state we can evaluate the result expression $e^c_{do}$, which gives us the object $\beta^{ab}$. The state $\sigma'''$ 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

$$C^c_{m_1, \ldots, m_n} : ClassDef^c_{m_1, \ldots, m_n} \rightarrow \Gamma \rightarrow \Gamma$$

which is defined as follows:

$$C^c_{m_1, \ldots, m_n}[c : \langle m_{1d_1}^c = \mu_{1d_1}^c, \ldots, m_{nd_n}^c = \mu_{nd_n}^c \rangle](\gamma) = \gamma \{M[\mu_{1d_1}^c](\gamma) / m_{1d_1}^c \} \ldots \{M[\mu_{nd_n}^c](\gamma) / m_{nd_n}^c \}$$

This means that in the environment $\gamma$ 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 $\gamma$. 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

$$U^{c_1, \ldots, c_n}_{m_1, \ldots, m_k} : Unit^{c_1, \ldots, c_n}_{m_1, \ldots, m_k} \rightarrow \Gamma \rightarrow \Gamma$$
which is defined by:

\[ U_{m_1, \ldots, m_n}[D_1^{c_1}, \ldots, D_n^{c_n}](\gamma) = \gamma' \]

where \( \gamma' = \gamma(\zeta_j/m_j)_{j=1}^{k} \)

\( (\zeta_1, \ldots, \zeta_k) = \mu \Psi \)

\( \Psi(\zeta'_1, \ldots, \zeta'_k) = (\gamma''(c_i, \tilde{d}_j)(m_j))_{j=1}^{k} \)

\( \gamma'' = C[D_1] \circ \ldots \circ C[D_n](\gamma(\zeta'_j/m_j)_{j=1}^{k}) \)

(we suppose that \( m_j = m_j^{c_j} \)).

The main point in this definition is the construction of an environment \( \gamma' \) from the least fixed point of the operator \( \Psi \). This operator \( \Psi \) 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 \( \gamma'' \) 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 \( \Psi \). The least fixed point of \( \Psi \) therefore consists of the meanings of the methods defined in the unit, where for the other methods the meanings recorded in \( \gamma \) are assumed.

Because we require that all the class names (the \( c_i \)) are different, each \( C[D_i^{c_i}] \) 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 \( C[D_1] \circ \ldots \circ C[D_n] \) because we want to preserve the meanings of methods not defined in \( D_1, \ldots, D_n \). This is important in the soundness of the recursion rule.

**Definition 3.18**

Finally we give the semantics of programs by defining a function

\[ P^c : \text{Prog}^c \to \Gamma \to \Delta^c \to \Sigma_\bot \to \Sigma_\bot \]

as follows:

\[ P[(U_{m_1, \ldots, m_n}[c : S^c])(\gamma)] = S^c[S](\gamma') \]

where \( \gamma' = U[U](\gamma) \)

If every method used in the program is defined in the unit then the meaning is independent of the environment \( \gamma \). One could take the "empty" environment \( \gamma_0 \), defined by

\[ \gamma_0(c, \tilde{d})(m_j^{c_j})(\tilde{d}_j)(\delta^c)(\sigma) = (\bot, \bot) \]

(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 $\Phi$, 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 $e_\delta^c$ and for every context $\delta^c$, the function $\mathcal{E}[e][\delta]$ is strict, i.e., that $\mathcal{E}[e][\delta](\bot) = \bot$.
- For every statement $S^c$, for every environment $\gamma$, and for every context $\delta^c$, the function $\mathcal{S}[S][\gamma][\delta]$ is also strict, i.e., $\mathcal{S}[S][\gamma][\delta](\bot) = \bot$.

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

Lemma 3.20
The function $\Psi$, 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 $s_\delta^c$, $\mathcal{E}[s]$ is a continuous function from $\Gamma$ to $\Delta^c \rightarrow \Sigma_\bot \rightarrow (\Sigma_\bot \times \mathcal{O}_\bot^4)$.
- For any statement $S^c$, $\mathcal{S}[S]$ is a continuous function from $\Gamma$ to $\Delta^c \rightarrow \Sigma_\bot \rightarrow \Sigma_\bot$.
- For any method definition $\mu_\delta^{\nu_0, \ldots, \nu_n}$, $\mathcal{M}[\mu]$ is a continuous function from $\Gamma$ to $\left(\prod_{i=1}^n \mathcal{O}_\bot^4\right) \rightarrow \Delta^c \rightarrow \Sigma_\bot \rightarrow (\Sigma_\bot \times \mathcal{O}^4_\bot)$.
- For any class definition $D_m^c$, $\mathcal{C}[D]$ is a continuous function from $\Gamma$ to $\Gamma$.
- Now we can prove that $\Psi$ is a continuous function.
In retrospect we can change the domain assignments of several entities as follows (where $\rightarrow_c$ stands for continuous functions and $\rightarrow_s$ for strict functions):

$$
\Gamma = \prod_{c_1, \ldots, c_n} (MName_c \rightarrow \left( \prod_{i=1}^{n} O_i \right) \rightarrow \Delta_c \rightarrow \Sigma_\perp \rightarrow_s (\Sigma_\perp \times O_i))
$$

$$
E_d^c : Exp_d \rightarrow \Delta_c \rightarrow \Sigma_\perp \rightarrow_s O_i
$$

$$
Z_d^c : SExp_d \rightarrow \Gamma \rightarrow \Delta_c \rightarrow \Sigma_\perp \rightarrow_s (\Sigma_\perp \times O_i)
$$

$$
S^c : Stat^c \rightarrow \Gamma \rightarrow \Delta_c \rightarrow \Sigma_\perp \rightarrow_s \Sigma_\perp
$$

$$
M_d^c : MethDef \rightarrow \Gamma \rightarrow \left( \prod_{i=1}^{n} O_i \right) \rightarrow \Delta_c \rightarrow \Sigma_\perp \rightarrow_s (\Sigma_\perp \times O_i)
$$

$$
C_{m_1, \ldots, m_n}^c : ClassDef \rightarrow \Gamma \rightarrow \Gamma
$$

$$
U_{m_1, \ldots, m_n}^c : UnitDef \rightarrow \Gamma \rightarrow \Gamma
$$

$$
P^c : Prog^c \rightarrow \Gamma \rightarrow \Delta_c \rightarrow \Sigma_\perp \rightarrow_s \Sigma_\perp
$$

**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 $e_d^c$, for any state $\sigma \in \Sigma$, and for any context $\delta \in \Delta_c$ such that $OK(\sigma, \delta)$, we have that $E[\sigma][\delta](\sigma) \in \sigma_\perp^{(d)}$.

- For any side effect expression $s_d^c$, for any agreement-preserving environment $\gamma$, for any state $\sigma \in \Sigma$, and for any context $\delta \in \Delta_c$ such that $OK(\sigma, \delta)$, we have that if $\langle \sigma', \beta \rangle = Z[s][\gamma](\delta)(\sigma)$ and $\sigma' \neq \perp$ then $OK(\sigma')$, $\sigma \leq \sigma'$ (therefore also $OK(\sigma', \delta)$), and $\beta \in \sigma_\perp^{(d)}$.

- For any statement $S^c$, for any agreement-preserving environment $\gamma$, for any state $\sigma \in \Sigma$, and for any context $\delta \in \Delta_c$ such that $OK(\sigma, \delta)$, if $\sigma' = S[S][\gamma](\delta)(\sigma)$ and $\sigma' \neq \perp$ then $OK(\sigma')$ and $\sigma \leq \sigma'$ (therefore also $OK(\sigma', \delta)$).

- For any method definition $\mu_d^c$, for any agreement-preserving environment $\gamma$, 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 $\beta = \langle \beta_1^{d_1}, \ldots, \beta_n^{d_n} \rangle \in \prod_{i=1}^{n} \sigma_\perp^{(d_i)}$ we have if $\langle \sigma', \beta^{d_0} \rangle = \mathcal{M}[\mu][\gamma](\beta)(\delta)(\sigma)$ and $\sigma' \neq \perp$ then $OK(\sigma')$, $\sigma \leq \sigma'$ (so also $OK(\sigma', \delta)$), and $\beta^{d_0} \in \sigma_\perp^{(d_0)}$.

- For any class definition $D$ and for any agreement-preserving environment $\gamma$ we have that $C[D][\gamma]$ is again an agreement-preserving environment.
• For any unit \( U \) and for any agreement-preserving environment \( \gamma \) we have that 
\[ U[U](\gamma) \] is again an agreement-preserving environment.

• For any program \( \rho^c \), for any agreement-preserving environment \( \gamma \), for any state \( \sigma \in \Sigma \), and for any context \( \delta^c \) such that \( OK(\sigma, \delta) \), if \( \sigma' = P[\rho](\gamma)(\delta)(\sigma) \) 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 (S1)), 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 \( \bot \). This does not affect essentially the expressive power of the assertion language, but in most practical cases one wants to exclude \( \bot \) 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 \( \{ d^* \mid d \in C^+ \} \), and we use \( C^+ \), with typical element \( a \), for the union \( C^+ \cup C^* \).

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

We have the following functions:

- \( \text{len}^d : O^{d*} \to \mathbb{Z} \) returns the number of elements in the sequence.
- \( \text{elt}^d : O^{d*} \times \mathbb{Z} \to O^d_\perp \) 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 \( a \in C^+ \) we have a set \( L\text{Var}_a \) of logical variables of type \( a \), with typical element \( z_a \).

**Definition 4.3**

We the set \( L\text{Exp}_a^c \) of logical expressions of type \( a \) in class \( c \), with typical element \( l_a^c \), as follows:

\[
l_a^c ::= \begin{align*}
e_a^c & \quad \text{if } a \in C^+ \\
z_a & \\
l_a^c \cdot x_a^c & \quad \text{if } a \in C^+ \\
\text{if } l_0^c \text{ Bool then } l_1^c \text{ else } l_2^c & \quad \text{if } a \in C^+ \\
l_a^c & \quad \text{if } a = \text{Bool} \\
l_a^c \text{Int} + l_2^c \text{Int} & \quad \text{if } a = \text{Int} \\
\vdots \\
l_a^c \text{Int} < l_2^c \text{Int} & \quad \text{if } a = \text{Bool} \\
\vdots \\
l_a^c | & \quad \text{if } a = \text{Int \ and \ } d \in C^+ \\
l_a^c \cdot l_1^c \text{Int} & \quad \text{if } a \in C^+ 
\end{align*}
\]

Note that the difference with the set \( \text{Exp}_a^c \) 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 \( .^i \) can be distinguished from the dereferencing operator \( .^1 \) by its higher vertical position on the line and by the type of its first argument.

**Definition 4.4**
The set \( \text{Ass}^c \) of assertions in class \( c \), with typical elements \( P^c \) and \( Q^c \), is defined by:

\[
P^c ::= \begin{array}{l}
\text{true}_\text{Bool} \\
P^c \rightarrow Q^c \\
\neg P^c \\
\forall z_a P^c \\
\exists z_a P^c
\end{array}
\]

This definition is rather conventional.

**Definition 4.5**
Of course, we shall freely use the logical connectives \( \land, \lor, \text{and} \rightarrow \), which we consider as abbreviations of appropriate constructions with \( \rightarrow \) and \( \neg \). Furthermore we shall use \( I^c_q^r \) as an abbreviation for \( I^c_q^r = \text{nil}_q \) and \( I^c_q^r \) for \( \neg I^c_q^r = \text{nil}_q \).

**Definition 4.6**
Finally we define the set \( \text{Corr}P^c \) of correctness formulae in class \( c \), with typical element \( F^c \), as follows:

\[
F^c ::= P^c \\
\{ P^c \} \rho^c(Q^c)
\]

**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 \( \Omega \) of valuations, with typical element \( \omega \), as follows:

\[
\Omega = \prod_a (L\text{Var}_a \rightarrow O^a_\perp).
\]

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

**Definition 4.8**
We call a valuation \( \omega \) compatible with a state \( \sigma \) if

- \( \forall c \in C \forall z_c \in L\text{Var}_c \quad \omega(z_c) \in \sigma^{(c)}_\perp \)
Again an abbreviation is useful: we shall write $OK(\sigma, \delta, \omega)$ meaning that $\sigma$ is consistent, $\delta$ agrees with $\sigma$, and $\omega$ is compatible with $\sigma$.

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 $\gamma$, for any state $\sigma \in \Sigma$, for any context $\delta^c$ and for any valuation $\omega$ such that $OK(\sigma, \delta, \omega)$ we have if $\sigma' = S[\delta](\gamma)(\delta)(\sigma)$ and $\sigma' \neq \bot$ then $OK(\sigma', \delta, \omega)$.

- For any program $\rho^c$, for any agreement-preserving environment $\gamma$, for any state $\sigma \in \Sigma$, for any context $\delta^c$ and for any valuation $\omega$ such that $OK(\sigma, \delta, \omega)$ we have if $\sigma' = P[\rho](\gamma)(\delta)(\sigma)$ and $\sigma' \neq \bot$ 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

$$L^c_a : LExpr^c_a \to \Omega \to \Delta^c \to \Sigma \to O^a_\bot$$

as follows:

$$L^c_a[e^c](\omega)(\delta)(\sigma) = E^c_a[e][\delta](\sigma)$$

$$L^c_a[z^c](\omega)(\delta)(\sigma) = \omega(a)(z)$$

$$L^c_a[l^c, x^c_a][\omega](\delta)(\sigma) = \bot \quad \text{if } \beta = \bot$$

$$= \sigma(z)[\omega, \delta, \beta](x^c_a) \quad \text{otherwise}$$

where $\beta^c = L^c_a[l][\omega](\delta)(\sigma)$

$$L^c_a[\text{if } l^c_0 \text{ then } l^c_1 \text{ else } l^c_2][\omega](\delta)(\sigma) = \bot \quad \text{if } \beta = \bot$$

$$= L^c_a[l_1][\omega](\delta)(\sigma) \quad \text{if } \beta = \text{t}$$

$$= L^c_a[l_2][\omega](\delta)(\sigma) \quad \text{if } \beta = \text{f}$$

where $\beta = L^c_a[l^c_0][\omega](\delta)(\sigma)$

$$L^c_a[l^c_1, l^c_2][\omega](\delta)(\sigma) = \text{t} \quad \text{if } L^c_a[l_1][\omega](\delta)(\sigma) = L^c_a[l_2][\omega](\delta)(\sigma)$$

$$= \text{f} \quad \text{otherwise}$$

(only if $d = \text{Bool}$)
\[ \mathcal{L}_d^c[l_1^{d_1} + l_2^{d_2}](\omega)(\delta)(\sigma) = \bot \quad \text{if} \quad \mathcal{L}_d^c[l_1](\omega)(\delta)(\sigma) = \bot \]
\[
\text{or} \quad \mathcal{L}_d^c[l_2](\omega)(\delta)(\sigma) = \bot
\]
\[
= \mathcal{L}_d^c[l_1](\omega)(\delta)(\sigma) - \mathcal{L}_d^c[l_2](\omega)(\delta)(\sigma)
\text{otherwise}
\]

\[\text{(only if } d = \text{ Int)}\]
\[\vdots\]
\[\mathcal{L}_d^c[l_1^{d_1} < l_2^{d_2}](\omega)(\delta)(\sigma) = \bot \quad \text{if} \quad \mathcal{L}_d^c[l_1](\omega)(\delta)(\sigma) = \bot \]
\[
\text{or} \quad \mathcal{L}_d^c[l_2](\omega)(\delta)(\sigma) = \bot
\]
\[
= \top \quad \text{if} \quad \mathcal{L}_d^c[l_1](\omega)(\delta)(\sigma) < \mathcal{L}_d^c[l_2](\omega)(\delta)(\sigma)
\]
\[
= \bot \quad \text{otherwise}
\]
\[\text{(only if } d = \text{ Bocl and } d' = \text{ Int)}\]
\[\mathcal{L}_{\text{int}}^c[l_1^{d_1}, l_2^{d_2}](\omega)(\delta)(\sigma) = \text{let}^{d'}(\mathcal{L}_{\text{int}}^c[l_1](\omega)(\delta)(\sigma))
\]
\[\mathcal{L}_{\text{int}}^c[l_1^{d_1} \cdot l_2^{d_2}](\omega)(\delta)(\sigma) = \text{elt}^{d'}(\mathcal{L}_{\text{int}}^c[l_1](\omega)(\delta)(\sigma), \mathcal{L}_{\text{int}}^c[l_2](\omega)(\delta)(\sigma))
\]

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 \(l \in LExpr_d^c\) we have \(L[l](\omega)(\delta)(\sigma) \in \sigma_1^{d}\), and for every expression \(l \in LExpr_d^c\) we have \(elt^{d'}(L[l](\omega)(\delta)(\sigma), n) \in \sigma_1^{d'}\) for every \(n\).

**Proof**
Induction on the complexity of \(l\). \(\Box\)

**Definition 4.12**
Now we can define the semantics of assertions in terms of the function
\[
\mathcal{A}^c : \text{Ass}^c \rightarrow \Omega \rightarrow \Delta^c \rightarrow \Sigma \rightarrow \mathbf{B}
\]
as follows:
\[\mathcal{A}^c[l_{\text{Bool}}](v)(\omega)(\delta)(\sigma) = \top \quad \text{if} \quad \mathcal{L}_d^c[l_{\text{Bool}}](\omega)(\delta)(\sigma) = \top
\]
\[= \bot \quad \text{otherwise}
\]

\[\mathcal{A}^c[\neg P^c](v)(\omega)(\delta)(\sigma) = \bot \quad \text{if} \quad \mathcal{A}^c[P^c](v)(\omega)(\delta)(\sigma) = \top
\]
\[= \top \quad \text{otherwise}
\]

\[\mathcal{A}^c[\forall z d^c P^c](v)(\omega)(\delta)(\sigma) = \top \quad \text{if for all } \beta \in \sigma^{d}\text{ we have}
\]
\[
\mathcal{A}^c[P^c](v)(\omega(\beta/z))(\delta)(\sigma) = \top
\]
\[= \bot \quad \text{otherwise}
\]
\[ A^c \exists z_d \ P^c \langle \omega \rangle(\delta)(\sigma) = t \text{ if there is a } \beta \in \sigma^{(d)} \text{ such that } \\
A^c \langle P^c \rangle \langle \omega \{ \beta/z \} \rangle(\delta)(\sigma) = t \]
\[ = f \text{ otherwise } \]
\[ A^c \forall z_d \ast \ P^c \langle \omega \rangle(\delta)(\sigma) = t \text{ if for all } \alpha \in \mathcal{O}^{d^*} \text{ such that } \]
\[ \forall n \in \mathbb{Z} \ \text{elt}(\alpha, n) \in \sigma_\perp^{(d)} \text{ we have } \\
A^c \langle P^c \rangle \langle \omega \{ \alpha/z \} \rangle(\delta)(\sigma) = t \]
\[ = f \text{ otherwise } \]
\[ A^c \exists z_d \ast \ P^c \langle \omega \rangle(\delta)(\sigma) = t \text{ if there is an } \alpha \in \mathcal{O}^{d^*} \text{ such that } \]
\[ \forall n \in \mathbb{Z} \ \text{elt}(\alpha, n) \in \sigma_\perp^{(d)} \text{ and } \\
A^c \langle P^c \rangle \langle \omega \{ \alpha/z \} \rangle(\delta)(\sigma) = t \]
\[ = f \text{ otherwise } \]

A few remarks should be made here.

- Note that the possible values of a boolean logical expression are \( t \), \( f \), and \( \bot \). If such an expression is viewed as an assertion, only \( t \) and \( f \) remain. If viewed as an expression it yields \( \bot \), as an assertion it delivers \( f \).

- 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

\[ v \xrightarrow{z} w \]

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

\[ \exists z_d^* \left( z \cdot 1 = v \land z \cdot |z| = w \land \forall n \ (0 < n \land n < |z|) \rightarrow (z \cdot n).x \equiv z \cdot (n + 1) \right) \]

Here \( n \) denotes a logical variable ranging over integers. This formula expresses that the object \( w \) can be reached from \( v \) by a "z-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:

\[
\exists z \in \Delta \mid |z| \equiv 1 \land z \cdot 1 \equiv l \land P(z)
\]

or equivalently:

\[
\forall z \in \Delta \mid (|z| \equiv 1 \land z \cdot 1 \equiv l) \rightarrow P(z).
\]

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 [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 \( \omega \), a context \( \delta^c \), and a state \( \sigma \), written as \( \omega, \delta, \sigma \models P \), if \( OK(\sigma, \delta, \omega) \) and

\[
\mathcal{A}[P](\omega)(\delta)(\sigma) = t
\]

- We call a correctness formula of the form \( P^c \) **valid**, written as \( \models P \), if it is true with respect to every \( \omega, \delta^c \), and \( \sigma \) such that \( OK(\sigma, \delta, \omega) \).

