Proposal for formal semantics for property and sequence instances

SV-AC

April 25, 2007

Replace at section E.1

c) The abstract syntax simplifies the assertion language by eliminating some features that tend to encumber the definition of the formal semantics.

1) The abstract syntax eliminates local variable declarations. The semantics of local variables is written with implicit types.

2) The abstract syntax eliminates instantiation of sequences and properties. The semantics of an assertion with an instance of a sequence or nonrecursive property is the same as the semantics of a related assertion obtained by replacing the sequence or nonrecursive property instance with an explicitly written sequence or property. The explicit sequence or property is obtained from the body of the associated declaration by substituting actual arguments for formal arguments. A separate subclause defines the semantics of instances of recursive properties in terms of the semantics of instances of nonrecursive properties.

with

c) The abstract syntax simplifies the assertion language by eliminating some features that tend to encumber the definition of the formal semantics.

1) The abstract syntax eliminates local variable declarations. The semantics of local variables is written with implicit types.

2) The abstract syntax eliminates instantiation of sequences and properties. The semantics of an assertion with an instance of a sequence or nonrecursive property is the same as the semantics of a related assertion obtained by replacing the sequence or nonrecursive property instance with an explicitly written sequence or property. ~~The explicit sequence or property is obtained from the body of the associated declaration by substituting actual arguments for formal arguments. A separate subclause defines the semantics of instances of recursive properties in terms of the semantics of instances of nonrecursive properties.~~ The definition of rewriting an assertion without property and sequence instances is given in clause .[Note to the editor: add link to the new sub section, which is described in the next item].

Add after E.3.1

.0.1 Rewriting property and sequence instances

This section describes an algorithm for rewriting a property_spec that contains property and/or sequence instances. The result of the algorithm is an assertion with one "flat" property expression. The semantics of a hierarchical property expression is then defined to be the semantics of the flat property_spec resulting from the rewriting algorithm. In particular, if the result is illegal, then so is the source.

For the rewriting algorithm, an auxiliary function *$var()* is defined. `$var` may be applied on every system verilog expression, returning the exact same expression. Let *e* be a system verilog expression. Then, every operation that is allowed on a variable with the same type of *e* is allowed on `$var(`*e*`)`. In particular, any operation that is allowed on a declared property or declared sequence, is allowed on *$var* applied on a property expression or a sequence expression respectively. Note that for some expressions *e*, there are operations that are allowed on variables *v* with the same type but not on *e*. For example, let `a, b` be variables of type `logic [0:1]`, let `v` be a variable of type `logic [0:3]`, and *e* the expression `(logic [3:0])'{a,b}`. Then, the result of applying a bit select operation on `v v[1:2]` is a valid expression, but `(logic [3:0])'{a,b}[0:1]` is not. Introducing the $var function enable usage of the form `$var((logic [3:0])'{a,b})[1:2]`. For expressions with undefined type, $var does not enable additional operations. $var is not a system verilog function and its only use is in the rewriting algorithm.

The rewrite algorithm
Given a property spec expression inside an assertion declaration:
While there are property instances in the property_spec do:
begin
    Select an arbitrary property instance `P`
    flatten_property(`P`)
end

While there are sequence instances in the property_spec do:
begin
    Select an arbitrary sequence instance R
    flatten_sequence(R)
end

flatten_property(P)
begin

1. Create a copy P' of the declaration of P.
2. For every local variable declared in P', make its name unique with respect to the names of the local variables declared in the property_exper of the assertion.
3. Replace every formal argument f in the property_expr of P' that is not on the left hand side of an assignment, with $var(f).
4. For every occurrence of a typed formal argument f of P' with a simple type t[see A.2.2.1] in the body of P', replace f with (t)'f (cast f to type t). See cast definition at (4.14).
5. Replace every formal argument f in the property_expr of P', with (a) where a is the corresponding actual argument expressions in the instance of P'.
6. In the property_expr of the assertion, replace the instance of P with property_expr of P' wrapped in parenthesis.

end

flatten_sequence(R)
begin

1. Create a copy R' of the declaration of R.
2. For every local variable declared in R', make its name unique with respect to the names of the local variables declared in the property_expr of the assertion.
3. Replace every formal argument f in the sequence_expr of R' that is not on the left hand side of an assignment, with $var(a), where a is the corresponding actual argument expressions in the instance of R'.
4. For every occurrence of a typed formal argument f of R' with a simple type t[see A.2.2.1] in the sequence_expr of R', replace f with (t)'f (cast f to type t). See cast definition at (4.14).
5. Replace every formal argument f in the sequence_expr of R', with (a) where a is the corresponding actual argument expressions in the instance of R'.

6. In the property_expr of the assertion, replace the instance of `R` with sequence_expr of `R'` wrapped in parenthesis.

end