```
concurrent_assertion_item ::=
[ block_identifier : ] concurrent_assertion_statement
| checker_instantiation

concurrent_assertion_statement ::=
assert property ( property_spec ) action_block
| assume property ( property_spec ) action_block
| cover property ( property_spec ) statement_or_null
| cover_sequence_statement

cover_sequence_statement::=
cover sequence ( [clocking_event ]
        [ disable iff ( expression_or_dist ) ]
        sequence_expr ) statement_or_null

property_instance ::=
ps_or_hierarchical_property_identifier
        [ ( [ property_list_of_arguments ] ) ]
property_list_of_arguments ::=
        [property_actual_arg] { , [property_actual_arg] }
        { , . identifier ( [property_actual_arg] ) }
        | . identifier ( [property_actual_arg] )
        { , . identifier ( [property_actual_arg] ) }
property_actual_arg ::= property_expr | sequence_actual_arg

assertion_item_declaration ::=
property_declaration
| sequence_declaration
| let_declaration

property_declaration ::=
property property_identifier [ ( [ property_port_list ] ) ] ;
{ assertion_variable_declaration }
property_spec [ ; ]
endproperty [ : property_identifier ]

property_port_list ::= property_port_item {, property_port_item}
property_port_item ::=
{ attribute_instance } [ local [ property_lvar_port_direction ] ]
        property_formal_type formal_port_identifier
        {variable_dimension} [ = property_actual_arg ]
property_lvar_port_direction ::= input
property_formal_type ::= sequence_formal_type | property

property_spec ::=
[clocking_event] [disable iff (expression_or_dist)] property_expr

property_expr ::= sequence_expr
| strong ( sequence_expr ) | weak ( 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 ]
| case ( expression_or_dist ) property_case_item
        { property_case_item } endcase
| sequence_expr #-# property_expr
| sequence_expr #=# property_expr
| nexttime property_expr
| nexttime [ constant _expression ] property_expr
| s_nexttime property_expr
| s_nexttime [ constant_expression ] property_expr
| always property_expr
| always [ cycle_delay_const_range_expression ] property_expr
| s_always [ constant_range] property_expr
| s_eventually property_expr
| eventually [ constant_range ] property_expr
| s_eventually [cycle_delay_const_range_expression] property_expr
| property_expr until property_expr
| property_expr s_until property_expr
| property_expr until_with property_expr
| property_expr s_until_with property_expr
| property_expr implies property_expr
| property_expr iff property_expr
| accept_on ( expression_or_dist ) property_expr
| reject_on ( expression_or_dist ) property_expr
| sync_accept_on ( expression_or_dist ) property_expr
| sync_reject_on ( expression_or_dist ) property_expr
| property_instance
| clocking_event property_expr
```

```
property_case_item ::=
expression_or_dist { , expression_or_dist } : property_expr ;
| default [ : ] property_expr ;

sequence_declaration ::=
sequence sequence_identifier [ ( [ sequence_port_list ] ) ] ;
{ assertion_variable_declaration }
sequence_expr [ ; ]
endsequence [ : sequence_identifier ]

sequence_port_list ::=
sequence_port_item {, sequence_port_item}
sequence_port_item ::=
{ attribute_instance } [ local
        [sequence_lvar_port_direction]] sequence_formal_type
formal_port_identifier{variable_dimension}
        [= sequence_actual_arg]
sequence_lvar_port_direction ::= input | inout | output
sequence_formal_type ::= data_type_or_implicit
| sequence | untyped

sequence_expr ::=
cycle_delay_range sequence_expr
        {cycle_delay_range sequence_expr}
| sequence_expr cycle_delay_range sequence_expr
        { cycle_delay_range sequence_expr }
| expression_or_dist [ boolean_abbrev ]
| sequence_instance [ sequence_abbrev ]
| (sequence_expr {, sequence_match_item }) [sequence_abbrev]
| sequence_expr and sequence_expr
| sequence_expr intersect sequence_expr
| sequence_expr or sequence_expr
| first_match ( sequence_expr {, sequence_match_item} )
| expression_or_dist throughout sequence_expr
| sequence_expr within sequence_expr
| clocking_event sequence_expr

cycle_delay_range ::=
## constant_primary
| ## [ cycle_delay_const_range_expression ]
| ##[*] | ##[+]
```

```
sequence_method_call ::= sequence_instance.method_identifier

sequence_match_item ::=
operator_assignment
| inc_or_dec_expression
| subroutine_call

sequence_instance ::=
ps_or_hierarchical_sequence_identifier
        [ ( [ sequence_list_of_arguments ] ) ]
sequence_list_of_arguments ::=
[sequence_actual_arg] { , [sequence_actual_arg] }
        { , . identifier ( [sequence_actual_arg] ) }
| . identifier ( [sequence_actual_arg] ) { , . identifier
        ( [sequence_actual_arg] ) }
sequence_actual_arg ::= event_expression | sequence_expr

boolean_abbrev ::=
consecutive_repetition
| non_consecutive_repetition
| goto_repetition

sequence_abbrev ::= consecutive_repetition

consecutive_repetition ::=
[* const_or_range_expression ] | [*] | [+]
non_consecutive_repetition ::= [= const_or_range_expression ]
goto_repetition ::= [-> const_or_range_expression ]

const_or_range_expression ::=
constant_expression
| cycle_delay_const_range_expression
cycle_delay_const_range_expression ::=
constant_expression : constant_expression
| constant_expression : $

expression_or_dist ::= expression [ dist { dist_list } ]

assertion_variable_declaration ::=
var_data_type list_of_variable_decl_assignments ;
```