

Boolean implication and equivalence

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. These operators can also be used on property expressions.

8.2 Operator syntax

Replace

```
assignment_operator ::= //from A.6.2
    = | += | -= | *= | /= | %= | &= | |= | ^= | <<= | >>= | <<<= | >>>=
conditional_expression ::= //from A.8.3
    cond_predicate ? { attribute_instance } expression : expression
unary_operator ::= //from A.8.6
    + | - | ! | ~ | & | ~& | | | ~| | ^ | ~^ | ^~
binary_operator ::=
    + | - | * | / | % | == | != | === | !== | ===? | !==? | && | || | **
    | < | <= | > | >= | & | | | ^ | ^~ | ~^ | >> | << | >>> | <<<
inc_or_dec_operator ::= ++ | --
unary_module_path_operator ::=
    ! | ~ | & | ~& | | | ~| | ^ | ~^ | ^~
binary_module_path_operator ::=
    == | != | && | || | & | | | ^ | ^~ | ~^
```

With

```
assignment_operator ::= //from A.6.2
    = | += | -= | *= | /= | %= | &= | |= | ^= | <<= | >>= | <<<= | >>>=
conditional_expression ::= //from A.8.3
    cond_predicate ? { attribute_instance } expression : expression
unary_operator ::= //from A.8.6
    + | - | ! | ~ | & | ~& | | | ~| | ^ | ~^ | ^~
binary_operator ::=
    + | - | * | / | % | == | != | === | !== | ===? | !==? | && | || | **
    | < | <= | > | >= | & | | | ^ | ^~ | ~^ | >> | << | >>> | <<<
    | -> | <->
inc_or_dec_operator ::= ++ | --
unary_module_path_operator ::=
    ! | ~ | & | ~& | | | ~| | ^ | ~^ | ^~
binary_module_path_operator ::=
    == | != | && | || | & | | | ^ | ^~ | ~^
```

Table 8-1

Replace

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

With

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

Table 8-3

Replace

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

With

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

Between Syntax 8-1 and Table 8-1

Insert

The operator `expression1 -> expression2` is a shorthand for writing `!(expression1) || expression2` and

`expression1 <-> expression2` is a shorthand for `(expression1 -> expression2) && (expression2 -> expression1)`

A.8.6

Replace

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

With

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

17.11 Syntax 17-14 and in A.2.10

Replace

```
...
property_expr ::=
sequence_expr
| ( property_expr )
| not property_expr
| property_expr or property_expr
| property_expr and property_expr
| sequence_expr -> property_expr
| sequence_expr => property_expr
| if ( expression_or_dist ) property_expr [ else property_expr ]
| property_instance
| clocking_event property_expr
...
```

With

```
...
property_expr ::=
sequence_expr
| ( property_expr )
| not property_expr
| property_expr or property_expr
| property_expr and property_expr
| property_expr -> property_expr
| property_expr <-> property_expr
| sequence_expr -> property_expr
| sequence_expr => property_expr
| if ( expression_or_dist ) property_expr [ else property_expr ]
| property_instance
```

| clocking_event property_expr

...

17.11, page 266

Add

h) A property is a boolean implication if it has the form

`property_expr1 -> property_expr2`

This is equivalent to writing

`(not property_expr1) or property_expr2`

i) A property is a boolean equivalence if it has the form

`property_expr1 <-> property_expr2`

This is equivalent to writing

`((not property_expr1) or property_expr2) and
((not property_expr2) or property_expr1)`

17.11, Table 17-2

Replace

Sequence operators	Property operators	Associativity
[*], [=], [->]		—
##		Left
throughout		Right
within		Left
intersect		Left
	not	—
and	and	Left
or	or	Left
	if...else	Right
	->, =>	Right

With

Sequence operators	Property operators	Associativity
[*], [=], [->]		—
##		Left
throughout		Right
within		Left
intersect		Left
	not	—
and	and	Left
or	or	Left
	->, <->	Right
	if...else	Right

	->, =>	Right
--	---------	-------