Atelier B

Proof Obligations

Reference Manual

version 3.7

CLEARSY
SYSTEM ENGINEERING

ATELIER B
Proof Obligations Reference Manual
version 3.7

Document made by CLEARSY.

CLEARSY
ATELIER B maintenance
Parc de la Duranne
320 avenue Archimède
Les Pléiades III - Bât.A
13857 Aix-en-Provence Cedex 3
France

Tél 33 (0)4 42 37 12 99
Fax 33 (0)4 42 37 12 71
email : maintenance.atelierb@clearsy.com

Contents

Chapter 1

Glossary

Vertical development: Set of B components linked by a **REFINES** clause.

Example:

MACHINE	**REFINEMENT**	**REFINEMENT**
MA	MA_1	MA_2
	REFINES	**REFINES**
	MA	MA_1
...	...	...

In this case, the vertical development is formed from the MA, MA_1 and MA_2 components.

Instancing: Assigns a value to the parameters of an abstract machine during an include/import. The parameters of the instanced abstract machine that is included or imported are called *formal parameters*, and the values that are assigned to them are *effective parameters*.

Example:

MACHINE	**MACHINE**
M1	M2(INTV, val)
INCLUDES	**CONSTRAINTS**
M2(1 .. 100, 35)	val $\in$ INTV
...	...

Effective parameter: Refer to "Instancing".

Formal parameter: Refer to "Instancing".

Predicate: A predicate is a logical expression that is read like a plain language affirmation. This kind of expression may be exact or inexact. Are considered predicates : equations, inequations, inequalities, tests for belonging or inclusion. Are also considered predicates, a conjunction of two predicates, the disjunction of two predicates and the negations of a predicate.

Example:

The following expressions are predicates:

$x3$	*"x is equal to 3"*
$5 < 2$	*"5 is strictly less than 2"*
$x \in \{1, 2, 4\}$	*"x belongs to the set* $\{1, 2, 4\}$*"*
$x + y^2 0 \ \vee \ y < x$	*"*$x + y^2$ *is equal to 0 or y is strictly less than x"*

The following expressions **are not** predicates:

$x + y$	*"the sum of x and y"*
$f(2)$	*"the value of f in 2"*
$\{1, 2, 4\}$	*"set* $\{1, 2, 4\}$*"*
$A \cup B$	*"the union of A and B"*

Checking Process: The Checking process checks the conformity of the product in relation to its specifications, throughout its specifications, throughout its development (*"Are we building the product correctly?"*).

Chapter 2

Introduction

This document describes the B theory proof obligations:

Definition:

> A *proof obligation* is a mathematical formula to be proven, in order to ensure that a B component is *correct.*

B theory specifies the proof obligations that must be proven to ensure the *correctness* of a given B component. To this end, the proof obligations assist the Checking process.

The proof obligations described in this document are mathematical formulas. To understand their effect properly, it is necessary to have a good working knowledge of B and of mathematical logic.

The proof obligations produced by the proof obligation generator in Atelier B are not exactly the ones described in this document. The proof obligation generator transforms theoretical formulas into normalized formulas (more numerous and simpler), that are ready for efficient use by the Atelier B prover.

2.1 General Format of Proof Obligations

The B proof obligations B are self-sufficient, i.e., no implicit information must be used in their demonstration. All of the proof obligations use the following structure:

$$\begin{array}{c} H \\ \Rightarrow \\ P \end{array}$$

where P and H are predicates. This formula means that it is necessary to prove the aim P applying assumption H, H being generally a conjunction of predicates.

In B, the P predicate and some H assumptions are built by applying one (or more) substitution(s) to a predicate. As B is a mathematical language, the substitutions and the predicates considered are directly taken from the B source.

The reader may refer to the B language Reference Manual to find the meaning of the application of a substitution to a predicate. Note that the application of an "operation call" substitution is made by replacing this call by the body of the operation specified in the source abstract machine.

2.2 Introductory Example

Let us consider the "Example" machine below: it contains a "val" status variable, that is a natural integer, without limit values; it also defines an "increment" operation, the role of which is to add a 1 to the "val" state:

```
MACHINE
    Example
VARIABLES
    val
INVARIANT
    val ∈ ℕ
OPERATIONS
    increment ≙
        BEGIN
            val := val + 1
        END
END
```

B semantics require that the services provided by a component (in this case the service is the "increment" operation) should never invalidate the invariant. In this case, this means that the application for the "increment" operation must preserve the val $\in \mathbb{N}$ predicate. The corresponding proof obligation is:

$$\begin{array}{l} \text{val} \in \mathbb{N} \\ \Rightarrow \\ [\text{val} := \text{val} + 1]\text{val} \in \mathbb{N} \end{array}$$

By applying the substitution, it becomes

$$\begin{array}{l} \text{val} \in \mathbb{N} \\ \Rightarrow \\ \text{val} + 1 \in \mathbb{N} \end{array}$$

If this formula is true (as well as all of the other proof obligations for the machine Example), then the B component above is correct.

2.3 Effect of the Proof Obligations

Some concepts such as the feasibility of performing substitutions, performing operations that do not belong to this definition of correctness in B. With one exception, the proof of existence of entities defined in a B specification is not requested by the theory. The justification for this is that the proof of existence, generally difficult to achieve, and therefore costly, will be performed by exhibiting a special case (the implementation) at the end of the B development process.

The exception we mentioned above applies to refineable constants (those declared in an **ABSTRACT_CONSTANTS** clause), never assigned values, the feasibility of which must be demonstrated explicitly.

As it is now established that the notion of B proof obligations is linked to this correctness, it is this latter perspective that is presented here: the theoretical proof obligations for the B language.

2.4 Overview of Proof Obligations

The description of the proof obligations is split into three parts: abstract machines, refinements and implementations.

At this point, the problem of syntax correctness that is a precondition for semantic correctness is ignored. The B components mentioned later are therefore assumed to be syntaxically correct (the B BOOK "type check" concept).

The proof obligations relating to an abstract machine cover:

- the correctness of instance assignments when including machines (using the **INCLUDES** clause): the effective instancing parameters must satisfy the constraints of the included machine parameters;
- the correctness of assertions;
- the correctness of the initialisation: the initialisation must establish the machine invariant (the machine invariant must be true after applying the initialisation substitution);
- the correctness of operations: the operations must preserve the invariant (the machine invariant must be true after applying the operation substitution, given that the invariant was true previously); the operations must establish their postcondition.

The four categories of abstract machine proof obligations are again found for refinements. However, in the case of a refinement, the correctness of the initialisation and the operations implies the initialisation/abstract operation procedure so that the relevance of the refinement may be demonstrated. In addition, in the case of the correctness of an operation, it is not necessary to prove the postcondition : the correctness of the refinement is enough.

- correcness of the instancing (a proof obligation similar to that of an abstract machine);
- correctness of assertions;
- correctness of the initialisation: the initialisation must establish the invariant of the refinement (properties of the new variables **and** link invariant) **without contradicting the specified initialisation**;
- correctness of operations: the operations must preserve the refinement invariant (properties of the new variables **and** the link invariant) **without contradicting the specified operation**;

The proof obligations for valuations and local operations are added to the four previous categories of proof obligations for implementations. The relevance of the implementation to the initialisation and the operations leads to proof obligations that are similar to the ones for the refinements:

- correctness of the instancing when an import is made (the correctness of the proof obligation is similar to that of an include action for an abstract machine);

- correctness of the valuing of constants and abstract sets: the values must verify the properties (explicit and implicit) of constants and abstract sets;
- correctness of assertions;
- correctness of the initialisation: initialisation must establish the invariant of the implementation (properties of the new **and** link invariant) **without contradicting the specified operation**;
- correctness of operations: operations must preserve the invariant of the implementation (properties of the new variables **and** the link invariant) **without contradicting the specified operation**;
- correctness of local operation specifications: local operations must preserve the invariant of the imported machines; they must establish their postcondition;
- correctness of local operation implementations: local operations must preserve the variables of the implementation and the imported machines **without contradicting their specification**.

Chapter 3

Correctness of the Abstract Machine

3.1 Correctness of Inclusions

Presentation:

The inclusion of machines in B is *correct* once the constraints on the parameters of the formal parameters of the included machine are checked by the effective parameters specified when instancing is defined.

The instancing of the machines referenced in the **INCLUDES** clause may be performed from:

- abstract machine sets and constants, machines referenced in the **USES** clause and machines referenced in the **SEES** clause,
- formal parameters for the abstract machine and the machines referenced in the **USES** clause.

The properties of the various entities are specified by predicates in the **PROPERTIES** clause for the abstract machine, machines referenced in the **SEES** clause and machines referenced in the clause **USES** clause, and in the **CONSTRAINTS** clause for the abstract machine and machines referenced in the **USES** clause. The various predicates must therefore be present in the assumptions relative to the correctness of inclusions.

Description of the proof obligation:

The proof obligation that defines the correctness of an inclusion, to be proved for each inclusion, is formed from the following assumptions:

- Parameter constraints for the abstract machine and machines referenced in the **USES** clause.
- Properties of abstract machine constants, machines referenced in the **SEES** clause and machines referenced in the **USES** clause.

To these assumptions are added the following implicit assumptions:

- Any abstract set is defined as a non-empty sub-set of relative integers.
- Any set of formal parameters is defined as a non-empty sub-set of relative integers.
- Any listed set is defined as a set made up of all of these elements, the elements being distinct two by two.

Within these assumptions, the aim is to prove:

The (instanced) constraint of the included machine.

Appendix § A.1 presents the mathematical formulation of this same proof obligation.

Example: Considering the following two machines:

```
MACHINE
        MAIN(param10)
CONSTRAINTS
        param10 < 10
INCLUDES
        COUNT(param10)
END
```

```
MACHINE
        COUNT(par)
CONSTRAINTS
        par < 15
VARIABLES
        tab
INVARIANT
        tab ∈ (1 ... par) → BOOL
END
```

The proof obligation relating to the inclusion of the COUNT machine above must enable the establishment that the constraint on this machine is correctly checked by the instancing provided. The result is therefore:

$\text{param10} < 10$	*"Parameter constraints for the MAIN machine"*
$\Rightarrow$	
$[\text{par} : \text{param10}](\text{par} < 15)$	*"Set parameter instance by param10"*

This becomes, by applying the substitution:

$$\begin{array}{l} \text{param10} < 10 \\ \Rightarrow \\ \text{param10} < 15 \end{array}$$

3.2 Correctness of Assertions

Presentation:

The assertions are lemmas for proof phases, i.e. they are intermediate theorems that may be deduced from the invariant used to establish proof that the component is correct. These predicates will be added (by conjunction) as an assumption for the proof obligations each time that the invariant is present in these assumptions.

