overloaded operator, depends on the context (see 12.5). For each predefined operator, the operand and result types are given in the following subclause.

NOTE 1—The syntax for an expression involving logical operators allows a sequence of binary **and**, **or**, **xor**, or **xnor** operators (whether predefined or user-defined), since the corresponding predefined operations are associative. For the binary operators **nand** and **nor** (whether predefined or user-defined), however, such a sequence is not allowed, since the corresponding predefined operations are not associative.

NOTE 2—The syntax for an expression involving a unary condition operator or unary logical operator in combination with any other operator requires that the unary operator and its operand be a parenthesized expression. For example, the expressions "(**and** A) **and** B" and "A **and** (**and** B)" are legal, whereas the expression "**and** A **and** B" and "A **and and** B" are not. Similarly, "**and** (**and** A)" is legal, whereas "**and and** A" is not. An expression consisting only of a unary condition oprator or unary logical operator and its operand need not be parenthesized.

NOTE 3—PSL extends the grammar of VHDL expressions to allow PSL expressions, PSL built-in function calls, and PSL union expressions as subexpressions. Such extended expressions can only appear in a VHDL description within PSL declarations and PSL directives, or in a verification unit.

9.2 Operators

9.2.1 General

The operators that may be used in expressions are defined as follows. Each operator belongs to a class of operators, all of which have the same precedence level; the classes of operators are listed in order of increasing precedence.

condition_operator ::= ??

logical_operator ::= **and** | **or** | **nand** | **nor** | **xor** | **xnor**

relational_operator ::= = | /= | < | <= | > | >= | ?= | ?/= | ?< | ?<= | ?> | ?>=

shift_operator ::= **sll** | **srl** | **sla** | **sra** | **rol** | **ror**

adding_operator ::= + | – | &

sign ::= + | –

multiplying_operator ::= * | / | **mod** | **rem**

miscellaneous_operator ::= ** | **abs** | **not**

Operators of higher precedence are associated with their operands before operators of lower precedence. Where the language allows a sequence of operators, operators with the same precedence level are associated with their operands in textual order, from left to right. The precedence of an operator is fixed and cannot be changed by the user, but parentheses can be used to control the association of operators and operands.

In general, operands in an expression are evaluated before being associated with operators. For certain operations, however, the right-hand operand is evaluated if and only if the left-hand operand has a certain value. These operations are called *short-circuit* operations. The binary logical operations **and**, **or**, **nand**, and **nor** defined for operands of types BIT and BOOLEAN are all short-circuit operations; furthermore, these are the only short-circuit operations.

Every predefined operator and every predefined MINIMUM and MAXIMUM operation is a pure function (see 4.2.1). No predefined operators have named formal parameters; therefore, named association (see 6.5.7.1) cannot be used when invoking a predefined operator.

NOTE 1—The predefined operators for the standard types are declared in package STANDARD as shown in 16.3.

NOTE 2—The operator **not** is classified as a miscellaneous operator for the purposes of defining precedence, but is otherwise classified as a logical operator.

9.2.2 Logical operators

The binary logical operators **and**, **or**, **nand**, **nor**, **xor**, and **xnor**, and the unary logical operator **not** are defined for predefined types BIT and BOOLEAN. They are also defined for any one-dimensional array type whose element type is BIT or BOOLEAN.

For the binary operators **and**, **or**, **nand**, **nor**, **xor**, and **xnor**, the operands shall both be of the same base type, or one operand shall be of a scalar type and the other operand shall be a one-dimensional array whose element type is the scalar type. The result type is the same as the base type of the operands if both operands are scalars of the same base type or both operands are arrays, or the same as the base type of the array operand if one operand is a scalar and the other operand is an array.

If both operands are one-dimensional arrays, the operands shall be arrays of the same length, the operation is performed on matching elements of the arrays, and the result is an array with the same index range as the left operand. If one operand is a scalar and the other operand is a one-dimensional array, the operation is performed on the scalar operand with each element of the array operand. The result is an array with the same index range as the array operand.

For the unary operator **not**, the result type is the same as the base type of the operand. If the operand is a one-dimensional array, the operation is performed on each element of the operand, and the result is an array with the same index range as the operand.

The effects of the logical operators are defined in the following tables. The symbol T represents TRUE for type BOOLEAN, '1' for type BIT; the symbol F represents FALSE for type BOOLEAN, '0' for type BIT.

A	B	A **and** B
T	T	T
T	F	F
F	T	F
F	F	F

A	B	A **or** B
T	T	T
T	F	T
F	T	T
F	F	F

A	B	A **xor** B
T	T	F
T	F	T
F	T	T
F	F	F

A	B	A **nand** B
T	T	F
T	F	T
F	T	T
F	F	T

A	B	A **nor** B
T	T	F
T	F	F
F	T	F
F	F	T

A	B	A **xnor** B
T	T	T
T	F	F
F	T	F
F	F	T

A	**not** A
T	F
F	T

For the short-circuit operations **and**, **or**, **nand**, and **nor** on types BIT and BOOLEAN, the right operand is evaluated only if the value of the left operand is not sufficient to determine the result of the operation. For operations **and** and **nand**, the right operand is evaluated only if the value of the left operand is T; for operations **or** and **nor**, the right operand is evaluated only if the value of the left operand is F.