- A correctness formula of the form \( \{P^c\}p\{Q^c\} \) is called true with respect to an environment \( \gamma \), a valuation \( \omega \), a context \( \delta^c \), and a state \( \sigma \), written as \( \gamma, \omega, \delta, \sigma \models \{P^c\}p\{Q^c\} \), if \( \omega, \delta, \sigma \models P \) implies that for the state \( \sigma' = P^c[\gamma](\delta)(\sigma) \) we have

\[
\sigma' \neq \bot \Rightarrow \omega, \delta, \sigma' \models Q.
\]

- We define a correctness formula of the form \( \{P^c\}p\{Q^c\} \) to be **valid** with respect to an environment \( \gamma \), written as \( \gamma \models \{P^c\}p\{Q^c\} \), if we have \( \gamma, \omega, \delta, \sigma \models \{P^c\}p\{Q^c\} \) for every \( \omega, \delta^c \), and \( \sigma \). 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 $\gamma$ the validity of the premises of the rule with respect to $\gamma$ implies the validity of the conclusion with respect to $\gamma$. 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:

$$\{ P_G[ e_u/u_d ] \} \langle U \{ \text{c : u_d \leftarrow e_u} \} \{ P_G \} \}$$ (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:

- $x \ [e/u] = x$
- $u \ [e/u] = e$
- $u' \ [e/u] = u'$ if $u' \neq u$
- $z \ [e/u] = z$
- $l \ [e/u] = l$ if $l = \text{nil}, \text{self}, \text{true}, \text{false}$
- $n \ [e/u] = n$
- $l \cdot x \ [e/u] = (l[e/u]) \cdot x$
if \( l_0 \) then \( l_1 \) else \( l_2 \) \( \hat{\text{if}} [e/u] \) = if \( l_0[e/u] \) then \( l_1[e/u] \) else \( l_2[e/u] \) \( \hat{\text{fi}} \)

\[
\begin{align*}
(l_1 \lor l_2)[e/u] &= (l_1[e/u]) \lor (l_2[e/u]) \\
(l_1 \land l_2)[e/u] &= (l_1[e/u]) \land (l_2[e/u]) \\
& \vdots \\
(l_1 < l_2)[e/u] &= (l_1[e/u]) < (l_2[e/u]) \\
& \vdots \\
|l| \ [e/u] &= |l[e/u]| \\
(l_1 \cdot l_2)[e/u] &= (l_1[e/u] \cdot l_2[e/u])
\end{align*}
\]

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

\[
\begin{align*}
(P \rightarrow Q)[e/u] &= (P[e/u]) \rightarrow (Q[e/u]) \\
(\neg P) \ [e/u] &= \neg (P[e/u]) \\
(\forall z \ P) \ [e/u] &= \forall z \ (P[e/u]) \\
(\exists z \ P) \ [e/u] &= \exists z \ (P[e/u])
\end{align*}
\]

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 \( u \rightarrow e \). Let \( \gamma \in \Gamma, \sigma \in \Sigma \) and \( \delta \in \Delta^{e} \) be arbitrary, and let

\[
\sigma' = S[u \rightarrow e](\delta)(\sigma).
\]

Then we have the following facts:

1. For every logical expression \( l_{e}^{\rightarrow} \) and every valuation \( \omega \)

\[
L[l_{e}[u]][\omega](\delta)(\sigma) = L[l_{e}][\omega](\delta)(\sigma').
\]

2. For every assertion \( P^{e} \), every valuation \( \omega \)

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

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 $l[e/u] = e$:

$$L[e][\omega](\delta)(\sigma) = E[e][\delta](\sigma)$$
$$= \beta$$
$$= \sigma'[\omega][\delta](u)$$
$$= E[u][\delta](\sigma')$$
$$= L[u][\omega](\delta)(\sigma')$$

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 $\gamma$ we have

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

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:

$$\{P[e'_u/x'_u]\} \langle U| c : x'_u \leftarrow e'_u \rangle \{P^c\}$$  \hspace{1cm} (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:

\[
\begin{align*}
x & \quad [e/x] = e \\
x' & \quad [e/x] = x' \quad \text{if } x' \neq x \\
u & \quad [e/x] = u \\
z & \quad [e/x] = z \\
l & \quad [e/x] = l \quad \text{if } l = \text{nil, self, true, false} \\
n & \quad [e/x] = n \\
l \cdot x & \quad [e/x] = \text{if } (l[e/x]) = \text{self then } e \text{ else } (i[e/x]) \cdot x \text{ fi} \\
l \cdot x' & \quad [e/x] = (l[e/x]) \cdot x' \quad \text{if } x' \neq x \\
\text{if } l_0 \text{ then } l_1 \text{ else } l_2 & \quad [e/x] = \text{if } l_0[e/x] \text{ then } l_1[e/x] \text{ else } l_2[e/x] \text{ fi} \\
(l_1 \equiv l_2) & \quad [e/x] = (l_1[e/x]) \equiv (l_2[e/x]) \\
(l_1 + l_2) & \quad [e/x] = (l_1[e/x]) + (l_2[e/x]) \\
\vdots & \\
(l_1 < l_2) & \quad (l_1[e/x]) < (l_2[e/x]) \\
\vdots & \\
|l| & \quad [e/x] = |l[e/x]| \\
(l_1 \cdot l_2) & \quad [e/x] = (l_1[e/x] \cdot l_2[e/x])
\end{align*}
\]

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

\[
\begin{align*}
(P \rightarrow Q) & \quad [e/x] = (P[e/x]) \rightarrow (Q[e/x]) \\
(\neg P) & \quad [e/x] = \neg (P[e/x]) \\
(\forall z \ P) & \quad [e/x] = \forall z \ (P[e/x]) \\
(\exists z \ P) & \quad [e/x] = \exists z \ (P[e/x])
\end{align*}
\]

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 \(l \cdot x\). 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 \(l \cdot x\), it is possible that, after substitution, \(l\) refers to the currently active object, so that \(l \cdot x\) 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 $x'_{\alpha} \leftarrow e_{\alpha}$. Let $\gamma \in \Gamma$, $\sigma \in \Sigma$, and $\delta \in \Delta^e$ be arbitrary, and let

$$\sigma' = S[\gamma \leftarrow e](\delta)(\sigma).$$

Then we have the following facts:

1. For every logical expression $l'_{\alpha}$ and every valuation $\omega$

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

2. For every assertion $P_{\alpha}$ and every valuation $\omega$

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

**Proof**

Like in lemma 5.4 we first note that $\sigma' = \sigma[\delta/\delta_{(1)}]; x$ where $\beta = S[e](\delta)(\sigma)$. 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 $L[l'[e/x]](\omega)(\delta)(\sigma) = L[l'][\omega](\delta)(\sigma')$. Let us call this object $\beta_0$. Now if $\beta_0 = \delta_{(1)}$ then $L[l'. x](\omega)(\delta)(\sigma') = \sigma'_{(1)}[\delta_{(1)}](x) = \beta = L[e](\omega)(\delta)(\sigma)$. Otherwise we have $L[l'. x](\omega)(\delta)(\sigma') = \sigma'_{(2)}[\beta_0](x) = L[l'[e/x]](\omega)(\delta)(\sigma)$. So $L[l'[e/x]](\omega)(\delta)(\sigma') = L[l'. x](\omega)(\delta)(\sigma')$.

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 $\gamma$ we have

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

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 $P$. 

\[\Box\]
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 \( u \leftarrow \text{new} \) we have an axiom similar to the previous two:

\[
\{ P^e | \text{new} \vdash u \leftarrow \text{new} \} \quad \langle U | c : u \leftarrow \text{new} \rangle \quad \{ P^e \} \quad \text{(NT)}
\]

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:

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

if \( l_0 \) then \( l_1 \) else \( l_2 \) \( \hat{h}[e/z] = \) if \( l_0[e/z] \) then \( l_1[e/z] \) else \( l_2[e/z] \) fi

\[
\begin{align*}
(l_1 \lor l_2)[e/z] & = (l_1[e/z]) \lor (l_2[e/z]) \\
(l_1 + l_2)[e/z] & = (l_1[e/z]) + (l_2[e/z]) \\
\vdots \\
(l_1 < l_2)[e/z] & = (l_1[e/z]) < (l_2[e/z]) \\
\vdots \\
|l| & \quad [e/z] = |l[e/z]| \\
(l_1 \cdot l_2)[e/z] & = (l_1[e/z] \cdot l_2[e/z])
\end{align*}
\]

We extend this definition to assertions other than logical expressions as follows:
\[(P \rightarrow Q)[e/z] = (P[e/z]) \rightarrow (Q[e/z])\]
\[(-P)[e/z] = \neg(P[e/z])\]
\[(\forall z\ P)[e/z] = \forall z\ P\]
\[(\forall z'\ P)[e/z] = \forall z'\ (P[e/z])\quad \text{if } z' \neq z\]
\[(\exists z\ P)[e/z] = \exists z\ P\]
\[(\exists z'\ P)[e/z] = \exists z'\ (P[e/z])\quad \text{if } z' \neq z\]

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 \(\sigma \in \Sigma\), \(\delta \in \Delta^e\), \(e \in \text{Exp}_d\), and \(z \in \text{LVar}_d\) be arbitrary, and let \(\beta = \mathcal{E}[e](\delta)(\sigma)\). Then we have

1. For all \(l \in \text{LExp}_d^e\) and for all \(\omega \in \Omega\):
   \[L[l[e/z]](\omega)(\delta)(\sigma) = L[l](\omega(\beta/z))(\delta)(\sigma)\]

2. For all \(P \in \text{Ass}^e\), for all \(\omega \in \Omega\):
   \[A[P[e/z]](\omega)(\delta)(\sigma) = A[P](\omega(\beta/z))(\delta)(\sigma)\]

**Proof**
A rather trivial induction on the complexity of \(l\) and \(P\). \(\square\)

Now we can define the substitution \(\text{new}_c/u_c\). 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 \textit{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 \textit{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 \textit{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 \([\text{new}/u]\) for logical expressions:

\[ x[\text{new}/u] = x \]
\[ u[\text{new}/u] \text{ is undefined} \]
\[ u'[\text{new}/u] = u \quad \text{if } u' \neq u \]
\[ z[\text{new}/u] = z \]
\[ l[\text{new}/u] = l \quad \text{if } l = \text{nil, self, true, false} \]
\[ n[\text{new}/u] = n \]

\[ x'.x[\text{new}/u] = x'.z \]
\[ u'.x[\text{new}/u] = \text{nil} \]
\[ u'.x[\text{new}/u] = u'.z \quad \text{if } u' \neq u \]
\[ z'.x[\text{new}/u] = z'.x \]
\[ l'.x[\text{new}/u] = l'.x \quad \text{if } l = \text{nil, self} \]
\[ l'.x'.x[\text{new}/u] = (l'.x'[\text{new}/u]).x \]

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

\[ \text{if } l_0 \text{ then } l_1 \text{ else } l_2 \text{ fi[new}/u] \]
\[ = \text{if } l_0[\text{new}/u] \text{ then } l_1[\text{new}/u] \text{ else } l_2[\text{new}/u] \text{ fi} \]
\[ \text{if the substitutions of the subexpressions are all defined, otherwise undefined} \]

\[ (l_1 \triangleq l_2)[\text{new}/u] = (l_1[\text{new}/u]) \triangleq (l_2[\text{new}/u]) \]
\[ \text{if neither } l_1 \text{ nor } l_2 \text{ is } u \text{ or of the form if } \ldots \text{ fi} \]

\[ (l_1 \triangleq l_2)[\text{new}/u] = \text{false} \]
\[ \text{if either } l_1 = u \text{ and } l_2 \text{ is not } u \text{ or of the form if } \ldots \text{ fi} \]
\[ \text{or } l_2 = u \text{ and } l_1 \text{ is not } u \text{ or of the form if } \ldots \text{ fi} \]

\[ (l_1 \triangleq l_2)[\text{new}/u] = \text{true} \]
\[ \text{if } l_1 = l_2 = u \]
\[
\begin{align*}
\text{if } l_0 \text{ then } l_1 \text{ else } l_2 \text{ fi} & \equiv l_3 \text{ [new/u]} \\
& = \text{ if } l_0 \text{[new/u]} \uparrow \\
& \quad \text{ then } (l_3 \uparrow) \text{[new/u]} \\
& \quad \text{ else if } l_0 \text{[new/u]} \\
& \quad \quad \text{ then } (l_1 \equiv l_3) \text{[new/u]} \\
& \quad \quad \text{ else } (l_2 \equiv l_3) \text{[new/u]} \\
& \quad \text{ fi} \\
\text{fi} \\
(l_1 \equiv l_0 \text{ then } l_2 \text{ else } l_3 \text{ fi}) \text{[new/u]} & = \text{ if } l_0 \text{[new/u]} \uparrow \\
& \quad \text{ then } (l_1 \uparrow) \text{[new/u]} \\
& \quad \text{ else if } l_0 \text{[new/u]} \\
& \quad \quad \text{ then } (l_1 \equiv l_3) \text{[new/u]} \\
& \quad \quad \text{ else } (l_1 \equiv l_2) \text{[new/u]} \\
& \quad \text{ fi} \\
\text{fi} \\
\end{align*}
\]

if \( l_1 \) is not of the form if \ldots fi

\[
\begin{align*}
(l_1 + l_2) \text{[new/u]} & = (l_1 \text{[new/u]}) + (l_2 \text{[new/u]}) \\
\vdots \\
(l_1 < l_2) \text{[new/u]} & = (l_1 \text{[new/u]}) < (l_2 \text{[new/u]}) \\
\vdots \\
|l| \text{[new/u]} & = |l| \text{[new/u]} \\
(l_1 \cdot l_2) \text{[new/u]} & = (l_1 \text{[new/u]}) \cdot (l_2 \text{[new/u]})
\end{align*}
\]

**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:

\[
lu \ ::= \ u \\
| \text{ if } l_0 \text{ then } lu \text{ else } l_1 \text{ fi} \\
| \text{ if } l_0 \text{ then } l_1 \text{ else } lu \text{ fi}
\]
2. If $\sigma \in \Sigma$, $\delta \in \Delta^e$, $\omega \in \Omega$, and $\gamma \in \Gamma$ are such that $OK(\sigma, \delta, \omega)$, and if $\sigma' = S[u \leftarrow \text{new}]\gamma(\delta)(\sigma)$ then for every logical expression $l$ such that $l[\text{new}/u]$ is defined we have

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

**Proof**

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

$$\sigma' = \sigma\{\sigma_{(1)}(d) \cup \{b\} / d\} \{b / u\}$$

where $b = \text{pick}_d(\sigma_{(1)}(d))$, so $b \notin \sigma_{(1)}(d) \cup \{\bot\}$ (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 $\beta$. Let us deal with one representative case: $l = x'.x$. Then $l[\text{new}/u] = l = x'.x$. 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 $\beta'$ then we know $\beta' \neq \beta$ because lemma 4.11 tells us that $\beta' \in \sigma_{(1)}(d) \cup \{\bot\}$. 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 $[\text{new}/u]$ to assertions other than logical expressions as follows (we assume that the type of $u$ is $d \in C$):

$$(P \rightarrow Q)[\text{new}/u] = (P[\text{new}/u]) \rightarrow (Q[\text{new}/u])$$

$$(\neg P)[\text{new}/u] = \neg (P[\text{new}/u])$$

$$(\forall z_d \ P)[\text{new}/u] = (\forall z(P[\text{new}/u])) \land (P[u/z][\text{new}/u])$$

$$(\forall z_{d*} \ P)[\text{new}/u] = (\forall z \forall z_{\text{Bool}^*}[z] = z' \rightarrow (P[z', u/z][\text{new}/u]))$$

$$(\forall z_a \ P)[\text{new}/u] = (\forall z(P[\text{new}/u])$$

if $a \neq d, d^*$

$$(\exists z_d \ P)[\text{new}/u] = (\exists z(P[\text{new}/u])) \lor (P[u/z][\text{new}/u])$$

$$(\exists z_{d*} \ P)[\text{new}/u] = (\exists z \exists z_{\text{Bool}^*}[z] = z' \land (P[z', u/z][\text{new}/u]))$$

$$(\exists z_a \ P)[\text{new}/u] = (\exists z(P[\text{new}/u])$$

if $a \neq d, d^*$

Here we choose for $z'$ the first variable from $\text{LVar}_{\text{Bool}^*}$ 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 $\bot$ the coded sequences also yields $\bot$.

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 $z' \in LVar_{Bool^*}$. For logical expressions we define the operation $[z', u/z]$ as follows:

$$
eq [z', u/z] = e$$

$$z \quad [z', u/z] \text{ is undefined}$$

$$z'' \quad [z', u/z] = z'' \quad \text{if } z'' \neq z$$

$$l \cdot x \quad [z', u/z] = ((([z', u/z]) \cdot x$$

$$|z| \quad [z', u/z] = |z|$$

$$|l| \quad [z', u/z] = |l([z', u/z])| \quad \text{if } l \neq z$$

$$(z \cdot l_2) [z', u/z] = \text{if } z' \cdot (l_2[z', u/z]) \text{ then } u \text{ else } z \cdot (l_2[z', u/z]) \text{ fi}$$

$$(l_1 \cdot l_2)[z', u/z] = (l_1[z', u/z]) \cdot (l_2[z', u/z]) \quad \text{if } l_1 \neq z$$

if $l_0$ then $l_1$ else $l_2$ fi $[z', u/z] =$

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

$$(l_1 \cdot l_2)[z', u/z] = (l_1[z', u/z]) \cdot (l_2[z', u/z])$$

$$(l_1 \leq l_2)[z', u/z] = (l_1[z', u/z]) \leq (l_2[z', u/z])$$

$$\vdots$$

$$|l|[z', u/z] = |l([z', u/z])|$$

$$|l_1 \cdot l_2|[z', u/z] = (l_1[z', u/z] \cdot l_2[z', u/z])$$

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

$$(P \rightarrow Q)[z', u/z] = (P[z', u/z]) \rightarrow (Q[z', u/z])$$

$$(\neg P) [z', u/z] = \neg (P[z', u/z])$$

$$(\forall z P) [z', u/z] = (\forall z P)$$

$$(\forall z'' P) [z', u/z] = (\forall z'' (P[z', u/z])) \quad \text{if } z'' \neq z$$

$$(\exists z P) [z', u/z] = (\exists z P)$$

$$(\exists z'' P) [z', u/z] = (\exists z'' (P[z', u/z])) \quad \text{if } z'' \neq z$$

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 $\alpha = \omega(d')(z)$, $\alpha' = \omega(Bool^*)(z')$, $\beta = \sigma(3)(d,u)$. Suppose that $\text{len}(\alpha) = \text{len}(\alpha')$. Define $\alpha'' \in O^d$ to be the sequence that satisfies (for all $n \in \mathbb{Z}$):
\[
\begin{align*}
\text{len}(\alpha'') &= \text{len}(\alpha) \\
\text{elt}(\alpha'', n) &= \beta & \text{if } \text{elt}(\alpha', n) = t \\
\text{elt}(\alpha'', n) &= \text{elt}(\alpha, n) & \text{if } \text{elt}(\alpha', n) = f \\
\text{elt}(\alpha'', n) &= \bot & \text{if } \text{elt}(\alpha', n) = \bot
\end{align*}
\]
and take \(\omega' = \omega\{\alpha''/z\}\).

Then we have:

1. For every \(l \in L\text{Expr}_n^c\) such that \(l \neq z\):
   \[
   L_0^c[l[z', u/z]][\omega](\delta)(\sigma) = L_0^c[l][\omega'](\delta)(\sigma).
   \]
2. For every \(P \in \text{Ass}^c\) such that \(z'\) does not occur in \(P\):
   \[
   A^c[P[z', u/z]][\omega](\delta)(\sigma) = A^c[P][\omega'](\delta)(\sigma).
   \]

**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 \(d \in C, u \in T\text{Var}_d, \gamma \in \Gamma\) and define \(\sigma' = S^c[u \leftarrow \text{new}][\gamma](\delta)(\sigma)\). Then for every assertion \(P \in \text{Ass}^c\) we have
\[
A^c[P[\text{new}/u]][\omega](\delta)(\sigma) = A^c[P][\omega](\delta)(\sigma').
\]

**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 \(\beta = \sigma^{(d)}(d)(u)\), and let \(z'\) be the first variable from \(L\text{Var}_{\text{Bool}^*}\) that does not occur in \(Q\).

Now suppose that
\[
A[(\forall z_d^* Q)[\text{rew}/u]][\omega](\delta)(\sigma) = t.
\]
We shall prove that
\[
A[\forall z_d^* Q][\omega](\delta)(\sigma') = t
\]
so we have to show that for every \(\alpha'' \in O^{d^*}\) such that \(\text{elt}(\alpha'', n) \in \sigma^{(d)}(\delta)\) for all \(n \in \mathbb{Z}\), it is the case that \(A[Q][\omega(\alpha''/z)])[\delta](\sigma') = t\). If we have such an \(\alpha''\), we can define \(\alpha \in O^{d^*}\) and \(\alpha' \in O^{\text{Bool}^*}\) as follows:
\[ \text{len}(\alpha) = \text{len}(\alpha') = \text{len}(\alpha'') \]
\[ \text{elt}(\alpha, n) = \bot, \quad \text{elt}(\alpha', n) = t \quad \text{if elt}(\alpha'', n) = \beta \]
\[ \text{elt}(\alpha, n) = \text{elt}(\alpha'', n), \quad \text{elt}(\alpha', n) = f \quad \text{if} \ 1 \leq n \leq \text{len}(\alpha'') \]
\[ \text{and elt}(\alpha'', n) \neq \beta \]

Now because
\[ A[\forall z \in \text{Bool} \mid |z| = |z'| \rightarrow Q[z', u/z'][\text{new}/u][\omega](\delta)(\sigma) = t \]
and because \( \alpha \) and \( \alpha' \) have equal length and do not have elements outside \( s_\bot^{(d)} \) and \( s_\bot^{(\text{Bool})} \) respectively, we know that
\[ A[Q[z', u/z'][\text{new}/u][\omega\{\alpha/z\}\{\alpha'/z'\}](\delta)(\sigma) = t. \]

The induction hypothesis then tells us that
\[ A[Q[z', u/z][\omega\{\alpha/z\}\{\alpha'/z'\}](\delta)(\sigma') = t.\]

Finally we can apply lemma 5.17 and use the fact that \( z' \) does not occur in \( Q \), to see that
\[ A[Q]\{\omega\{\alpha''/z\}\}(\delta)(\sigma') = t. \]

To prove that \( A[\forall z \in \text{Bool} \mid Q](\omega)(\delta)(\sigma') = t \) implies \( A[\forall z \in \text{Bool} \mid Q][\text{new}/u][\omega](\delta)(\sigma) = t \)

involves reasoning in the other direction. In particular to find a suitable \( \alpha'' \) for each pair \( \alpha, \alpha' \) that satisfies certain conditions. We omit further details. \( \square \)

Again we extend this result to the case of programs:

**Corollary 5.19**

The axiom (NT) is valid, that is, for every environment \( \gamma \) we have
\[ \gamma \models \{P[\text{new}/u]\} \langle U | c \leftarrow \text{new} \rangle \{P\}. \]

\( \square \)

### 5.2.2 Assigning a new object to an instance variable

**Definition 5.20**

If our assignment is of the form \( x \leftarrow \text{new} \) we have the following axiom:
\[ \{P^c[\text{new}^c/x^c]\} \langle U | c : x^c \leftarrow \text{new}^c \rangle \{P^c\} \quad (\text{NI}) \]

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 \([\text{new}_{\omega}, x^\omega_e]\) is defined by:

\[ P[\text{new}_{\omega}, x^\omega_e] = P[\text{u}_{\omega}, x^\omega_e] | \text{new}_{\omega} / u_{\omega} \]

where \(u_{\omega}\) 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 \(\gamma \in \Gamma\), \(d \in C\), \(x \in \text{IVar}_d^\delta\), and define \(\sigma' = \mathcal{S}[x \leftarrow \text{new}](\gamma)(\delta)(\sigma)\). Then for every assertion \(P^c\) we have

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

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

Now if \(\beta = \text{pick}^d(\sigma(d))\) 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. \(\square\)

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

\[ \gamma \models \{P[\text{new}/x]\} \langle U|c : x \leftarrow \text{new} \rangle \{P\}. \]

\(\square\)