The machine assertions must be proven from the invariant and the properties of the entities handled: local variables, variables of the machines referenced in the **USES**, **INCLUDES**, and **SEES** clauses, constants, sets and parameters.

The order of the assertions in the B specification text is important as the assertions are proven in sequence by adding the previously proven assertions as assumptions.

Description of the Proof Obligation:

The proof obligation that defines the correctness of the assertions, to be proven for each assertion in turn, is formed from the following assumptions:

- Constraints of abstract machine parameters and of machines referenced in the **USES** clause.
- Properties of the constants for the abstract machine and the machines referenced in the **SEES**, **USES**, **INCLUDES** clauses.
- Invariants and assertions for machines referenced in the **INCLUDES** clause, to which the corresponding instance definition was applied.
- Invariant of the abstract machine and the machines referenced in the **USES** clauses.
- Assertions for machines referenced in the **USES** clause.
- Previous assertions (in the order of the text) for the abstract machine.

To these assumptions the following implicit assumptions are added:

- Any abstract set is defined as a non-empty sub-set of relative integers.
- Any formal parameter set list is defined as a non-empty sub-set of relative integers.
- Any listed set is defined as a set comprising all of its elements, and the elements are distinct two by two.

Based on these assumptions, the aim is to prove.

the assertion

Appendix § A.2 presents the mathematical formulation of this same proof obligation.

Example: Considering the following abstract machine:

```
MACHINE
        EXAM
VARIABLES
        xx, yy, zz
INVARIANT
        xx < 0 ∧
        yy > 10 ∧
        zz yy ∗ xx
ASSERTIONS
        zz < 0 ;
        zz ∗ xx > 0
END
```

The proof obligations relating to the assertions are as follows:

$$\begin{array}{ll}
\quad \mathrm{xx} < 0 \wedge & \textit{“Machine invariant”} \\
\quad \mathrm{yy} > 10 \wedge & \\
\quad \mathrm{zzyy} * \mathrm{xx} & \\
\Rightarrow & \\
\quad \mathrm{zz} < 0 & \textit{“First assertion for the machine”}
\end{array}$$

and

$$\begin{array}{ll}
\quad \mathrm{xx} < 0 \wedge & \textit{“Machine invariant”} \\
\quad \mathrm{yy} > 10 \wedge & \\
\quad \mathrm{zzyy} * \mathrm{xx} \wedge & \\
\quad \mathrm{zz} < 0 & \textit{“First assertion for the machine”} \\
\Rightarrow & \\
\quad \mathrm{zz} * \mathrm{xx} > 0 & \textit{“Second assertion for the machine”}
\end{array}$$

3.3 Correctness of the Initialisation

Presentation:

The initialisation of an abstract machine is *correct* when it establishes the invariant. After applying this initialisation, the component invariant must be true; the aim of the proof is therefore the application of the initialisation to the invariant.

The initialisation of the abstract machine can be specified from:

- abstract machine sets and constants, machines references in the **USES**, **INCLUDES** and **SEES** clauses,
- formal parameters of the abstract machine and of the machines referenced in the **USES** clauses,
- machine variables referenced in the **USES** and **INCLUDES** clauses,
- consult operations and the concrete variables of the machines referenced in the **SEES** clause.

The properties of these entities are specified by predicates in the corresponding **PROPERTIES** clauses, and in the CONSTRAINTS clauses for the abstract machine and from the machines referenced in the **USES** clause. These predicates must therefore appear in the assumptions relating to the correctness of inclusions. The definitions and the properties of the variables are specified in the **INVARIANT** and **ASSERTIONS** clauses for the corresponding abstract machines, this is why these predicates are also found in the assumptions.

Description of the proof obligation:

The proof obligation that defines the correctness of the initialisation contains the following assumptions:

- Constraints for abstract machine parameters and for machine referenced in the **USES** clause.

- Properties for abstract machine constants and for machines references in the **SEES**, **INCLUDES** and **USES** clauses.
- Invariants and assertions for machines referenced in the **INCLUDES** clauses, that the corresponding instances have been applied to. [1]
- Invariants and assertions of machines referenced in the **USES** and **SEES** clauses.

To these assumptions are added the following implicit assumptions:

- Any abstract set is defined as a non-empty sub-set of relative integers.
- Any formal set list parameter is defined as a non-empty sub-part of the relative integers.
- Any listed set is defined as the set comprising all of its elements, and the elements are separate two by two.

Using these assumptions, the aim is to prove:

> The invariant of the abstract machine, after applying the initialisations of the machines referenced in the **INCLUDES** clause, then the initialisation of the machine.

Appendix § A.3 presents the mathematical formulation of this same proof obligation.

Example: Let us consider the following machine:

```
MACHINE
        OP01
VARIABLES
        v1
INVARIANT
        v1 ∈ ℕ ∧ v1 − 1 ∈ ℕ
INITIALISATION
        v1 : 2
END
```

The proof obligation relating to correctness of the initialisation of the OP01 machine above, must allow the establishment of the invariant. We will therefore obtain:

$$\begin{array}{l}\textit{“No assumption in this example”}\\ \Rightarrow\\ [\mathrm{v1} : 2](\mathrm{v1} \in \mathbb{N} \ \land \ \mathrm{v1} - 1 \in \mathbb{N})\end{array}$$

Which becomes, by applying the substitution:

$$\begin{array}{l}\Rightarrow\\ 2 \in \mathbb{N} \ \land \ 2 - 1 \in \mathbb{N}\end{array}$$

[1]Note that there is a special case (seldom encountered) that applies to groups of machines linked by USES clauses and simultaneously present in an INCLUDES clause: in this case, the invariants of these machines that link the variables of a number of components are part of the aim to be proved rather than the assumptions (cf. § A.3).

3.4 Correctness of Operations

Presentation:

Each operation is defined by a header and a general substitution. The header may have typing predicates for the output parameters, which constitute the postcondition of the operation. The effect produced by the application of an operation is defined as the application of the substitution to a given predicate. In order to demonstrate the correctness of an operation, the predicate used is the invariant of the machine and the postcondition of the operation:

> An abstract machine operation is *correct* when it preserves the invariant - i.e. the invariant was true prior to applying the operation, and it is still true afterwards - and establishes its postcondition.

The proof obligation contains the invariant of the component in the Assumption, and its aim is the invariant and the postcondition after application of the operation.

The substitution that defines an operation may be specified from:

- sets and constants from the abstract machine, and the machines referenced in the **USES** clauses, **INCLUDES** and **SEES**,
- formal parameters for the abstract machine and the referenced machines in the **USES** clause,
- variables for the abstract machine and the referenced machines in the **USES** and **INCLUDES** clauses,
- accessing operations and concrete variables for the machines referenced in the **SEES** clause.

The definitions and properties of the parameters and constants are the predicates specified in the **PROPERTIES** and **CONSTRAINTS** clauses of the corresponding abstract machines. These different predicates are therefore in the assumptions relating to the correctness of the operation. The definitions and properties of the variables are the predicates specified in the **INVARIANT** and **ASSERTIONS clauses** of the corresponding abstract machines, this is why they are also part of the assumptions.

Description of the proof obligation:

As the proof obligation defines the correctness of an operation, it must be demonstrated for each operation, and contain the following assumptions:

- Constraints of the abstract machine parameters and of the machines referenced in the **USES** clause.
- Properties of the constants of the abstract machine, of machines referenced in the **SEES** clause, machines referenced in the **INCLUDES** clauses and the machines referenced in the **USES** clause.
- Invariants and assertions for all of the machines referenced in the **INCLUDES** clause, to which the corresponding instance definitions were applied.

- Invariants and assertions from the abstract machine, machines referenced in the **USES** and **SEES** clauses,

To these assumptions the following implicit assumptions are added:

- Any abstract set is defined as a non-empty sub-part of the relative integers.
- Any set list formal parameter is defined as a non-empty sub-part of the relative integers.
- Any listed set is defined as the set comprising all of its elements, and the elements are separate two by two.

Using these assumptions, the aim to demonstrate is:

> The invariant of the abstract machine and the postcondition of the operation, after applying the substitution that defines the operation.

In Appendix § A.4 we present the mathematical formulation of this same proof obligation.

Example: Considering the following machine:

```
MACHINE
        OP02
VARIABLES
        v1
INVARIANT
        v1 ∈ ℕ ∧ v1 − 1 ∈ ℕ
INITIALISATION
        v1 : 2
OPERATIONS
        (ss ∈ ℕ) ⟵ increment ≙
                PRE
                        v1 − 2 ∈ ℕ
                THEN
                        v1, ss := v1 + 1, v1 − 1
                END
END
```

The proof obligation relating to the correctness of the operation on machine OP02 above must allow us to show that the invariant is preserved and that the postcondition is established. We will therefore obtain:

$$\begin{array}{ll} \mathrm{v1} \in \mathbb{N} \land & \textit{“Machine invariant”} \\ \mathrm{v1} - 1 \in \mathbb{N} & \\ \Rightarrow & \\ \multicolumn{2}{l}{[\textsc{pre}\ \mathrm{v1} - 2 \in \mathbb{N}\ \textsc{then}\ \mathrm{v1}, \mathrm{ss} := \mathrm{v1} + 1, \mathrm{v1} - 1\ \textsc{end}](\mathrm{v1} \in \mathbb{N}\ \land\ \mathrm{v1} - 1 \in \mathbb{N}\ \land\ \mathrm{ss} \in \mathbb{N})} \end{array}$$

This will be changed, by applying the PRE substitution, into

$$\begin{array}{l} \mathrm{v1} \in \mathbb{N} \land \\ \mathrm{v1} - 1 \in \mathbb{N} \\ \Rightarrow \\ \mathrm{v1} - 2 \in \mathbb{N}\ \Rightarrow\ ([\mathrm{v1}, \mathrm{ss} := \mathrm{v1} + 1, \mathrm{v1} - 1](\mathrm{v1} \in \mathbb{N}\ \land\ \mathrm{v1} - 1 \in \mathbb{N}\ \land\ \mathrm{ss} \in \mathbb{N})) \end{array}$$

And by applying the substitution:

$$\begin{array}{l} \mathrm{v1} \in \mathbb{N} \land \\ \mathrm{v1} - 1 \in \mathbb{N} \\ \Rightarrow \\ \mathrm{v1} - 2 \in \mathbb{N} \ \Rightarrow \ (\mathrm{v1} + 1 \in \mathbb{N} \ \land \ \mathrm{v1} + 1 - 1 \in \mathbb{N} \ \land \ \mathrm{v1} - 1 \in \mathbb{N}) \end{array}$$

Which is logically equivalent to

$$\begin{array}{l} \mathrm{v1} \in \mathbb{N} \land \\ \mathrm{v1} - 1 \in \mathbb{N} \land \\ \mathrm{v1} - 2 \in \mathbb{N} \\ \Rightarrow \\ \mathrm{v1} + 1 \in \mathbb{N} \ \land \ \mathrm{v1} + 1 - 1 \in \mathbb{N} \ \land \ \mathrm{v1} - 1 \in \mathbb{N} \end{array}$$

Chapter 4

Correctness of the Refinement

The correctness of the **INCLUDES** clauses and the proof of the assertions leads to proof obligations similar to those in abstract machines.