The unary logical operators **and**, **or**, **nand**, **nor**, **xor**, and **xnor** are referred to as logical reduction operators. The logical reduction operators are predefined for any one-dimensional array type whose element type is BIT or BOOLEAN. The result type for the logical reduction operators is the same as the element type of the operand.

The values returned by the logical reduction operators are defined as follows. In the remainder of this subclause, the values of their arguments are referred to as R.

— The **and** operator returns a value that is the logical and of the elements of R. That is, if R is a null array, the return value is '1' if the element type of R is BIT or TRUE if the element type of R is BOOLEAN. Otherwise, the return value is the result of a binary **and** operation. The left argument of the binary **and** operation is the leftmost element of R. The right argument of the binary **and** operation is the result of a unary **and** operation with the argument being the rightmost (R'LENGTH – 1) elements of R.

— The **or** operator returns a value that is the logical or of the elements of R. That is, if R is a null array, the return value is '0' if the element type of R is BIT or FALSE if the element type of R is BOOLEAN. Otherwise, the return value is the result of a binary **or** operation. The left argument of the binary **or** operation is the leftmost element of R. The right argument of the binary **or** operation is the result of a unary **or** operation with the argument being the rightmost (R'LENGTH – 1) elements of R.

— The **xor** operator returns a value that is the logical exclusive-or of the elements of R. That is, if R is a null array, the return value is '0' if the element type of R is BIT or FALSE if the element type of R is BOOLEAN. Otherwise, the return value is the result of a binary **xor** operation. The left argument of the binary **xor** operation is the leftmost element of R. The right argument of the binary **xor** operation is the result of a unary **xor** operation with the argument being the rightmost (R'LENGTH – 1) elements of R.

— The **nand** operator returns a value that is the negated logical and of the elements of R. That is, the return value is the result of a **not** operation. The argument of the **not** operation is the result of a unary **and** operation with the argument being R.

— The **nor** operator returns a value that is the negated logical or of the elements of R. That is, the return value is the result of a **not** operation. The argument of the **not** operation is the result of a unary **or** operation with the argument being R.

— The **xnor** operator returns a value that is the negated logical exclusive-or of the elements of R. That is, the return value is the result of a **not** operation. The argument of the **not** operation is the result of a unary **xor** operation with the argument being R.

NOTE—All of the binary logical operators belong to the class of operators with the lowest precedence. The unary logical operators belong to the class of operators with the highest precedence.

9.2.3 Relational operators

Relational operators include tests for equality, inequality, and ordering of operands. The operands of each relational operator shall be of the same type. The result type of each ordinary relational operator (=, /=, <, <=, >, and >=) is the predefined type BOOLEAN. The result type of each matching relational operator (?=, ?/=, ?<, ?<=, ?>, and ?>=) is the same as the type of the operands (for scalar operands) or the the element type of the operands (for array operands).

Operator	Operation	Operand type	Result type
=	Equality	Any type, other than a file type or a protected type	BOOLEAN
/=	Inequality	Any type, other than a file type or a protected type	BOOLEAN
< <= > >=	Ordering	Any scalar type or discrete array type	BOOLEAN
?=	Matching equality	BIT or STD_ULOGIC	Same type
		Any one-dimensional array type whose element type is BIT or STD_ULOGIC	The element type
?/=	Matching inequality	BIT or STD_ULOGIC	Same type
		Any one-dimensional array type whose element type is BIT or STD_ULOGIC	The element type
?< ?<= ?> ?>=	Matching ordering	BIT or STD_ULOGIC	Same type

The equality and inequality operators (= and /=) are defined for all types other than file types and protected types. The equality operator returns the value TRUE if the two operands are equal and returns the value FALSE otherwise. The inequality operator returns the value FALSE if the two operands are equal and returns the value TRUE otherwise.

Two scalar values of the same type are equal if and only if the values are the same. Two composite values of the same type are equal if and only if for each element of the left operand there is a *matching element* of the right operand and vice versa, and the values of matching elements are equal, as given by the predefined equality operator for the element type. In particular, two null arrays of the same type are always equal. Two values of an access type are equal if and only if they both designate the same object or they both are equal to the null value for the access type.

For two record values, matching elements are those that have the same element identifier. For two one-dimensional array values, matching elements are those (if any) whose index values match in the following sense: the left bounds of the index ranges are defined to match; if two elements match, the elements immediately to their right are also defined to match. For two multidimensional array values, matching elements are those whose indices match in successive positions.