OVERVIEW:

With mantis 928, formal arguments to properties and sequences are defined to apply to a list of arguments that follow, much like tasks and function arguments. Previously, the type had to be replicated for each list item. Untyped arguments must therefore be listed first before any typed arguments. The “context” type is introduced to allow arguments that do not have any data type restrictions to be mixed freely with those that do.

==========

REPLACE

A.2.10 Assertion declarations

sequence_formal_type ::=
    data_type_or_implicit

property_formal_type ::=
    data_type_or_implicit

WITH

A.2.10 Assertion declarations

sequence_formal_type ::=
    data_type_or_implicit
    | **context**

property_formal_type ::=
    ~~data_type_or_implicit~~
    sequence_formal_type

REPLACE Syntax 16-2 from section 16.6

sequence_formal_type ::=
    data_type_or_implicit

WITH

sequence_formal_type ::=
    data_type_or_implicit
    | **context**

REPLACE Syntax 16-4 from section 16.7.

sequence_formal_type ::=
    data_type_or_implicit

WITH

sequence_formal_type ::=
    data_type_or_implicit
    | **context**

REPLACE

16.7.1 Typed formal arguments in sequence declarations

Formal arguments of sequences can optionally be typed. To declare a type for a formal argument of a sequence, it is required to prefix the argument with a type. A formal argument that is not prefixed by a type shall be untyped.
A type name can refer to a comma-separated list of arguments. Untyped arguments must therefore be listed before any typed arguments.

Exporting values of local variables through typed formal arguments is not supported.

The supported data types for sequence formal arguments are the types that are allowed for operands in assertion
expressions (see 16.5.1). The assignment rules for assigning actual argument expressions to formal arguments, at the time of sequence instantiation, are the same as the general rules for doing assignment of a
typed variable with a typed expression (see Clause 6).

For example, two equivalent ways of passing arguments are shown below. The first has untyped arguments,
and the second has typed arguments:

```
sequence rule6_with_no_type(x, y);
##1 x ##[2:10] y;
endsequence

sequence rule6_with_type(bit x, bit y);
##1 x ##[2:10] y;
endsequence
```

WITH

16.7.1 Typed formal arguments in sequence declarations

~~Formal arguments of sequences can optionally be typed. To declare a type for a formal argument of a sequence, it is required to prefix the argument with a type. A formal argument that is not prefixed by~~

~~a type will be untyped. A type name can refer to a comma-separated list of arguments. Untyped arguments must therefore be listed before any typed arguments.~~

~~Exporting values of local variables through typed formal arguments is not supported.~~

~~The supported data types for sequence formal arguments are the types that are allowed for operands in assertion expressions (see 16.5.1).~~
A formal argument of a sequence may be typed by specifying the data type prior to the formal argument identifier. A data type shall apply to all formal arguments whose identifiers both follow the data type and precede the next data type, if any, specified in the formal argument list.

The data type specified for a formal argument of a sequence may be the keyword **`context`**. A formal argument of a sequence is said to be *contextual* if either its data type is **`context`** or there is no data type preceding its identifier in the formal argument list. The semantics of binding an actual argument expression to a contextual formal argument shall be the same regardless of which of these two criteria is satisfied. This semantics is described in Subclause 16.7. The keyword **`context`** shall be used if a contextual formal argument follows a data type in the formal argument list.

The supported data types for sequence formal arguments are the types that are allowed for operands in assertion expressions (see 16.5.1) and the keyword **`context`**.

Local variable values may be exported from a sequence only through contextual formal arguments.

The assignment rules for assigning actual argument expressions to formal arguments, at the time of sequence instantiation, are the same as the general rules for doing assignment of a typed variable with a typed expression (see Clause 6).

For example, two ways of declaring arguments are shown below. All of the formal arguments of foo1 are contextual. The formal arguments w and y of foo2 are contextual, while the formal argument x has data type bit.

```
sequence foo1(w, x, y);
  w ##1 x ##[2:10] y;
endsequence

sequence foo2(w, bit x, context y);
  w ##1 x ##[2:10] y;
endsequence
```

The following instances of foo1 and foo2 are equivalent:

```
foo1(.w(a), .x(bit'(b)), .y(c))

foo2(.w(a), .x(b), .y(c))
```

~~For example, two equivalent ways of passing arguments are shown below. The first has untyped arguments,~~

~~and the second has typed arguments~~

```
~~sequence rule6_with_no_type(x, y);~~
~~##1 x ##[2:10] y;~~
~~endsequence~~

~~sequence rule6_with_type(bit x, bit y);~~
~~##1 x ##[2:10] y;~~
~~endsequence~~
```

REPLACE Syntax 16-14 from section 16.12

property_formal_type ::=
    data_type_or_implicit

WITH

property_formal_type ::=
    ~~data_type_or_implicit~~
    sequence_formal_type

REPLACE

16.12.1 Typed formal arguments in property declarations

Formal arguments of properties can optionally be typed. To declare a type for a formal argument of a property, it is required to prefix the argument with a type. A formal argument that is not prefixed by a type shall be untyped. A type name can refer to a comma-separated list of arguments. Untyped arguments must therefore be listed before any typed arguments.

The supported types for property formal arguments include all the types that are allowed for sequences.

WITH

16.12.1 Typed formal arguments in property declarations

Formal arguments of properties can optionally be typed. ~~To declare a type for a formal argument of a property, it is required to prefix the argument with a type. A formal argument that is not prefixed by a type shall be untyped. A type can refer to a comma-separated list of arguments. Untyped arguments must therefore be listed before any typed arguments.~~ The supported types for property formal arguments include all the types that are allowed for sequences. Refer to 16.7.1.