As for abstract machines, correctness of the initialisation and the refinement operations is established by establishing and preserving the invariant. However, the invariant of a refinement is a **link** invariant that defines the properties of new variables in relation to variables of the refined component. It is no longer adequate to apply a substitution to the invariant, it is necessary to apply the substitution of the refinement and the substitution of the refined component in combination, according to the double negation formula (see below). It is then clear that correcting a refinement is not just a simple internal coherence check, but an assurance that the refinement was developed in accordance with the specification.

In all of the proof obligations described in this section, the properties of the constants of previous refinements and of the abstract machine comprise properties of constants that are specific to the component and the properties of the constants of the machines referenced in the **INCLUDES** clause. In the same way, the invariants of previous refinements and of the abstract machine are made up of specific invariants and instanced invariants of machines referenced in the **INCLUDES** clause.

4.1 Correctness of Inclusions

Presentation:

In the same way as for abstract machines, the inclusion of machines in B is *correct* once the constraints of the formal parameters of the included machine are checked by the effective parameters specified when instancing is done.

The instancing of machines referenced in the **INCLUDES** clause may be performed from:

- sets and constants in the vertical development and the machines referenced in the **SEES** clauses,
- formal parameters for the abstract machine.

The properties of these entities are specified in the **PROPERTIES** clauses of the corresponding machine, and in the **CONSTRAINTS** clause of the abstract machine. These clauses must therefore be present in the assumptions relating the correctness of inclusions.

Description of the proof obligation:

The proof obligation that defines the correctness of an inclusion, to be demonstrated for each inclusion of the refinement, is formed from the following assumptions:

- Constraints of the parameters of the abstract machine.
- Properties of the constants of the vertical development and of machines referenced in the **SEES** clause.

To these assumptions are added the following implicit assumptions:

- Any abstract set is defined as a non-empty sub-part of relative integers.
- Any formal set list parameter is defined as a non-empty sub-part of relative integers.
- Any listed set is defined as the set comprising all of its elements, and the elements are distinct, two by two.

For these assumptions, the aim is to prove:

The constraint (instanced) of the included machine.

Appendix § B.1 presents the mathematical formulation of this particular proof obligation.

Example: Considering the following components:

```
MACHINE
        MAIN(param10)
CONSTRAINTS
        param10 < 10
END
```

```
REFINEMENT
        MAIN_1(param10)
REFINES
        MAIN
CONSTANTS
        vscal
PROPERTIES
        vscal ∈ ℕ ∧
        vscal < param10 /2
INCLUDES
        COUNT(vscal)
END
```

```
MACHINE
        COUNT(par)
CONSTRAINTS
        par < 15
VARIABLES
        tab
INVARIANT
        tab ∈ (1 . . . par) → BOOL
END
```

The proof obligation relating to the inclusion of the COUNT machine in the MAIN_1 refinement below must allow establishing that the COUNT constraint is indeed checked by the instancing provided. This will show:

$$\begin{array}{ll}
\text{param10} < 10 \land & \textit{“Parameter constraints for the abstract machine”} \\
\text{vscal} < \text{param10}/2 \land & \textit{“Properties of the refinement”} \\
\text{vscal} \in \mathbb{N} & \\
\Rightarrow & \\
[\text{par} : \text{vscal}](\text{par} < 15) & \textit{“Constraints of COUNT and its instancing”}
\end{array}$$

This becomes, by applying the substitution:

$$\begin{array}{l} \;\text{param10} < 10 \land \\ \;\text{vscal} < \text{param10}/2 \land \\ \;\text{vscal} \in \mathbb{N} \\ \Rightarrow \\ \;\text{vscal} < 15 \end{array}$$

4.2 Correctness of Assertions

Presentation:

Assertions are lemmas for the proof phases. These predicates will be added (by conjunction) as an assumption for the proof obligations, each time the invariant is present in these assumptions.

Refinement assertions must be proven from the invariant and the properties of the entities handled: refinement variables, concrete variables from the vertical development, variables from the machines referenced in the **INCLUDES** clause, constants and sets.

The order of the assertions in the text of the B specification is Important, as the assertions are proven in sequence, by adding the previous assertions already proven as assumptions.

Description of the proof obligation:

The proof obligation that defines the correctness of the assertions to be proved for each assertion in the order, is formed from the following assumptions:

- Constraints of the abstract machine parameters,
- Properties of the vertical development constants for machines referenced in the **SEES** and **INCLUDES** clauses.
- Invariants and assertions for machines referenced in the **INCLUDES** clause, that the corresponding instancing was applied to.
- Invariants from the vertical development.
- Assertions from previous refinements and from the abstract machine.
- Previous assertions (in the order of the text) from the refinement.

To these assumptions are added the following implicit assumptions:

- Any abstract set is defined as a non-empty sub-part of the relative integers.
- Any formal set list parameter is defined as a non-empty sub-part of the relative integers.
- Any listed set is defined as a set comprising all of its elements and the elements taken two by two are distinct.

Based on these assumptions, the purpose to prove is:

> the assertion.

Appendix § B.2, presents the mathematical formulation of this proof obligation.

Example: Considering the following components:

```
MACHINE                 REFINEMENT
    EXAM                    EXAM_1
VARIABLES               REFINES
    xx, yy                  EXAM
INVARIANT               VARIABLES
    xx < 0 ∧                zz
    yy > 10 ∧           INVARIANT
END                         zz ∈ ℤ ∧
                            zz yy * xx
                        ASSERTIONS
                            zz < 0
                        END
```

The proof obligation relating to the assertion is as follows:

$$\begin{array}{ll}
\text{xx} < 0 \wedge & \textit{“Machine invariant”} \\
\text{yy} > 10 \wedge & \\
\text{zz} \in \mathbb{Z} \wedge & \textit{“Refinement invariant”} \\
\text{zzyy} * \text{xx} & \\
\Rightarrow & \\
\text{zz} < 0 & \textit{“Refinement assertion”}
\end{array}$$

4.3 Correctness of the Initialisation

Presentation:

The initialisation of a refinement is *correct* when it establishes the refinement invariant, *without contradicting the initialisation of the refined component.* This definition means that the new initialisation must not produce exactly the same results as the previous one but should rather ensure that the new initialisation values should not be in contradiction with the previous ones. Generally this means that the domain of the values of common variables may be restricted.

For example, if the initialisation of a *vv* variable in the abstract machine specifies an initial value in the 1..3 range, the initialisation of the refinement may assign the initial value 1 to *vv*.

When an abstract variable is not retained in the refinement, it is generally refined via the link invariant into a new variable. In this case, the initialisation of the new variable must be performed without contradicting the initialisation of the abstract variable.

For example, the initialisation of an abstract variable SS is the empty set ∅, and the link invariant specifies that the new variable num is the cardinal of SS, then the initialisation of num is 0.

The aim to prove is built from the invariant: to the contraposition of this invariant is applied the initialisation of the refined component (especially to initialize the abstract variables, in conjunction with the concrete variables); then, on the contraposition of the predicate obtained, we will apply the initialisation of the refinement.

The initialisation of the refinement may be specified from:

- sets and constants from the vertical development, and machines referenced in the **SEES** and **INCLUDES** clauses,
- abstract machine formal parameters,
- variables from machines referenced in the **INCLUDES** clause,
- concrete variables and operations for consulting machines referenced in the **SEES** clause.

The definitions and properties of the parameters and constants are specified in the **PROPERTIES** and **CONSTRAINTS clauses** for the corresponding abstract machines. These different clauses are therefore found in the assumptions relating to the correctness of the initialisation. The definitions and properties of the variables are specified in the clauses **INVARIANT** and **ASSERTIONS** clauses of the corresponding abstract components, this is why they are also in the assumptions.

Description of the proof obligation:

The proof obligation that defines the correctness of the initialisation contains the following assumptions:

- Abstract machine constraint,
- Properties of vertical development constants, and machines referenced in the **SEES** and **INCLUDES** clauses,
- Invariants and assertions for all machines referenced in the **INCLUDES** clause, to which the corresponding instancing was applied.
- Invariants and assertions for machines referenced in the **SEES** clause.

To these assumptions are added the following implicit assumptions:

- Any abstract set is defined as a non-empty sub-part of the relative integers.
- Any set list formal parameter is defined as a non-empty sub-part of the relative integers.
- Any listed set is defined as the set made up of all of its elements, and the elements are distinct two by two.

Taking these assumptions, the aim is to show:

> The refinement invariant, after contraposition, then the application of the initialisation of the refined component, contraposition, then application of the initialisation of the refinement.

In Appendix § B.3, we will present the mathematical formulation of this same proof obligation.

Example: Taking the following components:

```
MACHINE                          REFINEMENT
    OP01                             OP01_1
VARIABLES                        REFINES
    v1                               OP01
INVARIANT                        VARIABLES
    v1 ∈ 0 .. 10                     v2
INITIALISATION                   INVARIANT
    ANY value WHERE                  v2 2 * v1
        value ∈ 1 .. 5           INITIALISATION
    THEN                             v2 : 2
        v1 : value               END
    END
END
```

The proof obligation relating to the correctness of the initialisation of refinement OP01_1 above is built as follows: the contraposition of the invariant is:

$$\text{v2} \neq 2 * \text{v1}$$

The initialisation of the refined component, applied to this predicate is

$$[\textsc{any}\ \text{value}\ \textsc{where}\ \text{value} \in 1\,..\,5\textsc{then}\ \text{v1} : \text{value}\ \textsc{end}](\text{v2} \neq 2 * \text{v1})$$

Which becomes, by applying the substitution ANY:

$$\forall\,\text{value.}(\text{value} \in 1\,..\,5 \quad \Rightarrow \quad \text{v2} \neq 2 * \text{value})$$

The contraposition of the latter predicate is

$$\exists\,\text{value . }(\text{value} \in 1\,..\,5 \quad \wedge \quad \text{v22} * \text{value})$$

The application of the refinement initialisation allows us to instance v2 with 2, so that the proof obligation to be proven will then be obtained

$$\Rightarrow$$
$$\exists\,\text{value . }(\text{value} \in 1\,..\,5 \quad \wedge \quad 22 * \text{value})$$

4.4 Correctness of Operations

Presentation:

Each refinement operation is a new (more solid) version of a previously specified operation. The headings of the two operations are identical, only the generalized substitution that defines the effect of the operation is modified.

