# Understanding the `within` Operator

I noticed from several posts that the **`within`** operator is frequently misunderstood, leading to its incorrect usage. This common misconception can result in errors. This paper clarifies the issues and provides solutions.

### 1.0 Definition

1800'2023 *16.9.10 Sequence contained within another sequence*, defines the construct
`seq1` **`within`** `seq2` is an abbreviation for the following:

```
(1[*0:$] ##1 seq1 ##1 1[*0:$]) intersect seq2
```

The **`within`** operator expresses the containment of one sequence within another sequence. Loosely speaking, it is similar to the notion of asserting something like the following:

I hold $1 in my wallet for 3 consecutive days out of 6 days, implying I do not hold it consecutively for less than 3 days. However, I may hold it for more than 3 days.
In SVA, I would write this as:

```
bit dollar,day;
assert property(@(posedge day)
  dollar[*3] within 1[*6]);
```

The assertion will fail if `dollar` is repeated less than 3 times in 6 consecutive days. The assertion will pass if `dollar` is repeated 3 or more times.

### 2.0 Understanding (a[*3] within 1[*6])

`a[*3]` **`within`** `1[*6]`
Expanding the abbreviation of the **`within`**, we get:

```
(1[*0:$] ##1 a[*3] ##1 1[*0:$]) intersect 1[*6]
```

The LHS sequence is multithreaded and is a match if in 6 cycles after its attempt we have a thread with 3 consecutive `a`'s; for example:

a a a 1 1 1, 1 a a a 1 1, 1 1 a a a 1, 1 1 1 a a a

There is a misconception that the property (`a[*3]` **`within`** `1[*6])` will flag as false if `a` is repeated more than 3 times; it does not based on the definition of the **`within`** operator and that can be misleading.

```
ap_3in6_noncontiguous_BAD: assert property( // Does not test for more than 3 a's
      @(posedge clk) $rose(b) |->
            a[*3] within 1[*6]) p_withn++; else f_withn++;
// See simulation results in section 5.0
```

**3.0 a[=3] within 1[*6]**

Question: Since the `a[=3]` expands to `a[->3] ##1 !a[*0:$]`, wouldn't the `last !a` term cause a no-match if `a` is nonconsecutively repeated more than 3 times?

Answer: `NO`

```
a[=3] within 1[*6] // is same as
  1[*0:$] // The within container
       !a[*0:$] ##1 a // same as a[->1]
   ##1 !a[*0:$] ##1 a // same as a[->1]
   ##1 !a[*0:$] ##1 a // same as a[->1]
   ##1 !a[*0:$]  // This and the previous 3 lines represent a[=3]
 ##1 1[*0:$] // the within container
```

Analyzing the last 3 lines starting with the last `a:`

```
 a ##1 !a[*0:$] ##1 1[*0:$]  // we get
 a ##1 !a[*0]   ##1 1[*0:$] or a ##1 !a[*1:$] ##1 1[*0:$]
```

Considering the last thread with the empty match

```
a ##1 !a[*0]  ##1 1[*0:$]  // is same as
a ##1 empty   ##1 1[*0:$] // is same as
a ##0 1       ##1 1[*0:$]
```

If this thread is a match after the 3rd occurrence of `a`, then
the other treads that check for `!a` are not needed.
These other threads are `!a[*1:$] ##1 1[*0:$]`
Therefore this property does not check if more than 3 `a`'s occur.

**4.0 Exactly 3 a's in 6 cycles**

**Non-consecutive a's**

Using the non-consecutive repeat [=n] along with the **`intersect`**

```
   ap_3in6_noncontiguous: assert property(
      @(posedge clk) $rose(b) |->
            a[=3] intersect 1[*6]) p_insct++; else f_insct++;
// There is no issue here with empty matches
```

**consecutive a's**

With the goto followed by `!a[*0:$]` and with the `intersect`

```
    ap_3in6_contiguous_OK: assert property(
      @(posedge clk) $rose(b) |->
      a[->1] ##1 a[*2] ##1 !a[*0:$] intersect 1[*6]) p_goto++; else f_goto++;
// There is no issue here with empty matches
```

### 5.0 Simulation Results

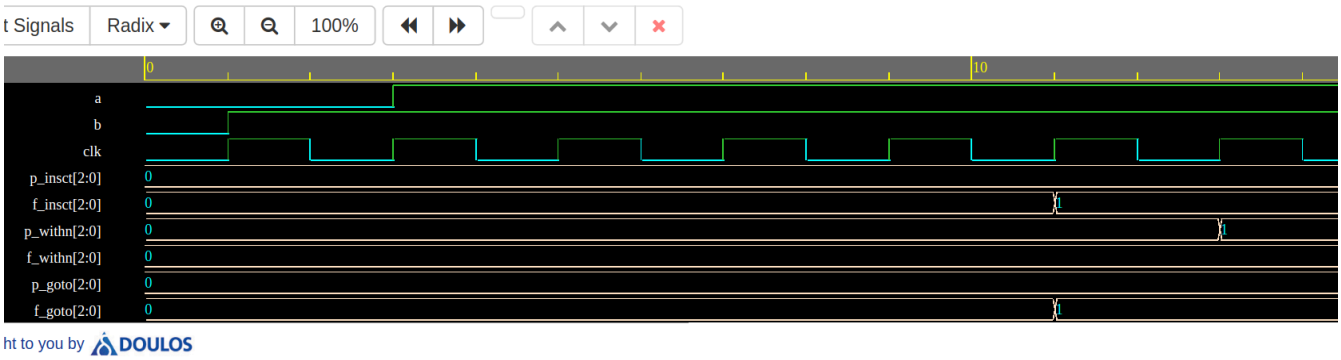
Code https://www.edaplayground.com/x/SPja Wave https://www.edaplayground.com/w/x/N2H

```
module m;
  bit clk, a, b;
  bit[2:0] p_insct, f_insct, p_withn, f_withn, p_goto, f_goto;
    always  #1  clk  =  !clk ;
    ap_3in6_noncontiguous: assert property(
      @(posedge clk) $rose(b) |->
            a[=3] intersect 1[*6]) p_insct++; else f_insct++;

    ap_3in6_noncontiguous_BAD: assert property(
      @(posedge clk) $rose(b) |->
            a[*3] within 1[*6]) p_withn++; else f_withn++;

    ap_3in6_contiguous_OK: assert property(
      @(posedge clk) $rose(b) |->
      a[->1] ##1 a[*2] ##1 !a[*0:$] intersect 1[*6]) p_goto++; else f_goto++;

  initial  begin
     $dumpfile("dump.vcd"); $dumpvars;
    @(posedge clk) b<=1;
    @(posedge clk) a<=1;
     #25 $finish();
  end
 endmodule
```




ht to you by DOULOS

### 6.0 Conclusions

Be cautious when using the **`within`** operator and understand how its abbreviations may impact the model. IMHO, the use of the **intersect** is clearer to understand and prefer it over the **`within`**.

Note: The following paper is related to degeneracy and use of the empty match.

**Paper: Understanding SVA Degeneracy**

**https://systemverilog.us/vf/Degeneracy111723Ben.pdf**

**https://lnkd.in/gABW3y-j**

**Ben Cohen**

**Link to the list of papers and books that I wrote, many are now donated.**

**https://systemverilog.us/vf/Cohen_Links_to_papers_books.pdf**