

PREAMBLE

On 2007-08-18, the P1800 Champions offered the following feedback for this proposal.

1601 new keyword for untyped formal arguments

svac Passed by e-mail vote on 2007-07-31, 8y/0n/0a This is an enhancement

- I am opposed to this enhancement at the current time. It is unfortunate that we ever allowed un-typed formals. The language should be fixed properly so that un-typed formal are obsolete rather than allowing them to be mixed with typed formals.
- this proposal could affect future changes with respect to a variable number of arguments.
- untyped formals weren't well thought out originally.
- is there a real need to mix typed and untyped formals?
- can put the untyped arguments first.

The only reason for this proposal is to allow them in the middle.

Doesn't see the ordering of args making a strong case for this.

- doesn't like adding new keywords using the word context in a different way here (not good either).
- we need untyped arguments in other parts of the language. we need to make sure that we are ok with whatever is allowed in svac.
- 'untyped' could possibly be used in place of context
- still opposed, even by using the new keyword untyped.

By rejecting this proposal, only effect is to force untyped to be first. Not a major restriction.

Resolution from Champions:

Send 1601 back to the sv-ac to use 'untyped' in place of 'context' and request a justification for the proposal.

The 1601 proposal has been changed to use the keyword "untyped" rather than "context" as suggested by the Champions. In response to these comments and in defense of the proposal, SV-AC make the following arguments to the Champions.

1. We are not in a position to rewrite the history of the assertions constructs within SystemVerilog. There is a basic matter of fact that untyped arguments to sequences and properties exist and have existed since the Accellera SystemVerilog 3.1 standard in June 2003. In that version of the standard, only untyped arguments were allowed to sequences and properties.

2. Since the release of the Accellera SystemVerilog 3.1 standard, there has been growing support of SystemVerilog Assertions in tools available commercially and otherwise. In parallel, there has been increasing industrial development of assertion code in alignment with the Accellera SystemVerilog 3.1 standard and its subsequent revisions. Regardless of one's estimation of the technical merit of untyped arguments in SystemVerilog Assertions, the elimination or punitive treatment of this capability has significant consequences for users, especially early adoptors, of the language.

3. In IEEE Std 1800-2005, typed formal arguments to sequences and properties were introduced as an option. There were examples in that version of the standard that showed that a type in a formal argument list of a sequence or property applies only to the next formal argument identifier, not to further comma-separated identifiers. In the 1800-2008 effort, SV-AC have clarified that types in formal argument lists of sequences and properties apply to successive identifiers that are separated only by commas. We recognize that this is disruptive to existing tools and code that followed IEEE Std 1800-2005 and its examples. The current proposal is partly intended to mitigate the disruptive impact by allowing the insertion of the untyped keyword in the declarations of the affected constructs.

4. This proposal is intended to enable the migration from untyped to typed arguments by providing the user the flexibility to write the formal arguments in any order and to specify each argument as either typed or untyped as the user sees fit. Existing libraries may be reimplemented to use types on some arguments and not on others without disrupting the argument order, which may be chosen for reasons that deserve our respect. For some kinds of arguments, it is not yet possible to pass them through a typed formal argument.

5. The comments of the Champions notwithstanding, we believe that there is technical merit in the untyped argument capability itself. Specifying a type without the ability to parameterize the type is problematic when the same declared form is required for a number of different types. Untyped arguments help to avoid this dilemma. For example,

```
sequence ones(v);  
  
    (2 * $countones(v) > $bits(v))*2];  
  
Endsequence
```

matches if the majority of the vector bits of v are ones in two consecutive cycles. One can argue that adding parameters to the sequence and property constructs is an alternative solution, but this solution requires significant work, including consideration of corresponding enhancement for tasks and functions, is not obviously technically preferable, and is not universally preferred.

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 “untyped” 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
 | **untyped**

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
 | **untyped**

REPLACE Syntax 16-4 from section 16.7.

sequence_formal_type ::=
 data_type_or_implicit

WITH

```
sequence_formal_type ::=  
    data_type_or_implicit  
    | untyped
```

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 **untyped**. A formal argument is said to be *untyped* if either its data type is **untyped** or there is no data type preceding its identifier in the formal argument list. The semantics of binding an actual argument expression to an untyped 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 **untyped** shall be used if an untyped 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 **untyped**.

Local variable values may be exported from a sequence only through untyped 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 untyped. The formal arguments `w` and `y` of `foo2` are untyped, 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, untyped 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.