> A refinement operation is *correct* when it preserves the invariant *without contradicting the specified operation*, and when its precondition is less restrictive than the specified precondition.

It should be understood by this definition, that the new operation must not produce exactly the same results as the previous one, but rather that the effects of the new operation must not be in contradiction with the effects specified in the abstract operation. The aim of the proof obligation will therefore be built around a double negative.

This definition means that the range of output values for the operation's output variables may be restricted.

For example, if the abstract operation returns a value in the range of 1..10, then the new operation may return a value in the range 2..4.

As for operations on abstract machines, the aim to prove is based on the component invariant; in the case of refinements an equals predicate is added (by conjunction), between the output variables from the refinement operation, and the renamed output variables from the specified operation. The aim to prove will then comprise

- the application of the refined operation to the negation of the application of the abstract operation to the negation of the invariant.

The example below will clarify this complex definition.

The substitution that defines an operation may be specified from:

- sets and constants in the vertical development, and the machines referenced in the **INCLUDES** and **SEES** clauses,
- formal parameters of the abstract machine,
- concrete variables from the vertical development,
- variables for machines referenced in the **INCLUDES** clause,
- concrete variables and operations for accessing machines referenced in the **SEES** clause.

The definitions and properties of the parameters and constants are specified in the **PROPERTIES** and **CONSTRAINTS clauses** of the corresponding operation. The different clauses are therefore in the assumptions relating to operation correctness. The definitions and properties of the variables are specified in the **INVARIANT** and **ASSERTIONS** clauses of the corresponding components, this is why they are also stated as assumptions.

Description of the proof obligation:

The proof obligation that defines the correctness of an operation, to be proven for each operation, contains the following assumptions:

- Parameter constraints from the abstract machine,
- Properties of the vertical development constants, and of machines referenced in the **SEES** and **INCLUDES** clauses,
- Invariants and assertions for machines referenced in the **INCLUDES** clause, to which the corresponding instance assignment was applied,

- Invariants and assertions from the vertical development and the machines referenced in the **SEES** clause,
- Precondition of the operation in the abstract machine.

To these assumptions are added the following implicit assumptions:

- Any abstract set is defined as a non-empty sub-part of the relative integers.
- Any formal set list of parameters is defined as a non-empty sub-part of the relative integers.
- Any listed set is defined as a set made up of all of its elements, and the elements are distinct taken two by two.

Given these assumptions, the aim to prove is:

> the refinement operation applied to the negation of the application of the abstract operation on the negation of the invariant.

Appendix § B.4 presents the mathematical formulation of this same proof obligation.

Example: Given the following components:

```
MACHINE
    OP01
VARIABLES
    v1
INVARIANT
    v1 ∈ ℕ
INITIALISATION
    v1 : 0
OPERATIONS
    /* The operation returns the new value of v1,
       that is greater than the previous one */
    out ⟵ valv1 ≙
        BEGIN
            ANY value WHERE
                value > v1
            THEN
                out, v1 : value, value
            END
        END
END
```

```
REFINEMENT
        OP01_1
REFINES
        OP01
VARIABLES
        v2
INVARIANT
        v2 > v1
INITIALISATION
        v2 : 1
OPERATIONS
        /* The new value of v1 is already stored in v2,
             it is therefore enough to return v2 and to prepare v2 for the next call */
        out ⟵ valv1 ≙
                BEGIN
                      out, v2 : v2, v2+1
                END
END
```

The first operation **valv1** returns values that are always larger. The second operation always returns the (arithmetical) successor of the previous return. The proof obligation must therefore be able to show that the arithmetical successor is in fact a larger value than the previous one.

The proof obligation relative to the correctness of operation **valv1** on the OP01_1 refinement above is built as follows: The invariant in conjunction with the predicate of equality between the operation output parameter in OP01 and the operation output parameter in OP01_1 is:

$$\mathrm{v2} > \mathrm{v1} \ \wedge \ out out'$$

This predicate shows what must always be true.

The contraposition of this formula is

$$\mathrm{v2} \leq \mathrm{v1} \ \vee \ out \neq out'$$

The predicate shows what must never occur.

The refined component operation (the abstract machine in this case), applied to this predicate is

$$[\text{ANY value WHERE value} > \text{v1 THEN out, v1 : value, value END}](\mathrm{v2} \leq \mathrm{v1} \ \vee \ out \neq out')$$

This becomes, by applying ANY and : substitutions:

$$\forall \ \text{value.}(\text{value} > \text{v1} \ \ \Rightarrow \ \ (\text{v2} \leq \text{value} \ \vee \ \text{value} \neq out'))$$

This predicate shows the effect of the specified operation on what must never happen.

The contraposition of this latter predicate is

$$\exists \, \text{value . (value} > \text{v1} \ \wedge \ (\text{v2} > \text{value} \ \wedge \ \text{value}out'))$$

The "value out'"equation allows us to logically simplify this predicate to become

$$out' > \mathrm{v1} \ \wedge \ \mathrm{v2} > out'$$

This predicate represents what is never established by the operation specified on what must never occur.

The application of the refinement operation allows us to instance *out′* with v2 and v2 with v2+1, this will give the following proof obligation:

$$\begin{array}{l}\mathrm{v1} \in \mathbb{N} \wedge \\ \mathrm{v2} > \mathrm{v1} \\ \Rightarrow \\ \mathrm{v2} > \mathrm{v1} \ \wedge \ \mathrm{v2} + 1 > \mathrm{v2}\end{array}$$

Note that v1 is the value returned by the previous call to the operation, v2 is the value that will be sent back by the current call and v2+1 is the value that will be sent back by the next call. There is in fact between these three values, an order relationship specified in operation **valv1** on the abstract machine.

Chapter 5

Correctness of the Implementation

As when correcting a refinement, correcting an implementation is not just a simple check on internal coherence, but the insurance that the implementation was developed in accordance with the the specification. The proof obligations relating to the correctness of implementations are similar to those for refinements, to which are added the proof obligation on the existence of refineable constants.

In the proof obligations described in this section, the properties of constants of previous refinements and of the abstract machine are made up of properties of constants that are specific to each of the components and of properties of constants of the machine referenced in the **INCLUDES** clause by its components. In the same way, the invariants of the previous refinements and of the abstract machine are made up of specific invariants and of instanced invariants of machines referenced in the **INCLUDES** clause.

5.1 Correctness of Imports

Presentation:

In the same way as an inclusion in abstract machines, importing machines into B is *correct* when the parameter constraints on the formal parameters of the imported machine are checked by the effective parameters specified when the instances are set.

The instance setting for machines referenced in the **IMPORTS** clause may be performed from:

- sets and constants from the vertical development and from machines referenced in the **SEES** clause,
- formal parameters of the abstract machine.

The properties of these entities are specified in the **PROPERTIES** clauses on the corresponding components and in the **CONSTRAINTS** clause of the abstract machine. These clauses must therefore be demonstrated within the assumptions relating to correcting imports.

Description of the proof obligation:

The proof obligation that defines the correctness of an importation, to be proven for each implementation import, is made up of the following assumptions:

- Abstract machine parameter constraints.
- Properties of the constants from the vertical development and of the machines referenced in the **SEES** clause.

To these assumptions are added the following implicit assumptions:

- Any abstract set is defined as a non-empty sub-set of relative integers.
- Any formal set list parameter is defined as a non-empty sub-set of relative integers.
- Any listed set is defined as the set comprising all of these elements, and the elements are distinct two by two.

Using these assumptions, the aim to prove is:

> The (instanced) constraint of the imported machine.

Appendix § C.1 the mathematical formulation of this same proof obligation.

5.2 Correctness of Valuations

Presentation:
The valuation of constants and abstract sets defined in the abstract machine, in the refinements and in the implementation must verify the properties of the various entities.
The valuation of the constants and sets may be performed from:

- sets and constants of machines referenced in the **IMPORTS** and **SEES** clauses.

The properties of these entities are specified in the **PROPERTIES** clauses of the machines referenced in the **IMPORTS** and **SEES** clauses. These clauses must therefore be present in the assumptions relating to correcting valuations.

Description of the proof obligation:
The proof obligation relating to the correctness of constants contains the following assumptions:

- Properties of the constants of machines referenced in the **SEES** clause,
- Properties of constants of machines referenced in the **IMPORTS** clause.

To these assumptions are added the following implicit assumptions:

- Any abstract set is defined as a non-empty sub-part of the relative integers.
- Any listed set is defined as the set made up of all of its elements, and with elements that are distinct two by two.

Given these assumptions, the aim to prove is:

The existence of constants that may be refined for the conjunction of properties of abstract machine constants, refinements and the implantation to which the valuation substitution was applied.

Appendix § C.2 presents the mathematical formulation of this same proof obligation.

Example: Taking the following components:

```
MACHINE                       IMPLEMENTATION
        OP01                          OP01_1
CONSTANTS                     REFINES
        S1, c1                        OP01
ABSTRACT_CONSTANTS            VALUES
        c2                            S1 1 .. 5 ;
PROPERTIES                            c1 0
        c1 ∈ 0 .. 10 ∧        END
        c2 ∈ S1
END
```

The proof obligation relating to the correctness of the valuation is:

$$\Rightarrow$$
$$\exists \, c2 \,.\, [\mathrm{S1}, \mathrm{c1} : 1 \,..\, 5, 0](\mathrm{c1} \in 0 \,..\, 10 \;\wedge\; \mathrm{c2} \in S1)$$

By applying the substitution we will obtain:

$$\Rightarrow$$
$$\exists \, c2 \,.\, (0 \in 0 \,..\, 10 \;\wedge\; \mathrm{c2} \in 1 \,..\, 5)$$

5.3 Correctness of Assertions

Presentation:

The implementation assertions must be proven from the invariant and the properties of the entities handled: local variables, abstract variables, variables of machines referenced in the **IMPORTS** and **SEES** clauses, constants and sets.

