Verification Conditions Generated by **Perfect Developer**

## 1 Revision history

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>02-05-2002</td>
<td>NE</td>
<td>Initial version</td>
</tr>
<tr>
<td>1.2</td>
<td>26-06-2002</td>
<td>NE</td>
<td>Updated to latest “that” and “any” obligations</td>
</tr>
<tr>
<td>1.3</td>
<td>09-08-2002</td>
<td>DC</td>
<td>Added introduction, made minor corrections, reformatted</td>
</tr>
<tr>
<td>1.4</td>
<td>10-12-2003</td>
<td>DC</td>
<td>Updated to version 2.10. Ordered the obligations alphabetically by name.</td>
</tr>
<tr>
<td>1.5</td>
<td>05-11-2004</td>
<td>DC</td>
<td>Updated for version 3.0</td>
</tr>
</tbody>
</table>

## 2 Introduction

This document lists all the verification conditions generated by **Perfect Developer** version 3.0.

The following, each type of verification condition is introduced by name. The names of verification conditions in this document correspond to the names that **Perfect Developer** uses in its informational output and in the proof and unproven output files. For each verification condition, we give a description, a primary location (i.e. source file/row/column) and sometimes also a secondary location. The primary location is the location given at the start of the warning message. The secondary location is the one that appears in brackets, typically preceded by the words “defined at”.

## 3 List of verification conditions

### 3.1 All qualifying elements in operand of ‘that’ are equal

Generated for every expression to ensure that all elements in the collection that satisfy the predicate are equal. Primary location is the location of the expression.

### 3.2 Assertion satisfies inherited assertion

Generated wherever: a method declared with `define` or `redefine` also declares an assertion, does not use the ‘...’ form of assertion, and the original method was declared with an assertion. In this case the new assertion must imply the original assertion (i.e. be stronger). Primary location is that of the new assertion; secondary location is that of the original assertion.

### 3.3 Assertion valid

Generated for every `assert` statement, whether within an expression, a postcondition or a statement list (but not a post-assertion). The location is that of the assertion.

### 3.4 At least one guard is true

Generated for every conditional expression, conditional postcondition and if statement with no empty guard. The location is that of the expression or statement.

### 3.5 Cannot break constraint via selector

Generated for every selector that may return a variable of a constrained type, or a sub-component of a variable of constrained type, to check that all values of the type of the selector satisfy the constraint. The primary location is that of the selector; the secondary location is that of the declaration of the constraint.

### 3.6 Class invariant satisfied

Generated after the postcondition of every modifying schema and constructor, and when `self` after `...` has been satisfied, for each invariant declared in the enclosing class. Primary location is that of the final part of the postcondition; secondary location is that of the class invariant.

### 3.7 Expressions modified by schema are independent

Generated for each call to a schema that modifies more than one object, to check that these objects are independent of each other. The location is that of the schema call.

### 3.8 Guarded variable [...] is initialised, or its ‘when’ guard has not become true

Generated every time a guarded data member of a class is accessed. Primary location is that of the call; secondary location is that of the guard condition.

### 3.9 Implementation has not changed abstract data

Generated for every implementation of a method whose specification cannot modify `self` (e.g. a function), but whose implementation modifies internal data. The verification condition checks that the values of the abstract data are unchanged. The location is that of the end of the implementation.

### 3.10 Inherited precondition satisfies new precondition

Generated wherever a method declared with `define` or `redefine` also declares a precondition. In this case the new precondition must be implied by the precondition of the original method (i.e. the new precondition must be the same as the old or be weaker). Primary location is that of the new precondition; secondary location is that of the original precondition.

### 3.11 Intermediate object satisfies class invariant

Generated every time an intermediate instance of `self` or `it` for which it is possible to break the class invariant is used in any way, except to access a variable member or as an operand of the equality operator. Primary location is that of the `self` or `it` expression; secondary location is that of the invariant.

### 3.12 Internal class invariant satisfied

Generated at the end of every implementation that modifies internal data of a class, to check that each internal invariant is satisfied. The primary location is that of the end of the implementation; the secondary location is that of the invariant being checked.

