

Proposal for 1420

SV-AC

August 24, 2006

1. SECTION 17.11.4, REPLACE:

RESTRICTION 4: For every recursive instance of property q in the declaration of property p , each actual argument expression e of the instance satisfies at least one of the following conditions:

- e is itself a formal argument of p .
- No formal argument of p appears in e .
- e is passed to a formal argument of q that is typed and the set of values for the type is bounded.

For example :

```
property p1(int i, j);
  (j == 7) or
  (
    (i > 0) and
    ((a ##1 b) |-> p1(i+2, j))
  );
endproperty
```

is a legal declaration, but

```
property p1(int i, j);
  (j == 7) or
  (
    (i > 0) and
    ((a ##1 b) |-> p1(i+2, j+2))
  );
endproperty
```

is not legal because the second formal argument of $p1$ is not typed, the actual argument expression $j+2$ in the recursive instance $p1(i+2, j+2)$ is not itself a formal argument of $p1$, and this actual argument expression has an appearance of the formal argument j of $p1$.

WITH:

RESTRICTION 4: For every recursive instance of property q in the declaration of property p , each actual argument expression e of the instance satisfies ~~at least~~ one of the following conditions:

- e is itself a formal argument of p .
- No formal argument of p appears in e .
- ~~e is passed to a formal argument of q that is typed and the set of values for the type is bounded.~~

For example :

```
property fibonacci1(int a, b, n, fib_sig);
  int l_a, l_b, l_n;
  (n > 0, l_a = a, l_b = b, l_n = n)
  |->
  (
    (fib_sig == l_a)
    and
    (1'b1 | => fibonacci1(l_b, l_a + l_b, l_n - 1, fib_sig))
  );
endproperty
```

is a legal declaration, but

```
property fibonacci2(int a, b, n, fib_sig);
  (n > 0)
  |->
  (
    (fib_sig == a)
    and
    (1'b1 | => fibonacci2(b, a + b, n - 1, fib_sig))
  );
endproperty
```

is not legal because, in the recursive instance `fibonacci2(b, a+b, n-1, fib_sig)`, the actual argument expressions `a+b`, `n-1` are not themselves formal arguments of `fibonacci2` and yet formal arguments of `fibonacci2` appear in these expressions.

2. SECTION 17.11.4, REPLACE:

```

property check_write_data_beat
(
    expected_data, // [0:127]
    tag, // [3:0]
    i // [3:0]
);
first_match
(
    ##[0:$]
    (
        (data_valid && (data_valid_tag == tag))
        ||
        (retry && (retry_tag == tag))
    )
)
|->
(
    (
        (data_valid && (data_valid_tag == tag))
        |->
        (data == expected_data[i*8+:8])
    )
    and
    (
        if (retry && (retry_tag == tag))
        (
            1'b1 |> check_write_data_beat(tag, expected_data, 4'h0)
        )
        else if (!last_data_valid)
        (
            1'b1 |> check_write_data_beat(tag, expected_data, i+4'h1)
        )
        else
        (
            ##1 (retry && (retry_tag == tag))
            |>
            check_write_data_beat(tag, expected_data, 4'h0)
        )
    )
);
endproperty

```

WITH:

```

property check_write_data_beat

```

```

(
  logic [0:127] expected_data, // [0:127]
  logic [3:0] tag, // [3:0]
  logic [3:0] i // [3:0]
);
first_match
(
  ##[0:$]
  (
    (data_valid && (data_valid_tag == tag))
    ++
    (retry && (retry_tag == tag))
  )
)
logic [3:0] l_i;
(1'b1, l_i = i) ##0
(
  (data_valid && (data_valid_tag == tag))
  ||
  (retry && (retry_tag == tag))
)[->1]
|->
(
  (
    (data_valid && (data_valid_tag == tag))
    |->
    (data == expected_data[i*8+:8 l_i*8+:8])
  )
  and
  (
    if (retry && (retry_tag == tag))
    (
      1'b1 | => check_write_data_beat(tag, expected_data, 4'h0)
    )
    else if (!last_data_valid)
    (
      1'b1 | =>
      check_write_data_beat(tag, expected_data, i+4'h1 l_i+4'h1)
    )
    else
    (
      ##1 (retry && (retry_tag == tag))
      | =>
      check_write_data_beat(tag, expected_data, 4'h0)
    )
  )
)
)

```

```
);  
endproperty
```

3. ANNEX E.5: REPLACE:

RESTRICTION 4: For every recursive instance of property q in the declaration of property p , each actual argument expression e of the instance satisfies at least one of the following conditions:

- e is itself a formal argument of p .
- No formal argument of p appears in e .
- e is passed to a formal argument of q that is typed and the set of values for the type is bounded.

WITH:

RESTRICTION 4: For every recursive instance of property q in the declaration of property p , each actual argument expression e of the instance satisfies ~~at least~~ one of the following conditions:

- e is itself a formal argument of p .
- No formal argument of p appears in e .
- ~~e is passed to a formal argument of q that is typed and the set of values for the type is bounded.~~