Description of the proof obligation:

The proof obligations that define the correctness of an assertion, to show that for each assertion in the order of text, is formed from the following assumptions:

- Constraints for abstract machine parameters,
- Properties of vertical development constraints and of machines referenced in the **SEES** and **IMPORTS** clauses.
- Invariants and assertions for machines referenced in the **IMPORTS** clause, to which the corresponding instancing was applied.
- Invariant in vertical development.
- Assertions on previous refinements and for the abstract machine.
- Previous assertions (in the order of the text) in the implementation.

To these assumptions are added the following implicit assumptions:

- Any abstract set is defined as a non-empty sub-part of the relative integers.
- Any set list formal parameter is defined as a non-empty sub-part of the relative integers.
- Any listed set is defined as the set made up of all of its elements, and the elements are distinct two by two.

Based on these assumptions, the aim to prove is:

the assertion.

Appendix § C.3 presents the mathematical formulation of these same proof obligations.

5.4 Correctness of the Initialisation

Presentation:

The initialisation of an implementation is *correct* when it establishes the invariant of the implementation, *without contradicting the initialisation of the implemented component.* This definition must be taken to mean that the new initialisation must not produce exactly the same results as the previous one, but rather that the new initialisation values must not contradict the previous ones. Generally this means that the value fields of the common variables (the "concrete_variables"in any vertical development) may be restricted.

The aim to prove is built from the invariant: to the contraposition of this invariant is applied the initialisation of the refined component (especially to initialize abstract variables, in accordance with the concrete variables); then on the contraposition of the predicate obtained, will be applied the initialisation of the implementation.

The initialisation of the implementation may be specified from:

- sets and constants from the vertical development and the machines referenced in the **IMPORTS** and **SEES** clauses,
- formal parameters of the abstract machine,
- variables from the machines referenced in the **IMPORTS** clause,
- concrete variables and access operations on the machines referenced in the **SEES** clause.

The definitions and properties of the parameters and constants are specified in the **PROPERTIES** and **CONSTRAINTS** clauses of the corresponding abstract machines. These different clauses are therefore present in the assumptions relating to initialisation correctness. The definitions and properties of the variables are specified in the **INVARIANT** and **ASSERTIONS** clauses of the corresponding abstract machines, this is why they are also present in the assumptions.

Description of the proof obligation:

The proof obligation that defines the correctness of the initialisation contains the following assumptions:

- Abstract machine parameter constraints,
- Properties of vertical development constants, and of machines referenced in the **SEES** and **IMPORTS** clauses.
- Invariants and assertions for machines referenced in the **IMPORTS** clauses, to which the corresponding instancing was applied.
- Invariants and assertions of machines referenced in the **SEES** clause.

To these assumptions are added the following implicit assumptions:

- Any abstract set is defined as a non-empty sub-part of the relative integers.
- Any set list formal parameter is defined as a non-empty sub-part of the relative integers
- Any listed set is defined as the set comprising all of its elements, and the elements are distinct taken two by two.

Given these assumptions, the aim to prove is:

> The implementation invariant, after contraposition, then the application of the initialisation of the refined component, contraposition, then the application of implementation initialisation.

Appendix § C.4 presents the mathematical formulation of this proof obligation.

Example: Taking the following components:

```
MACHINE
        OP01
VARIABLES
        v1
INVARIANT
        v1 ∈ 0 . . 10
INITIALISATION
        v1 :∈ 1..5
END
```

```
IMPLEMENTATION
        OP01_i
REFINES
        OP01
VARIABLES
        v2
INVARIANT
        v2 2 * v1
INITIALISATION
        v2 : 2
END
```

The proof obligation relating to the correctness of the initialisation of implementation OP01_i above is built as follows: the contraposition of the link invariant is:

$$\mathrm{v2} \neq 2 * \mathrm{v1}$$

The initialisation of the specification, applied to this predicate is

$$[\mathrm{v1} :\in 1 \mathinner{..} 5](\mathrm{v2} \neq 2 * \mathrm{v1})$$

Which becomes, by applying the substitution :∈:

$$\forall\, \mathrm{v1\_0}.(\mathrm{v1\_0} \in 1 \mathinner{..} 5 \quad \Rightarrow \quad \mathrm{v2} \neq 2 * \mathrm{v1\_0})$$

The contraposition of the latter predicate is

$$\exists\, \mathrm{v1\_0} \ . \ (\mathrm{v1\_0} \in 1 \mathinner{..} 5 \quad \wedge \quad \mathrm{v22} * \mathrm{v1\_0})$$

The application of the refinement initialisation allows setting instance v2 with 2, which will give the proof obligation

$$\begin{array}{l} \Rightarrow \\ \ \exists\, \mathrm{v1\_0} \ . \ (\mathrm{v1\_0} \in 1 \mathinner{..} 5 \quad \wedge \quad 22 * \mathrm{v1\_0}) \end{array}$$

5.5 Correctness of Operations

Presentation: Each implementation operation is a new (concrete) version of a previously specified operation. The headings of the two operations are identical, only the generalized substitution that defines the effect of the operation is modified.

> An implementation operation is *correct* when it preserves the invariant *without contradicting the refined operation.*

This definition should be understood to mean that the new operation must not produce exactly the same results as the previous one, but rather that the effects of the new operation must not contradict the effects specified in the abstract operation. The aim of the proof obligation will therefore be built around a double negative.

As for refinement operations, the aim to prove is based on the implementation (link) invariant in conjunction with an equals predicate between the implementation refinement operation output variables and the specified operation's renamed output variables. The aim to prove will then comprise

- the application of the implementation operation up to the negation of the application of the abstract operation, or up to the negation of the invariant.

The substitution that defines an operation may be specified from:

- sets and constants from the vertical development and machines referenced in the **IMPORTS** and **SEES** clauses,
- abstract machine formal parameters,
- concrete variables (**only**) taken from the vertical development and the machines referenced in the **IMPORTS** and **SEES** clauses.

The definitions and properties of the parameters and constants are specified in the **PROPERTIES** and **CONSTRAINTS** clauses of the corresponding components. These clauses are therefore present in the assumptions relating to operation correctness. The definitions and properties of the variables are specified in the **INVARIANT** and **ASSERTIONS** clauses of the corresponding clauses, this is why they are also in the assumptions.

Description of the proof obligation:

The proof obligation that defines the correctness of an operation, to be proven for each operation, contains the following assumptions:

- Abstract machine parameter constraints,
- Properties of the vertical development constants and of the machines referenced in the **SEES** and **IMPORTS** clauses,
- Invariants and assertions of machines referenced in the **IMPORTS** clause, to which the corresponding instancing was applied,
- Invariants and assertions from the vertical development, and machines referenced in the **SEES** clause,
- Precondition of the operation in the abstract machine.

To these assumptions are added the following implicit assumptions:

- All abstract sets are defined as a non-empty sub-part of the relative integers.
- Any set list formal parameter is defined as a non-empty sub-part of the relative integers.
- Any listed set is defined as the set comprising all of its elements, and the elements are distinct two by two.

Based on these assumptions, the aim to prove is:

> the implementation operation applied to the negation of the application of the abstract operation to the negation of the invariant.

Appendix § C.5 presents the mathematical formulation of this same proof obligation.

5.6 Correctness of Specifications of Local Operations

Presentation: Local operations can be used in an implementation in order to factorise B code. Local operations are both specified and implemented in the same implementation and can only be used within this implementation.

> A specification of a local operation is *correct* when it preserves the invariants of the machines imported by the implementation, and establishes its postcondition.

On the one hand, a local operation can modify directly the variables of an imported machine, hence this proof obligation. On the other hand, a local operation need not preserve the implementation invariant.

Description of the proof obligation:
The proof obligation relating to the correctness of the specification of a local operation, to be demonstrated for every local operation, contains the following hypothesis:

- Abstract machine parameter constraints,
- Properties of the vertical development constants and of the machines referenced in the **SEES** and **IMPORTS** clauses,
- Invariants and assertions of machines referenced in the **IMPORTS** clause, to which the corresponding instancing was applied,
- Invariants and assertions from the machines referenced in the **SEES** clause,
- B typing of the implementation concrete variables,
- Precondition of the specification of the local operation,

To these assumptions are added the following implicit assumptions:

- Every abstract set is defined as a non-empty sub-part of the relative integers.
- Every set formal parameter is defined as a non-empty sub-part of the relative integers.
- Every enumerated set is defined as the set comprising all of its elements, and the elements are distinct two by two.

Based on these assumptions, the aim to prove is:

> The body of the local operation specification, without its possible precondition, preserves the invariants of imported machines and establishes its postcondition.

Appendix § C.6 gives the mathematical formulation of this proof obligation.

5.7 Correctness of Implementations of Local Operations

Présentation : The local operation specified in the **LOCAL_OPERATIONS** clause are implemented in the **OPERATIONS** clause of the same implementation.

> An implementation of a local operation is *correct* when it preserves the equality predicates between the modifiable variables of the local operation specification and these same variables in the local operation implementation, *without contradicting* the local operation specification.

Description of the proof obligation:

The proof obligation relating to the correctness of the implementation of a local operation, to be demonstrated for every local operation, contains the following hypothesis:

- Abstract machine parameter constraints,
- Properties of the vertical development constants and of the machines referenced in the **SEES** and **IMPORTS** clauses,
- Invariants and assertions of machines referenced in the **IMPORTS** clause, to which the corresponding instancing was applied,
- Invariants and assertions from the machines referenced in the **SEES** clause,
- B typing of the implementation concrete variables,
- Precondition of the specification of the local operation,

To these assumptions are added the following implicit assumptions:

- Every abstract set is defined as a non-empty sub-part of the relative integers.
- Every set formal parameter is defined as a non-empty sub-part of the relative integers.
- Every enumerated set is defined as the set comprising all of its elements, and the elements are distinct two by two.

Based on these assumptions, the aim to prove is:

> The implementation of the local operation applied to the negation of the application of the specification of the local operation to the negation of the implicit invariant of equality for the variables of the implementation and imported machines, and for the output parameters of the local operation.

Appendix § C.7 presents the mathematical formulation of this proof obligation.