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 \(x \in \text{IVar}_d^\delta\), \(m \in \text{MName}_d^\delta, \ldots, d_n\), \(e_0 \in \text{Exp}_d^\gamma\) and \(e_i \in \text{Exp}_d^\gamma\) for \(i = 1, \ldots, n\), we have the following proof rule:

\[ \frac{P^c \land \land_{i=1}^k v_i = \text{nil} \langle U|c' : S \rangle \{Q^c[e/r]\}, \quad Q[\bar{e}/\text{self}, \bar{u}][\bar{f}/\bar{z}] \rightarrow R^c[r/x]} {P[\bar{e}/\text{self}, \bar{u}][\bar{f}/\bar{z}] \langle U|c : x \leftarrow e_0!m(e_1, \ldots, e_n) \rangle \{R\}} \quad (\text{MI}) \]
where $S \in \text{Stat}'$ and $e \in \text{Exp}_{d_0}'$ 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 $R$, $f$ 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, $[\bar{e}/\text{self}, \bar{u}]$ stands for a simultaneous substitution having the "components" $[e_0/\text{self}], [e_1/u_1], \ldots, [e_n/u_n]$ (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 $[\bar{e}/\text{self}, \bar{u}]$ 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 $[e_0/\text{self}]$. A little more precisely, an assertion $P$ as seen from the receiver's viewpoint is equivalent to $P[e_0/\text{self}]$ from the viewpoint of the sender. Note that this substitution also changes the class of the assertion: $P[e_0/\text{self}] \in \text{Ass}'$ whereas $P \in \text{Ass}'. 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 $P[\bar{e}/\text{self}, \bar{u}]$ 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 premise 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 $[f/\bar{z}]$. 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 $[e/self]$. First we do this for logical expressions:

\[
\begin{align*}
x & \quad [e/self] = e \cdot x \\
u & \quad [e/self] = u \\
z & \quad [e/self] = z \\
self & \quad [e/self] = e \\
l & \quad [e/self] = l & \text{ if } l = \text{nil, true, false, n} \\
l \cdot x & \quad [e/self] = (l[e/self]) \cdot x \\
\text{if } l_0 & \text{ then } l_1 \text{ else } l_2 & \quad [e/self] = \text{if } l_0[e/self] \text{ then } l_1[e/self] \text{ else } l_2[e/self] \text{ fi} \\
l_1 \cdot l_2 & \quad [e/self] = (l_1[e/self]) \cdot (l_2[e/self]) \\
l_1 + l_2 & \quad [e/self] = (l_1[e/self]) + (l_2[e/self]) \\
\vdots \\
l_1 < l_2 & \quad [e/self] = (l_1[e/self]) < (l_2[e/self]) \\
\vdots \\
[l] & \quad [e/self] = [l[e/self]] \\
l_1 \cdot l_2 & \quad (l_1[e/self] \cdot l_2[e/self])
\end{align*}
\]
Now we extend this to assertions other than logical expressions:

\[(P \to Q)[e/self] = (P[e/self]) \to (Q[e/self])\]

\[(\neg P)[e/self] = \neg (P[e/self])\]

\[(\forall \varepsilon P)[e/self] = \forall \varepsilon(P[e/self])\]

\[(\exists \varepsilon P)[e/self] = \exists \varepsilon(P[e/self])\]

Lemma 5.26
Let \(\sigma \in \Sigma, \delta \in \Delta^e, \varepsilon \in \text{Exp}^e\), and define \(\beta^{e'} = \mathcal{E}[\varepsilon](\delta)(\sigma)\). Let \(\delta' \in \Delta^{e'}\) be such that \(\delta'(1) = \beta\). Then we have

1. For every logical expression \(l^{e'}_0\) and every valuation \(\omega\)
   \[\mathcal{L}[l](\omega)(\delta')(\sigma) = \mathcal{L}[l[e/self]](\omega)(\delta)(\sigma)\]

2. For every assertion \(P^{e'}\) and every valuation \(\omega\)
   \[\mathcal{A}[P](\omega)(\delta')(\sigma) = \mathcal{A}[P[e/self]](\omega)(\delta)(\sigma)\]

Proof
An easy induction on the complexity of \(l\) and \(P\).

\[\square\]

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 \(\varepsilon = e_0, \ldots, e_n\) and \(\bar{u} = u_1, \ldots, u_n\). Then we define:

\[x\ [\varepsilon/self, \bar{u}] = e_0 \cdot x\]

\[u_i\ [\varepsilon/self, \bar{u}] = e_i\quad \text{for } i = 1, \ldots, n\]

\[u\ [\varepsilon/self, \bar{u}] = u\quad \text{if } u \notin \{u_1, \ldots, u_n\}\]

\[z\ [\varepsilon/self, \bar{u}] = z\]

\[\text{self}[\varepsilon/self, \bar{u}] = e_0\]

\[l\ [\varepsilon/self, \bar{u}] = l\quad \text{if } l = \text{nil, true, false, \text{n}}\]

\[l \cdot [\varepsilon/self, \bar{u}] = (l[e/self, \bar{u}]) \cdot x\]

If \(l_0\) then \(l_1\) else \(l_2\) fi \([\varepsilon/self, \bar{u}] = \{l_0[\varepsilon/self, \bar{u}]\text{ if } l_0\ [\varepsilon/self, \bar{u}]\text{ then } l_1[\varepsilon/self, \bar{u}]\text{ else } l_2[\varepsilon/self, \bar{u}]\}

\[(l_1 \equiv l_2)[\varepsilon/self, \bar{u}] = (l_1[\varepsilon/self, \bar{u}]) \equiv (l_2[\varepsilon/self, \bar{u}])\]

\[(l_1 + l_2)[\varepsilon/self, \bar{u}] = (l_1[\varepsilon/self, \bar{u}]) + (l_2[\varepsilon/self, \bar{u}])\]

\[\vdots\]

\[(l_1 < l_2)[\varepsilon/self, \bar{u}] = (l_1[\varepsilon/self, \bar{u}]) < (l_2[\varepsilon/self, \bar{u}])\]

\[\vdots\]
\[ |l[\bar{e}/\text{self, } \bar{u}]| = |l[\bar{e}/\text{self, } \bar{u}]| \]
\[ (l_1 \cdot l_2)[\bar{e}/\text{self, } \bar{u}] = (l_1[l[\bar{e}/\text{self, } \bar{u}] \cdot l_2[\bar{e}/\text{self, } \bar{u}]) \]

Now we extend this to assertions other than logical expressions:

\[(P \rightarrow Q)[\bar{e}/\text{self, } \bar{u}] = (P[\bar{e}/\text{self, } \bar{u}]) \rightarrow (Q[\bar{e}/\text{self, } \bar{u}]) \]
\[ (\neg P)[\bar{e}/\text{self, } \bar{u}] = \neg(P[\bar{e}/\text{self, } \bar{u}]) \]
\[ (\forall z P)[\bar{e}/\text{self, } \bar{u}] = \forall z(P[\bar{e}/\text{self, } \bar{u}]) \]
\[ (\exists z P)[\bar{e}/\text{self, } \bar{u}] = \exists z(P[\bar{e}/\text{self, } \bar{u}]) \]

Of course we also have a corresponding lemma:

**Lemma 5.28**

Let \( \sigma \in \Sigma \), \( \delta \in \Delta^c \) and \( e_i \in \text{Exp}_{\delta}^c \) 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 \( l_{d_0}^{\delta} \) and every valuation \( \omega \)
   \[ \mathcal{L}[l](\delta')(\sigma') = \mathcal{L}[l[\bar{e}/\text{self, } \bar{u}]](\omega)(\delta)(\sigma). \]

2. For every assertion \( P_{d_0}^{\delta} \) and every valuation \( \omega \)
   \[ \mathcal{A}[P](\delta')(\sigma') = \mathcal{A}[P[\bar{e}/\text{self, } \bar{u}]](\omega)(\delta)(\sigma). \]

**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
\( U = c : (m \leftarrow (u_0) : x_1 = u_0 \uparrow x_2) \) and the program \( \rho = (U|c : x_1 = u_1 \uparrow m(x_2)) \). We want to show
\[ \{ u_1 . x_1 \doteq x_1 \wedge \neg u_1 \doteq \text{self} \} \rho \{ u_1 . x_1 \doteq x_2 \wedge x_1 \doteq u_1 . x_2 \}. \]

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

- \( P = x_1 \doteq x_1 \wedge \neg \text{self} \doteq x_2 \)
- \( Q = x_1 \doteq u_0 \wedge \neg u_0 \doteq x_2 \)
- \( R = u_1 . x_1 \doteq x_2 \wedge x_1 \doteq u_1 . x_2 \)
- \( k = 0 \) (we shall use no \( v_i \))
- \( f_1 = x_1 \) (represented by \( z_1 \) in \( P \) and \( Q \))
- \( f_2 = \text{self} \) (represented by \( z_2 \) in \( P \) and \( Q \))
First notice that \( P[u_1, x_2/self, u_0][x_1, self/z_1, z_2] = u_1 \cdot x_1 \equiv x_1 \wedge \neg u_1 \equiv self \) so that the result of the rule is precisely what we want.

For the first premise we have to prove

\[
\{ x_1 \equiv z_1 \wedge \neg self \equiv z_2 \} \langle \Uparrow | c : x_1 \leftarrow u_0 \rangle \{ x_1 \equiv u_0 \wedge x_2 \equiv z_2 \wedge \neg self \equiv z_2 \}.
\]

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 premise, we have

\[
Q[u_1, x_2/self, u_0][x_1, self/z_1, z_2] = u_1 \cdot x_1 \equiv x_2 \wedge r \equiv u_1 \cdot x_2 \equiv \neg u_1 \equiv self
\]

\[
R[r/x_1] = \text{if } u_1 \equiv \text{self } \text{then } r \text{ else } u_1 \cdot x_1 \equiv x_2 \wedge r \equiv u_1 \cdot x_2
\]

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 premises are valid with respect to \( \gamma \). We shall prove that the conclusion is valid with respect to \( \gamma \). So let \( \sigma \in \Sigma, \delta \in \Delta^c, \) and \( \omega \in \Omega \) be such that \( \sigma, \delta, \omega \models P[\bar{e}/self, \bar{u}] [\bar{f}/\bar{z}] \). Let \( \gamma' = \Upsilon[U] (\gamma) \) and let \( \sigma' = P[U, \{ c : x \leftarrow e_0!m(z_1, \ldots, e_n) \}] (\gamma')(\delta) (\sigma) \). So \( \sigma' = S[z \leftarrow e_0!m(e_1, \ldots, e_n)](\gamma')(\delta)(\sigma) \). We have to prove \( \sigma', \delta, \omega \models R \).

Let \( \omega' = \omega \{ E[f_i](\delta')(\sigma)/z_i \}_{i=1}^f \). Then Lemma 5.12 gives us \( \sigma, \delta, \omega' \models P[\bar{e}/self, \bar{u}] \). Let \( \beta_i = E[e_i](\delta)(\sigma) \) 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 \( \delta' = \langle \beta_0, \delta_2 \rangle \cup \{ \delta_1 \} \) and \( \sigma_1 = (\sigma_{(1)}, \sigma_{(2)}, \sigma_{(3)}) \) where \( \sigma_1(3 \times d)(u_i) = \beta_i \) 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
\[ \sigma, \delta, \omega \models P[e/\text{self}, \bar{u}][f/z] \Rightarrow \sigma, \delta, \omega' \models P[e/\text{self}, \bar{u}] \Rightarrow \sigma_1, \delta', \omega' \models P \]
\[ \downarrow \]
\[ \sigma_2, \delta', \omega' \models Q[e/r] \]
\[ \downarrow \]
\[ \sigma'', \delta, \omega_1 \models Q[e/\text{self}, \bar{u}][f/z] \iff \sigma'', \delta, \omega_1' \models Q[e/\text{self}, \bar{u}] \iff \sigma_2, \delta', \omega_1' \models Q \]
\[ \downarrow \]
\[ \sigma'', \delta, \omega_1 \models R[r/x] \]
\[ \downarrow \]
\[ \sigma', \delta, \omega_1 \models R \]
\[ \downarrow \]
\[ \sigma', \delta, \omega \models R \]

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 \( \sigma_1, \delta', \omega' \models v_i \equiv \text{nil} \) for \( i = 1, \ldots, k \) so \( \sigma_1, \delta', \omega' \models P \land \bigwedge_{i=1}^{k} v_i \equiv \text{nil} \).

Now because of the construction of \( \gamma' \) in definition 3.17 we know that \( \gamma'(c', \bar{d})(m) = M[(\bar{u}) : S \mapsto e](\gamma) \) 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 \( \sigma_2 = P[\{U|c' : S\}](\gamma)(\delta')(\sigma_1) \), then \( \sigma_2 = S[S; \gamma'](\delta')(\sigma_1) \). Assume that \( \sigma_2 \neq \perp \), otherwise we have \( \sigma' = \perp \) and we are ready. The validity of the first premise with respect to \( \gamma \) tells us that \( \sigma_2, \delta', \omega' \models Q[e/r] \). Let \( \beta = e'[\delta'(\sigma_2), \omega_1 = \omega'[\beta/r] \), and \( \omega'_1 = \omega'[\beta/r] \). Then because of lemma 5.12 we have \( \sigma_2, \delta', \omega'_1 \models Q \).

Let \( \sigma'' = \{ \sigma_2(1), \sigma_2(2), \sigma_3(3) \} \) (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 \( f_i \) are unchanged in \( \sigma'' \) in comparison with \( \sigma \). Otherwise stated, \( E[e_1](\delta)(\sigma) = E[e_1](\delta)(\sigma'') \) and the same for \( f_i \). (Of course, this can also be proved formally.) Then we know from lemma 5.28 that \( \sigma'', \delta, \omega'_1 \models Q[e/\text{self}, \bar{u}] \) and from lemma 5.12 together with the observation that \( \omega'_1 = \omega_1[\{E[f_i]\}(\delta)(\sigma)/z_{i=1}^{|f|}] \) we get \( \sigma'', \delta, \omega_1 \models Q[e/\text{self}, \bar{u}][f/z] \).

From the second premise we can conclude that \( \sigma'', \delta, \omega_1 \models R[r/x] \). Now for the final state \( \sigma' \) we know that \( \sigma' = \sigma''[\beta/\delta_{(1)}] \), 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 \( u \leftarrow e_0 \mid m(e_1, \ldots, e_n) \), where \( u \in \text{TVar}_{d_0} \), \( m \in \text{MNames}_{d_0, \ldots, d_n} \),
$e_0 \in Exp^p_\omega$ and $e_i \in Exp^p_\omega$ 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 $\bot$, 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:

\[
\frac{\hat{F}_1, \ldots, \hat{F}_n}{F, \ldots, F_n \vdash F'_1, \ldots, F'_n}
\]

where
\[ F_i = \{ F_i^n(x_i/\text{self}, \bar{u}_i)[\bar{f}_i/\bar{z}_i]\} \langle U^- | c_i : x_i \leftarrow e_i \rangle m_i(e_i^1, \ldots, e_i^n)) \{ R_i^n \} \]
\[ F'_i = \{ F_i^n \land \bigwedge_{j=1}^n v_j = \text{nil} \} \langle U^- | c_i : S_i \} \{ Q_i^n[e_i/r_i] \} \]
\[ \bar{F}_i = Q_i[\bar{z}_i/\text{self}, \bar{u}_i][\bar{f}_i/\bar{z}_i] \rightarrow R_i[r_i/x_i] \]
\[ F = \{ P_i \land \bigwedge_{i=1}^n v_i = \text{nil} \} \langle U | c_i : S_i \} \{ Q_i^n[e_i/r_i] \} \]
\[ \bar{u}_i, S_i, \text{and } e_i \text{ are instance or temporary variables} \]
\[ U^- \text{ results from } U \text{ by deleting the definitions of } m_i, \ldots, m_n \]
\[ P_i, Q_i, \bar{f}_i, \bar{z}_i, \bar{u}_i, k_i, \text{and } r_i \text{ are just like in definition 5.24} \]

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 \( s^c \) functioning as a statement we have the following rule:

\[
\{ P^c \} \langle U | c : u_d \leftarrow s^c \} \{ Q^c \}
\]

\[
\{ P \} \langle U | c : s \} \{ Q \}
\]

(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:

\[
\{ P^c \} \langle U | S_1^c \} \{ Q^c \}
\]

\[
\{ P^c \} \langle U | S_2^c \} \{ R^c \}
\]

\[
\{ P \} \langle U | S_1; S_2 \} \{ R \}
\]

(SC)

**Definition 5.36**

For the conditional statement we have this rule:

\[
\{ P^c \land e_{B\text{bool}} \} \langle U | S_1^c \} \{ Q^c \}
\]

\[
\{ P^c \land \neg e \} \langle U | S_2^c \} \{ Q^c \}
\]

\[
\{ P \} \langle U | \text{if } e \text{ then } S_1 \text{ else } S_2 \} \{ Q \}
\]

(C)
Definition 5.37
For the while loop we have the following rule:
\[
\begin{align*}
\{ P^c \land e^c_{\text{Boo}} \} & \{ U | S^c \} \{ P^c \} \\
\{ P \} & \{ U | \text{while } e \text{ do } S \text{ od} \} \{ P \land \neg e \}
\end{align*}
\] 
(W)

Definition 5.38
For every valid (see definition 4.15) assertion \( P^c \) we have the axiom:
\[
P
\] 
(TR)

Definition 5.39
Finally, we have the so-called rule of consequence:
\[
P_1^c \rightarrow P_2^c & \{ P_2 \} \{ U | e : S \} \{ Q_2 \} \quad Q_2^c \rightarrow Q_1^c \\
\{ P_1 \} \{ U | e : S \} \{ Q_1 \}
\] 
(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 \( \gamma \) 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 \( \bar{F}_1, \ldots, \bar{F}_n \), 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 \( \gamma \) 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 \( \gamma \) and define \( \gamma' = U[U](\gamma) \). Now \( \gamma' \) is the limit of an increasing sequence \( \gamma_0, \gamma'_1, \ldots \), where \( \gamma_0 = \gamma \{ \lambda \bar{\beta} . \lambda \delta . \lambda \sigma . (\bot, \bot) / m_i \}_{i=1}^n \) and \( \gamma_{i+1}' \) is obtained from \( \gamma_i' \) by calculating and filling

\[
\begin{align*}
\{ P \} & \{ U | e : S \} \{ Q \}
\end{align*}
\]
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 \( U[U^{-}[\gamma_i')(m)] = \gamma_i'(m) \) because \( m \) is not defined in \( U^- \).

Now for \( \gamma_0' \) 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 \( \gamma_{i+1}' = F_j' \) 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 \( \gamma \models F. \) \( \square \)
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 \( \rho = \langle U | c : x \leftarrow \text{selfMark}() \rangle \) be closed and \( m() = \text{nil} \uplus \text{nil} \) occur in \( U \). We obviously have \( \models \{ \text{true} \} \rho \{ \text{false} \} \). But we do not have the derivability of this correctness formula. For otherwise there would exist assertions \( P, Q \) and \( R \) such that:

1. \( \models \{ P \land \land u_i = \text{nil} \} \langle U | c : \text{nil} \rangle \{ Q[\text{nil}/r] \} \)
2. \( \models Q[\text{self/self}][\bar{f}/\bar{z}] \rightarrow R[r/z] \)
3. \( \models \text{true} \rightarrow P[\text{self/self}][\bar{f}/\bar{z}] \) and \( \models R \rightarrow \text{false} \)

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 \( = R \rightarrow \text{false} \), we have \( \models R[r/x] \rightarrow \text{false} \). So from clause 2 it then follows that \( \models Q[\text{self/self}][\bar{f}/\bar{z}] \rightarrow \text{false} \). Furthermore, we have \( \models Q[\text{self/self}] \rightarrow Q \) so we infer \( \models Q[\bar{f}/\bar{z}] \rightarrow \text{false} \). From clause 1 in turn it is not difficult to deduce that \( \models P \rightarrow Q[\text{nil}/r] \) (use \( \bar{v}_i \cap \overline{TVar(P,Q)} = \emptyset \) and the truth of the correctness formula of clause 1). So we have \( P[\bar{f}/\bar{z}] \rightarrow Q[\text{nil}/r][\bar{f}/\bar{z}] \). Note next that \( \models Q[\bar{f}/\bar{z}] \rightarrow \text{false} \) implies \( \models Q[\text{nil}/r][\bar{f}/\bar{z}] \rightarrow \text{false} \), from which we infer that \( \models P[\bar{f}/\bar{z}] \rightarrow \text{false} \), which in turn, using \( \models P[\text{self/self}] \rightarrow P \), would imply by clause 3 \( \models \text{true} \rightarrow \text{false} \). We thus have reached a contradiction. So we conclude that \( \nexists \{ \text{true} \} \rho \{ \text{false} \} \).
Note that adding the conjunct \(-(\text{self} \equiv \epsilon_0)\) 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 \(\sigma, \delta, \omega\) we define \(OK(\sigma, \delta, \omega)\) iff \(\sigma\) is consistent, \(\delta\) agrees with \(\sigma, \omega\) is compatible with \(\sigma\) and for an arbitrary \(\epsilon\) we have
\[
\delta(2) = \{ \alpha : \exists n \in N(elt(b_c, n) = \alpha \neq \bot) \}.
\]

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 \(x \leftarrow \epsilon_0 m(e_1, \ldots, e_n)\), where \(x \in IVar_{d_0}\), \(m \in MName_{d_0} e_{d}, e_c \in Exp_e\) and \(e_i \in Exp_\delta\), for \(i = 1, \ldots, n\), we have the following proof rule:

\[
\{ P' \land \bigwedge_{i=1}^{k} u_i \in \text{nil} \land \neg(\text{self} \in b_c) \} \langle U | c' : S \} \{ Q[c'/x] \}, \quad Q' \rightarrow R' [r/x] \}
\]

\[
\{ P' \langle U | c : x \leftarrow \epsilon_0 m(e_1, \ldots, e_n) \} \{ R \}
\]

(MI)

where \(P' = P[e/\text{self}, \bar{u}] [f/z] [b_c \circ (\text{self}) / b_c], \quad Q' = Q[e/\text{self}, \bar{u}] [f/z] [b_c \circ (\text{self}) / b_c], \quad S \in \text{Stat}\) and \(e \in Exp_\delta\) are the statement and expression occurring in the definition of the method \(m\) in the unit \(U\), \(e_1, \ldots, e_n\) are its formal parameters, \(u_1, \ldots, u_k\) 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 \(R\), \(f\) is an arbitrary row of expressions \(\text{not}\)
logical expressions) in class $c$, and $\bar{z}$ is a row of logical variables, mutually different and different from $\tau$, 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 $l_i \in l_2$ abbreviates $\exists i (l_1 \equiv l_2 \cdot i)$, where $i$ is some fresh logical integer variable. $P[b_c \circ \langle \text{self} \rangle/b_c]$, for an arbitrary assertion $P$, equals the assertion

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

where $z \in \text{LVar}_c$, $i \in \text{LVar}_{\text{int}}$ are some fresh variables.

