```
bit clk, a=1;
int unsigned k=16'hF0F0, m;
initial forever #10 clk=!clk;
default clocking cb_clk @ (posedge clk); endclocking
sequence q_with_default(
    logic w, untyped d=16'h0000, int unsigned x, y=16'hFF00);
    w ##2 x==y ##1 x==d;
endsequence : q_with_default
ap_q_with_default: assert property(q_with_default(a, , k));
// The "y" actual takes the default value of 16'hFF00
```

2.7 Local variables in formal arguments and in sequence and property declarations

A powerful feature of SystemVerilog Assertions is the ability to declare dynamically created variables local to properties and sequences. Some simple examples were already demonstrated and additional examples in the application of local variables in sequences and properties are also demonstrated throughout this book (see index). SystemVerilog allows individual copies of those local variables for each successful attempt of the asserted property. The local variables can be initialized, assigned (and reassigned) a value, operated on, and compared to other expressions. However, because complex sequences and properties are often composed of simpler sequences and properties, it is often necessary to interconnect those local variables among those sequences and properties. This methodology facilitates the construction of assertion statements as it provides for a divide-and-conquer approach to the problem

🕮 **Rule:** There are two positions where local variables can be declared:

1. In the assertion variable declaration section of a property or sequence
2. In the *sequence_port_list or property_port_list* as formal arguments.

This section provides the rules in applying local variables in properties and sequences. It then addresses the rules in applying local variables defined in formal arguments. The syntax of a sequence with formal arguments and local variables is shown below.

property property_identifier
[**(** [property_port_list] **)**] **;**
{ assertion_variable_declaration }
[clocking_event] [**disable iff (** expression_or_dist **)**]
property_statement
endproperty [**:** property_identifier]

sequence sequence_identifier
[**(** [sequence_port_list] **)**] **;**
{ assertion_variable_declaration }
sequence_expr **;**
endsequence [**:** sequence_identifier]

Local variables declared and optionally initialized (with some restrictions) here.
Direction for properties is limited to input only.

property_port_list ::= property_port_item {**,** property_port_item}
property_port_item ::=
{ attribute_instance }
[**local** [property_lvar_port_direction]] property_formal_type
port_identifier {variable_dimension} [**=** property_actual_arg]
property_lvar_port_direction ::= **input**

If `local` or `untyped` argument then treated as formal variable arguments.

```
sequence_port_list ::= sequence_port_item {, sequence_port_item}
sequence_port_item ::=
   { attribute_instance }
    [ local [ sequence_lvar_port_direction ] ] sequence_formal_type
      port_identifier {variable_dimension} [ = sequence_actual_arg ]
sequence_lvar_port_direction ::= input | inout | output
assertion_variable_declaration ::=
   var_data_type list_of_variable_decl_assignments ;
```

If `local` then treated as formal variable arguments. Direction for sequences can be `input | inout | output`

🕮 **Rule:** *[1] In general, a local variable formal argument behaves in the same way as a local variable declared in an assertion_variable_declaration. Thus, the rules for the variables declared in the assertion_variable_declaration region* (Section 2.6) *also apply to those variables declared as formal arguments.* From here on, the term "local variable" shall mean either a local variable formal argument or a local variable declared in the *assertion_variable_declaration.*

🕮 **Rule:** A non-local formal argument is by default of direction `input`, and can have a default value; however, a non-local formal argument cannot be written into it and is not considered a local variable. In the following example, formal arguments `i` and `j` are of direction `input`, and cannot be written into them. Their values are samples in the Preponed region.

```
sequence q_non_local_formal_arguments(int i=0, bit j);
      // (a, i=9, j=b) ##1 c==1 && j==c;  // 💣 Illegal, i and j are not local variables
      i>10 ##1 j;   // ✓ i and j are inputs
endsequence
```

🕮 **Rule:** A local variable formal argument acts as a local variable of the sequence or property. It can be exported out only if direction `inout` or `output` **(***in sequence declarations only, and not in property declaration*). An `untyped` formal argument cannot have a direction or a type; it can be treated as a local formal argument of direction `input` or `output` depending on how it is used within the sequence or property. An `untyped` formal argument can also be treated as a non-local formal argument if it is not assigned a value in the body of the sequence or property in which it is used. For example (***Ch/2.7/type_m.sv***)

```
bit clk, a, b, c;
default clocking cb_clk @ (posedge clk); endclocking
sequence q_local_formal_arguments2(
     local input int i=0,
     untyped j, k,
     local output bit t);
  (i>10, j=i) ##1 (1, j=data, t=1'b1) ##1 k;
endsequence
property p_test_untype;
    int x, z; // local variable
    bit r; // local variable
      (a, x=10) ##1 q_local_formal_arguments2(
                    .i(x), .j(z), .k(a), .t(r)) ##1 x==z ##0 r;
endproperty : p_test_untype
ap_test_untype: assert property(p_test_untype);
```

j is treated as an output formal argument because it is assigned in the sequence matched item. *k* is treated as non-local formal argument. *k* is never assigned.

🕮 **Rule:** A local formal argument can be of any type provided it is used such that the resulting expression results in a type that is cast compatible with an integral type. *[1] The term integral is used throughout the standard to refer to the data types that can represent a single basic integer data type, packed array, packed structure, packed union, enum variable, or time variable.* Therefore, a local formal argument can be of types `int`, `bit`, `byte`, `vectors (e.g., logic[15:0])`, `shortreal`, `real`, `realtime`, packed array, packed structure, packed union, dynamic arrays, queues, and associative arrays, `enum`. The following example demonstrates legal assertion constructs.

```
byte q1[$], y1=3, y2;  // q1 is a queue of bytes
sequence q_test(byte q[$]);  // (Ch/2.7/type_m.sv)
  byte v_q[$];
    ($rose(a), v_q.push_back(y1)) ##1 q[0]==v_q.pop_front();
endsequence
ap_q: assert property(q_test(q1));
// -------------
string s1="TEST", s2="test";
ap_test: assert property(s1==s2.toupper());
// -------------
typedef enum {BK_RED , BK_GREEN, BK_BLUE } colors_e;
colors_e color;
ap_colors: assert property(color==BK_BLUE);
// -------------
int mem_aarray[*]; // associative array
ap_test : assert property (mem_aarray[addr]==rdata);
// -------------
real r1=1.03, r2=1.03;
shortreal sr1, sr2;
int data;
realtime rt1=101.11, rt2;
sequence q_r(real ra, rb, realtime rt);
    ra==rb ##1 rt > rt1;
endsequence
ap_q_r: assert property(q_r(r1, r2, rt2));
always @ (posedge clk) rt2<= $realtime;
ap_test: assert property(s1==s2);
```

🕮 **Rule:** Table 2.7 summarizes the rules in the use of local variables. Further explanation on these rules is provided afterwards. The table uses the following abbreviations:

- FA stands for formal arguments.
- MT stands for empty.
- AVD stands for assertion variable declaration section (the region inside the property or sequence (and not in the port list) where variables can be declared.
- LV stands for local variable, including those in FA and in AVD.
    - Notation: Local variables in the formal argument region are prefixed with *lv*
      Local variables in the AVD region are prefixed with *v*.