

Mantis 928: The purpose is to fix issues with the existing BNF. Specifically:

1. removal of `list_of_formals`, `formal_list_item`, and `actual_arg_expr` that were not referenced
2. `list_of_arguments` was redefined as `sequence_list_of_arguments` and `property_list_of_arguments`. `actual_arg` was defined as `sequence_actual_arg` and `property_actual_arg`. They are different in that sequences cannot have property arguments.
3. `tf_port_list` was replaced by `sequence_port_list` and `property_port_list` to fix the issue that `tf_port_list` does not allow default values assignment other than expression. The new definition allows for initialization of all args in the definition.. Named or positional association of arguments is allowed when the sequence or property is instantiated.
4. Actual args for a sequence include `sequence_instance` or `event_expression` (which includes expression). Actual args for a property are the same as for a sequence with the addition of a `property_instance`.

=====

REPLACE (and also 17-14)

A.2.10 Assertion declarations

```
...
property_instance ::=
    property_identifier [ ( [ list_of_arguments ] ) ]
concurrent_assertion_item_declaration ::=
    property_declaration
    | sequence_declaration
property_declaration ::=
    property property_identifier [ ( [ tf_port_list ] ) ] ;
    { assertion_variable_declaration }
    property_spec ;
    endproperty [ : property_identifier ]
property_spec ::=
    [ clocking_event ] [ disable iff ( expression_or_dist ) ] property_expr
property_expr ::=
    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 ]
    | property_instance
    | clocking_event property_expr
sequence_declaration ::=
    sequence sequence_identifier [ ( [ tf_port_list ] ) ] ;
    { assertion_variable_declaration }
    sequence_expr ;
    endsequence [ : sequence_identifier ]
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 ]
| ( expression_or_dist { , sequence_match_item } ) [ 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 ::=
    ## integral_number
    | ## identifier
    | ## ( constant_expression )
    | ## [ 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 ::=
    sequence_identifier [ ( [ list_of_arguments ] ) ]
formal_list_item ::=
    formal_identifier [ = actual_arg_expr ]
list_of_formals ::= formal_list_item { , formal_list_item }
actual_arg_expr ::=
    event_expression
    | $

```

WITH

A.2.10 Assertion declarations

```

property_instance ::=
    ps_property_identifier [ ( [ list_of_arguments 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_instance
    | sequence_actual_arg
concurrent_assertion_item_declaration ::=
    property_declaration
    | sequence_declaration
property_declaration ::=
    property property_identifier [ ( [ #_port_list 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 }
    property_formal_type
    port_identifier {variable_dimension} [=expression]
property_formal_type ::=
    data_type_or_implicit
property_spec ::=
    [clocking_event] [ disable iff ( expression_or_dist ) ] property_expr
property_expr ::=
    sequence_expr
    | sequence_instance
    | ( 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 ]
    | property_instance
    | clocking_event property_expr
sequence_declaration ::=
    sequence sequence_identifier [ ( [ tf_port_list 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 }
    sequence_formal_type
    port_identifier {variable_dimension} [=expression]
sequence_formal_type ::=
    data_type_or_implicit
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 ]
    | ( expression_or_dist {, sequence_match_item } ) [ 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

...
sequence_instance ::=
    ps_sequence_identifier [ ( [ list_of_arguments sequence_list_of_arguments ] ) ]
formal_list_item ::=

```

```
formal_identifier [= actual_arg_expr]
list_of_formals ::= formal_list_item { , formal_list_item }
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 ::=
    sequence_instance
    | event_expression
```

....