The idea of this substitution $[b_c \circ \langle \text{self} \rangle/b_c]$ can be explained roughly as follows: Occurrences of the variable $b_c$ in the assertions $F^c$ 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 $[b_c \circ \langle \text{self} \rangle/b_c]$ can be considered simply as part of the simultaneous substitution $[f/z]$). 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:

\[
\frac{\{P_1^c\} \rho^c \{Q_1^c\} \quad \{P_2^c\} \rho^c \{Q_2^c\}}{\{P_1^c \land P_2^c\} \rho^c \{Q_1^c \land Q_2^c\}} \quad \text{(CR)}
\]

**Definition 6.6**

Elimination rule 1:

\[
\frac{\{P^c\} \rho^c \{Q^c\}}{\exists z_d \ P^c \lor P[\text{nil}/z_d] \rho^c \{Q^c\}} \quad \text{(ER1)}
\]

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:

\[
\frac{\{P^c\} \rho^c \{Q^c\}}{\exists z_a \ P^c \rho^c \{Q^c\}} \quad \text{(ER2)}
\]

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

**Definition 6.8**

Initialization rule 1:

\[
\frac{\{P^c\} \rho^c \{Q^c\}}{\{P^c[t/z]\} \rho^c \{Q^c\}} \quad \text{(IR1)}
\]

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

**Definition 6.9**

Initialization rule 2:

\[
\frac{\{P^c\} \rho^c \{Q^c\}}{\{P^c[u/l]\} \rho^c \{Q^c\}} \quad \text{(IR2)}
\]

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

**Definition 6.10**

Substitution rule:

\[
\frac{\{P^c\} \rho^c \{Q^c\}}{\{P^c[z'/z]\} \rho^c \{Q^c[z'/z]\}} \quad \text{(SR)}
\]

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 $c$,
$$\models \{ x \in b_c \} \rho \{ \text{false} \},$$
where $\models x \in b_c$ abbreviates the assertion $\exists i (x \equiv b_c \cdot i)$. 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:
$$\{ x \in z \} \rho \{ \text{false} \}.$$
Applying next the elimination rule (ER2), assuming $z \not\in BVar$, then gives us the derivability of the formula:
$$\exists z (x \in z) \rho \{ \text{false} \}.$$
Finally, we apply the consequence rule:
$$\{ \text{true} \} \rho \{ \text{false} \}.$$
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 $\bigcup_{d \in d} IVar_d$, where $\epsilon$ ranges over $C'$, and $\delta$ ranges over the set $C'^+ = C' \cup \{ \text{int}, \text{Bool} \}$. Next we fix the temporary integer variables $u, u'$, and for every $d \in C'^+$ the temporary variables $re_d, re'_d$. Let $\tilde{r}$ denote a sequence of these variables. Now let $TVar'$ be a finite subset of $\bigcup_{d} TVar_d$ (again, $\delta$ ranging over $C'^+$), such that $\tilde{r} \subseteq TVar'$. Given these sets $C'$, $IVar'$, and $TVar'$ we have the following definition.

Definition 6.12
We define an expression $c^+_d$ to be restricted iff $c \in C'$, $a = d, d'$, with $d \in C'^+$, $IVar(c^+_d) \subseteq IVar'$, and $TVar(c^+_d) \subseteq TVar'$. 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 if $c \in C'$, every expression occurring in $\rho$ is restricted, $u, u' \not\in TVar(\rho)$, and, finally, the temporary variables $re_d, re'_d$ are only allowed in the main statement $S$ itself, where $S = re_d - s_d$ or $S = re'_d - s_d$, with $TVar(s) \cap \tilde{r} = \emptyset$. A correctness formula $\{ P \} \rho \{ Q \}$ is called restricted if $P, Q$, and $\rho$ 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 $\tilde{r}$ 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 formulæ. 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 \( \sigma \), which we denote by \( \sigma \downarrow \), to be an element of

\[
\Sigma \downarrow = \prod_{c \in C'} P^c \times \prod_{c \in C', d \subseteq C'^+} \left( O^c \rightarrow IVar'_d \rightarrow O^d_1 \right) \times \prod_{d \subseteq C'^+} \left( TVar'_d \rightarrow O^d_1 \right)
\]

such that

- \( \sigma \downarrow^{(c)} = \sigma^{(c)}, c \in C' \).
- \( \sigma \downarrow (\alpha)(x) = \sigma(\alpha)(x), \alpha \in O^c, \) 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 \( \rho \), and \( \sigma, \sigma', \delta, \gamma \) such that

1. \( \sigma^{(c)} = \sigma^{(c)}, c \not\in C' \).
2. \( \sigma(\alpha) = \sigma'(\alpha), \) for \( \alpha \in O^c, c \not\in C' \).
3. \( \sigma(\alpha) = \sigma'(\alpha), \) for \( \alpha \in O^c \setminus \sigma^{(c)}, c \in C' \).
4. \( \sigma(\alpha)(x) = \sigma'(\alpha)(x), \) for \( \alpha \in \sigma^{(c)}, c \in C', x \not\in IVar' \).
5. \( \sigma(\alpha)(x) = \bot, \) for \( \alpha \in \sigma^{(c)} \setminus \sigma^{(c)}, c \in C', x \not\in IVar' \).
6. \( \sigma(u) = \sigma'(u), u \not\in TVar' \).

we have

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

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

**Proof**

Induction on the structure of the program $\rho$.

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 $\sigma, \delta, \omega$ such that $OK(\sigma, \delta, \omega)$ we have

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

where $\omega \models \bigwedge \omega \models LVar_{a} \rightarrow O_{a}^{\omega}$, 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 $\sigma, \delta, \omega$ such that $OK(\sigma, \delta, \omega)$. We have for an arbitrary restricted correctness formula $\{P\} \rho\{Q\}$

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

**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 $\overline{r}$ as defined above. A program $\rho$ from now on will denote, when not stated otherwise, a program such that the temporary variables $r$, $r'$ are allowed to occur in it only in assignments $r \leftarrow s, r' \leftarrow s$, with $r, r' \not\in TVar(s)$, and $u, u' \not\in TVar(\rho)$. 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 $f = (f^d)_{d \in \mathbb{C}^+}$, where $f^d \in O_1^d \rightarrow O_2^d$ is a bijection, $j^d(\bot) = \bot$ and $f^d$, for $d = \text{int, bool}$, 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 $\sigma$ the state $f(\sigma)$ as follows:

- For every $c$: $f(\sigma)^{(c)} = f^c(\sigma^{(c)})$.
- For every $c, d, \sigma^c, x^d_\alpha$: $f(\sigma)^(c)(x^d_\alpha) = f^d(\sigma(f^{-1}^c(\alpha))(x^d_\alpha))$, where the osi $f^{-1}$ denotes the inverse of $f$: $f^{-1} = (f^d)^{-1}$.
- 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 $\sigma, \delta, \omega$ be such that $OK(\sigma, \delta, \omega)$. Then for every logical expression $l^c_\alpha$ and assertion $P^c$ we have:
• $f^*(L[z_1]((\omega)(\delta))(\sigma)) = L[z_1](f(\omega))(f(\delta))(f(\sigma))$,

• $A[P^c]((\omega)(\delta))(\sigma) = A[P^c](f(\omega))(f(\delta))(f(\sigma))$.

where $f(\delta)(1) = f^c(\delta(1))$, $f(\delta)(2)(c') = f^c(\delta(2)(c'))$, for an arbitrary $c'$, and $f(\omega)(z_2) = f^d(\omega(z_2))$, $f^c(\alpha_1, \ldots, \alpha_n) = (f^d(\alpha_1), \ldots, f^d(\alpha_n))$.

Proof
Straightforward induction on the structure of $I^c$, $P^c$. We only treat the case $l = z^c_2$:

$L[x](f(\omega))(f(\delta))(f(\sigma)) = f(\sigma)f^c(\delta(1))(x) = f^d(\sigma(\delta(1))(x)) = f^d(L[x](\omega)(\delta))(\sigma)$

We are now sufficiently prepared to analyze the notion of a strongest postcondition.

Given a program $\rho^c$ and an assertion $P^c$, we denote by $sp(\rho^c, P^c)$ the set of final states of executions of $\rho^c$ 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 $\rho^c$. 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 $\rho^c$ 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 $\rho^c = (U | c : x \leftarrow \text{new})$, with $\rho^c$ closed, and $\sigma, \sigma', \delta$ such that $\sigma'(c) = \{\alpha, \beta\}$, $\sigma(c) = \{\alpha\}$, $\delta(1) = \alpha$ and $\sigma' = P[\rho^c](\gamma)(\delta)(\sigma)$. Let $P^c = \text{true}$. 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 $pick^c(\{f^c(\beta)\}) \neq f^c(\beta)$. So we have that $f(\sigma')(c) = \{f^c(\alpha), f^c(\beta)\}$. Now suppose that there is a $\sigma_0$ such that $f(\sigma') = P[\rho^c](\gamma)(f(\delta))(f(\sigma_0))$. 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 $\sigma_0$ does not exist and $f(\sigma') \notin sp(\rho^c, \text{true})$.

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 $\rho^c$ 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 $\sigma, \delta, \omega$ 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 $\sigma_0$ such that:

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

- $\sigma_0, \delta', \omega' \models P^c$.

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

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

Note that in the above theorem we cannot take $f(\omega)$, where $f(\omega)(z) = f(\omega(z))$, for $\omega'$. This would require that $f(\omega)$ and $\sigma_0$ are compatible, which cannot be expressed by our assertion language. For suppose there exists an $\alpha \in \sigma'_{(z')}$, for some $z'$, such that $f^c(\alpha) \notin \sigma'_{0(z')}$. Let $z' \notin L$, it then follows that $\sigma, \delta, \omega_{\{\alpha/z'\}} \models SP^c_L(\rho, P^c)$, but on the other hand it is not the case that $f(\omega_{\{\alpha/z'\}})$ and $\sigma_0$ are compatible, so we do not have $\sigma_0, \delta', f(\omega_{\{\alpha/z'\}}) \models P^c$. 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 $\sigma_0$.

**Proof**

See appendix B.

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 $\rho^c$ and assertion $P^c$ such that $LVar(P^c) \subseteq L$, we have

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

**Proof**

Let $\sigma, \sigma', \delta, \omega$ ($\sigma, \sigma' \neq \perp$) be such that $OK(\sigma, \delta, \omega)$, $\sigma' = \mathcal{P}[\rho^c](\gamma)(\delta)(\sigma)$ ($\gamma$ 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 $\sigma_0$ the state $\sigma$, 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 $\rho^c$, assertions $P^c, Q^c$, $BVar \subseteq L \subseteq LVar$ (L finite) such that $LVar(P^c, Q^c) \subseteq L$ we have

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

Proof
Assume $\models \{ P \} \rho \{ Q \}$ and let $\sigma, \delta, \omega$ 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 $\sigma_0$ such that:

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

where $\delta' = f(\delta)$ and $\omega' = f(\omega) \downarrow L$. From $\models \{ P^c \} \rho^c \{ Q^c \}$ 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, $\rho$ denote a program of some simple procedural language (like the ones treated in [3] or [10]) and $\sigma, \sigma'$ denote some simple functions assigning values to program variables. Let $\bar{x}$ denote the set of program variables occurring in $\rho$, $\bar{z}$ denote a corresponding sequence of logical variables and $\bar{x} \equiv \bar{z}$ abbreviate $\bigwedge_i (x_i \equiv z_i)$. Furthermore let $SP(\rho, \bar{x} \equiv \bar{z})$ be an assertion describing the set of final states resulting from executions of $\rho$ starting in a state satisfying $\bar{x} \equiv \bar{z}$. In the standard completeness proof an important consequence of the definition of the notion of strongest postcondition is that the assertion $SP(\rho, \bar{x} \equiv \bar{z})$ in the following sense describes the graph of $\rho$:

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

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

Note that the logical variables $\bar{x}$ 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 $i_{x_d} \in LVar_d$, and, finally, for each temporary variable $u_d$ we fix a logical variable $i_{u_d} \in LVar_d$. The sequence denoted by $i_{x_d}$, $x \in IVar^c$, 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 $i$th element of the sequence $i_{x_d}$ is the value of the variable $x$ in the object that is the $i$th element of the sequence $cr_c$. The value of $i_{u_d}$, $u \in TVar$, just equals that of $u$.

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

**Definition 6.24**

We define the assertion $\text{init}$ as follows:

$$
\text{init} = \bigwedge_c cr_c \cdot 1 \equiv \text{nil} \land \forall z_x \exists i (z_x \equiv cr_x \cdot i) \land \\
\bigwedge_c \forall i (\bigwedge_{x \in IV_x} ((cr_x \cdot i) \cdot x \equiv i_{x_d} \cdot i)) \land \\
\bigwedge_{u \in TVar} (u \equiv i_{u_d}) \land \\
\bigwedge_d (b_c \equiv bl_c)
$$

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 $d^r$, 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 $l_1 \equiv l_2$ can be expressed as $\forall i (l_1 \cdot i \equiv l_2 \cdot i)$, where $i$ is some logical integer variable. Furthermore we remark that for every class name $c$ we have $\text{init} \in Ass^c$. 
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 \( \vec{s}t \) 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 \( \text{self} \). But to look up this value in the logical environment one has to know where the object denoted by \( \text{self} \) occurs in the sequence denoted by \( \text{cr} \), assuming \( x \in \text{IVar}_d \) 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 \( \text{cr}_c \). Our solution to this problem consists essentially of using a second logical expression, of type \( \text{Bool} \), 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 \( \text{cr}_c \). In our example above, the expression \( x \) is then translated into the triple \( \langle \langle i \rangle, \text{self} \equiv \text{cr}_c \cdot i, \text{iv}_x \cdot i \rangle \), where \( i \) is some logical integer variable. This is interpreted as follows: Whenever the variable \( i \) takes such a value that the Boolean expression \( \text{self} \equiv \text{cr}_c \cdot i \) is true, then the expression \( \text{iv}_x \cdot i \) takes the desired value.

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

**Definition 6.26**

We define \( I^\vec{a}_c [\vec{s}t] = (\vec{i}, l_1^\text{Bool}, l_2^\text{a}) \) for an arbitrary logical expression \( I^\vec{a}_c \) by induction on the structure of \( I^\vec{a}_c \). Let \( \epsilon \) denote the empty sequence. We treat the following cases:

- \( x^\vec{a}_c[\vec{s}t] = \langle \langle i \rangle, \text{self} \equiv \text{cr}_c \cdot i, \text{iv}_x \cdot i \rangle \)
  where \( i \) is some fresh logical integer variable (it does not occur in \( \vec{s}t \)).

- \( u^\vec{a}_c[\vec{s}t] = (\epsilon, \text{true}, l) \)

- \( l[\vec{s}t] = (\epsilon, \text{true}, l) \)
  where \( l = \text{nil}, \text{self}, \text{true}, \text{false}, \text{a}, \text{or}, \text{z} \).

- \( (l_c \cdot x^\vec{a}_c)[\vec{s}t] = (\vec{i} \cdot \langle j \rangle, l_1 \wedge l_2 \equiv \text{cr}_c \cdot j, \text{iv}_x \cdot j) \)
  where \( l_c[l] = \vec{i}, l_1, l_2 \) and \( j \not\in \vec{i} \).

- \( (l_1 + l_2)[\vec{s}t] = \vec{\tilde{i}}, l_1, l_2 \) where \( l_1[l_1] = (\vec{i}_1, l_1, l_1), l_2[l_2] = (\vec{i}_2, l_2, l_2) \), \( \vec{\tilde{i}} = \vec{i}_1 \circ \vec{j}, \vec{j} \) is some sequence of fresh logical integer variables of the same length as \( \vec{i}_2, l'_1 = l_2[\vec{j} / l_2], \) and \( l'_2 = l_2[\vec{j} / l_2] \).
(1) if \( l_1 \) then \( l_2 \) else \( l_3 \) \( \hat{s}t \) = \( \langle \hat{i}, l_1, \land l_2, \land l_3, \rangle \) if \( l_{12} \) then \( l'_{23} \), else \( l'_{32} \).

Where \( l_1[\hat{s}t] = \langle \hat{i}, l_1, l_1, l_1 \rangle \), \( l_2[\hat{s}t] = \langle \hat{i}, l_2, l_2, l_2 \rangle \), \( l_3[\hat{s}t] = \langle \hat{i}, l_3, l_3, l_3 \rangle \), \( \hat{i} = \hat{i} \circ \hat{j} \circ \hat{\hat{j}} \), \( \hat{\hat{j}} \) and \( \hat{j} \) are mutually disjoint sequences of fresh logical variables of the same length as \( \hat{i} \) and \( \hat{i} \), respectively, \( l'_{12} = l_2[\hat{j}_2/i_2] \), \( l'_{22} = l_2[\hat{j}_2/i_2] \), \( l'_{32} = l_3[\hat{j}_3/i_3] \), and \( l'_{32} = l_3[\hat{j}_3/i_3] \).

(2) \( l_1 \cdot l_2 \) \( \hat{s}t \) = \( \langle \hat{i}, l_1, \land l_2, l_1, \cdot l_2 \rangle \)

Where \( l_1[\hat{s}t] = \langle \hat{i}, l_1, l_1, l_1 \rangle \), \( l_2[\hat{s}t] = \langle \hat{i}, l_2, l_2, l_2 \rangle \), \( \hat{i} = \hat{i} \circ \hat{j} \), \( \hat{j} \) is some sequence of fresh logical integer variables of the same length as \( \hat{i} \), \( l'_{12} = l_2[\hat{j}_2/i_2] \), and \( l'_{22} = l_2[\hat{j}_2/i_2] \).

(3) \( l_1 \dagger l_2 \) \( \hat{s}t \) = \( \langle \hat{i}, l_1, \land l_2, \dagger l_2 \rangle \)

Where \( l_1[\hat{s}t] = \langle \hat{i}, l_1, l_1, l_1 \rangle \), \( l_2[\hat{s}t] = \langle \hat{i}, l_2, l_2, l_2 \rangle \), \( \hat{i} = \hat{i} \circ \hat{j} \), \( \hat{j} \) is some sequence of fresh logical integer variables of the same length as \( \hat{i} \), \( l'_{12} = l_2[\hat{j}_2/i_2] \), and \( l'_{22} = l_2[\hat{j}_2/i_2] \).

Note that in \( l_1[\hat{s}t] = \langle \hat{i}, l_1, l_1 \rangle \), the expression \( l_1 \) describes where the relevant existing objects, with respect to the evaluation of \( l_1 \), are stored in that part of the logical environment as specified by \( \text{cr}_c, c \in C \). 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 \( z \cdot x \cdot y \), where \( z \in L\text{Var}_c, x \in L\text{Var}_c \). We have

\[
(z \cdot x \cdot y)[\hat{s}t] = \langle (i, j), z \equiv \text{cr}_c \cdot i \land iv_x \cdot i \equiv \text{cr}_c \cdot j, iv_y \cdot j \rangle
\]

where \( i \) and \( j \) are distinct logical integer variables.

**Definition 6.28**

Next we define the transformation \( P^c[\hat{s}t] \) for an arbitrary assertion \( P^c \) by induction on the structure of \( P^c \). We treat the following cases:

- \( l_1[\hat{s}t] = l_2[\hat{s}t] \)
- \( (P_1 \lor P_2)[\hat{s}t] = P_1[\hat{s}t] \lor P_2[\hat{s}t], \ldots \)
- \( (\forall z_a P)[\hat{s}t] = \forall z_a P[\hat{s}t] \),

where \( a = d, d^*, d = \text{Int, Bool} \).
- \( (\forall z_c P)[\hat{s}t] = \forall z_c (z_c \in \text{cr}_c \rightarrow P[\hat{s}t]) \).
• $(\forall z_a P)[\bar{s}t] = \forall z_a(z_a \subseteq cr_c \rightarrow P[\bar{s}t])$, 
  where $a = c^*$.

• $(\exists z_a P)[\bar{s}t] = \exists z_a P[\bar{s}t]$, 
  where $a = d^*, d = \text{Int, Bool}$.

• $(\exists z_c P)[\bar{s}t] = \exists z_c(z_c \in cr_c \land P[\bar{s}t])$.

• $(\exists z_a P)[\bar{s}t] = \exists z_a(z_a \subseteq cr_c \land P[\bar{s}t])$, 
  where $a = c^*$.

Here $l_1 \subseteq l_2$ abbreviates $\exists i(l_1 \subseteq l_2 \cdot i)$ and $l_1 \subseteq l_2$ abbreviates $\forall i(l_1 \cdot i \in l_2)$. Note that, although nil $\in cr_c$, the quantification in $(\forall z_a P)[\bar{s}t]$ and $(\exists z_a P)[\bar{s}t]$ 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 $\sigma, \delta, \omega$ such that $OK(\sigma, \delta, \omega)$ and $\sigma, \delta, \omega \models init$. Then:

$\sigma, \delta, \omega \models P^c$ iff $\sigma, \delta, \omega \models P^c[\bar{s}t]$.

**Proof**

The proof proceeds by induction on the structure of $P^c$. The case that $P^c$ equals $I^c_{\text{Bool}}$ is treated as follows: We prove that for every logical expression $I^c_{\bar{s}t}$ 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 $L[I^c_{\bar{s}t}](\omega)(\delta)(\sigma) = L[I^c_{\bar{s}t}](\omega(\bar{n}/\bar{i}))(\sigma)$, where $I^c_{\bar{s}t} = (l, l_1, l_2)$. This is proved by induction on the structure of $I^c_{\bar{s}t}$. \(\square\)

### 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 $\exists z \forall z'[z \equiv z']$, 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 $\rho^c = \langle U | x \leftarrow \text{new} \rangle$, $U$ arbitrary and $x \in IVar^c$. Then it is not the case that $\models \{P\} \rho^c \{P\}$, 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

$$P^c ::= I^c_{\text{Bool}}$$

$$\vdots$$

$$\exists z_a P \mid \forall z_a P$$

where $a = d, d^*, d = \text{Int, Bool}$

$$\exists z_c (z_c \in z^*_c \land P^c)$$

$$\exists z^*_c (z^*_c \subseteq z^*_c \land P^c)$$

$$\forall z_c (z_c \in z^*_c \rightarrow P^c)$$

$$\forall z^*_c (z^*_c \subseteq z^*_c \rightarrow P^c)$$

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 $\models P \leftrightarrow P[\text{new}/v]$.

