

Boolean implication and equivalence - Changes are relative to P1800-2008 Draft 4

Two new Boolean operators implication $\rightarrow$ and equivalence $\leftrightarrow$ are introduced. The operators can be used in any expressions. In fact, in constraints the operator $\rightarrow$ already exists with the same meaning as the one to be introduced.

Syntax 11-7—Operator syntax (excerpt from Annex A)

Replace

```
binary_operator ::=
+ | - | * | / | % | == | != | === | !== | ==? | !=? | && | || | **
| < | <= | > | >= | & | | | ^ | ^~ | ~^ | >> | << | >>> | <<<
```

With

```
binary_operator ::=
+ | - | * | / | % | == | != | === | !== | ==? | !=? | && | || | **
| < | <= | > | >= | & | | | ^ | ^~ | ~^ | >> | << | >>> | <<<
| -> | <->
```

Table 11-1: Operators and data types

Replace - note that Mantis 1035 deleted "other", stricken through here

&&	other binary logical operators	integral,	real, shortreal
----	---	-----------	--------------------

With

&& -> <->	other binary logical operators	integral,	real, shortreal
------------------	---	-----------	--------------------

Table 11-2—Legal operators for use in real expressions

Replace - note that Mantis 1035 deletes this table so this change is not needed

~~! && | |~~ ~~logical operators~~

With

~~! && | |~~ ~~logical operators~~

In 11.3.2 Operator precedence

Table 11-4—Operator precedence and associativity

Replace

<code>() [] :: .</code>	left
<code>+ - ! ~ & ~& ~ ^ ~^ ^~ ++ --</code> (unary)	
<code>**</code>	left
<code>* / %</code>	left
<code>+ - (binary)</code>	left
<code><< >> <<< >>></code>	left
<code>< <= > >=</code> inside dist	left
<code>== != === !== ==? !=?</code>	left
<code>& (binary)</code>	left
<code>^ ~^ ^~ (binary)</code>	left
<code> (binary)</code>	left
<code>&&</code>	left
<code> </code>	left
<code>?: (conditional operator)</code>	right
<code>-></code>	right
<code>= += -= *= /= %= &= ^= =</code>	none
<code><<= >>= <<<= >>>= := :/ <=</code>	
<code>{ } { }</code>	concatenation

With

<code>() [] :: .</code>	left
<code>+ - ! ~ & ~& ~ ^ ~^ ^~ ++ --</code> (unary)	
<code>**</code>	left
<code>* / %</code>	left
<code>+ - (binary)</code>	left
<code><< >> <<< >>></code>	left
<code>< <= > >=</code> inside dist	left
<code>== != === !== ==? !=?</code>	left
<code>& (binary)</code>	left
<code>^ ~^ ^~ (binary)</code>	left
<code> (binary)</code>	left
<code>&&</code>	left
<code> </code>	left
<code>?: (conditional operator)</code>	right
<code>-> <-></code>	right
<code>= += -= *= /= %= &= ^= =</code>	none
<code><<= >>= <<<= >>>= := :/ <=</code>	
<code>{ } { }</code>	concatenation

Replace

All operators shall associate left to right with the exception of the conditional operator, which shall associate right to left. Associativity refers to the order in which the operators having the same precedence are evaluated. Thus, in the following example, B is added to A, and then C is subtracted from the result of A+B.

With

All operators shall associate left to right with the exception of the conditional (?:) **implication (->)** and **equivalence (<->)** operators, which shall associate right to left. Associativity refers to the order in which the operators having the same precedence are evaluated. Thus, in the following example, B is added to A, and then C is subtracted from the result of A+B.

11.4.7 Logical operators

Replace

The operators *logical and* (&&) and *logical or* (| |) are logical connectives. The result of the evaluation of a logical comparison shall be 1 (defined as true), 0 (defined as false), or, if the result is ambiguous, the unknown value (x). The precedence of && is greater than that of | |, and both are lower than relational and equality operators.

A third logical operator is the unary *logical negation* operator (!). The negation operator converts a nonzero or true operand into 0 and a zero or false operand into 1. An ambiguous truth value remains as x.

With

The operators *logical and* (&&), ~~and~~ *logical or* (| |), *logical implication* (->) and *logical equivalence* (<->) are logical connectives. The result of the evaluation of a logical ~~comparison~~ **operation** shall be 1 (defined as true), 0 (defined as false), or, if the result is ambiguous, the unknown value (x). The precedence of && is greater than that of | |, and both are lower than relational and equality operators. **The precedence of -> and <-> is at the same level, the binding of operands between the two operations is governed by associativity (right), both are lower than other logical operators and the conditional operator.**

The logical implication `expression1 -> expression2` is logically equivalent to `(!expression1 || expression2)`, and the logical equivalence `expression1 <-> expression2` is logically equivalent to `((expression1 -> expression2) && (expression2 -> expression1))`. ~~A third logical operator is the unary logical negation operator (!).~~ The unary *logical negation* operator (!) ~~negation operator~~ converts a nonzero or true operand into 0 and a zero or false operand into 1. An ambiguous truth value remains as x.

Table 11-23—Bit lengths resulting from self-determined expressions

Replace the following table row

i op j, where op is: &&	1 bit	All operands are self-determined
----------------------------	-------	----------------------------------

With

i op j, where op is: && -> <->	1 bit	All operands are self-determined
---------------------------------------	-------	----------------------------------

A.8.6

Replace

binary_operator ::=

+ | - | * | / | % | == | != | === | !== | ==? | !=? | && | || | **
| < | <= | > | >= | & | | | ^ | ^~ | ~^ | >> | << | >>> | <<<

With

binary_operator ::=

+ | - | * | / | % | == | != | === | !== | ==? | !=? | && | || | **
| < | <= | > | >= | & | | | ^ | ^~ | ~^ | >> | << | >>> | <<<
| -> | <->