### 3.13 Jump satisfies precondition at label [...] [Page 2]
3.14 Left identity declared for operator [...] is valid
Generated for any operator declaration that includes the declaration of a left identity, to verify that the expression given really is a left identity. The primary location is that of the operator property declaration.

3.15 Loop body establishes end condition or decreases variant
Generated at the end of a loop body to check that either the until condition is true (i.e. the loop has terminated) or the variant has decreased. The primary location is that of the end of the loop body; the secondary location is that of the variant.

3.16 Loop body establishes end condition or preserves validity of variant
Generated at the end of a loop body to check that either the until condition is true (i.e. the loop has terminated) or that each integer variant component is non-negative. The primary location is that of the end of the loop body; the secondary location is that of the variant component being checked.

3.17 Loop body only modifies objects in 'change' list
Generated at the end of a loop body to check that only objects or parts of objects specified in the change list were actually changed by the loop body. The primary location is that of the end of the loop body; the secondary location is that of the change list.

3.18 Loop body preserves loop invariant
Generated at the end of a loop body to check that the invariant is true after each iteration. The primary location is that of the end of the loop body; the secondary location is that of the invariant being checked.

3.19 Loop initialisation establishes end condition or a valid variant
Generated at every loop statement to check that when the loop statement is reached (i.e. before the body has been executed at all), either the until condition is true (i.e. the loop body will not be executed at all), or all integer components of the variant (decrease part) are non-negative. The primary location is that of the loop; the secondary location is that of the variant component.

3.20 Loop initialisation establishes loop invariant
Generated at every loop statement to check that the stated invariant is true when the loop statement is reached (i.e. before the body has been executed at all). The primary location is that of the loop statement; the secondary location is that of the invariant being checked.

3.21 Method assertion implies 'require' assertion
Generated where a template with require declarations is instantiated. We check that each of the methods' assertions are implied by the assertions declared by the actual class methods. The primary location is that of the point of instantiation of the template; the secondary location is that of the require declaration.

3.22 Method precondition is implied by 'require' precondition
Generated where a template with require declarations is instantiated. We check that each of the actual class methods' preconditions is implied by the preconditions given in the require declaration. The primary location is that of the point of instantiation of the template; the secondary location is that of the require declaration.

3.23 Objects modified in parallel are independent
Generated for parallel postconditions (i.e. where conditions are combined with ' & ' or ' & ', and forall postconditions). The verification condition checks that the objects modified by each component of the condition are independent of each other. The forall postcondition with bag or seq bound also generates the verification condition that the collection is unique. Location is that of the postcondition.

3.24 Only variables modified in specification are modified by implementation
Generated for every schema with an implementation, to check that every variable and component of a variable not specified as changed in the postcondition (or specified as changed only in certain circumstances) is unchanged by the implementation (or is changed only in the same circumstances as the postcondition specifies). The primary location is that of the done statement (or the end of the implementation); the secondary location is that of the postcondition.

3.25 Operand of [that | any] has at least one qualifying element
Generated at every expression and every that expression with a condition. For a that or any with a condition, the verification condition checks that there exists an element of the collection that satisfies the condition. For an any with no condition we simply check that the collection is non-empty. The location is that of the any or that expression.

3.26 Operand of 'is' within specified type
Generated for every is cast, to check that the type of the expression is as stated. Location is that of the cast.

3.27 Operand of 'over' has at least one element
Generated for every op over expression if no left identity has been declared for op, to check that the collection is non-empty. The location is that of the expression.

3.28 Operator [...] is associative
Generated for every operator declaration which is declared as having the associative property to verify that a op (b c) = (a op b) op c. The location is that of the property declaration.

3.29 Operator [...] is commutative
Generated for every operator declaration which is declared as having the commutative property to verify that a op b = b op a. The location is that of the property declaration.

3.30 Operator [...] is idempotent
Generated for every operator declaration which is declared as having the idempotent property to verify that a op a = a. The location is that of the property declaration.

3.31 Operator '~~' is 'total'
Generated wherever the user defines the operator ~~ and declares it to be total to check the property x~~y = same@rank == x = y. Location is that of the operator declaration.
3.32 Operator '~~' is reflexive
Generated wherever the user defines the operator ~~ to check the property x~~x = same@rank. Location is that of the operator declaration.

