```
action ::=
  basic_action
  | behavior_action_block
  | alternative
  | for_loop
  | forall_action
  | while_loop
  | do_until_loop
actual_parameter ::=
  [ formal_parameter_identifier : ]
  ( variable_name | value_constant )
add_subtract ::=
  multiply_divide
  [ { + multiply_divide }+
  | - multiply_divide ]
alternative ::=
  if guarded_action { [] guarded_action }+ fi
  |
  if ( boolean_expression ) behavior_actions
  { elsif ( boolean_expression )
    behavior_actions }*
  [ else behavior_actions ]
  end if
asserted_action ::=
  [ precondition_assertion ]
  action
  [ postcondition_assertion ]
assignment ::=
  assignment_target := ( expression | record_term | any )
(for subprograms)
assignment_target ::=
  variable_name
  | outgoing_subprogram_parameter_identifier
(for subprograms)
basic_action ::=
  skip
  | assignment
  | simultaneous_assignment
  | when_throw
  | subprogram_call
  | combinable_operation
behavior_action_block ::=
  [ local_variables ] { behavior_actions }
  [ timeout behavior_time ]  [ catch_clause ]
behavior_actions ::=
  asserted_action
  | sequential_composition
  | concurrent_composition
behavior_time ::=
  numeric_constant
  | variable_name
  | port_value
  | parenthesized_expression
case_choice ::= ( boolean_expression -> expression )
case_expression ::= case { case_choice }+
catch_clause ::= catch { ( exception_label : basic_action ) }+
combinable_operation ::=
 fetchadd
 ( target_variable_name ,
   arithmetic_expression [, result_identifier] )
 | ( fetchor | fetchand | fetchxor )
 ( target_variable_name , boolean_expression
   [, result_identifier] )
 | swap
 ( target_variable_name , reference_variable_name
   , result_identifier )
component_element_reference ::=
  subcomponent_identifier
  | bound_prototype_identifier
  | feature_identifier
  | self
concurrent_composition ::=
  asserted_action { & asserted_action }+
conditional_expression ::=
  ( boolean_expression ?? expression : expression )
  | ( if boolean_expression then expression else expression )
conjunction ::=
  relation
  [ { and relation }+
  | { and then relation }+ ]
disjunction ::=
  conjunction
  [ { or conjunction }+
  | { or else conjunction }+
  | { xor conjunction }+ ]
do_until_loop ::=
  do
  [ invariant assertion ]
  [ bound integer_expression ]
  behavior_actions
  until ( boolean_expression )
enumeration_value ::=
  enumeration_type_identifier ' enumeration_literal_identifier
exception_label ::= { exception_identifier }+ | all
exponentiation ::=
  subexpression
  [ ** subexpression ]
```

```
expression ::=
  disjunction
  [ ( iff | implies ) disjunction ]
for_loop ::=
  for integer_identifier
    in integer_expression .. integer_expression
  [ invariant assertion ]
  { behavior_actions }
forall_action ::=
  forall { variable_identifier }+
    in integer_expression .. integer_expression
    behavior_action_block
function_call ::=
  { package_identifier :: }*
  function_identifier (
    [ function_parameter {  , function_parameter }+
    | expression ] )
function_parameter ::= formal_identifier : actual_expression
guarded_action ::= ( boolean_expression )~> asserted_action
integer_expression_or_range ::=
  integer_expression [ ..  integer_expression ]
issue_exception ::=
  exception ( exception_identifier [ message_string_literal ] )
local_variables ::= declare { variable }+
multiply_divide ::=
  exponentiation
  [ { * exponentiation }+
  | ( / | div | mod | rem ) exponentiation ]
name ::=
  root_identifier { [ integer_expression_or_range ] }*
    { . field_identifier { [ integer_expression_or_range ] }* }*
numeric_constant ::=
  quantity
  | property_constant
  | property_reference
parameter_list ::= actual_parameter { , actual_parameter }*
property_constant ::=
  property_set_identifier :: property_constant_identifier
property_field ::=
  [ integer_value ]
  | . field_identifier
  | . upper_bound
  | . lower_bound
property_name ::=
  [ property_set_identifier ::  ]
  property_identifier { property_field }*
property_reference ::=
  ( #
  | component_element_reference #
  | unique_component_classifier_reference #
  | self #  )
  property_name
quantity ::= numeric_literal [ unit_identifier | scalar ]
record_term ::= ( { record_value }+ )
record_value ::= field_identifier => value [ ; ]
relation ::=
  add_subtract
  [ relation_symbol add_subtract
  | in range ]
sequential_composition ::= asserted_action { ; asserted_action }+
simultaneous_assignment ::=
  | variable_name [ ' ] { , variable_name [ ' ] }+
  := ( expression | record_term | any )
    { , ( expression | record_term | any ) }+ |
subexpression ::=
  [ - | not | abs | truncate | round ]
  ( value
  | ( expression )
  | conditional_expression
  | case_expression )
subprogram_annex_subclause ::=
  annex Action {** subprogram_behavior **} ;
subprogram_behavior ::=
  [ throws { exception_identifier }+ ]
  [ assert { assertion }+ ]
  [ pre assertion ]
  [ post assertion ]
  [ invariant assertion ]
  behavior_action_block
subprogram_call ::= subprogram_name ( [parameter_list] )
subprogram_name ::= required_subprogram_access_name
unique_component_classifier_reference ::=
  { package_identifier :: }* component_type_identifier
  [ . component_implementation_identifier ]
(for subprograms)
value ::=
  variable_name | value_constant | function_call
  | enumeration_value | incoming_subprogram_parameter_identifier
value_constant ::=
  true | false | null | quantity | string_literal
  | property_constant | property_reference
when_throw ::=
  when ( boolean_expression ) throw exception_identifier
  [ message_string_literal ]
while_loop ::=
  while ( boolean_expression )
  [ invariant assertion ]
  [ bound integer_expression ]
  behavior_action_block
```