Note: The first_match() is needed because without it, if b==0 when `local_v<=0` matches, the repeat loop will keep on going forever.

NOTED OPTION: If the variable used to define the delay has values that are within a constraint range, such as between 0 and 7 (*or 15, or at most 32*) one can use the **generate** statement, which appears much simpler than the use of local variables and the *sequence_match_item*. Example:

```
generate for (genvar g_i=0; g_i<8; g_i++) begin
  ap_delay_gen:  assert property (v==g_i && $rose(a) |-> ##g_i b);
end endgenerate
```

11.4.2 Range delay (e.g., ##[2:v] b

The first element of the range must be a constant, and cannot be a variable. The second element is a variable. Below is an equivalent assertion for : `// /11.4/m5067_gen_options.sv`

```
ap_range_fix: assert property($rose(a) |-> ##[N:v] b);
 property p_range_equivalent; // Equivalent implementation
  int local_v; // this is an internal local variable defined by the tool
   $rose(a) |-> (1, local_v = v-N)
        ##N 1'b1
         ##0 first_match((1, local_v=local_v - 1)[*0:$] ##1  (b || local_v<=0))
         ##0 b;
 endproperty
 ap_range_equivalent: assert property(p_range_equivalent);

generate option
  generate for (genvar g_i=0; (g_i<8); g_i++) begin
   if (g_i >= N)
      ap_range_gen:  assert property (v==g_i && $rose(a) |-> ##[N:g_i] b);
 end endgenerate
```

11.4.3 Simple repeat (e.g., a[*v]]

```
ap_repeat_fix: assert property( $rose(a) |-> b[*v] ##1 c); //1800'2018
 property p_repeat_equivalent; // /11.4/m5067_gen_options.sv
  int local_v; // this is an internal local variable defined by the tool
  $rose(a) |-> (1, local_v = v)
        ##0 first_match((b, local_v=local_v - 1)[*0:$] ##1  local_v<=0)
         ##0 c;
 endproperty
 ap_repeat_equivalent: assert property(p_repeat_equivalent);

generate option
  generate for (genvar g_i=0; g_i<8; g_i++) begin
   ap_repeat_gen:  assert property (v==g_i && $rose(a) |-> b[*g_i] ##1 c);
 end endgenerate
```

11.4.4 Range repeat (e.g., a[*1:v]

The first element of the range must be a constant, and cannot be a variable. The second element is a variable. Below is an equivalent assertion: `// m5067.sv`

```
ap_repeat_range_fix: assert property($rose(a) |-> b[*N:v] ##1 c); // 1800'2018
 property p_repeat_range_equivalent;
  int local_v; // this is an internal local variable defined by the tool
  $rose(a) |-> (1, local_v = v-N)  // /11.4/m5067_gen_options.sv
        ##0 b[*N]
        ##1 first_match((b, local_v=local_v - 1)[*0:$] ##1  (c || local_v<=0))
         ##0 c;
 endproperty
 ap_repeat_range_equivalent: assert property(p_repeat_range_equivalent);
```

```
generate option
 generate for (genvar g_i=0; g_i<8; g_i++) begin
    if (g_i >= N)
      ap_repeat_range_gen: assert property (v==g_i && $rose(a) |-> b[*N:g_i] ##1 c);
 end endgenerate
```

11.4.5 GOTO repeat (e.g., a[->v]

The following emulates the goto with a range set by a variable, the value of which is sampled when the goto statement is executed.

```
   ap_goto_fix: assert property( $rose(a) |-> b[-> v]); // p1800'2018
   property p_goto_equivalent; // Equivalent implementation
     int local_v; // this is an internal local variable defined by the tool
     $rose(a) |-> (1, local_v = v)  // /11.4/m5067_gen_options.sv
       ##0 first_match(b[->1] ##0 (1, local_v=local_v - 1)[*1:$] ##0  local_v<=0) ##1 c;
    endproperty
    ap_goto_equivalent: assert property(p_goto_equivalent);

generate option
     generate for (genvar g_i=0; g_i<8; g_i++) begin
       ap_goto_gen:  assert property (v==g_i && $rose(a) |-> b[-> g_i] );
    end endgenerate
```

11.4.6 Non-consecutive repetition (e.g., a[=v])

The following emulates the non-consecutive repetition with a range set by a variable, the value of which is sampled when the statement is executed

```
     ap_nonconsec_fix: assert property( $rose(a) |-> b[=v] ##1 c); // p1800'2018
     property p_nonconsec_equivalent; // Equivalent implementation
       int local_v; // this is an internal local variable defined by the tool
       $rose(a) |-> (1, local_v = v)  ##0  // /11.4/m5067_gen_options.sv
          first_match((b[->1] ##0 (1, local_v=local_v - 1))[*1:$] ##0 local_v<=0)
                      ##1 !b[*0:$] ##1 c;
       endproperty
       ap_nonconsec_equivalent: assert property(p_nonconsec_equivalent);
generate option
   generate for (genvar g_i=0; g_i<8; g_i++) begin
      ap_nonconsec_gen:  assert property (v==g_i && $rose(a) |-> b[=g_i] ##1 c);
   end endgenerate
```