3.33 Operator '~~' is transitive
Generated wherever the user defines the operator ~~ to check the property x~~y = y~~z ==> x~~y = x~~z. Location is that of the operator declaration.

3.34 Operator '~~' refines inherited definition
Generated wherever the user defines the operator ~~ in a class with an ancestor that also defines the operator ~~ to check that the new definition is a refinement of the ancestor definition, i.e. for any pair of operands for which the ancestor definition returns above@rank or below@rank, the new definition returns the same value. The primary location is that of the operator declaration; the secondary location is that of the ancestor operator declaration.

3.35 Operator '~~' satisfies first symmetry condition
Generated wherever the user defines the operator ~~ to check the properties:

x~~y = same@rank ==> y~~x = same@rank

Location is that of the operator declaration.

3.36 Operator '~~' satisfies second symmetry condition
Generated wherever the user defines the operator ~~ to check the properties:

x~~y = above@rank ==> y~~x = below@rank

Location is that of the operator declaration.

3.37 Operator '~~' satisfies third symmetry condition
Generated wherever the user defines the operator ~~ to check the properties:

x~~y = below@rank ==> y~~x = above@rank

Location is that of the operator declaration.

3.38 Post-assertion valid
Generated whenever a post-assertion is declared (other than in a deferred method) or inherited by a method declaration. The primary location is that of the end of the postcondition or function definition; the secondary location is that of the post-assertion being checked.

3.39 Postcondition specifies value for uninitialised data
Generated for all constructors and schemas with out parameters. A check is generated for each uninitialised data member or out parameter to ensure that it has a value specified for it before being used. The location is that of the postcondition.

3.40 Precondition at label [...] satisfied after preceding statement
Generated at each labelled statement that can be reached by fall-through from the preceding statement, to check that the precondition is true at this point. The location is that of the label.

3.41 Precondition of [...] satisfied
Generated at the point of call to any function, operator, selector, constructor or schema with a precondition. Primary location is that of the call; secondary location is that of the precondition.

3.42 Precondition of 'absurd' declaration is always false
Generated for every 'absurd' declaration to check that the precondition inherited for that method is always false given the invariants in the class containing the declaration. The primary location is that of the declaration; the secondary location is that of the inherited precondition.

3.43 Property satisfied
Generated for every property declaration. Location is that of the assertion being checked.

3.44 Return value satisfies specification
Generated at every value statement to check that the given value satisfies the specification. The primary location is that of the value statement; the secondary location is that of the specification of the value.

3.45 Selector still returns assigned value after assignment
Generated for every selector declaration to check that changing the value returned cannot affect which object the selector should return (e.g. if the selector contains a conditional, the branch selected cannot be changed by changing the value returned by the selector). The location is that of the selector declaration.

3.46 Specification satisfied at 'done'
Generated at every done statement to check that the postcondition has been achieved. The primary location is that of the done statement; the secondary location is that of the postcondition.

3.47 Specification satisfied at end of implementation
Generated at the end of every implementation that does not end with a done or value statement to check that the postcondition has been achieved. The primary location is that of the done or value statement; the secondary location is that of the postcondition.

3.48 Type constraint satisfied
Generated for parameters, return values, variables written to by schemas and objects modified in part by a postcondition with a constrained type. The primary location is that of the expression or postcondition in question; the secondary location is that of the declaration of the constraint.

3.49 Type constraint satisfied in assignment
Generated wherever a postcondition requires a value be assigned to a variable with a more constrained type. The primary location is that of the postcondition; the secondary value is that of the declaration of the constraint.
3.50 Variable [...] is not accessed unless its 'when' guard is true
Generated at each access to a variable that was declared with a when-guard, to ensure that the
guard is true. Primary location is that of the variable access; secondary location is that of the
variable declaration.

3.51 Variant decreases
Generated at the point of each recursive call. Checks the variant of the called method is less
than the variant of the calling method. Primary location is that of the recursive call; secondary
location is that of the variant of the called method.

3.52 Variant non-negative
Generated for all recursion variants with integer components. Assumes the method
precondition. Location is that of the variant component.

End of document