```
actual_assertion_parameter ::=
  formal_identifier : actual_assertion_expression
actual_assertion_parameter_list ::=
  actual_assertion_parameter { , actual_assertion_parameter }*
assertion ::=
  << ( assertion_predicate
  | assertion_function
  | assertion_enumeration ) >>
assertion_add_subtract ::=
  assertion_multiply_divide
  [  { + assertion_multiply_divide }+
     |  - assertion_multiply_divide ]
assertion_annex_library ::=
  annex Assertion {** [ ghost_variables ]
   { labeled_assertion }+ **} ;
assertion_enumeration ::=
  asserion_enumeration_label_identifier :
    parameter_identifier ~ enumeration_type_identifier +=>
  enumeration_pair { , enumeration_pair }*
assertion_exponentiation ::=
  assertion_subexpression [ ** assertion_subexpression ]
assertion_expression ::=
  sum logic_variables [ logic_variable_domain ]
    of assertion_expression
  | product logic_variables [ logic_variable_domain ]
    of assertion_expression
  | numberof logic_variables [ logic_variable_domain ]
    that subpredicate
  | assertion_add_subtract
assertion_function ::=
  [ label_identifier : [ variable_list ] ]
   returns type_or_reference
   := ( assertion_expression | conditional_assertion_function )
assertion_function_invocation ::=
  assertion_function_identifier
    ( [ assertion_expression |
    actual_assertion_parameter
    { , actual_assertion_parameter }* ] )
assertion_multiply_divide ::=
  assertion_exponentiation
  [ ( / | div | mod | rem ) assertion_exponentiation
    | { * assertion_exponentiation }+ ]
assertion_predicate ::=
  [ label_identifier : [ variable_list ] : ]
  predicate
assertion_range ::=
  assertion_subexpression range_symbol assertion_subexpression
assertion_subexpression ::=
  [ - | abs | truncate | round ] timed_expression
assertion_value ::=
  now
  | tops
  | timeout
  | value_constant
  | variable_name
  | port_value
conditional_assertion_expression ::=
  ( predicate ?? assertion_expression : assertion_expression )
  | ( if predicate then assertion_expression
    else assertion_expression )
conditional_assertion_function ::=
    [ condition_value_pair { , condition_value_pair }* ]
condition_value_pair ::= ( predicate )->  assertion_expression
enumeration_pair ::=
  enumeration_literal_identifier -> predicate
existential_quantification ::=
  exists logic_variables
  ( in assertion_subexpression range_symbol
      assertion_subexpression
  | which predicate )
  that predicate
ghost_variables ::= ghost variables { ghost_variable }+
ghost_variable ::= def ghost_variable
index_expression ::=
  period_shift
  [ - period_shift
  | div period_shift
  | mod period_shift
  | { + period_shift }+
  | { * period_shift }+ ]
logic_variable_domain ::=
  in assertion_expression range_symbol assertion_expression
  | which predicate
logic_variables ::= logic_variable { , logic_variable }*
parenthesized_assertion_expression ::=
  ( assertion_expression )
  | conditional_assertion_expression
  | record_term
parenthesized_predicate ::= ( predicate )
period_shift ::= [ - ] ( integer_value | ( index_expression ) )
predicate ::=
  universal_quantification
  | existential_quantification
  | predicate_disjunction
    [ ( implies | iff ) predicate_disjunction ]
predicate_conjunction ::=
  subpredicate
  [ { and subpredicate }+
  | { and then subpredicate }+ ]
```

```
predicate_disjunction ::=
  predicate_conjunction
  [ { or predicate_conjunction }+
  | { or else predicate_conjunction }+
  | { xor predicate_conjunction }+ ]
predicate_invocation ::=
  assertion_identifier ( [ assertion_expression
   | actual_assertion_parameter_list  ] )
predicate_relation ::=
  assertion_subexpression relation_symbol assertion_subexpression
  | assertion_subexpression in assertion_range
  | shared_integer_name += assertion_subexpression
range_symbol ::= .. | ,. | ., | ,,
relation_symbol ::= = | < | > | <= | >= | != | <>
subpredicate ::=
  predicate_relation
  | [ not ] timed_predicate
```

```
timed_expression ::=
  ( assertion_value
  | parenthesized_assertion_expression
  | assertion_function_invocation )
  [ ' | ^ period_shift | @ time_subexpression ]
timed_predicate ::=
  ( name
  | parenthesized_predicate
  | predicate_invocation )
  [ ' | @ time_subexpression | ^ period_shift ]
universal_quantification ::=
  all logic_variables logic_variable_domain
  are predicate
variable ::= identifier ~ type_or_reference
variable_list ::= variable { , variable }*
```