Appendix

Appendix A

Abstract Machine Proof Obligations

The following machines introduce the naming conventions that will be used to describe the proof obligations linked to machine M1:

MACHINE	**MACHINE**	**MACHINE**
M1(X_1,x_1)	Ms(X_s,x_s)	Mu(X_u,x_u)
CONSTRAINTS	**CONSTRAINTS**	**CONSTRAINTS**
C_1	C_s	C_u
SEES		
Ms		
SETS	**SETS**	**SETS**
S_1 ;	S_s ;	S_u ;
T_1 $\{a_1, b_1\}$	T_s $\{a_s, b_s\}$	T_u $\{a_u, b_u\}$
ABSTRACT_CONSTANTS	**ABSTRACT_CONSTANTS**	**ABSTRACT_CONSTANTS**
ac_1	ac_s	ac_u
CONCRETE_CONSTANTS	**CONCRETE_CONSTANTS**	**CONCRETE_CONSTANTS**
cc_1	cc_s	cc_u
PROPERTIES	**PROPERTIES**	**PROPERTIES**
P_1	P_s	P_u
INCLUDES		
Mi1(N_{i_1}, n_{i_1}),		
Mi2(N_{i_2}, n_{i_2})		
USES		
Mu		
ABSTRACT_VARIABLES	**ABSTRACT_VARIABLES**	**ABSTRACT_VARIABLES**
av_1	av_s	av_u
CONCRETE_VARIABLES	**CONCRETE_VARIABLES**	**CONCRETE_VARIABLES**
cv_1	cv_s	cv_u
INVARIANT	**INVARIANT**	**INVARIANT**
$I_1 \wedge L_1(av_u, cv_u)$	I_s	I_u
INITIALISATION	**INITIALISATION**	**INITIALISATION**
U_1	U_s	U_u
ASSERTIONS	**ASSERTIONS**	**ASSERTIONS**
J_1	J_s	J_u
OPERATIONS	**OPERATIONS**	**OPERATIONS**
$u_1 \leftarrow op_1(w_1) \widehat{=}$	$u_s \leftarrow op_s(w_s) \widehat{=}$	$u_u \leftarrow op_u(w_u) \widehat{=}$
PRE	PRE	PRE
Q_1	Q_s	Q_u
THEN	THEN	THEN
V_1	V_s	V_u
END	END	END
END	**END**	**END**

MACHINE	**MACHINE**
Mi1(X_{i_1},x_{i_1})	Mi2(X_{i_2},x_{i_2})
CONSTRAINTS	**CONSTRAINTS**
C_{i_1}	C_{i_2}
SETS	**SETS**
S_{i_1} ;	S_{i_2} ;
T_{i_1} $\{a_{i_1}, b_{i_1}\}$	T_{i_2} $\{a_{i_2}, b_{i_2}\}$
ABSTRACT_CONSTANTS	**ABSTRACT_CONSTANTS**
ac_{i_1}	ac_{i_2}
CONCRETE_CONSTANTS	**CONCRETE_CONSTANTS**
cc_{i_1}	cc_{i_2}
PROPERTIES	**PROPERTIES**
P_{i_1}	P_{i_2}
USES	
Mi2	
ABSTRACT_VARIABLES	**ABSTRACT_VARIABLES**
av_{i_1}	av_{i_2}
CONCRETE_VARIABLES	**CONCRETE_VARIABLES**
cv_{i_1}	cv_{i_2}
INVARIANT	**INVARIANT**
$I_{i_1} \wedge L_{i_1}(cv_{i_2}, av_{i_2})$	I_{i_2}
INITIALISATION	**INITIALISATION**
U_{i_1}	U_{i_2}
ASSERTIONS	**ASSERTIONS**
J_{i_1}	J_{i_2}
OPERATIONS	**OPERATIONS**
$u_{i_1} \leftarrow op_{i_1}(w_{i_1}) \mathrel{\widehat{=}}$	$u_{i_2} \leftarrow op_{i_2}(w_{i_2}) \mathrel{\widehat{=}}$
PRE	PRE
Q_{i_1}	Q_{i_2}
THEN	THEN
V_{i_1}	V_{i_2}
END	END
END	**END**

The following notations are used:

$$A_1 \ ^{\mathrm{def}}\ \mathrm{C}_1 \ \wedge\ X_1 \in \mathbb{P}_1(INT)$$
$$B_1 \ ^{\mathrm{def}}\ \mathrm{P}_1 \ \wedge\ S_1 \in \mathbb{P}_1(INT) \ \wedge\ T_1 \in \mathbb{P}_1(INT) \ \wedge\ T_1\{a_1, b_1\} \ \wedge\ a_1 \neq b_1$$

In the same way for the Mu, Ms, Mi1 and Mi2 machines.

A.1 Inclusion in an Abstract Machine

Formula of the proof obligation:

The proof obligation below must be proven for each included machine (Mi1 and Mi2), this is presented here for Mi1:

$$
\begin{array}{ll}
A_1 \ \wedge & \textit{“Parameter constraints for components”} \\
A_u \ \wedge & \textit{“Parameter constraints for components used”} \\
B_1 \ \wedge & \textit{“Properties of component constants”} \\
B_u \ \wedge & \textit{“Properties of constants in components used”} \\
B_s & \textit{“Properties of constants in components seen”} \\
\Rightarrow & \\
[X_{i_1}, x_{i_1} : N_{i_1}, n_{i_1}]C_{i_1} & \textit{“Instanced constraint of the included machine”}
\end{array}
$$

A.2 Assertion in an Abstract Machine

The assertion J_1 is a succession of predicates that will be written as J_{1_1}, J_{1_2}, $\cdots$, J_{1_k}.

Proof obligation formula:

The proof obligation below is to be proven for each assertion of J_1 ; it is presented here for the J_{1_j} for the $1 \leq j \leq k$ assertion:

$$
\begin{array}{ll}
A_1 \ \wedge & \textit{“Parameter constraints for components”} \\
A_u \ \wedge & \textit{“Parameter constraints for components used”} \\
B_1 \ \wedge & \textit{“Properties of component constants”} \\
B_u \ \wedge & \textit{“Properties of constants in components used”} \\
B_s \ \wedge & \textit{“Properties of constants in components seen”} \\
B_{i_1} \ \wedge \ B_{i_2} \ \wedge & \textit{“Properties of constants in included components”} \\
[X_{i_1}, x_{i_1} : N_{i_1}, n_{i_1}](I_{i_1} \ \wedge \ L_{i_1} \ \wedge \ J_{i_1}) \ \wedge & \textit{“Invariants and assertions”} \\
[X_{i_2}, x_{i_2} : N_{i_2}, n_{i_2}](I_{i_2} \ \wedge \ J_{i_2}) \ \wedge & \textit{“of included components”} \\
(I_u \ \wedge \ J_u) \ \wedge & \textit{“Invariants and assertions for components used”} \\
(I_1 \ \wedge \ L_1) \ \wedge & \textit{“Machine invariant”} \\
J_{1_1} \ \wedge \ \cdots \ \wedge \ J_{1_{j-1}} & \textit{“Previous assertions”} \\
\Rightarrow & \\
J_{1_j} & \textit{“Assertion to prove”}
\end{array}
$$

A.3 Initialisation in an Abstract Machine

Mathematical formula of the proof obligation:

$A_1 \wedge$ *"Parameter constraints for components"*

$A_u \wedge$ *"Parameter constraints for components used"*

$B_1 \wedge$ *"Properties of component constants"*

$B_u \wedge$ *"Properties of constants in components used"*

$B_s \wedge$ *"Properties of constants in components seen"*

$B_{i_1} \wedge B_{i_2} \wedge$ *"Properties of constants in included components"*

$(I_u \wedge J_u) \wedge$ *"Invariants and assertions for components used"*

$[X_{i_1}, x_{i_1} : N_{i_1}, n_{i_1}](I_{i_1} \wedge J_{i_1}) \wedge$ *"Invariants and assertions"*
$[X_{i_2}, x_{i_2} : N_{i_2}, n_{i_2}](I_{i_2} \wedge J_{i_2}) \wedge$ *"for components included"*

$(I_s \wedge J_s)$ *"Invariants and assertions of seen components"*

$\Rightarrow$

"Invariant after initialisation of machines included for the considered operation"
$[[X_{i_1}, x_{i_1} : N_{i_1}, n_{i_1}]U_{i_1} \,;\, [X_{i_2}, x_{i_2} : N_{i_2}, n_{i_2}]U_{i_2} \,;\, U_1](I_1 \wedge L_{i_1})$

A.4 Operations in an Abstract Machine

Mathematical formula of the proof obligation:

Let D_1 be the typing predicates of the output parameters u_1.

$A_1 \wedge$	*"Parameter constraints for components"*
$A_u \wedge$	*"Parameter constraints for components used"*
$B_1 \wedge$	*"Properties of component constants"*
$B_u \wedge$	*"Properties of component constants used"*
$B_s \wedge$	*"Properties of component constants seen"*
$B_{i_1} \;\wedge\; B_{i_2} \wedge$	*"Properties of constants in included components"*
$(I_u \;\wedge\; J_u) \wedge$	*"Invariants and assertions in components used"*
$[X_{i_1}, x_{i_1} : N_{i_1}, n_{i_1}](I_{i_1} \;\wedge\; L_{i_1} \;\wedge\; J_{i_1}) \;\wedge$	*"Invariants and assertions"*
$[X_{i_2}, x_{i_2} : N_{i_2}, n_{i_2}](I_{i_2} \;\wedge\; J_{i_2}) \;\wedge$	*"of included components"*
$(I_s \;\wedge\; J_s) \wedge$	*"Invariants and assertions of seen components"*
$(I_1 \;\wedge\; L_1 \;\wedge\; J_1) \wedge$	*"Machine invariants and assertions"*
Q_1	*"Operation precondition"*
$\Rightarrow$	
$[V_1](I_1 \;\wedge\; L_{i_1} \;\wedge\; D_1)$	*"Operation applied to the invariant and the postcondition"*

Appendix B

Refinement Proof Obligations

The following machines introduce naming conventions that will be used to describe the proof obligations linked to refinement Rn. M1 is the abstract machine, and R2, . . ., Rn-1 the refinements prior to Rn.

