REPLACE section 7.17

7.17 Set membership

SystemVerilog supports singular value sets and set membership operators.

The syntax for the set membership operators is:

```
inside_expression ::=                                              // from Annex A.8.3
        expression inside open_range_list
```

Syntax 7-2—inside expression syntax (excerpt from Annex A)

The *expression* on the left-hand side of the **inside** operator is any singular expression.

The set-membership *open_range_list* on the right-hand side of the **inside** operator is a comma-separated list of expressions or ranges. If an expression in the list is an aggregate array, its elements are traversed by descending into the array until reaching a singular value. The members of the set are scanned until a match is found and the operation returns 1'b1. Values can be repeated, so values and value ranges can overlap. The order of evaluation of the expressions and ranges is non-deterministic.

```
int a, b, c
if ( a inside {b, c} ) ...
int array[$] = {3,4,5};
if( ex inside { 1, 2, array} ) … // same as { 1, 2, 3, 4, 5}
```

The **inside** operator uses the equality (==) operator on non-integral expressions to perform the comparison. If no match is found, the **inside** operator returns 1'b0. Integral expressions also use the equality operator, except that an z inside a value in the set is treated as a don't care and that bit position shall not be considered. Note that unlike comparisons performed by the casez statement, z values in the expression on the left-hand-side are not treated as a don't-care; the don't-care is unidirectional.

```
logic [2:0] val;
while ( val inside {3'b1?1} ) … // matches 3'b101,3'b111,3'b1x1,3'b1z1
```

If no match is found, but some of the comparisons result in x, the **inside** operator shall return 1'bx. The return value is effectively the *or* reduction of all the comparisons in the set with the expression on the left-hand side.

```
wire r;
assign r= 3'bz11 inside {3'b1?1, 3'b011}; // r = 1'bx
```

A range may be specified with a low and high bound enclosed by square braces `[ ]`, and separated by a colon (`:`), as in `[low_bound:high_bound]`. A bound specified by `$` shall represent the lowest or highest value for the type of the expression on the left-hand side. A match is found if the expression on the left-hand-side is inclusively within the range. When specifying a range, the expressions must be of a singular type for which the relational operators (<=, >=) are defined. If the bound to the left of the colon is greater than the bound to the right, the range is empty and contains no values.

For example:

```
bit ba = a inside { [16:23], [32:47] };
string I;
if (I inside {"a rock":"hard place"})…//I between "a rock" and a "hard place"
```

REMOVE from section 17.9

— `$inset (<expression>, <expression> {, <expression> } )` returns true if the first expression is equal to at least one of the subsequent expression arguments.
— `$insetz (<expression>,<expression> {, <expression> } )` returns true if the first expression is equal to at least other expression argument. The comparison is performed using casez semantics, so ‘z’ or ‘?’ bits are treated as don’t-cares.

REMOVE from section 22.7 (and from BNF box 22-5)

— `$inset` returns true if the first expression is equal to at least one of the subsequent expression arguments.
— `$insetz` returns true if the first expression is equal to at least one other expression argument. Comparison is performed using **`casez`** semantics, so `Z` or `?` bits are treated as don’t-cares.

REPLACE BNF in A.8.3

```
inside_expression ::=
        expression inside { open_range_list }
```