**Proof**

Induction on the complexity of $P$. We treat the representative case of $P = \exists z_c (z_c \in z^*_c \land Q)$, assuming the type of the variable $v$ to be $c$: Now $P[\text{new}/v] = \exists z_c (z_c \in z^*_c \land Q[\text{new}/v]) \lor (v \in z^*_c \land Q[v/z_c])[\text{new}/v]$. But as $(v \in z^*_c)[\text{new}/v]$ 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 $Q[\text{new}/v]$ is equivalent to $Q$. Putting these observations together gives us the equivalence of $P$ and $P[\text{new}/v]$. The case $P = \forall z_c (z_c \in z^*_c \rightarrow Q)$ is treated analogously. The cases of $P = \exists z^*_c (z^*_c \subseteq z^*_c \land Q) \lor \forall z^*_c (z^*_c \subseteq z^*_c \rightarrow Q)$ 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 \( \rho' = \langle U | c' : v' \leftarrow e_0!m'(e_1', \ldots, e_n') \rangle \in M \)
  then \( \rho_i = \langle U | c_i : v_i \leftarrow e_0!m_i(e_1^i, \ldots, e_n^i) \rangle \in M \),

where \( v_i = e_0!m_i(e_1^i, \ldots, e_n^i) \) or \( e_0!m_i(e_1^i, \ldots, e_n^i) \) occurs in \( S' \), \( S' \) being the body of the method \( m' \). In the latter case we have \( v_i = \text{rec}_{d_i} \), 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: \( \rho_i = \langle U | c_i : v_i \leftarrow e_0!m_i(e_1^i, \ldots, e_n^i) \rangle \in M \) and \( m_i(u_1^i, \ldots, u_n^i) \leftarrow S_i \uparrow e_i \) occurs in \( U \), \( i = 1, \ldots, k \). Furthermore, \( \bar{e}^i \) denotes the sequence \( e_1^i, \ldots, e_n^i \) and \( \bar{u}^i \) the sequence \( u_1^i, \ldots, u_n^i \). Next we introduce for every class name \( c \) a new variable \( z_c \). We let \( \bar{v} \) denote a sequence (without repetitions) of these variables and \( \bar{b} \) denote the corresponding sequence of the variables \( b_i \in BVar \). Finally we put for \( i = 1, \ldots, k \):

\[
F_i = \{ P' \} \rho_i \{ P' \},
\]

where \( P' = P'[\bar{z}/\bar{b}][z_c/self], z_c \) being a new variable.

Now we have that

\[
F_1, \ldots, F_k \vdash \{ P' \} \langle U | c'_1 : S_1 \rangle \{ P' \}
\]

(\( c'_1 \) being the type of \( e_0^1 \)). This is established by induction on the complexity of \( S_i \).

The only slightly less straightforward case of \( S_i = v \leftarrow \text{new} \) is taken care by the previous lemma.

Putting \( P_1, Q_1, R_A = P' \) and introducing some logical variable \( r_i \not\in LVar(P') \) of the same type as the variable \( v_i \), \( i = 1, \ldots, k \), and observing that \( P'[\bar{e}^i/self, \bar{u}^i][b_{c_i} \circ (\text{self})/b_{c_i}] = P' \) we infer by (NMR) that:

\[
\vdash \{ P' \} \langle U | c'_i : S_i \rangle \{ P' \}.
\]

Next we put \( P_1, Q_1 = P' \) and \( R_A = P^c[\bar{z}/\bar{b}] \). We have that:

\[
\models P_1[\bar{e}^1/self, \bar{u}^1][\text{self}/z_{c_1}][b_{c_1} \circ (\text{self})/b_{c_1}] \rightarrow P^c[\bar{z}/\bar{b}]
\]
and
\[ | Q_1[\varepsilon^1/\text{self}, \bar{u}^1][\text{self}/\bar{z}_{1\mathrm{a}}][b_{c_1} \cdot (\text{self})/b_{z_1}] \rightarrow R_1[r_1/v_1]. \]

Thus applying (MI) (or (MT)) gives us that:
\[ \vdash \{ P^c[z/\bar{b}] \} \rho^c \{ P^c \}. \]

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

\( \Box \)

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 \( \{ \text{init} \} \rho^c \{ \text{SP}_L^e(\rho^c, \text{init}) \} \), for some \( L \subseteq L\text{Var} \), 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 \( \{ \text{init} \} \rho^c \{ \text{SP}_L^e(\rho^c, \text{init}) \} \), 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 \( s\bar{t} \).
This is necessary because the variables of \( s\bar{t} \) have a fixed interpretation as specified by the assertion \( \text{init} \). But every valid correctness formula in which variables of \( s\bar{t} \) 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 \( s\bar{t} \) by some corresponding new variable.

So we introduce for each \( c \) fresh logical variables \( cr_1^c, bl_1^c \in L\text{Var}_d \). For each instance variable \( x \in I\text{Var}_d \) we introduce the fresh logical variable \( int_{x} \in L\text{Var}_d \), and with each temporary variable \( u \in T\text{Var}_d \) we associate the fresh logical variable \( tu^u \in L\text{Var}_d \). We assume again that all these newly introduced logical variables are distinct. We let \( s\bar{t} \) denote a sequence (without repetitions) of these variables. We can thus assume that \( s\bar{t} \cap s\bar{t} = \emptyset \).

Furthermore we introduce for every temporary variable \( r_e \) (defined in the introduction to justify the assumption of the finiteness of the sets \( C, I\text{Var} \), and \( T\text{Var} \)) a fresh logical variable \( t_e \). Let \( t_e \) denote a sequence of these logical variables. We will use the variable \( t_e \) when applying the rule (ES): Applications of this rule will make use of the variable \( t_e \) to store temporarily the result of the expression \( s \). Therefore we have to substitute occurrences of \( t_e \) in the precondition and the postcondition by the corresponding variable \( t_e \). 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 formulas about simple assignments.

**Lemma 6.34**
For an arbitrary program \( \rho = \langle U | c : v \leftarrow e \rangle \) we have
\[
\models \{ \rho \} \rho \{ Q^c \} \implies \{ \rho \} \rho \{ Q^c \}.
\]

**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 \( \models \{ \rho \} \rho \{ Q \} \) it follows that \( \models \rho \rightarrow Q^c[e/u] \). So an application of the axiom (SAT) and the consequence rule gives us the derivability of the correctness formula \( \{ \rho \} \rho \{ Q \} \).

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

**Lemma 6.35**
For an arbitrary program \( \rho = \langle U | c : v \leftarrow \text{new} \rangle \) we have
\[
\models \{ \rho \} \rho \{ Q^c \} \implies \{ \rho \} \rho \{ Q^c \}.
\]

**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 \( \models \{ \rho \} \rho \{ Q \} \) it follows that \( \models \rho \rightarrow Q^c[\text{new}/v] \). So an application of the axiom (NT) and the consequence rule gives us the derivability of the correctness formula \( \{ \rho \} \rho \{ Q \} \).

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

**Lemma 6.36**
Let \( \rho = \langle U | c : v \leftarrow \text{e}_0!m(\text{e}_1, \ldots, \text{e}_n) \rangle \) be a closed program. Furthermore let \( \rho, Q^c \) and \( BVar \subseteq L \subseteq LVar \) (\( L \) finite) such that \( LVar(P, Q) \subseteq L \setminus \text{st}I \), and \( \text{st} \cup \text{st}I \subseteq L \). Then:
\[
\models \{ \rho \} \rho \{ Q^c \} \implies \{ \rho \} \rho \{ \text{init} \} \rho \{ \text{SP}_L(\rho, \text{init}) \} \models \{ \rho \} \rho \{ Q^c \}.
\]
Proof
Let \( P' = P[s\tilde{t}/\tilde{s}t] \) and \( Q' = Q[s\tilde{t}/\tilde{s}t] \). Furthermore we introduce the following abbreviation: \( P'' = P'[\tilde{s}t] \). We start with the assumption:

\[
\{ \text{init} \} \rho \{ SP_L^2(\rho, \text{init}) \}.
\]

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:

\[
\{ P'' \} \rho \{ p'' \}.
\]

Applying the conjunction rule gives us:

\[
\{ P'' \land \text{init} \} \rho \{ P'' \land SP_L^2(\rho, \text{init}) \}.
\]

We next prove that \( \models P'' \land SP_L^2(\rho, \text{init}) \rightarrow Q' \):
Let \( \sigma, \delta, \omega \models P'' \land SP_L^2(\rho, \text{init}) \). So there exist a state \( \sigma_0 \) and an osi \( f \) such that

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

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 \( IVar(P'') \subseteq L \), so we have \( \models f(\sigma), \delta', \omega' \models P'' \). Furthermore we have that \( \models \{ \neg P'' \} \rho \{ \neg P'' \} \) (by theorem 6.33 we have \( \models \{ \neg P'' \} \rho \{ \neg P'' \} \) 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 6.2, note that \( \sigma_0, \delta', \omega' \models \text{init} \), we then infer \( \sigma_0, \delta', \omega' \models P' \). By the soundness of the substitution rule (SR) we have that \( \models \{ P \} \rho \{ Q \} \) implies the truth of the correctness formula \( \{ P' \} \rho \{ Q' \} \). 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:

\[
\{ P'' \land \text{init} \} \rho \{ Q' \}.
\]

Next we apply the initialization rule (IR1):

\[
\{ (P'' \land \text{init})[\widetilde{u}/\widetilde{v}] \} \rho \{ Q' \},
\]

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

$$\{\exists \bar{x}(P'' \land \text{init})[\bar{u}/\bar{v}]\}P'\{Q'\},$$

where $\bar{x}$ 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 $\bar{v}$ 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 $\models P' \Rightarrow \exists \bar{x}(P'' \land \text{init})[\bar{u}/\bar{v}]$: Let $\sigma, \delta, \omega$ be such that $OK(\sigma, \delta, \omega)$ and $\sigma, \delta, \omega \models P'$. It is not difficult to see that there exists an $\omega'$ such that $\omega'$ differs from $\omega$ only with respect to the variables of $\bar{u}$ and $\sigma, \delta, \omega' \models \text{init}$. As $LVar(P') \cap \bar{u} = \emptyset$ we have $\sigma, \delta, \omega' \models P'$. Applying theorem 6.29 then gives us $\sigma, \delta, \omega' \models P'[\bar{u} / \bar{v}]$. For every temporary variable $u$ we have $\sigma(u) = \omega'(tv_u)$, so we infer $\sigma, \delta, \omega' \models (P'' \land \text{init})[\bar{u} / \bar{v}]$. So we conclude $\sigma, \delta, \omega \models \exists \bar{x}(P'' \land \text{init})[\bar{u} / \bar{v}]$.

We thus have by the consequence rule:

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

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

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

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

$$\models \{P\}P\{Q\} \implies \{P'\}P\{Q'\},$$

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

**Proof**

Let $\sigma, \delta, \omega \models P'$ and $\sigma' = P[\rho]'(\gamma)(\delta)(\sigma)$. We have that $\sigma' = \sigma''[\beta / re]$, with $(\sigma'', \beta) = U[s][\gamma')(\delta)(\sigma)$, $\gamma' = U[U'](\gamma)$. As $re \notin TVar(s)$ ($\rho$ being restricted) we have $(\sigma_1, \beta) = U[s][\gamma')(\delta)(\sigma_0)$, with $\sigma_1 = \sigma''[\omega(re)/re]$ and $\sigma_0 = \sigma[\omega(re)/re]$. 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, \( \sigma^n, \delta, \omega \models Q' \). Finally, as \( re \notin TVar(Q') \), we conclude that \( \sigma', \delta, \omega \models Q' \). \( \Box \)

Lemma 6.38
Let \( \rho = \{U|c : s\} \), where \( s = e, \text{new} \). Furthermore let \( P, Q \) such that \( LVar(P, Q) \cap \text{lre} = \emptyset \). Then:
\[
\models \{P\} \rho \{Q\} \implies \vdash \{P\} \rho \{Q\}.
\]

Proof
Let \( P' = P[\text{lre}/re] \) and \( Q' = Q[\text{lre}/re] \), where \( \text{lre} \) and \( re \) are of the same type as the expression \( s \). By lemma 6.37 we have \( \models \{P'\} \rho' \{Q'\} \), where \( \rho' = \{U|c : re \leftarrow s\} \). By lemma 6.34, in case \( s = e \), and lemma 6.35, if \( s = \text{new} \), we then have
\[
\vdash \{P'\} \rho' \{Q'\}.
\]
So by rule (ES) it follows that
\[
\vdash \{P\} \rho \{Q\}.
\]
Furthermore we have \( \models \{\text{lre} \equiv re\} \{U|c : re' \leftarrow s\} \{\text{lre} \equiv re\} \). So again by lemmas 6.34 and 6.35 we have
\[
\vdash \{\text{lre} \equiv re\} \{U|c : re' \leftarrow s\} \{\text{lre} \equiv re\}.
\]
Applying again the rule (ES) then gives
\[
\vdash \{\text{lre} \equiv re\} \rho \{\text{lre} \equiv re\}.
\]
Next we apply the conjunction rule
\[
\vdash \{\text{lre} \equiv re \land P'\} \rho \{\text{lre} \equiv re \land Q'\}.
\]
Now \( \models (\text{lre} \equiv re \land Q') \rightarrow Q \) and \( \vdash P \rightarrow (\exists ! r P'' \lor P''[\text{nil}/\text{lre}]) \), where \( P'' = \text{lre} \equiv re \land P' \). (Note that \( \text{lre} \notin LVar(P) \).) So applying first the consequence rule for \( Q \), then the elimination rule (ER.1) (note that \( \text{lre} \notin LVar(Q) \)), and finally the consequence rule for \( P \), gives us the derivability of
\[
\vdash \{P\} \rho \{Q\}.
\]
\( \Box \)

We have a similar lemma for valid correctness formulae about a program \( \rho \) of the form \( \{U|c : e_0!m(e_1, \ldots, e_n)\} \).
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 $LVar(P, Q) \subseteq L \setminus (sI \cup lre)$, $st \cup sI \cup lre \subseteq L$. Then we have

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

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 $P' = P[re_d/rd]$ and $Q' = Q[re_d/rd]$. An application of lemma 6.37 gives us

$$\models \{ P' \} \rho' \{ Q' \}$$

(remember that $\rho$ is assumed to be restricted). By lemma 6.36 we have

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

Applying next the rule (ES) gives us

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

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

$$\vdash \{ lre_d \equiv re_d \} \rho \{ lre_d \equiv rd \}.$$

So by an application of the conjunction rule we have

$$\{ init \} \rho \{ SP_L(\rho, init) \} \vdash \{ P' \land lre_d \equiv re_d \} \rho \{ Q' \land lre_d \equiv re_d \}.$$

Now we have $\models \{ Q' \land lre_d \equiv re_d \} \vdash Q$. Furthermore for $P'' = P' \land lre_d \equiv re_d$ we have $\models P \vdash (\exists lre_d P'' \lor P''[nil/ld])$ (note that $lre_d \notin LVar(P)$). So first applying the consequence rule for $Q$, then the elimination rule (EE:1) (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 $\{ init \} \rho \{ SP_L(\rho, init) \}$, 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 $LVar(P^c, Q^c) \subseteq L \setminus (sI \cup lre)$, $st \cup sI \cup lre \subseteq L$. Then:

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

where $F_i = \{ init \} \rho_i \{ SP^c_L(\rho_i, init) \}$, $\rho_i = \langle U|c_i : v_i \leftarrow s_i \rangle$, 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 $v_i = re_d$, 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 $L^- = L \setminus (s \bar{I}f \cup l \bar{r}e)$. We have by lemma 6.22

$$\models \left\{ P^c \right\} \rho_1 \left\{ SP_L-(\rho_1, P^c) \right\},$$

and

$$= \left\{ SP_L-(\rho_1, P^c) \right\} \rho_2 \left\{ SP_L-(\rho_2, SP_L-(\rho_1, P^c)) \right\},$$

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

$$F_1, \ldots, F_n \models \left\{ P^c \right\} \rho_1 \left\{ SP_L-(\rho_1, P^c) \right\},$$

and

$$F_1, \ldots, F_n \models \left\{ SP_L-(\rho_1, P^c) \right\} \rho_2 \left\{ SP_L-(\rho_2, SP_L-(\rho_1, P^c)) \right\}.$$ 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 $\text{OK}(\sigma, \delta, \omega)$. By theorem 6.21 there exist a state $\sigma_0$ and an osi $f$ such that

- $f(\sigma) = \mathcal{P}[\rho_2](\gamma)(\delta')(\sigma_0)$, $\gamma$ arbitrary,

- $\sigma_0, \delta', \omega' \models SP_L-(\rho_1, P^c),$

where $\delta' = f(\delta)$ and $\omega' = f(\omega) \upharpoonright L^-$. Now $\sigma_0, \delta', \omega' \models SP_L-(\rho_1, P^c)$ in turn implies that there exist a state $\sigma'_0$ and an osi $g$ such that

- $g(\sigma_0) = \mathcal{P}[\rho_1](\gamma)(\delta'')(\sigma'_0)$, $\gamma$ arbitrary,

- $\sigma'_0, \delta'', \omega'' \models P^c,$
where $\delta'' = g(\delta')$ and $\omega'' = g(\omega') \upharpoonright L^-$.

To relate these computations of $\rho_1$ and $\rho_2$ 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 $\gamma$ is arbitrary.

Since $g(\delta') = \delta''$ it follows that $h(f(\sigma)) = \mathcal{P}[\rho](\gamma)(\delta'')(\sigma_0)$, with $\gamma$ 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 $\mathcal{O}K(\sigma_0, \delta')$. So we have $h(\delta'_{(1)}) = g^c(\delta'_{(1)}) = \delta''_{(1)}$ and $h(\delta'_{(2)}) = g^c(\delta'_{(2)}) = \delta''_{(2)}$, 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 $\mathcal{O}K(\sigma_0, \delta', \omega')$. So by theorem 6.19 and the fact that $LVar(Q^c) \subseteq L^-$ we conclude $\sigma, \delta, \omega \models Q^c$.

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

$S = \text{while } e \text{ do } S_1 \text{ od:}$

In order to deal with this case we construct a loop invariant $R$ as follows. Let $L^- = L \setminus (sI \cup \beta e)$ 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 $\rho' = \{Uc : e \land u < u' \text{ do } S_1; u \leftarrow u + 1 \text{ od}\}$. Furthermore let $R' = SP_{L^+}(\rho', P' \land u = 0)$ and define $R = \exists z P'[z, z'/u, u']$, where $z \in LVar_{L^+}$ is a new variable. Note that $LVar(R) \subseteq L^+$. Furthermore we have $\models \{P^c\}\rho \{Q^c\}$ (note that $u, u' \notin TVar(\rho)$, $\rho$ being restricted).

We have $\models P' \rightarrow R$:

Let $\sigma, \delta, \omega \models P'$, with $\mathcal{O}K(\sigma, \delta, \omega)$. We prove that for $\omega' = \omega\{0/z\}$ we have $\sigma', \delta, \omega' \models P'[z, z'/u, u']$. 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 $\sigma', \delta, \omega \models P' \land u = 0$. Furthermore it is easy to see that $\sigma' = \mathcal{P}[\rho'][\gamma](\delta')(\sigma')$, with $\gamma$ arbitrary. Finally, as $LVar(P') \subseteq L^+$ we have by theorem 6.21 $\sigma', \delta, \omega \models R'$.

Next we prove $\models R \land \lnot e \rightarrow Q'$:

Let $\sigma, \delta, \omega \models R \land \lnot e$. So let $\alpha \in \mathbb{N}$ such that $\sigma', \delta, \omega = R'$, where $\sigma' = \sigma\{\alpha, \alpha/u, u'\}$. So there exist $f, \sigma_0$ such that

- $f(\sigma') = \mathcal{P}[\rho'][\gamma](\sigma_0)$, where $\gamma$ arbitrary,

- $\sigma_0, \delta', \omega' \models P' \land u = 0$,

where $\delta' = f(\delta)$ and $\omega' = f(\omega) \upharpoonright L^+$. Now $u, u' \notin TVar(e)$ so $\sigma, \delta, \omega \models \lnot e$ implies $\sigma', \delta, \omega = \lnot e$. By theorem 6.19 we have $f(\sigma'), \delta', f(\omega) \models \lnot e$. So from $LVar(e) = 0$
it follows that \( f(\sigma'), \delta', \omega' \models \neg e \). From this it is not difficult to derive that \( f(\sigma') = \mathcal{P}[\rho](\gamma)(\delta')(\sigma'_0) \), 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 \( \models \{ P' \} \rho \{ Q' \} \) 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 \( \models \{ R \land e \} \rho_1 \{ R \} \), where \( \rho_1 = (U | c : S_1) \):

Let \( \sigma, \delta, \omega \models R \land e \), with \( O K(\sigma_0, \delta, \omega) \), and \( \sigma_1 = \mathcal{P}[\rho_1]((\gamma)\delta)(\sigma_0) \), with \( \gamma \) arbitrary.
Let \( \alpha \in N \) such that \( \sigma'_0, \delta, \omega \models R' \), where \( \sigma'_0 = \sigma_0\{\alpha, \alpha/u, u'\} \). So there exist \( f, \sigma \) such that

- \( f(\sigma'_0) = \mathcal{P}[\rho']((\gamma)\delta')(\sigma), \gamma \) arbitrary,
- \( \sigma, \delta', \omega' \models P' \land u \geq 0 \),

where \( \delta' = f(\delta) \) and \( \omega' = f(\omega) \). So we have the following situation:

\[
\begin{array}{c}
\sigma_0, \delta \xrightarrow{\rho_1} \sigma_1 \\
\mid \\
\sigma'_0 \\
\mid \\
\sigma, \delta' \xrightarrow{\rho'_1} f(\sigma'_0)
\end{array}
\]

Here \( \sigma, \delta \xrightarrow{\rho_1} \sigma' \) should be interpreted as \( \sigma' = \mathcal{P}[\rho((\gamma)\delta)(\sigma)], \gamma \) 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 \( \sigma'_1 = \mathcal{P}[\rho_1][((\gamma)\delta)(\sigma'_0)] \). We now have the following situation:

\[
\begin{array}{c}
\sigma_0, \delta \xrightarrow{\rho_1} \sigma_1 \\
\mid \\
\sigma'_0, \delta \xrightarrow{\rho_1} \sigma'_1 \\
\mid \\
\sigma, \delta' \xrightarrow{\rho'_1} f(\sigma'_0)
\end{array}
\]

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

\[
\begin{align*}
\sigma_0, \delta & \xrightarrow{\rho_1} \sigma_1 \\
\sigma_0', \delta & \xrightarrow{\rho_1} \sigma_1' \\
\sigma, \delta' & \xrightarrow{\rho_2} f(\sigma_0'), \delta' & \xrightarrow{\rho_1} g(\sigma_1')
\end{align*}
\]

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 \( \sigma_3 = P[\rho'][(\gamma)(\delta')(\sigma_2)] \), with \( \gamma \) 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 5.22 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 \( g(\sigma_1'), \delta', g(\delta') \models R \). 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 \( \models \{ R \wedge \epsilon \} \rho_1 \{ R \} \) it follows that \( \models \{ R'' \wedge \epsilon \} \rho_1 \{ R'' \} \) (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:

\[
F_1, \ldots, F_n \vdash \{ R'' \wedge \epsilon \} \rho_1 \{ R'' \}.
\]

