Insert as Section 8.5.3

8.5.3 The foreach loop

The **foreach** construct specifies iteration over the elements of an array. Its argument is an identifier that designates any type of array (fixed-size, dynamic, or associative) followed by a list of loop variables enclosed in square brackets. Each loop variable corresponds to one of the dimensions of the array. The **foreach** construct is similar to a **repeat** loop that uses the array bounds to specify the repeat count instead of an expression.

```
loop_statement ::=                                              // from Annex A.6.8
        …
        | foreach ( array_identifier [ loop_variables ] ) statement

loop_variables ::= [ index_indentifier ]{ , [ index_indentifier ] }
```

Examples:

```
string words [2] = { "hello", "world" };
int prod [1:8] [1:3];

foreach( words [ j ] )
    $display( j , words[j] );    // print each index and value

foreach( prod[ k, m ] )
    prod[k][m] = k * m;          // initialize
```

The number of loop variables must match the number of dimensions of the array variable. Empty loop variable may be used to indicate no iteration over that dimension of the array, and contiguous empty loop variables towards the end may be omitted. Loop variables are automatic, read-only, and their scope is local to the loop. The type of each loop variable is implicitly declared to be consistent with the type of array index. It shall be an error for any loop variable to have the same identifier as the array.

The mapping of loop variables to array indexes is determined by the dimension cardinality, as described in Section 22.4. The **foreach** arranges for higher cardinality indexes to change more rapidly.

```
//      1   2   3             3     4        1    2     -> Dimension numbers
int A [2][3][4];     bit [3:0][2:1] B [5:1][4];

foreach( A [ i, j, k ] ) …
foreach( B [ q, r, , s ] ) …
```

The first **foreach** causes `i` to iterate from 0 to 1, `j` from 0 to 2, and `k` from 0 to 3. The second **foreach** causes `q` to iterate from 5 to 1, `r` from 0 to 3, and `s` from 2 to 1 (iteration over the 3rd index is skipped).

Multiple loop variables correspond to nested loops that iterate over the given indexes. The nesting of the loops is determined by the dimension cardinality: outer loops correspond to lower cardinality indexes. In the first example above, the outermost loop iterates over `i` and the innermost loop iterates over `k`.

When loop variables are used in expressions other than as indexes to the designated array, they are auto-cast into a type consistent with the type of index. For fixed-size and dynamic arrays the auto-cast type is **int**. For associative arrays indexed by a specific index type the auto-cast type is the same as the index type. For associative arrays indexed by a wildcard index (*) the auto-cast type is **unsigned longint**. To use different types, an explicit cast may be used.