REFINEMENT	**MACHINE**	**MACHINE**
Rn(X_1,x_1)	M1(X_1,x_1)	Ms(X_s,x_s)
REFINES		
Rn-1		
	CONSTRAINTS	**CONSTRAINTS**
	C_1	C_s
SEES		
Ms		
SETS	**SETS**	**SETS**
S_n ;	S_1 ;	S_s ;
T_n $\{a_n, b_n\}$	T_1 $\{a_1, b_1\}$	T_s $\{a_s, b_s\}$
ABSTRACT_CONSTANTS	**ABSTRACT_CONSTANTS**	**ABSTRACT_CONSTANTS**
ac_n	ac_1	ac_s
CONCRETE_CONSTANTS	**CONCRETE_CONSTANTS**	**CONCRETE_CONSTANTS**
cc_n	cc_1	cc_s
PROPERTIES	**PROPERTIES**	**PROPERTIES**
P_n	P_1	P_s
INCLUDES	**INCLUDES**	
Mi1(N_{i_1}, n_{i_1}),	$Minc_1$	
Mi2(N_{i_2}, n_{i_2})		
ABSTRACT_VARIABLES	**ABSTRACT_VARIABLES**	**ABSTRACT_VARIABLES**
av_n	av_1	av_s
CONCRETE_VARIABLES	**CONCRETE_VARIABLES**	**CONCRETE_VARIABLES**
cv_n	cv_1	cv_s
INVARIANT	**INVARIANT**	**INVARIANT**
I_n	I_1	I_s
INITIALISATION	**INITIALISATION**	**INITIALISATION**
U_n	U_1	U_s
ASSERTIONS	**ASSERTIONS**	**ASSERTIONS**
J_n	J_1	J_s
OPERATIONS	**OPERATIONS**	**OPERATIONS**
$u_1 \leftarrow op_1(w_1) \widehat{=}$	$u_1 \leftarrow op_1(w_1) \widehat{=}$	$u_s \leftarrow op_s(w_s) \widehat{=}$
PRE	PRE	PRE
Q_n	Q_1	Q_s
THEN	THEN	THEN
V_n	V_1	V_s
END	END	END
END	**END**	**END**

Components M1, R2, ..., Rn-1 respectively include the machines $Minc_1$, $Minc_2$, ..., $Minc_{n-1}$. No description of these machines will be provided here because they take up too much space. However, the $M1inc_n$ and $M2inc_n$ machines included by refinement Rn is describe (especially due to the L_{i_1} part of the invariant in $M1inc_n$).

The abstract machine M1 above is different from machine M1 in Appendix A as it does not contain a **USES** clause.

MACHINE	**MACHINE**
Mi1(X_{i_1},x_{i_1})	Mi2(X_{i_2},x_{i_2})
CONSTRAINTS	**CONSTRAINTS**
C_{i_1}	C_{i_2}
SETS	**SETS**
S_{i_1} ;	S_{i_2} ;
T_{i_1} $\{a_{i_1}, b_{i_1}\}$	T_{i_2} $\{a_{i_2}, b_{i_2}\}$
ABSTRACT_CONSTANTS	**ABSTRACT_CONSTANTS**
ac_{i_1}	ac_{i_2}
CONCRETE_CONSTANTS	**CONCRETE_CONSTANTS**
cc_{i_1}	cc_{i_2}
PROPERTIES	**PROPERTIES**
P_{i_1}	P_{i_2}
USES	
Mi2	
ABSTRACT_VARIABLES	**ABSTRACT_VARIABLES**
av_{i_1}	av_{i_2}
CONCRETE_VARIABLES	**CONCRETE_VARIABLES**
cv_{i_1}	cv_{i_2}
INVARIANT	**INVARIANT**
$I_{i_1} \wedge L_{i_1}(v_{i_2})$	I_{i_2}
INITIALISATION	**INITIALISATION**
U_{i_1}	U_{i_2}
ASSERTIONS	**ASSERTIONS**
J_{i_1}	J_{i_2}
OPERATIONS	**OPERATIONS**
$u_{i_1} \leftarrow op_{i_1}(w_{i_1}) \;\widehat{=}$	$u_{i_2} \leftarrow op_{i_2}(w_{i_2}) \;\widehat{=}$
PRE	PRE
Q_{i_1}	Q_{i_2}
THEN	THEN
V_{i_1}	V_{i_2}
END	END
END	**END**

The following notations are used:

"Explicit and implicit properties of the machine"

$B_1 \quad {}^{\text{def}} \quad B_{inc1} \wedge P_1 \wedge S_1 \in \mathbb{P}_1(INT) \wedge$
$\quad T_1 \in \mathbb{P}_1(INT) \wedge T_1\{a_1, b_1\} \wedge a_1 \neq b_1$

$\vdots$

"Explicit and implicit properties of the previous refinement"

$B_{n-1} \quad {}^{\text{def}} \quad B_{incn-1} \wedge P_{n-1} \wedge S_{n-1} \in \mathbb{P}_1(INT) \wedge$
$\quad T_{n-1} \in \mathbb{P}_1(INT) \wedge T_{n-1}\{a_{n-1}, b_{n-1}\} \wedge a_{n-1} \neq b_{n-1}$

"Explicit and implicit properties of the considered refinement"

$B_n \quad {}^{\text{def}} \quad P_n \wedge S_n \in \mathbb{P}_1(INT) \wedge T_n \in \mathbb{P}_1(INT) \wedge T_n\{a_n, b_n\} \wedge a_n \neq b_n$

In addition, in the proof obligation formulas presented below, the invariants $I_1, \ldots, I_{n-1}$ are formed from the conjunction of the invariant of the corresponding component (M1 ... Rn-1) and possible included instanced invariants.

B.1 Inclusion in a Refinement

Proof obligation formula:

The proof obligation below must be proven for each included machine (Mi1 and Mi2), it is presented here for Mi1:

$A_1 \wedge$ *"Machine parameter constraints"*

$B_1 \ \wedge \ \ldots \ \wedge \ B_{n-1} \wedge$ *"Properties of constants from previous refinements"*

$B_n \wedge$ *"Properties of refinement constants"*

B_s *"Properties of constants in components seen"*

$\Rightarrow$

$[X_{i_1}, x_{i_1} : N_{i_1}, n_{i_1}]C_{i_1}$ *"Instanced constraint"*

B.2 Assertion in a Refinement

Assertion J_n is a succession of predicates that are noted J_{n_1}, J_{n_2}, $\cdots$, J_{n_k}.

Proof obligation formula:

The proof obligation below must be proven for each assertion of J_n :

$A_1 \wedge$ *"Machine parameter constraints"*

$B_1 \ \wedge \ \ldots \ \wedge \ B_{n-1} \wedge$ *"Properties of constants from previous refinements"*

$B_n \wedge$ *"Properties of refinement constants"*

$B_s \wedge$ *"Properties of constants of components seen"*

$B_{i_1} \ \wedge \ B_{i_2} \ \wedge$ *"Properties of constants in included components"*

$[X_{i_1}, x_{i_1} : N_{i_1}, n_{i_1}](I_{i_1} \ \wedge \ L_{i_1} \ \wedge \ J_{i_1}) \ \wedge$ *"Invariants and assertions"*
$[X_{i_2}, x_{i_2} : N_{i_2}, n_{i_2}](I_{i_2} \ \wedge \ J_{i_2}) \ \wedge$ *"of included components"*

$(I_1 \ \wedge \ J_1) \ \wedge \ \ldots \ \wedge \ (I_{n-1} \ \wedge \ J_{n-1}) \ \wedge$ *"Invariants and assertions of previous refinements"*

$I_n \wedge$ *"Refinement invariant"*

$J_{n_1} \ \wedge \ \cdots \ \wedge \ J_{n_{j-1}}$ *"Previous assertions"*

$\Rightarrow$

J_{n_j} *"Assertions to prove"*

B.3 Initialisation in a Refinement

Mathematical formula of the proof obligation:

$A_1 \wedge$ *"Machine parameter constraints"*

$B_1 \ \wedge \ \ldots \ \wedge \ B_{n-1} \ \wedge$ *"Properties of constants of previous refinements"*

$B_n \wedge$ *"Properties of refinement constants"*

$B_s \wedge$ *"Properties of constants of components seen"*

$B_{i_1} \ \wedge \ B_{i_2} \ \wedge$ *"Properties of constants of components included"*

$[X_{i_1}, x_{i_1} : N_{i_1}, n_{i_1}](I_{i_1} \ \wedge \ L_{i_1} \ \wedge \ J_{i_1}) \ \wedge$ *"Invariants and assertions"*
$[X_{i_2}, x_{i_2} : N_{i_2}, n_{i_2}](I_{i_2} \ \wedge \ J_{i_2}) \ \wedge$ *"of included components"*

$I_s \ \wedge \ J_s$ *"Invariants and assertions of components seen"*

$\Rightarrow$

"Initialisation of included components, then of the refinement applied to the negation of"
"the application of the refined initialisation applied to the negation of the invariant"
$[[X_{i_1}, x_{i_1} : N_{i_1}, n_{i_1}]U_{i_1} \,;\, [X_{i_2}, x_{i_2} : N_{i_2}, n_{i_2}]U_{i_2} \,;\, U_n]\neg\ [U_{n-1}]\neg\ (I_n \ \wedge \ L_{i_1})$

B.4 Operations in a Refinement

Mathematical formula of the proof obligation:

$A_1 \wedge$ *"Machine parameter constraints"*

$B_1 \ \wedge \ \ldots \ \wedge \ B_{n-1} \ \wedge$ *"Properties of constants of previous refinements"*

$B_n \wedge$ *"Properties of refinement constants"*

$B_s \wedge$ *"Properties of constants of components seen"*

$B_{i_1} \ \wedge \ B_{i_2} \ \wedge$ *"Properties of constants of components included"*

$[X_{i_1}, x_{i_1} : N_{i_1}, n_{i_1}](I_{i_1} \ \wedge \ L_{i_1} \ \wedge \ J_{i_1}) \ \wedge$ *"Invariants and assertions"*
$[X_{i_2}, x_{i_2} : N_{i_2}, n_{i_2}](I_{i_2} \ \wedge \ J_{i_2}) \ \wedge$ *"of included components"*

$I_s \ \wedge \ J_s \ \wedge$ *"Invariants and assertions of components seen"*

$(I_1 \ \wedge \ J_1) \ \wedge \ \ldots \ \wedge \ (I_n \ \wedge \ J_n) \ \wedge$ *"Invariants and assertions of the vertical development"*

Q_1 *"Precondition of the abstract operation"*

$\Rightarrow$

$Q_n \wedge$ *"Precondition of the refinement operation"*

"Refinement operation applied to the negation of the specified operation"
"applied to the negation of the invariant"
$[[u_1 : u_1']V_n]\neg\ [V_{n-1}]\neg\ (I_n \ \wedge \ u_1 u_1')$

Remark on this proof obligation: in order to check that the result of the refinement operation is the same as the result of the previous refinement operation, a predicate $u_1 u'_1$ is added to the invariant I_n. u_1 will take as its value the result of operation V_{n-1}; using the substitution $[u_1 : u'_1]$ applied to V_n, u'_1 will take as its value the result of operation V_n.

Appendix C

Implementation Proof Obligations

The following machines introduce the naming convention that will be be used to describe the proof obligations linked to the Mn implementation. M1 is the notation for the abstract machine, and R2, . . ., Rn-1 the refinements prior to Mn. These components are an inclusion of machines Minc1, Minc2, . . ., Mincn-1 respectively.