By theorem 6.33 we have

\[
\models \{ z_u \vdash u \wedge z_u' \vdash u' \} \rho_1 \{ z_u \vdash u \wedge z_u' \vdash u' \}.
\]

Furthermore we have \( \models (R'' \wedge z_u \vdash u \wedge z_u' \vdash u') \rightarrow R \) and \( R \rightarrow (R'' \wedge z_u \vdash u \wedge z_u' \vdash u') \) (note that \( u, u' \notin TVar(R) \)). So applying the conjunction rule, the consequence rule for the postcondition, the initialization rule (IR:2), and the consequence rule for the precondition gives us

\[
F_1, \ldots, F_n \vdash \{ R \wedge \epsilon \} \rho_1 \{ R \}.
\]

From an application of the rule (W) and the consequence rule, using the truth of the implications \( P' \rightarrow R \) and \( R \wedge \neg \epsilon \rightarrow Q' \), 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

\[
F_1, \ldots, F_n \vdash \{ P' \wedge z_u \vdash u \wedge z_u' \vdash u' \} \rho \{ Q' \wedge z_u \vdash u \wedge z_u' \vdash u' \}.
\]

We have \( \models (Q' \wedge z_u \vdash u \wedge z_u' \vdash u') \rightarrow Q \) and \( \models P \rightarrow (P' \wedge z_u \vdash u \wedge z_u' \vdash u') \). 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. \( \Box \)
6.6 The context switch

In this subsection we prove the derivability of the correctness formula \( \{ \text{init} \} \rho \{ S^P_L(\rho, \text{init}) \} \), where \( \rho = (U|c : v \leftarrow e'_0!m'(e'_1, \ldots, e'_n)) \) closed and \( \text{BVar} \subseteq L \subseteq \text{LVar} \) such that \( \text{st} \cup \text{sI} \cup \text{lre} \subseteq L \). From now on until the end of this section unless stated otherwise we assume \( \rho \) 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 \( \rho' = (U|c' : v' \leftarrow e'_0!m'(e'_1, \ldots, e'_n)) \in M \) then \( \rho_i = (U|c_i : v_i \leftarrow e'_0!m_i(e'_{i,1}, \ldots, e'_{i,n})) \in M \),
- where \( v_i \leftarrow e'_0!m_i(e'_{i,1}, \ldots, e'_{i,n}) \) or \( e'_0!m_i(e'_{i,1}, \ldots, e'_{i,n}) \) occurs in \( S' \) as a statement (in this latter case we have \( v_i = \text{re}_{d_i} \), 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: \( \rho_i = (U|c_i : v_i \leftarrow e'_0!m_i(e'_{i,1}, \ldots, e'_{i,n})) \in M \), and \( m_i(u'_{i,1}, \ldots, u'_{i,n}) \leftarrow S_i \upharpoonright e_i \) occurs in \( U \), \( i = 1, \ldots, k \). We let \( \bar{e}^i \) denote the sequence \( e'_0, \ldots, e'_{i} \). 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 \( \text{LVar}(P_i, Q_i) \subseteq L \setminus (\text{sI} \cup \text{lre}) \), and

\[
\models \{ P_i \land \bigwedge_j v_j \uparrow = \text{nil} \land \text{self} \not\in b_{c'_i}(U|c'_i : S_i) \} \{ Q_i[e_i/r_i] \}, \tag{6.1}
\]

where \( \bar{v}^i = \bar{u} \setminus \bar{u}^i \) and \( c'_i \) is the type of \( e'_0 \),

\[
\models \text{init} \rightarrow P_i[\bar{e}^i/\text{self}, \bar{u}^i][\bar{g}^i/\bar{z}^i][b_{c_i} \circ (\text{self})/b_{c_i}], \tag{6.2}
\]

and

\[
\models Q_i[\bar{e}^i/\text{self}, \bar{u}^i][\bar{g}^i/\bar{z}^i][b_{c_i} \circ (\text{self})/b_{c_i}] \rightarrow S^P_L(\rho_i, \text{init})[r_i/u_i], \tag{6.3}
\]

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

\[
F'_1, \ldots, F'_k \vdash \{ P_i \land \bigwedge_j v_j \uparrow = \text{nil} \land \text{self} \not\in b_{c'_i}(U|c'_i : S_i) \} \{ Q_i[e_i/r_i] \}
\]

where
\[ F'_i = \{ \text{init} \} \rho_i \{ SP_L^e(\rho_i, \text{init}) \}. \]

Furthermore by an application of the consequence rule, using (6.2), we have \( F_i \vdash F'_i \)
where
\[ F_i = \left\{ P[\tilde{e}^i/\text{self}, \tilde{u}^i][\tilde{g}^i/\tilde{z}^i][b_{c_i} \circ \langle \text{self} \rangle/b_{c_i}] \right\} \rho_i \left\{ SP_L^e(\rho_i, \text{init}) \right\}. \]

So we have
\[ F_1, \ldots, F_k \vdash \left\{ P_i \land \bigwedge_j v_j \equiv \text{nil} \land \text{self} \not\in b_{c_i} \right\} (U|e'_i : S_i) \left\{ Q_i : e_i/r_i \right\}. \]

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 \( u \in \tilde{u} \) a new logical variable \( tv^u \) of the same type
and with each \( c \in C \) a new logical variable \( id^c \). We define \( \tilde{tv} \) to be
the sequence of logical variables \( tv^u \) corresponding to the sequence \( \tilde{u} \). Finally let \( \tilde{id}^i, i = 1, \ldots, k, \)
denote the sequence consisting of the variable \( id^c_i \), followed by the elements of \( \tilde{tv} \).

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

**Lemma 6.43**
For any \( i \in \{1, \ldots, k\} \) and every assertion \( P \in Ass^e \), we have
\[ P^{ci}[\tilde{id}^i/\text{self}, \tilde{u}][\tilde{e}^i/\text{self}, \tilde{u}^i] = P^{ci}[\tilde{id}^i/\text{self}, \tilde{u}]. \]

**Proof**
Straightforward induction on the complexity of \( P^{ci} \).

Note that the substitution \([\tilde{id}^i/\text{self}, \tilde{u}]\) transforms the assertion \( P^{ci} \) into an assertion
in \( Ass^e \) for arbitrary \( c \). Furthermore it is easy to see that if \( LVar(P) \cap \tilde{id}^i = \emptyset \)
then \( \models P^{ci} \rightarrow P^{ci}[\tilde{id}^i/\text{self}, \tilde{u}][\tilde{f}/\tilde{id}^i] \), where \( \tilde{f} \) denotes
the sequence consisting of the expression \( \text{self} \) followed by the elements of \( \tilde{u} \). Note that in general
we do not have that \( P^{ci} \) is syntactically equal to \( P^{ci}[\tilde{id}^i/\text{self}, \tilde{u}][\tilde{f}/\tilde{id}^i] \), as is shown
by the following example:
Example 6.44
Take for $P^{c_i} = z \vdash z.y$, where $z \notin \overline{\text{id}}^\dagger$. We have $P^{c_i}[^\text{id}^\dagger/\text{self}, \overline{u}] = \overline{\text{id}}_c, x \equiv z.y$ and $(\overline{\text{id}}_c, x \equiv z.y)[\overline{f}/\overline{\text{id}}^\dagger] = \text{self}.x \equiv z.y$.

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

$$\models [P^{c_i}[b_{c_i}/\text{self}] \land b_{c_i} = b_{c_i} \circ \langle \text{self} \rangle]][b_{c_i} \circ \langle \text{self} \rangle]/b_{c_i}] \rightarrow P^{c_i}.$$  

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

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

To summarize the argument above we introduce the following definition:

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

$$R(P^{c_i}) = (P^{c_i}[b_{c_i}/b_{c_i}] \land b_{c_i} = b_{c_i} \circ \langle \text{self} \rangle)[\overline{\text{id}}^\dagger/\text{self}, \overline{u}]$$

We have the following lemma about this reverse context switch:

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

$$\models R(P^{c_i})[^i/\text{self}, \overline{u}][\overline{f}/\overline{\text{id}}^\dagger][b_{c_i} \circ \langle \text{self} \rangle]/b_{c_i}] \rightarrow P^{c_i}.$$  

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

$$\models P^{c_i} \rightarrow R(P^{c_i})[^i/\text{self}, \overline{u}][\overline{f}/\overline{\text{id}}^\dagger][b_{c_i} \circ \langle \text{self} \rangle]/b_{c_i}].$$

Here $\overline{f} = \text{self}, \overline{u}$.

Proof
Clear from the above.

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

$$\models \{R(P) \land \bigwedge_j v_j \equiv \text{nil} \land \text{self} \notin b_{c_j}\} \{U|c_i : S_i\} \{R(Q') | v_i/r_i\},$$
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 \( R(\) the information \( \text{self} \models e_j^{i}[\tilde{id}^i/\text{self}, \tilde{u}] \) and \( u_j^{i} \models e_j^{i}[\tilde{id}^i/\text{self}, \tilde{u}] \), \( j = 1, \ldots, n_i \). We have the following lemma:

**Lemma 6.47**

\[ \models (f_j^{i} \models (e_j^{i}[\tilde{id}^i/\text{self}, \tilde{u}]))(\tilde{e}^i/\text{self}, \tilde{u}][\tilde{f}/\tilde{id}^i] \]

where \( \tilde{f} = \text{self}, \tilde{u}^i \) and \( \tilde{f} = \text{self}, \tilde{u} \).

**Proof**

Easy. \( \square \)

Note that from lemma 6.46 and lemma 6.47 it follows that for every \( P^{c_i} \) such that \( \models P^{c_i} \models b_{c_i} \models b_{c_i} \), we have

\[ \models P \models R(\) \wedge \bigwedge_j f_j^{i} \models (e_j^{i}[\tilde{id}^i/\text{self}, \tilde{u}])(\tilde{e}^i/\text{self}, \tilde{u}][\tilde{f}/\tilde{id}^i][b_{c_i} \wedge (\text{self})/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 \( P, Q \in \text{Ass}^{c_i} \) such that \( \models \{ P \} \rho_i \{ Q \} \) we have

\[ \models \{ P' \wedge \bigwedge_j v_j^{i} \models \text{nil} \wedge \text{self} \not\models b_{c_i} \}(U|c_i : S_i)\{ Q'[\tilde{e}^i/r_i] \}, \]

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

**Proof**

Let \( \sigma, \delta, \omega \models P' \wedge \bigwedge_j v_j^{i} \models \text{nil} \wedge \text{self} \not\models b_{c_i} \), for \( \sigma, \delta, \omega \) such that \( OK(\sigma, \delta, \omega) \), and \( \sigma' = P|U|c_i : S_i|)(\gamma)(\delta)(\sigma) \), with \( \gamma \) arbitrary and \( \sigma' \not\models \bot \).

We define \( \sigma_1 = \sigma\{\omega(tu_2/u)/u\}_{u \in \mathbb{A}} \) and \( \delta_1^{(c)} = \omega(id_{c_1}) \), \( \delta_2^{(c)} = \delta_2^{(c)} \) for every \( c \).

It follows from lemma 5.28 that \( \sigma_1, \delta', \omega \models P[b_{c_i}/b_{c_i}] \wedge b_{c_i} \models b_{c_i} \) (self).

Next we define \( \delta'' \) as follows: \( \delta''_1 = \delta_1^{(1)} \), \( \delta''_2^{(c)} = \delta_2^{(c)} \setminus \{\omega(id_{c_1})\} \), and \( \delta''_2^{(c)} = \delta_2^{(c)} \) for any \( c \neq c_1 \). Furthermore we put \( \omega_1 = \omega(\omega(b_{c_i})/b_{c_i}) \). It then follows that \( \omega_1 \models P \).
\[ \sigma, \delta, \omega \models P' \quad \Downarrow \quad \sigma', \delta, \omega \models Q'[e_i/r_i] \]

\[
\sigma', \delta', \omega_2 \models P'[b_{c_i}/b_{c_i}] \land b_{c_i} = b_{c_i} \circ (\text{self}) \quad \Downarrow \quad \sigma'', \delta', \omega_3 \models Q'[b_{c_i}, r_i/b_{c_i}, v_i] \land b_{c_i} = b_{c_i} \circ (\text{self}) \quad \Uparrow \quad \sigma_2, \delta'', \omega_1 \models P \quad \Rightarrow \quad \sigma_2, \delta'', \omega_1 \models Q
\]

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

Let on the other hand \( \sigma'' = \sigma'[\omega(tv2u)/u] \) for 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 \( \beta = E[e_i](\delta'(\sigma')) \) (remember that \( e_i \) is the resultant expression of the method \( m_i \)). Now from \( \sigma, \delta, \omega \models \land_j f_j^* = (e'[id'/self, \bar{u}]) \) it follows from lemma 5.28 that:

\[ \delta_{(1)} = L[e_i][\delta'(\sigma)](\omega)(\delta)(\sigma) = L[e_i][\delta''(\sigma)](\omega)(\delta'')(\sigma_1) \]

and \( \sigma_{(3)}(u_j^i) = L[e_i][\delta''(\sigma)](\omega)(\delta'')(\sigma_1) \).

Furthermore from \( \sigma, \delta, \omega \models \omega \not\models b_{c_i} \) it in turn follows that \( \delta_{(1)} \not\models \delta_{(2)}(\sigma') \). Now putting this together with the assumption that \( \sigma' = P'[U^c_i : S_i](\gamma)(\delta)(\sigma) \), using \( \sigma, \delta, \omega \models \land_j v_j^i \models \text{nil} \), enables one to infer that \( \sigma_2 = P[\rho_i][\gamma](\delta''(\sigma_1)) \).

Furthermore we are given that \( \models \{ P \rho_i \} \gamma \{ Q \} \) so from \( \sigma_1, \delta'', \omega_1 \models P \) and \( \sigma_2 = P[\rho_i][\gamma](\delta'')(\sigma_1) \) 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 \( \omega_2(b_{c_i}) = \omega_1(b_{c_i}) = \omega(b_{c_i}) \) we have \( \sigma'', \delta'', \omega_3 \models Q[r_i, b_{c_i}/v_i, b_{c_i}] \) where \( \omega_3 = \omega(\beta/r_i) \).

From \( \sigma, \delta, \omega \models R(P) \) we infer that \( \omega(b_{c_i}) = \omega(\beta(r_{c_i}) \circ (\omega(id_{c_i})) \). But \( \omega(\beta) = \delta'_{(1)} \) so we have \( \sigma'', \delta', \omega_3 \models Q[r_i, b_{c_i}/v_i, b_{c_i}] \land b_{c_i} = b_{c_i} \circ (\text{self}) \).

Now an application of lemma 5.28 gives us \( \sigma', \delta, \omega_3 \models R(Q[r_i/v_i]) \). 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 \text{init} for \( P \) and \( SP^t_L(p_i, \text{init}) \) for \( Q \). Note that by lemma 6.22 we have \( \models \{ \text{init} \} p_i \{ SP^t_L(p_i, \text{init}) \} \). Now taking for \( P_i \) the assertion \( R(\text{init}) \land \land_j f_j^* = (e'[\bar{z}/\text{self}, \bar{u}]) \) and for \( Q_i \) the assertion \( R(SP^t_L(p_i, \text{init})[r_i/v_i]) \) we have by lemma 6.46 and lemma 6.47 that equations (6.2) and (6.3) are satisfied. However since in the assertions \( P \) and \( Q \), new logical variables occur which are not contained in \( L \), we must apply theorem 6.40 for \( F_i = \{ \text{init} \} p_i \{ SP^t_L(p_i, \text{init}) \} \), where \( L^+ = L \cup \{ id_c : c \in C \} \cup \{ tv2u : u \in \bar{u} \} \). But to apply the rule \{\text{NMR}\} we then have to take for \( Q_i \) the assertion \( R(SP^t_L(p_i, \text{init})[r_i/v_i]) \). An application
of (NMR) and (MI) or (MT) would then give us the derivability of the correctness formula \( \{ \text{init} \} = SP_{L^+}^\rho (\rho, \text{init}) \). However, as \( \models SP_{L^+}^\rho (\rho, \text{init}) \rightarrow SP_{L}^\rho (\rho, \text{init}) \) (use LVar(\text{init}) \subseteq L \subseteq L^+), we have by an application of the consequence rule the derivability of \( \{ \text{init} \} = SP_{L}^\rho (\rho, \text{init}) \).

But there is one problem we did not discuss yet. As \( s \bar{I} \cup \bar{I} \subseteq L \text{Var}(SP_{L^+}^\rho (\rho, \text{init})) \) we cannot apply theorem 6.40! This problem is solved as follows: First we define \( L^- = L^+ \setminus (s \bar{I} \cup \bar{I}) \). Next we define the following abbreviation:

**Definition 6.49**
Let \( \text{Subs}(\bar{I} \bar{e}, s \bar{I} \bar{I}, \bar{c} \bar{r}) \) abbreviate the assertion:

\[
\bigwedge \ e (cr_\bar{e} \subseteq cr_\bar{e} \land bl_\bar{e} \subseteq cr_\bar{e} \land \bar{I}re_\bar{e} \in cr_\bar{e} \land \bigwedge_{d \in C} \bigwedge_{x \in lVar_d} iuv_x \subseteq cr_d \land \bigwedge_{u \in TVar} tuv_u \in cr_u).
\]

The assertion \( \text{Subs}(\bar{I} \bar{e}, s \bar{I} \bar{I}, \bar{c} \bar{r}) \) states that all the objects which are denoted by a variable of \( \bar{I} \bar{e} \) or \( s \bar{I} \bar{I} \), or which occur in a sequence denoted by some variable of \( s \bar{I} \bar{I} \), are stored in the corresponding variable of \( \bar{c} \bar{r} \). We have the following proposition:

**Proposition 6.50**
Let \( P_i = R(\text{init}) \land \bigwedge_{j} \bigwedge_{x} (f_{i_j} = \langle c_j \mid id^j / \text{self}, \bar{u} \rangle) \), \( Q^-_i = R(SP_{L^-}^\rho (\rho, \text{init})[r_i / v_i]) \) and \( Q^+_i = R(SP_{L^+}^\rho (\rho, \text{init})[r_i / v_i]) \). We have

\[
\models P_i \land \text{Subs}(\bar{I} \bar{e}, s \bar{I} \bar{I}, \bar{c} \bar{r}) \iff P_i
\]

and

\[
\models Q^-_i [c_i / r_i] \land \text{Subs}(\bar{I} \bar{e}, s \bar{I} \bar{I}, \bar{c} \bar{r}) \rightarrow Q^+_i [c_i / r_i].
\]

**Proof**
The first assertion follows immediately from the fact that the assertion \( \text{init} \) (and so the assertion \( R(\text{init}) \)) implies the assertion \( \forall z \in cr_z \), for every \( c \).

Now we prove the second assertion. Let \( \sigma_1, \sigma, \omega \models Q^-_i [c_i / r_i] \land \text{Subs}(\bar{I} \bar{e}, s \bar{I} \bar{I}, \bar{c} \bar{r}) \). For \( \omega_1 = \omega \{ E[c_i] (\delta) (\sigma) / r_i \} \), we then have \( \sigma, \delta, \omega_1 \models Q^-_i \land \text{Subs}(\bar{I} \bar{e}, s \bar{I} \bar{I}, \bar{c} \bar{r}) \).

Next we define \( \sigma' = \sigma [\omega_1 (tv_{2u}) / u]_{u \in U} \), and \( \delta'_1 = \omega_1 (id_{c_i}) \), \( \delta'_{3(c)} = \delta_{3(c)} \), for every \( c \). It then follows by lemma 5.28 that: \( \sigma', \delta', \omega_1 \models SP_{L^-}^\rho (\rho, \text{init})[r_i, bl_{c_i} / v_i, b_{c_i}] \land \bigwedge_{b_{c_i} = bl_{c_i} \circ (\text{self}) \land \text{Subs}(\bar{I} \bar{e}, s \bar{I} \bar{I}, \bar{c} \bar{r}).
\]

For \( \omega_2 = \omega_1 (bl_{c_i} / b_{c_i}) \) 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 \( \sigma', \delta'', \omega_2 \models SP_{L^-}^\rho (\rho, \text{init})[r_i / v_i] \land \text{Subs}(\bar{I} \bar{e}, s \bar{I} \bar{I}, \bar{c} \bar{r}) \).
Next, let
\[ \sigma'' = \begin{cases} 
\sigma' \omega_2(r_i)/\delta'_{(1)}, \nu_i \} & \text{if } \nu_i \in \text{IVar} \\
\sigma' \omega_2(r_i)/\nu_i \} & \text{if } \nu_i \in \text{TVar}.
\end{cases} \]
It follows that \( \sigma'', \delta'', \omega_2 \models SP_{L-}^\omega (\rho_i, \text{init}) \land \text{Subs}(\text{ire}, \text{s11}, \text{cr}). \)

So by theorem 6.21 there exist \( f \) and \( \sigma_0 \) such that:

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

where \( \omega' = f(\omega_2) \sqsubseteq L^- \). Let \( \sigma_1 = f(\sigma'') \). Now by theorem 6.19 we have that \( \sigma_1, f(\delta''), f(\omega_2) \models \text{Subs}(\text{ire}, \text{s11}, \text{cr}) \). So from \( \{ \text{cr}_c : c \in C \} \subseteq L^- \) and the compatibility of \( \omega' \) and \( \sigma_0 \) we then infer the compatibility of \( f(\omega_2) \sqsubseteq L^+ \) and \( \sigma_0 \). Let \( \omega'' = f(\omega_2) \sqsubseteq L^+ \). We have that \( \sigma_0, f(\delta''), \omega'' \models \text{init} \), so we have \( \sigma'', \delta'', \omega_2 \models SP_{L-}^\omega (\rho_i, \text{init}) \).

From this it follows, by "reversing" the part of the above argument which led to the statement \( \sigma'', \delta'', \omega_2 \models SP_{L-}^\omega (\rho_i, \text{init}) \), that \( \sigma, \delta, \omega \models Q_+^\omega [e_i/r_i] \). □

Now we are ready for the following theorem.

