```
(for threads)
assignment_target ::=
  ( variable_name
  | outgoing_port_name
  | internal_port_name )
  [ ' ]
(for threads)
basic_action ::=
  skip
  | assignment
  | simultaneous_assignment
  | when_throw
  | combinable_operation
  | communication_action
  | setmode mode_identifier
behavior ::=
  [ assert { assertion }+ ]
  [ invariant assertion ]
  [ variables { variable_declaration }+ ]
  [ states { behavior_state }+ ]
  [ transitions { behavior_transition }+ ]
behavior_state ::=
  behavior_state_identifier :
  [ initial | complete | final ] state
  [ assertion ] [ ; ]
bless_annex_subclause ::= annex BLESS {** behavior **} ;
communication_action ::=
  subprogram_call
  | output_port_identifier ! [ ( expression ) ]
  | input_port_identifier ? ( local_variable_name )
completion_relative_timeout ::= timeout behavior_time
dispatch_condition ::= on dispatch [ dispatch_expression ]
dispatch_conjunction ::=
  dispatch_trigger { and dispatch_trigger }*
dispatch_expression ::=
  dispatch_conjunction { or dispatch_conjunction }*
dispatch_relative_timeout ::= timeout
dispatch_trigger ::=
  in_event_port_name
  | in_event_data_port_name
  | port_event_timeout_catch
  | dispatch_condition_reletive_timeout
  | completion_reletive_timeout
  | stop
execute_condition ::= boolean_expression
internal_condition ::=
  on internal internal_port_name { or internal_port_name }*
modifier ::=
  nonvolatile | constant | shared | spread | final
port_name ::= port_identifier [ [ natural_literal ] ]
port_event_timeout ::=
  timeout ( port_identifier { [ or ] port_identifier }* )
    behavior_time
port_value ::=
  in_port_name [ ? | 'count | 'fresh | 'updated ]
transition ::=
  transition_label :
  source_state_identifier { , source_state_identifier }*
  -[ [ transition_condition ] ]->
  destination_state_identifier
  [ { [ behavior_actions ] } ] [ ; ]
transition_condition ::=
  dispatch_condition
  | execute_condition
  | internal_condition
(for threads)
value ::=
  now | tops | value_constant | port_value
  | variable_name | function_call | enumeration_value
variable_declaration ::=
  variable [ modifier ] [ := constant_expression ]
    [ assertion ] [ ; ]
```