IMPLEMENTATION Mn(X_1,x_1) **REFINES** R_{n-1}	**MACHINE** M1(X_1,x_1)	**MACHINE** Ms(X_s,x_s)	**MACHINE** Mi(X_i,x_i)
	CONSTRAINTS C_1	**CONSTRAINTS** C_s	**CONSTRAINTS** C_i
SEES Ms			
SETS S_n ; T_n $\{a_n, b_n\}$	**SETS** S_1 ; T_1 $\{a_1, b_1\}$	**SETS** S_s ; T_s $\{a_s, b_s\}$	**SETS** S_i ; T_i $\{a_i, b_i\}$
	ABSTRACT_ CONSTANTS ac_1	**ABSTRACT_ CONSTANTS** ac_s	**ABSTRACT_ CONSTANTS** ac_i
CONCRETE_ CONSTANTS cc_n	**CONCRETE_ CONSTANTS** cc_1	**CONCRETE_ CONSTANTS** cc_s	**CONCRETE_ CONSTANTS** cc_i
PROPERTIES P_n	**PROPERTIES** P_1	**PROPERTIES** P_s	**PROPERTIES** P_i
VALUES S_1 E_1 ; ⋮ S_n E_n ; cc_1 d_1 ; ⋮ cc_n d_n			
IMPORTS Mi(N_i, n_i)	**INCLUDES** Minc1		
	ABSTRACT_ VARIABLES av_1	**ABSTRACT_ VARIABLES** av_s	**ABSTRACT_ VARIABLES** av_i
CONCRETE_ VARIABLES cv_n	**CONCRETE_ VARIABLES** cv_1	**CONCRETE_ VARIABLES** cv_s	**CONCRETE_ VARIABLES** cv_i
INVARIANT I_n	**INVARIANT** I_1	**INVARIANT** I_s	**INVARIANT** I_i
INITIALISATION U_n	**INITIALISATION** U_1	**INITIALISATION** U_s	**INITIALISATION** U_i
ASSERTIONS J_n	**ASSERTIONS** J_1	**ASSERTIONS** J_s	**ASSERTIONS** J_i
LOCAL_OPERATIONS $u_{l1} \leftarrow op_{l1}(w_{l1}) \widehat{=}$ PRE Q_{l1} THEN V_{l1} END			
OPERATIONS $u_{l1} \leftarrow op_{l1}(w_{l1}) \widehat{=}$ V_{l2} ; $u_1 \leftarrow op_1(w_1) \widehat{=}$ V_n	**OPERATIONS** $u_1 \leftarrow op_1(w_1) \widehat{=}$ PRE Q_1 THEN V_1 END	**OPERATIONS** $u_s \leftarrow op_s(w_s) \widehat{=}$ PRE Q_s THEN V_s END	**OPERATIONS** $u_i \leftarrow op_i(w_i) \widehat{=}$ PRE Q_i THEN V_i END
END	**END**	**END**	**END**

In this section, we will use the following notation:

"Explicit and implicit properties of the machine"

$$B_1 \quad \text{def} \quad B_{inc1} \ \land\ \mathrm{P}_1 \ \land\ S_1 \in \mathbb{P}_1(INT) \ \land \\ T_1 \in \mathbb{P}_1(INT) \ \land\ T_1\{a_1, b_1\} \ \land\ a_1 \neq b_1$$

⋮

"Explicit and implicit properties of the refined component"

$$B_{n-1} \quad \text{def} \quad B_{incn-1} \ \land\ \mathrm{P}_{n-1} \ \land\ S_{n-1} \in \mathbb{P}_1(INT) \ \land \\ T_{n-1} \in \mathbb{P}_1(INT) \ \land\ T_{n-1}\{a_{n-1}, b_{n-1}\} \ \land\ a_{n-1} \neq b_{n-1}$$

"Explicit and implicit properties of the implementation"

$$B_n \quad \text{def} \quad \mathrm{P}_n \ \land\ S_n \in \mathbb{P}_1(INT) \ \land\ T_n \in \mathbb{P}_1(INT) \ \land\ T_n\{a_n, b_n\} \ \land\ a_n \neq b_n$$

C.1 Importing into an Implementation

Proof obligation formula:

$A_1 \wedge$ *"Machine parameter constraints"*

$B_1 \wedge \ldots \wedge B_n \wedge$ *"Vertical development properties"*

B_s *"Properties of constants and components seen"*

$\Rightarrow$

$[X_i, x_i : N_i, n_i]C_i$ *"Instanced constraint of the imported machine"*

C.2 Valuation in an Implementation

Proof obligation formula:

$B_s \wedge$ *"Properties of constants of components seen"*

B_i *"Properties of constants of components imported"*

$\Rightarrow$

"Valuing constants applied to properties"
$\exists(ac_1, \ldots, ac_{n-1}) \ . \ [S_1 := E_1; cc_1 := d_1; \ldots; S_n := E_n; cc_n := d_n](B_1 \ \wedge \ \cdots \ \wedge \ B_n)$

C.3 Assertion in an Implementation

The assertion J_n is a succession of predicates that will be noted J_{n_1}, J_{n_2}, $\cdots$, J_{n_k}.

Proof obligation formula:
The proof obligation below must be proven for each assertion of J_n:

$A_1 \wedge$	*"Machine parameter constraints"*
$B_1 \wedge \ldots \wedge B_n \wedge$	*"Properties of the vertical development"*
$B_s \wedge$	*"Properties of constants of components seen"*
$B_i \wedge$	*"Properties of constants of imported components"*
$[X_i, x_i : N_i, n_i](I_i \wedge J_i) \wedge$	*"Invariants and assertions of imported components"*
$(I_1 \wedge J_1) \wedge \ldots \wedge (I_{n-1} \wedge J_{n-1}) \wedge$	*"Invariants and assertions of previous refinements"*
$I_n \wedge$	*"Invariant of the implementation"*
$J_{n_1} \wedge \cdots \wedge J_{n_{j-1}}$	*"Previous assertions"*
$\Rightarrow$	
J_{n_j}	*"Assertion to prove"*

C.4 Initialisation in an Implementation

Mathematical formula of the proof obligation:

$A_1 \wedge$	*"Machine parameter constraints"*
$B_1 \wedge \ldots \wedge B_n \wedge$	*"Properties of the vertical development"*
$B_s \wedge$	*"Properties of constants of components seen"*
$B_i \wedge$	*"Properties of constants of components imported"*
$[X_i, x_i : N_i, n_i](I_i \wedge J_i) \wedge$	*"Invariants and assertions for imported components"*
$I_s \wedge J_s$	*"Invariants and assertions for components seen"*
$\Rightarrow$	

"Initialisations of imported components then initialisation of the implementation"
"applied to the negation of the specified initialisation applied to the invariant"
$[[X_i, x_i : N_i, n_i]U_i \,\mathbf{;}\, [X_{i_2}, x_{i_2} : N_{i_2}, n_{i_2}]U_{i_2} \,\mathbf{;}\, U_n]\neg\ [U_{n-1}]\neg\ I_n$

C.5 Operations on an Implementation

Mathematical formula of the proof obligation:

$A_1 \wedge$	*"Machine parameter constraints"*
$B_1 \wedge \ldots \wedge B_n \wedge$	*"Vertical development properties"*
$B_s \wedge$	*"Properties of constants of components seen"*
$B_i \wedge$	*"Properties of constants of imported components"*
$I_s \wedge J_s \wedge$	*"Invariants and assertions of components seen"*
$[X_i, x_i : N_i, n_i](I_i \wedge J_i) \wedge$	*"Invariants and assertions of imported components"*
$(I_1 \wedge J_1) \wedge \ldots \wedge (I_n \wedge J_n) \wedge$	*"Invariants and assertions from the vertical development"*
Q_1	*"Abstract operation precondition"*

$\Rightarrow$

"Implementation operation applied to the negation of the specified operation"
"applied to the negation of the link invariant"
$[[u_1 : u_1']V_n]\neg\,[V_{n-1}]\neg\,(I_n \wedge u_1 u_1')$

C.6 Specification of Local Operations in an Implementation

Mathematical formula of the proof obligation:
Let D_{l1} be the typing predicates of the outputs parameters u_1.

$A_1 \wedge$	*"Machine parameter constraints"*
$B_1 \wedge \ldots \wedge B_n \wedge$	*"Vertical development properties"*
$B_s \wedge$	*"Properties of constants of components seen"*
$B_i \wedge$	*"Properties of constants of imported components"*
$I_s \wedge J_s \wedge$	*"Invariants and assertions of components seen"*
$[X_i, x_i : N_i, n_i](I_i \wedge J_i) \wedge$	*"Invariants and assertions of imported components"*
$BTyping(vc_n) \wedge$	*"B typing of the concrete variables of the implementation"*
Q_{l1}	*"Precondition of the local operation specification"*

$\Rightarrow$

"Local operation applied to the invariants of imported components and to the postcondition"
$[V_{l1}][X_i, x_i := N_i, n_i](I_i \wedge D_{l1})$

C.7 Implementation of Local Operation in an Implementation

Mathematical formula of the proof obligation:

$A_1 \wedge$	*“Machine parameter constraints”*
$B_1 \wedge \ldots \wedge B_n \wedge$	*“Vertical development properties”*
$B_s \wedge$	*“Properties of constants of components seen”*
$B_i \wedge$	*“Properties of constants of imported components”*
$I_s \wedge J_s \wedge$	*“Invariants and assertions of components seen”*
$[X_i, x_i : N_i, n_i](I_i \wedge J_i) \wedge$	*“Invariants and assertions of imported components”*
$BTyping(vc_n) \wedge$	*“B typing of the concrete variables of the implementation”*
$cv_n = cv'_n \wedge$	*“Implicit invariant of equality for the concrete variables of the implementation”*
$av_i = av'_i \wedge$	*“Implicit invariant of equality for the abstract variables of the imported components”*
$cv_i = cv'_i \wedge$	*“Implicit invariant of equality for the concrete variables of the imported components”*
Q_{l1}	*“Precondition of the local operation specification”*

$\Rightarrow$

“Implementation of the local operation applied to the negation of the specification of the local operation applied to the negation of the invariant of equality for the variables of the implementation and of imported components and for the output parameters of the local operation.”
$[[u_{l1} := u'_{l1}]V_{l2}]\neg\ [V_{l1}]\neg\ (cv_n = cv'_n \ \wedge\ av_i = av'_i \ \wedge\ cv_i = cv'_i \ \wedge\ u_{l1} = u'_{l1})$