Assertion Syntax Card v0.9 (May 3, 2016)

```
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_enumeration_invocation ) >>

assertion_annex_library ::=
  annex Assertion {** { assertion }+ **} ;

assertion_enumeration ::=
  assertion_enumeration_label_identifier :  parameter_identifier
  +=> enumeration_pair { , enumeration_pair }*

assertion_enumeration_invocation ::=
  +=> asserion_enumeration_label_identifier
  ( actual_assertion_parameter )

assertion_expression ::=
  assertion_subexpression
    [ { + assertion_subexpression }+
    | { * assertion_subexpression }+
    | - assertion_subexpression
    | / assertion_subexpression
    | ** assertion_subexpression
    | mod assertion_subexpression
    | rem assertion_subexpression ]
  | 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_function ::=
  [ label_identifier : [ formal_assertion_parameter_list ] ]
   := ( assertion_expression | conditional_assertion_function )

assertion_function_invocation ::=
  assertion_function_identifier ( [ assertion_expression |
  actual_assertion_parameter { , actual_assertion_parameter }* ] )


assertion_predicate ::=
  [ label_identifier : [ formal_assertion_parameter_list ] : ]
  predicate

assertion_range ::=
  assertion_subexpression range_symbol assertion_subexpression

assertion_subexpression ::=
  [ - | abs ]  timed_expression
  | assertion_type_conversion

assertion_type_conversion ::=
  ( natural | integer | rational | real | complex | time )
  parenthesized_assertion_expression

assertion_value ::=
  now | tops | timeout | value_constant | variable_name
  | assertion_function_invocation | port_value

component_element_reference ::=
  subcomponent_identifier | bound_prototype_identifier
  | feature_identifier | self

conditional_assertion_expression ::=
  ( predicate ?? assertion_expression : assertion_expression )

conditional_assertion_function ::=
    condition_value_pair { , condition_value_pair }*

condition_value_pair ::=
    parenthesized_predicate ->  assertion_expression

enumeration_pair ::= enumeration_literal_identifier -> predicate

event ::= < port_variable_or_state_identifier >

event_expression ::=
  [not] event
  | event_subexpression (and event_subexpression)+
  | event_subexpression (or event_subexpression)+
  | event - event

event_subexpression ::=
  [ always | never ] ( event_expression ) | event

existential_quantification ::=
  exists logic_variables logic_variable_domain
  that predicate

formal_assertion_parameter ::= parameter_identifier [ ~ type_name ]


formal_assertion_parameter_list ::=
  formal_assertion_parameter { , formal_assertion_parameter }*

index_expression_or_range ::=
    integer_expression [ ..  integer_expression ]

integer_expression ::=
  [ - ]
  ( integer_assertion_value
  | ( integer_expression - integer_expression )
  | ( integer_expression / integer_expression )
  | ( integer_expression { + integer_expression }+ )
  | ( integer_expression { * integer_expression }+ ) )
```

```
logic_variable_domain ::=
  in ( assertion_expression range_symbol assertion_expression
    | predicate )

logic_variables ::=
  logic_variable_identifier { , logic_variable_identifier }*
  : type

name ::=
  root_identifier { [ index_expression_or_range ] }*
    { . field_identifier { [ index_expression_or_range ] }* }*

parenthesized_assertion_expression ::=
  ( assertion_expression )
  | conditional_assertion_expression
  | record_term

parenthesized_predicate ::= ( predicate )

port_name ::=
  { subcomponent_identifier . }* port_identifier
    [ [ natural_literal ] ]

port_value ::=
  in_port_name ( ? | 'count | 'fresh | 'updated )

predicate ::=
  universal_quantification
  | existential_quantification
  | subpredicate
    [ { and subpredicate }+
    | { or subpredicate }+
    | { xor subpredicate }+
    | implies subpredicate
    | iff subpredicate
    | -> subpredicate ]

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

property_constant ::=
  property_set_identifier :: property_constant_identifier

property_field ::=
  [ integer_value ]  | . field_identifier
  | . upper_bound | . lower_bound

property_name ::= property_identifier { property_field }*

property_reference ::=
  ( # [ property_set_identifier :: ]
  | component_element_reference #
  | unique_component_classifier_reference #
  | self #  )
  property_name

range_symbol ::= .. | ,. | ., | ,,

relation_symbol ::= = | < | > | <= | >= | != | <>

subpredicate ::=
  [ not ]
  ( true | false | stop
  | predicate_relation
  | timed_predicate
  | event_expression
  | def logic_variable_identifier )

time_expression ::=
  time_subexpression
  | time_subexpression - time_subexpression
  | time_subexpression / time_subexpression
  | time_subexpression { + time_subexpression }+
  | time_subexpression { * time_subexpression }+

time_subexpression ::= [ - ]
  ( time_assertion_value
  | ( time_expression )
  | assertion_function_invocation )

timed_expression ::=
    ( assertion_value
      | parenthesized_assertion_expression
      | predicate_invocation )
    [ ' | ^ integer_expression | @ time_expression ]

timed_predicate ::=
  ( name | parenthesized_predicate | predicate_invocation )
  [ ' | @ time_expression | ^ integer_expression ]

type_name ::=
  { package_identifier :: }* data_component_identifier
    [ . implementation_identifier ]
  | natural | integer | rational | real
  | complex | time | string

unique_component_classifier_reference ::=
  { package_identifier :: }* component_type_identifier
  [ . component_implementation_identifier ]

universal_quantification ::=
  all logic_variables logic_variable_domain
  are predicate

value_constant ::=
  true | false | numeric_literal | string_literal
  | property_constant | property_reference
```