**Theorem 6.51**

Let the program \( \rho = (U|c : v \leftarrow e_0 ! m(e_1, \ldots, e_n)) \) be closed and let \( \text{BVar} \subseteq L \subseteq \text{LVar} \) such that \( \text{sl} \cup \text{s11} \cup \text{ire} \subseteq L \). Then we have

\[ \models \{ \text{init} \} \rho \{ \text{SP}_{L}^\omega (\rho, \text{init}) \}. \]

**Proof**

Let \( P_i = R(\text{init}) \land \bigwedge_j f_j^1 = (e_i^1 [\tilde{\nu}^i/\text{self}, \tilde{u}]) \), \( Q_i^- = R(\text{SP}_{L-}^\omega (\rho_i, \text{init})(r_i/v)) \) and \( Q_i^+ = R(\text{SP}_{L+}^\omega (\rho_i, \text{init})(r_i/v)) \). Now by lemma 6.22 we get

\[ \models \{ \text{init} \} \rho_i \{ \text{SP}_{L-}^\omega (\rho_i, \text{init}) \}. \]

So we have, by lemma 6.48,

\[ \models \{ P_i \land \bigwedge_j v_j^i = \text{nil} \land \text{self } \notin b_{c_i^1} \} \{ U|c_i^1 : S_i \} \{ Q_i^- [e_i/r_i] \}. \]

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

\[ F'_1, \ldots, F'_n \models \{ P_i \land \bigwedge_j v_j^i = \text{nil} \land \text{self } \notin b_{c_i^1} \} \{ U|c_i^1 : S_i \} \{ Q_i^- [e_i/r_i] \}, \]
where
\[ F'_i = \{ \text{init} \} \rho_i \{ SP_{L+}^c (\rho_i, \text{init}) \}. \]

Now by lemma 6.46 and lemma 6.47 an application of the consequence rule gives us
\[ F_i \vdash F'_i \] where
\[ F_i = \{ P_i[e^i / \text{self}, \tilde{u}^i][\tilde{f} / \tilde{v}^i][b_{c_i} \circ \langle \text{self} \rangle / b_{e_i}] \} \rho_i \{ SP_{L+}^c (\rho_i, \text{init}) \}. \]

So we have
\[ F_1, \ldots, F_k \vdash \{ P_i \land \bigwedge_j v_j \stackrel{\text{nil}}{=} \text{self} \land \neg b_{c_i} \} \langle U | c'_i : S_i \rangle \{ Q_i^+ [e_i / r_i] \}. \]

By theorem 6.33 we have
\[ \vdash \{ \text{Subs}(\tilde{r} \tilde{e}, s \tilde{I}, \tilde{c} \tilde{r}) \} \langle U | c'_i : S_i \rangle \{ \text{Subs}(\tilde{r} \tilde{e}, s \tilde{I} \tilde{c}, \tilde{c} \tilde{r}) \}. \]

So by the conjunction rule we infer
\[ F_1, \ldots, F_m \vdash \{ P_i \land \bigwedge_j v_j \stackrel{\text{nil}}{=} \text{self} \land \neg b_{c_i} \land \text{Subs}(\tilde{r} \tilde{e}, s \tilde{I}, \tilde{c} \tilde{r}) \} \langle U | c'_i : S_i \rangle \{ Q_i^+ [e_i / r_i] \}. \]

By proposition 6.50 an application of the consequence rule gives us
\[ F_1, \ldots, F_m \vdash \{ P_i \land \bigwedge_j v_j \stackrel{\text{nil}}{=} \text{self} \land \neg b_{c_i} \} \langle U | c'_i : S_i \rangle \{ Q_i^+ [e_i / r_i] \}. \]

We now can apply rule (NMR), making use of lemma 6.46, yielding the derivability of the correctness formula:
\[ \{ P_i \land \bigwedge_j v_j \stackrel{\text{nil}}{=} \text{self} \land \neg b_{c_i} \} \langle U | c'_i : S_i \rangle \{ Q_i^+ [e_i / r_i] \}. \]

Applying next (MI) or (MT) gives us the derivability of
\[ \{ P_i[e^i / \text{self}, \tilde{u}^i][\tilde{f} / \tilde{z}^i][b_{c_i} \circ \langle \text{self} \rangle / b_{e_i}] \} \rho_1 \{ SP_{L+} (\rho_1, \text{init}) \}. \]

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

We conclude with the completeness theorem:
Theorem 6.52
Let $\rho' = \langle U | c : S \rangle$ be a closed program. We have for an arbitrary correctness formula
$$\{ P^c \} \rho'^c \{ Q^c \} :$$
$$\models \{ P^c \} \rho^c \{ Q^c \} \text{ implies } \vdash \{ P^c \} \rho^c \{ Q^c \}.$$

Proof
Let $P'$ and $Q'$ result from substituting for every variable of $st$ and $lre$ a corresponding new variable (new with respect to the sets $LVar(P^c, Q^c), st, stl, lre$). Let $L \subseteq LVar$ ($L$ finite) be such that $BVar \subseteq L, LVar(P', Q') \subseteq L$ and $st \cup stl \cup lre \subseteq L$. By the soundness of the substitution rule we have $\models \{ P' \} \rho^c \{ Q' \}$, so applying theorem 6.49 gives us
$$F_1, \ldots, F_n \vdash \{ P' \} \rho^c \{ Q' \},$$
where $F_i = \{ \text{init}\} \rho_i \{ SP^c_i'(\rho_i, \text{init}) \}, \rho_i = \langle U | c_i : v_i \gets e_0^i \! / \! m_i(e_1^i, \ldots, e_n^i) \rangle$ and $e_0^i \! / \! m_i(e_1^i, \ldots, e_n^i), i = 1, \ldots, n$, are all the send-expressions occurring in $S$, and if such an expression $e_0^i \! / \! m_i(e_1^i, \ldots, e_n^i)$ occurs in $S$ as a statement we have that $v_i = r_{e_d^i}$, 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 $\vdash \{ P' \} \rho^c \{ Q' \}$. Finally an application of the substitution rule gives us the derivability of $\{ P^c \} \rho^c \{ Q^c \}$.  \qed
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


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 \( (\text{NMR}) \) denote the recursion rule resulting from (MR) by replacing all occurrences of \( U^- \) by \( U \). Furthermore let \( \vdash \) denote the derivability using \( (\text{NMR}) \) \( \vdash \) 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):

\[
\begin{array}{c}
\frac{\vec{F}_1, \ldots, \vec{F}_n \quad F_1, \ldots, F_n \vdash F_1', \ldots, F_n'}{F_1'}
\end{array}
\]

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 \( \rho_i = \{ \text{U} | c_i : v_i \leftarrow c_0^i \mid m_i(e_1^i, \ldots, e_n^i) \} \), be all the send statements occurring in \( U \). Now simply add to \( F_1, \ldots, F_n \) for \( i = 1, \ldots, k \), \( G_i = \{ \text{true} \} \rho_i \{ \text{true} \} \), and note that

\[
G_1, \ldots, G_k \vdash \{ \text{true} \} \langle U | c' : S_i \rangle \{ \text{true} \}
\]

where \( c'_i \) is the type of \( c_0^i \) 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 \( \vec{H}_1, \ldots, \vec{H}_k, H_1, \ldots, H_k, H_1', \ldots, H_k' \) such that for \( 1 \leq i \leq k \)

\[
\frac{H_1'}{H_i'}\frac{\vec{H}_1, \ldots, \vec{H}_k}{\vec{H}_i, \ldots, \vec{H}_k}
\]

is an instance of (MI) or (MT), we have:

\[
\frac{\vec{F}_1, \ldots, \vec{F}_n, \vec{H}_1, \ldots, \vec{H}_k \vdash \vec{F}_1', \ldots, \vec{F}_n', \vec{H}_1', \ldots, \vec{H}_k'}
\]

where, for \( G = \{ P \} \langle U | c : S \rangle \{ Q \}, \vec{G} \) 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 $\bar{H}_1, \ldots, \bar{H}_k$, $H_1, \ldots, H_k, H'_1, \ldots, H'_k$ we have

$$F_1, \ldots, F_n, \bar{H}_1, \ldots, \bar{H}_k \vdash \bar{G}, H'_1, \ldots, H'_k,$$

where for $i = 1, \ldots, k$

$$\frac{H'_i \quad \bar{H}_i}{\bar{H}_i}$$

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 $F_1, \ldots, F_n \vdash \bar{G}', \bar{G}$, where

$$\frac{G' \quad \bar{G}}{G}$$

is an instance of (MI) or (MT). Now by the induction hypothesis we know that for some $\bar{H}_1, \ldots, \bar{H}_k, H_1, \ldots, H_k, H'_1, \ldots, H'_k$:

$$F_1, \ldots, F_n, \bar{H}_1, \ldots, \bar{H}_k \vdash G', \bar{H}'_1, \ldots, \bar{H}'_k,$$

such that for $i = 1, \ldots, k$

$$\frac{H'_i \quad \bar{H}_i}{\bar{H}_i}$$

is an instance of (MI) or (MT). Now let $\bar{H}_{k+1} = \bar{G}$, $H_{k+1} = G$, and $H'_{k+1} = G'$. We then have that

$$F_1, \ldots, F_n, \bar{H}_1, \ldots, \bar{H}_{k+1} \vdash \bar{G}, \bar{H}'_1, \ldots, \bar{H}'_{k+1}.$$

**Induction step:** Let for $i = 1, \ldots, m$

$$\frac{\bar{G}'_1, \ldots, \bar{G}'_m}{\bar{G}'_i}$$

be all the applications of (NMR) in the derivation $F_1, \ldots, F_n \vdash F'_1, \ldots, F'_n$ such that

$$F_1, \ldots, F_n, G'_1, \ldots, G'_m \vdash F'_1, \ldots, F'_n.$$

By the same induction argument as used in the basis step above we have for some $\bar{H}_1, \ldots, \bar{H}_k$, $H_1, \ldots, H_k$, $H'_1, \ldots, H'_k$ (such that for $i = 1, \ldots, k$

$$\frac{H'_i \quad \bar{H}_i}{\bar{H}_i}$$

is an instance of (MI) or (MT)) that

$$F_1, \ldots, F_n, H_1, \ldots, H_k, G'_1, \ldots, G'_m \vdash F'_1, \ldots, F'_n, H'_1, \ldots, H'_k.$$
where $\vdash$ denotes derivability from $\vdash$ minus the rules (MR), (MI), and (MT). Now, applying the induction hypothesis gives us for $i = 1, \ldots, m$: $\bar{H}_i^1, \ldots, \bar{H}_i^{k_i}, \bar{H}_1^i, \ldots, \bar{H}_k^i$. $H_i^1, \ldots, H_i^{k_i}$ such that

$$G_1^i, \ldots, G_n^i, H_1^i, \ldots, H_k^i \vdash \bar{G}_1^i, \ldots, \bar{G}_n^i, \bar{H}_1^i, \ldots, \bar{H}_k^i.$$ 

Now it follows by a straightforward induction on the length of the derivation

$$F_1^i, \ldots, F_n^i, H_1^i, \ldots, H_k^i, G_1^i, \ldots, G_1^{m_i} \vdash F_1^i, \ldots, F_n^i, H_1^i, \ldots, H_k^i$$

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

- $\mathcal{F} = \{ F_1^i, \ldots, F_{n+k_i}^i \}$. $F_{n+i}^i = H_i$, $i = 1, \ldots, k$,
- $\mathcal{F}' = \{ F_1^i, \ldots, F_{n+k_i}^i \}$. $F_{n+i}^i = H_i^i$, $i = 1, \ldots, k$,
- $\mathcal{G}_i = \{ \bar{G}_1^i, \ldots, \bar{G}_m^i \}$, $1 \leq i \leq m$,
- $\mathcal{G}'_i = \{ \bar{G}_1^i, \ldots, \bar{G}_m^i \}$, $1 \leq i \leq m$,
- $\mathcal{H}_i = \{ \bar{H}_1^i, \ldots, \bar{H}_k^i \}$, $1 \leq i \leq m$,
- $\mathcal{H}'_i = \{ \bar{H}_1^i, \ldots, \bar{H}_k^i \}$, $1 \leq i \leq m$.

$\vdash$: This is proved in a similar way as the other direction. $\Box$
B Expressibility

In this section we show how to formulate the assertion $SP_L^c(\rho^c, P^c)$ in our assertion language, for an arbitrary closed program $\rho^c$, $BVar \subseteq L \subseteq LVar$ ($L$ 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 $v \in IVar$ we have $[v] \in N$, and for an arbitrary program $\rho$ we have $[\rho] \in N$.
- For every $d \in C^+$, $[.]_d : O^d \to N$ denotes an injection such that $[\bot]_d = 0$. In addition, we assume that the function $[.]_{Int}$ is surjective.
- For every state $\sigma \in \Sigma$ such that $O\kappa(\sigma), [\sigma] \in N$.
- For every context $\delta^c \in \Delta^c$; $[\delta^c] \in N$.

Furthermore we assume that the mappings $[.]_{Int}$ and $[.]_{Bool}$ 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 $[n]_{Int} = m$.
- $Bc(b) = m$ (mnemonic: Boolean coding) iff $[b]_{Bool} = m$.
- $Id(n) = m$ (mnemonic: integer decoding) iff $[m]_{Int} = n$.

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

$$\sigma, \delta, \omega \models Ic(z_1) = z_2 \quad \text{ iff } \quad [\omega(z_1)]_{Int} = \omega(z_2).$$

In fact from now on for every $c \in C$ and $\alpha \in O^c$ we identify $[\alpha]_c$ with $\alpha$. So we assume $O^c \subseteq N$.

In the same way we assume the following predicates and functions to be expressible in our assertion language:
- $E^c(n, m)$ (mnemonic: 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 $[\delta] = m$.

- $Val^c_\Sigma(k, l, m) = n$ (mnemonic: value) iff there exist $\sigma \in \Sigma$, $\alpha \in \sigma^{(c)}$, and $x_\delta \in IVar^c_\Delta$ such that $[\alpha]_c = k$, $[x_\delta] = l$, $[\sigma] = m$, and $[\sigma(\alpha)(x_\delta)]_d = n$.

- $Val_d(l, m) = n$ 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 $\gamma$ 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 $bij$ denote a sequence of some variables $bij^c \in IVar^c_c$, $c \in C$. 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 $bij$ can be used as a kind of decoding tables. To that end we assume that we
have a certain state $\sigma$ such that for every $c \in C$ and $\alpha \in O^c_\bot$

$$elt(\beta, [\alpha]_c) = \begin{cases} & \alpha \quad \text{if } \alpha \in \sigma^{(c)} \\ & \bot \quad \text{otherwise.} \end{cases}$$

where $\beta \in O^c$ is the value of $bij^c$ in a certain $\omega$. 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 $\sigma, \delta, \omega$ 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 $z^\sigma, z'^\sigma$ be some logical integer variables. We assume that the value of $z^\sigma$ equals the code $[\sigma]$ of some state $\sigma$, and that the value of $z'^\sigma$ equals $[\alpha]_c$ for some $\alpha \in \sigma^{(c)}$. For every logical expression $l^d_\partial$ we define $l^d_\partial[z^\sigma, z'^\sigma]$ as a triple $(\tilde{i}, l_1^\text{Bool}, l_2^\text{Int})$, where $\tilde{i}$ 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 $d^a$ with $d \in C^+$. The idea behind this transformation $l^d_\partial[z^\sigma, z'^\sigma] = (\tilde{i}, l_1, l_2)$ can be described as follows: The expression $l_1$ is constructed such that it is only true if the variables $\tilde{i}$ contain the code numbers of certain objects that are relevant for the evaluation of $l^d_\partial$. To do this, $l_1$ can consult the variables $\text{bij}$ as a translation table from code numbers to actual objects. Using this information, $l_2$ is a translation of $l^d_\partial$ such that every operation on objects described by $l^d_\partial$ is translated into a corresponding arithmetical operation on code numbers.

Here is the formal definition:

- $z^\sigma_\partial[z^\sigma, z'^\sigma] = (\epsilon, \text{true}, \text{Val}_d(z^\sigma, k, z'^\sigma))$, where $k = [z^\sigma_\partial]$ and $\epsilon$ is the empty sequence.

- $u_\partial[z^\sigma, z'^\sigma] = (\epsilon, \text{true}, \text{Val}_d(k, z'^\sigma))$, where $k = [u_\partial]$.

- $\text{nil}[z^\sigma, z'^\sigma] = (\epsilon, \text{true}, 0)$.

- $\text{self}[z^\sigma, z'^\sigma] = (\epsilon, \text{true}, z^\sigma)$.

- $l[z^\sigma, z'^\sigma] = (\epsilon, \text{true}, Bc(l))$, where $l = \text{true, false}$.

- $n[z^\sigma, z'^\sigma] = (\epsilon, \text{true}, [n]_{\text{int}})$. 
\[ z_{\text{Int}}[z^\alpha, z^\sigma] = (\epsilon, \text{true}, \text{Int}(z_{\text{Int}})) \]
\[ z_{\text{Bool}}[z^\alpha, z^\sigma] = (\epsilon, \text{true}, \text{BC}(z_{\text{Bool}})) \]
\[ z_c[z^\alpha, z^\sigma] = (\langle i \rangle, \text{if } z_c = \text{nil} \text{ then } i = 0 \text{ else } \text{bij}^c \cdot i = z_c \cdot fi, i) \]
\[ (\langle i \rangle_c \cdot z_d^c)[z^\alpha, z^\sigma] = (\tilde{i}, l_1, \text{Val}_d^c(l_2, k, z^\sigma)) \]
where \( k = [z_d^c] \) and \( \langle i \rangle_c [z^\alpha, z^\sigma] = (\tilde{i}, l_1, l_2) \).
\[ (l_{\text{Int}} \cdot l_{\text{Int}})[z^\alpha, z^\sigma] = (\tilde{i}, l_1, \text{Int}(l_{\text{Int}} \cdot \text{Id}(l_2))) \]
\[ (l_{\text{Bool}} \cdot l_{\text{Int}})[z^\alpha, z^\sigma] = (\tilde{i}, l_1, \text{BC}(l_{\text{Bool}} \cdot \text{Id}(l_2))) \]
where \( l_{\text{Int}}[z^\alpha, z^\sigma] = (\tilde{i}, l_1, l_2) \).
\[ (l_{d^*} \cdot l_{\text{Int}})[z^\alpha, z^\sigma] = (\tilde{i} \circ j, l_1 \land \text{if } l_{d^*} \cdot \text{Id}(l_2) \neq \text{nil} \text{ then } j = 0 \text{ else } \text{bij}^d \cdot j = l_{d^*} \cdot \text{Id}(l_2) \cdot fi, j) \]
where \( d \in C, l_{\text{Int}}[z^\alpha, z^\sigma] = (\tilde{i}, l_1, l_2) \) and \( j \) is a fresh integer logical variable.
\[ (l_{1} + l_{2})[z^\alpha, z^\sigma] = (\tilde{i}, l_{1} \land l_{1}', l_{2}' \land l_{2}', \text{Id}(l_{1}) + \text{Id}(l_{2}')) \]
where \( l_{1}[z^\alpha, z^\sigma] = (\tilde{i}, l_{1}, l_{1}', l_{1}'') \) and \( l_{2}[z^\alpha, z^\sigma] = (\tilde{i}, l_{2}, l_{2}', l_{2}'') \)
where \( \tilde{i} = i_1 \circ j_2 \circ j_3 \) and \( l_1, l_2, l_3, l_{1}', l_{1}' \) are sequences of new logical integer variables of the same length as \( l_1, l_2, l_3 \), respectively, such that \( i_1, j_2 \) and \( j_3 \) are mutually disjoint.
\[ (l_{1} \mid l_{2})[z^\alpha, z^\sigma] = (\tilde{i}, l_{1}, l_{1}' \mid l_{2}, l_{2}' \mid l_{2}') \]
where \( l_{1}[z^\alpha, z^\sigma] = (i_1, l_{1}, l_{1}', l_{1}'') \) and \( l_{2}[z^\alpha, z^\sigma] = (i_2, l_{2}, l_{2}', l_{2}'') \)
where \( \tilde{i} = i_1 \circ j_2 \circ j_3 \) and \( j_2 \) is some sequence of new logical integer variables of the same length as \( l_{1}, l_{2} \), respectively.

Next we define for every assertion \( P^c \) its transformation \( P^c[z^\alpha, z^\sigma] \).

\[ l_{\text{Bool}}[z^\alpha, z^\sigma] = \exists i(l_1 \land l_2 \neq \text{BC}(\text{true})) \]
where \( l_{\text{Bool}}[z^\alpha, z^\sigma] = (i, l_1, l_2) \).
\[ (P_1 \land P_2)[z^\alpha, z^\sigma] = P_1[z^\alpha, z^\sigma] \land P_2[z^\alpha, z^\sigma] \]
\[ (\exists z_a P)[z^\alpha, z^\sigma] = \exists z_a P[z^\alpha, z^\sigma] \]
for \( a = \text{Int}, \text{Bool}, \text{Int}^*, \text{Bool}^* \).
\[ (\exists z_d P)[z^\alpha, z^\sigma] = \exists z_d(z_d \in \text{bij}^d \land P[z^\alpha, z^\sigma]) \]
for every \( d \in C \). Here \( z_d \in \text{bij}^d \) abbreviates \( \exists i z_d \neq \text{bij}^d \cdot i \) (cf. definition 6.28).
\[ (\exists z_d P)[z^\alpha, z^\sigma] = \exists z_d(z_d \subseteq \text{bij}^d \land P[z^\alpha, z^\sigma]) \]
for every \( d \in C \). Here \( z_d \subseteq \text{bij}^d \) abbreviates \( \forall i z_d \cdot i \in \text{bij}^d \) (cf. definition 6.28).
• \((\forall z^a)P[z^a, z^\sigma] = \forall z^a P[z^a, z^\sigma]\)
  for \(a = \text{Int}, \text{Bool}, \text{Int}^*, \text{Bool}^*\).

• \((\forall z^d P)[z^\alpha, z^\sigma] = \forall z^d (z_d \in \text{bij}^d \rightarrow P[z^\alpha, z^\sigma])\)
  for every \(d \in C\).

• \((\forall z^{d^*} P)[z^\alpha, z^\sigma] = \forall z^{d^*} (z^{d^*} \subseteq \text{bij}^d \rightarrow P[z^\alpha, z^\sigma])\)
  for every \(d \in C\).

In this transformation we assume that the quantified variables are distinct from any of
the variables of \(\text{bij}\). 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
\(\text{Code}(\omega, \sigma, f)\) iff for every \(c \in C\) we have

• \(\sigma^{(c)} \subseteq \{\text{el}t(\beta^{\omega(c)}, n) : n \in \mathbb{N}\}\)

• for all \(\alpha \in \sigma^{(c)}\) and for all \(n \in \mathbb{N}\) we have
  \[\text{el}t(\beta^{\omega(c)}, n) = \alpha\quad \text{iff}\quad f^c(\alpha) = n\]

where \(\beta^{\omega(c)} = \omega(\text{bij}^c)\).

We write \(\text{Code}_L(\omega, \sigma, f)\) if \(\text{Code}(\omega, \sigma, f)\) and additionally for every \(c \in C\) we have

• \(\omega(z) \in \omega(\text{bij}^c)\) for every \(z \in L \cap L\text{Var}_c\)

• \(\omega(z) \subseteq \omega(\text{bij}^c)\) for every \(z \in L \cap L\text{Var}_c^*\).

In a sense \(\text{Code}(\omega, \sigma, f)\) can be interpreted as saying that \(\omega(\text{bij}^c)\) codes the restriction
of the osi \(f\) to the existing objects of \(\sigma\).

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
\[ \text{Ok}(\omega, \delta, \sigma^m) \] and \[ \text{Code}_L(\omega, \sigma, f) \], 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

\[ \sigma', \delta', \omega \models P^c \iff \sigma'^t, \delta, \omega \{ m / z^\alpha, z^\sigma \} \models P^c[z^\alpha, z^\sigma], \]

where \( \delta' = f(\delta) \), \( \omega' = f(\omega) \downarrow L \), \( n = [f(\delta(1))]_c \), \( m = [\sigma'] \), and \( z^\alpha, z^\sigma \) are new logical integer variables.

**Proof**

Induction on the complexity of \( P^c \). The case \( P^c = l^c_{\text{Bool}} \) is treated as follows. For every logical expression \( l^c \) such that \( LVar(l^c) \subseteq L \) and \( LVar(l^c) \cap \overline{bij} = \emptyset \) we prove, by induction on the complexity of \( l^c \), the following: Let \( l^c[z^\alpha, z^\sigma] = (i_1, i_1, i_2) \) where \( i = i_1, \ldots, i_q \). Then there exists a unique sequence of natural numbers \( k = k_1, \ldots, k_\gamma \) such that

\[ L[l_1](\omega \{ k, n, m / i, z^\alpha, z^\sigma \})(\delta)(\sigma^m) = t \]

and for this \( k \) we have

\[ [L[l_2](\omega')(\delta')(\sigma')]_d = L[l_2](\omega \{ k, n, m / i, z^\alpha, z^\sigma \})(\delta')(\sigma^m). \]

\[ \square \]

### 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 \( B_{ij}(z^\sigma) \), which expresses, as accurately as possible, that the current state is coded by the value of \( z^\sigma \) and that the logical variables \( \overline{bij} \) form a correct decoding table. However, it is only possible to express this up to isomorphism, as we shall see in lemma B.7.

**Definition B.5**

First we define the following auxiliary assertions:

- \( CT_{\text{Int}}(x^c_{\text{Int}}, z^\alpha, z^\sigma) = Ic((\text{bij}^c \cdot z^\alpha) \cdot x^c_{\text{Int}}) \models Val_{\text{Int}}(z^\alpha, k, z^\sigma), \)
  \( CT_{\text{Int}}(x^c_{\text{Int}}, z^\alpha, z^\sigma) = Ic((\text{bij}^c \cdot z^\alpha) \cdot x^c_{\text{Int}}) \models Val_{\text{Int}}(z^\alpha, k, z^\sigma), \)
  \[ \text{where } k = [x]. \]

- \( CT_d(x_d^c, z^\alpha, z^\sigma) = \langle ((\text{bij}^c \cdot z^\alpha) \cdot x_d^c \models \text{Val}_d(z^\alpha, k, z^\sigma) \models 0) \rangle \)
  \[ \langle ((\text{bij}^c \cdot z^\alpha) \cdot x_d^c \models \text{Val}_d(z^\alpha, k, z^\sigma) \models p) \rangle, \]
  \[ \text{where } d \in C \text{ and } k = [x_d]. \]

- \( CT_{\text{Int}}(u_{\text{Int}}, z^\sigma) = Ic(u_{\text{Int}}) \models Val_{\text{Int}}(k, z^\sigma), \)
  \( CT_{\text{Bool}}(u_{\text{Bool}}, z^\sigma) = Ic(u_{\text{Bool}}) \models Val_{\text{Bool}}(k, z^\sigma), \)
  \[ \text{where } k = [u]. \]
Definition B.6
Next we define the assertion $Bij(z')$, where $z'$ is some logical integer variable, as follows.

$$Bij(z') = \forall_c \forall_z \exists_i (bij^c \cdot i = z) \land$$
$$\forall_c \forall_i (E^c(i, z) \leftrightarrow bij^c \cdot i \neq \text{nil}) \land$$
$$\forall_c \forall_i (bij^c \cdot i \neq \text{nil} \rightarrow \exists_d \exists_{x \in Var^d} CT_d(x, i, z')) \land$$
$$\forall_d \forall_{u \in TVar_d} CT_d(u, z')$$

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 $z'$. 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 $\sigma, \omega, f$ 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 $\delta$ such that $OK(\sigma, \delta, \omega)$.

Proof
Straightforward.

B.4 Expressing the strongest post condition

Finally we are ready for the theorem stating the expressibility of the strongest post-condition.
Theorem B.8
Let \( \rho^c \) be closed, \( BVar \subseteq L \subseteq LVar, P^c \) such that \( LVar(P^c) \subseteq L \) and \( b \in L \cap L = \emptyset \). Then: \( SP^c_L(\rho^c, P^c) = \exists b_{ij}^{c_1}, \ldots, b_{ij}^{c_n}, z_1, z_2, z_3(Q) \) (assuming \( C = \{c_1, \ldots, c_n\} \)), where \( Q = \bigwedge_{1 \leq p \leq 5} Q_p \), and

- \( Q_1 = T(\rho^c, z_1, z_2, z_3) \),
- \( Q_2 = \text{Bij}(z_3) \),
- \( Q_3 = b_{ij}^{c} \cdot A^c(z_1) \models \text{self} \),
- \( Q_4 = \bigwedge_c \forall i (B^c(i, z_1) \leftrightarrow b_{ij}^{c} \cdot i \in b_c) \),
- \( Q_5 = \exists z_{c_1}, \ldots, z_{c_n} \bigwedge_{1 \leq p \leq 4} R_p, \)

where \( R_1 = \bigwedge_c (z_c \subseteq b_{ij}^{c}) \)

\( R_2 = \bigwedge_c \forall i (E^c(i, z_2) \leftrightarrow z_c \cdot i \neq \text{nil}) \)

\( R_3 = \bigwedge_1 \bigwedge_{i \in L} (z_c \in z_c \cdot i) \land \bigwedge_1 \bigwedge_{i \in L} (z_c \cdot i \subseteq z_c^c) \)

\( R_4 = P^c[z, z'] [\bar{e}/\bar{b}], A^c(z_1)/z, z_2/z' \)

where \( l_1 \leq l_2 \), for \( a = d^* \), abbreviates the assertion \( \forall i (l_1 \cdot i \in \text{nil} \lor l_1 \cdot i \equiv l_2 \cdot i) \), and \( \bar{e} \) denotes a sequence \( z_{c_1}, \ldots, z_{c_n} \) of fresh logical variables.

The quantification \( \exists b_{ij}^{c_1}, \ldots, b_{ij}^{c_n} \) 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 \( c'_i, \sigma_0, \) and \( f(\sigma) \), respectively. The conjunction \( \bigwedge_{1 \leq p \leq 4} Q_p \), then expresses \( f(\sigma) = P[\rho^c](\gamma)(\delta')(\sigma_0) \).

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 \( R_1 \land R_2 \) states that the variable \( z_c, 1 \leq i \leq n \), stores all the existing objects of \( \sigma_0 \) (of class \( c_i \)) at a position which equals its code. The assertion \( R_3 \) then states that \( \omega' \) is compatible with \( \sigma_0 \). 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 \( i = 1, \ldots, n, \alpha_i \in \text{O}^{\alpha_i}, \) and \( \beta_1, \beta_2, \beta_3 \in \text{N} \) such that \( \sigma, \delta, \omega' \models Q \), where \( \omega' = \omega(\alpha_i/b_{ij}^{c_i}), \{\beta_1, \beta_2, \beta_3/z_1, z_2, z_3\} \).

As \( \sigma, \delta, \omega' \models Q_1 \) there exists \( \sigma_0, \sigma_1, \delta' \) such that \( \sigma_1 = P[\rho^c](\gamma)(\delta')(\sigma_0), \gamma \) 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: \( f(\alpha) = \beta \iff \text{elt}(\omega'(b_{ij}^{c_i}), \beta) = \alpha \). (Note that as \( \sigma, \delta, \omega' \models Q_2 \) we have that for \( \alpha \in \sigma^{c_i} \) there exists some \( \beta \in \text{N} \) such that \( \text{elt}(\alpha, \beta) = \alpha \), furthermore we have \( E^c(\beta, \beta_3) \) so \( \beta \in \sigma^{c_i}_1 \)). So we have \( \text{Code}(\omega', \sigma, f) \) 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 $\delta'(2)(c) = \{f'(\alpha) : \alpha \in \omega(b_c)\}$. 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 $i = 1, \ldots, n$, $\alpha' \in \mathcal{O}^c, \omega'' = \omega'(\alpha'/z_c)i$, such that $\sigma, \delta, \omega'' \models \Lambda_{1 \leq j \leq 3} R_j$. Let $\sigma'$ such that, for an arbitrary $c$, $\sigma'(c) = f^{-1}(\sigma_0(c))$. It then follows that $\sigma' \succeq \sigma$ and by $\sigma, \delta, \omega'' \models \Lambda_{1 \leq j \leq 3} R_j$ we have $Code_L(\omega'', \sigma', f)$, where $\omega'' = \omega'\{\alpha'/bij^v\}_p;\{d'_{1}, \beta_{z}, z, z'\}$. Furthermore we have $\sigma, \delta, \omega'' \models P_c[z, z']$, so we have by theorem B.4: $\sigma_0, \delta', \omega \models P_c$, where $\omega = f(\omega'') \downarrow L = f(\omega) \downarrow L$. This finishes one part of the proof.

On the other hand, let $\sigma, \sigma_0, \delta, \omega, f$ such that:

- $f(\sigma) = P[\rho^c][\gamma](\delta')(\sigma_0), \gamma$ 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 $\alpha_i \in \mathcal{O}^c$, for $i = 1, \ldots, n$ (assuming $C = \{c_1, \ldots, c_n\}$, such that $elt(\alpha, m) = \alpha(\neq \bot)$ iff $\alpha \in \sigma^{(c)}$ and $f(\alpha) = m$. Furthermore let $\omega'' = \omega'\{\alpha'/bij^v\}_p;\{\beta_1, \beta_2, \beta_3, z_1, z_2, z_3\}$.

Now $f(\sigma) = P[\rho^c][\gamma](\delta')(\sigma_0)$ 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 $\sigma, \delta, \omega'' \models Q_3 \land Q_4$.

Let, for $i = 1, \ldots, n$, $\alpha'_i$ be a subsequence of $\alpha_i$ such that $\sigma_0^{(c)} = \{\alpha : elt(\alpha', \alpha) \neq \bot\}$. Furthermore let $\omega'' = \omega'\{\alpha'/bij^v\}_p$. Now from $\alpha_i'$ being a subsequence of $\alpha_i$ it immediately follows that $\sigma, \delta, \omega'' \models R_1$.

From $\sigma_0^{(c)} = \{\alpha \in \mathcal{O}^c : elt(\alpha', \alpha) \neq \bot\}$ it in turn follows that $\sigma, \delta, \omega'' \models R_2$. Furthermore we have that $\sigma_0$ and $\omega'$ 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 $\sigma'$ be such that for an arbitrary $c$ we have $\sigma'(c) = f^{-1}(\sigma_0(c))$ and $\omega = \omega''\{\alpha'/bij^v\}_p;\{\delta'_{1}, \beta', z, z'\}$. We then have that $Code_L(\omega, \sigma', f)$ and $\sigma' \succeq \sigma$. So from $\sigma_0, \delta', \omega' \models P_c$ and $\omega' = f(\omega') \downarrow L$ applying theorem B.4 it follows that $\sigma, \delta, \omega \models P_c[z, z']$. So we infer that $\sigma, \delta, \omega'' \models R_4$.

Summarizing 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 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

- $\mathcal{F} \in \left( \prod_{i=1}^{n} O_{\perp}^{d_i} \right) \rightarrow \Delta^c \rightarrow \Sigma_{\perp} \rightarrow \left( \Sigma_{\perp} \times O_{\perp}^{d_0} \right)$, for some $c, n, d_0, \ldots, d_n$,
- $\mathcal{G} \in \Delta^c \rightarrow \Sigma_{\perp} \rightarrow \left( \Sigma_{\perp} \times O_{\perp}^{d} \right)$, for some $c, d$,
- $\mathcal{H} \in \Delta^c \rightarrow \Sigma_{\perp} \rightarrow \Sigma_{\perp}$, 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 $\mathcal{F}(\beta_1, \ldots, \beta_n)(\delta)(\sigma) = (\sigma', \beta_0)$ then there exists an osi $g$ such that $f^c \downarrow \sigma^{(c)} = g^c \downarrow \sigma^{(c)}$, for an arbitrary $c$, and $\mathcal{F}(f^{d_1}(\beta_1), \ldots, f^{d_n}(\beta_n))(f(\delta))(f(\sigma)) = (g(\sigma'), g^{d_0}(\beta_0))$,

- $Cl(\mathcal{G})$ iff for an arbitrary $\beta, \delta, \sigma, \sigma'$, $f$ such that $OK(\delta, \sigma)$: if $\mathcal{G}(\delta)(\sigma) = (\sigma', \beta)$ then there exists an osi $g$ such that $f^c \downarrow \sigma^{(c)} = g^c \downarrow \sigma^{(c)}$, for an arbitrary $c$, and $\mathcal{G}(f(\delta))(f(\sigma)) = (g(\sigma'), g^{d}(\beta))$,

- $Cl(\mathcal{H})$ 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 $s \in SEz$ satisfies this property assuming it holds for the meaning assigned to an arbitrary method:
Lemma C.3
Let \( \gamma \) 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 \( Cl(Z[s](\gamma)) \).

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 \( \delta, \sigma \) such that \( OK(\delta, \sigma) \).

\( s = new_d \): Let \( Z[new_d]\{\gamma\}(\delta)(\sigma) = \langle \sigma', \beta \rangle \). So we have \( pick^d(\sigma^{(c)}) = \beta \). Let \( \beta' = \text{pick}^{(d)}(f(\sigma)^{(c)}) \) and \( g \) be an osi such that \( f^c \upharpoonright \sigma^{(c)} = g^c \upharpoonright \sigma^{(c)} \), for an arbitrary \( c \), and \( g^d(\beta) = \beta' \). It follows that \( Z[new_d]\{\gamma\}(f(\delta))(f(\sigma)) = \langle g(\sigma'), \beta' \rangle \).

\( s = e_0 \{ m \} \{ e_1, \ldots, e_n \} \): 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

\[
\begin{align*}
\delta'_{(1)} &= \beta_0 \\
\delta'_{(2)}(c') &= \delta(2)(\sigma) \{ \delta(2)(\sigma') \cup \delta'_{(1)}/c' \} \\
\delta''_{(2)}(c') &= \delta(2)(\sigma')
\end{align*}
\]

assuming \( s \in SExp^\mathcal{E}_d \), 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 \( g^c \upharpoonright \sigma^{(c)} = f^c \upharpoonright \sigma^{(c)} \), \( c \) 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

\[
\begin{align*}
f(\delta'_{(1)}) &= f(\beta_3) \\
f(\delta'_{(2)}(c')) &= f^c(\delta(2)(\sigma)) \{ f^c(\delta(2)(\sigma')) \cup f^c(\delta'_{(1)})/c' \} \\
f(\delta''_{(2)}(c')) &= f^c(\delta(2)(\sigma'))
\end{align*}
\]

So we conclude \( \mathcal{E}[s]\{\gamma\}(f(\delta))(f(\sigma)) = \langle g(\sigma'), g(\beta) \rangle \). \( \square \)

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 \( \gamma \) be an agreement-preserving environment such that for an arbitrary \( s \in SExp \) we have \( Cl(Z[s](\gamma)) \). Then we have \( Cl(S[S](\gamma)) \) for an arbitrary \( S \in Siat \).

Proof
The proof proceeds by induction on the complexity of \( S \). We treat the following cases:
$S = s_\delta' \leftarrow s_\delta$: Let $\mathcal{S}[S](\gamma)(\delta)(\sigma) = \sigma'' (\text{OK}(\delta, \sigma))$ and $f$ be some osi. So we have $\mathbb{Z}[s](\gamma)(\delta)(\sigma) = \langle \sigma', \beta \rangle$ such that: $\sigma'' = \sigma' \langle \beta / \delta_{11} \rangle, x \rangle$. By $\text{Cl}(\mathbb{Z}[s](\gamma))$ 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 $\mathbb{Z}[s](\gamma)(f(\delta))(f(\sigma)) = \langle g(\sigma'), g(\beta) \rangle$. Now $g(\sigma'') = g(\sigma') \langle g^d(\beta) / g^c(\delta_{11}), z \rangle$, so we conclude $\mathcal{S}[S](\gamma)(f(\delta))(f(\sigma)) = g(\sigma'')$.

$S = S_1; S_2$: Let $\mathcal{S}[S](\gamma)(\delta)(\sigma) = \sigma' (\text{OK}(\delta, \sigma))$ and $f$ be some osi. So there exists a $\sigma''$ 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 $h$ such that $h^c \downarrow \sigma^{(c)} = g^c \downarrow \sigma^{(c)}$, for an arbitrary $c$, and $\mathcal{S}[S_2](\gamma)(h(\delta))(g(\sigma')) = h(\sigma')$. 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 $\mathcal{S}[S_1](\gamma)(g(\delta))(h(\sigma'')) = h(\sigma')$. Putting these applications of the induction hypothesis together gives us $h^c \downarrow \sigma^{(c)} = g^c \downarrow \sigma^{(c)}$, for an arbitrary $c$, and $\mathcal{S}[S](\gamma)(f(\delta))(f(\sigma)) = h(\sigma')$. (Note that by lemma 3.21 $\sigma \leq \sigma''$ and, as $\text{OK}(\delta, \sigma)$. $f(\delta) = g(\delta)$)

$S = \text{while } e \text{ do } S_1 \text{ od: Let } \mathcal{S}[S](\gamma)(\delta)(\sigma) = \sigma'$. So we have $\mu \Phi(\delta)(\sigma) = \sigma'$, where $\Phi$ is as defined in definition 3.14. Now it suffices to prove that for an arbitrary $\varphi \in \Delta^c \rightarrow (\Sigma \rightarrow \Sigma)$ such that $\text{Cl}(\varphi)$ we have $\text{Cl}(\Phi(\varphi))$. So assume for some $\varphi$ we have $\text{Cl}(\varphi)$. Let $\Phi(\delta)(\sigma) = \sigma' (\text{OK}(\delta, \sigma)$ and $f$ be some osi. We consider the case that $\mathcal{E}[e](\delta)(\sigma) = t$. By theorem 6.21 we then have $\mathcal{E}[e](f(\delta))(f(\sigma)) = t$. Furthermore we have $\mathcal{S}[S_1](\gamma)(\delta)(\sigma) = \sigma''$. Let $\mathcal{S}[S_1](\gamma)(\delta)(\sigma) = \sigma''$, by the induction hypothesis it then follows that for some osi $h$ 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 $\mathcal{S}[S_1](\gamma)(f(\delta))(f(\sigma)) = h(\sigma')$. 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 $\sigma \leq \sigma''$ and, as $\text{OK}(\delta, \sigma)$. $f(\delta) = g(\delta)$)

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 $\gamma$ be an agreement-preserving environment such that $\text{Cl}(\mathcal{S}[S](\gamma))$ for an arbitrary statement $S$. Then we have for every method name $m$ defined by $D \text{ Cl}(C[D](\gamma)(m))$ for an arbitrary class definition $D$.

**Proof**

Let $\gamma' = C[D](\gamma)$ and $f$ be some osi. Now let the method name $m$ be defined by $D$, say $m$ is declared as $\mu_{\delta_0, \ldots, \delta_k}$. We have $\gamma'(m) = \mathcal{M}[\mu_{\delta_0, \ldots, \delta_k}](\gamma)$. Let $\mu_{\delta_0, \ldots, \delta_k} = (u_1, \ldots, u_k) : S \uparrow e$. Moreover let
\[ M[[u_1, \ldots, u_k] : S \uparrow e][\gamma](\beta_1, \ldots, \beta_k)(\delta)(\sigma) = (\sigma''', \beta) \]

where \[ \sigma' = (\sigma_{(1)}, \sigma_{(2)}, \sigma'_{(3)}) \]

\[ \sigma'_{(3)}(u) = \beta_i \quad \text{if } u = u_i \]

\[ = \bot \quad \text{otherwise} \]

\[ \sigma'' = S^c[S][\gamma](\delta)(\sigma') \]

\[ \beta = E[e](\delta)(\sigma'') \]

\[ \sigma''' = (\sigma''_{(1)}, \sigma''_{(2)}, \sigma''_{(3)}) \]

Note that we assume \( \sigma \neq \bot \) and \( \delta_{(1)} \) not to be blocked. If one of these do hold we have \( \sigma' = \bot \) from which follows that \( \sigma''', \beta = \bot \). By the assumption about \( \gamma \) we have for some \( osi g \downarrow g^c \downarrow \sigma'(c) = f^c \downarrow \sigma'(c) \), for every arbitrary \( c \), and \( S^c[S][\gamma](f(\delta))(f(\sigma')) = g(\sigma'') \). (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 \( g'(\sigma'(c)) = f^c(\sigma(c)) \) for an arbitrary \( c \). By theorem 6.21 we have \( g(\beta) = E[e][g(\delta)](g(\sigma'')) \). (Note that as \( \gamma \) is agreement-preserving we have by lemma 3.21 \( \sigma \leq \sigma'' \), and so \( OK(\delta, \sigma'') \).) Putting this together gives us \[ M[[u_1, \ldots, u_k] : S \uparrow e][\gamma](f(\beta_1), \ldots, f(\beta_k))(f(\delta))(f(\sigma)) = (g(\sigma'''), g(\beta)). \]

\[ \square \]

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 \( U[U](\gamma_0) = \gamma' \) and \( \gamma_0 \) is the "empty" environment defined by \( \gamma_0(\beta)(\delta)(\sigma) = (\bot, \bot) \).

**Proof**

We have \( \gamma' = \bigcup_i \gamma_i \), \( \gamma_0 \) being the "empty" environment and \( C[D_1] \circ \cdots \circ C[D_n](\gamma_i) = \gamma_{i+1} \). 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 \( \gamma_i \) 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 \( i \in N \) and \( m \) not defined by \( U \) this suffices.) \( \square \)
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 \( \rho = (U|c : \mathcal{E}) \), environment \( \gamma \) we have \( Cl(P[\rho](\gamma)) \).

**Proof**
First note that as \( \rho \) is a closed program we have \( P[\rho](\gamma) = P[\rho](\gamma_0) \). We have by definition 3.18 that \( P[\rho](\gamma_0) = S[S](\gamma') \), where \( \gamma' = U[U](\gamma_0) \). 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 \( Cl(S[S](\gamma')) \). (Note that by lemma 3.21 \( \gamma' \) is agreement-preserving.)

**Corollary C.8**
For an arbitrary closed program \( \rho, \sigma, \sigma', \delta, f \) 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 \( g(\sigma') = [\rho](\gamma)(f(\delta))(f(\sigma)) \).