# 1800’2023 Degeneracy Issue to be Addressed in 1800’2028

**1.0 Issue**:

When faced with degenerate sequences, 1800’2023 is unclear on what a simulator should do, particularly in situations resulting from parameter substitution that creates a degenerate sequence. In that situation, it would be undesirable to have to rewrite the code to get to a successful compilation. Existing tools differ in their interpretation of degenerate sequences - some report it as a degenerate situation, some issue a warning, and some proceed with the simulation.

This document identifies the sections in the LRM that define degeneracy and highlights cases that pose challenges for users and tool vendors. Additionally, it proposes solutions for the next revision of the LRM.

## 2.0 1800’2023 description of degeneracy

### 2.1 16.7 Sequences

A property may involve checking of one or more sequential behaviors beginning at various times. An attempted evaluation of a sequence is a search for a match of the sequence beginning at a particular clock tick. To determine whether such a match exists, appropriate Boolean expressions are evaluated beginning at the particular clock tick and continuing at each successive clock tick until either **a match is found or it is deduced that no match can exist**.

**A sequence may admit an *empty match*, a match that occurs over an interval of length 0.** (See a formal definition at 16.12.22, and see 16.9.2.1 for more discussion of empty matches). An *end point* of a sequence is the time step of any nonempty match of the sequence. An end point is reached whenever the ending clock tick of a match of the sequence is reached, regardless of the starting clock tick of the match. A *match point* includes both empty and nonempty matches, and is reached either at an end point or, in the case of an empty match, at the length-0 time interval at the beginning of the time step when sequence evaluation begins. A sequence that admits only empty matches is referred to as an *empty sequence*.

### 2.2 16.9.2.1 Repetition, concatenation, and empty matches

Using 0 as a sequence repetition number, an empty sequence (see 16.7) results, as in this example: a [*0]
Because empty matches occur over an interval of zero clock ticks and are thus of length 0, they follow the set of concatenation rules specified below. In the following rules, an empty sequence is denoted as *empty*, and another sequence (which may be empty or nonempty) is denoted as *seq*.
— (*empty* ##0 *seq*) does not result in a match.
— (*seq* ##0 *empty*) does not result in a match.
— (*empty* ##n *seq*), where n is greater than 0, is equivalent to (##(n-1) *seq*).
— (*seq* ##n *empty*), where n is greater than 0, is equivalent to (*seq* ##(n-1) `true).

### 2.3 16.12.22 Nondegeneracy

It is possible to define sequences that can never be matched. For example:
(1'b1) **intersect** (1'b1 ##1 1'b1)
It is also possible to define sequences that admit only empty matches. For example:
1'b1[*0]

## 3.0 Discussion

Summary of terminology regarding sequence match or no match:

1) Sequence matches on a trace
   @(posedge clk) a ##1 b – when a is true followed by b true on the next clock tick.
2) Empty match
   a[*0]-- only empty match
3) Empty and non-empty matches
   a[*0:1] -- empty and a possible non-empty match
4) Does not result in a match, admits no match
   a[*0] ##0 b-- empty and at the same time non-empty - impossible
5) Never be matched, no match can exist
   (i) a intersect (b ##1 c) -- structurally impossible - different lengths
   (ii) a ##1 1'b0 -- logically impossible - a followed by false

From 16.12.22: *A sequence that admits no match or that admits only empty matches is called degenerate*
*a) Any sequence that is used as a property shall be nondegenerate and shall not admit any empty match*.

This clearly involves cases (2) (3) and (4).

There is potential for differing interpretations of case (5).
Case (5.i) might be viewed as an inconsistent statement due to its structure, while case (5.ii) is attributed to a false logical value resulting from parameter substitution. In both scenarios, if utilized in a property, they would lead to an unsatisfied property. Existing tools vary in their handling of degenerate sequences: some identify it as a degenerate situation, some issue warnings, and some continue with the simulation.

*b) Any sequence that is used as the antecedent of an overlapping implication (|->) shall be nondegenerate.*
*c) Any sequence that is used as the antecedent of a nonoverlapping implication (|=>) shall admit at least one match. Such a sequence can admit only empty matches.*

In these scenarios where the sequence is employed as an antecedent, either case of (5), whether degenerate by construction or impossibility, may lead the implication to be evaluated as vacuously true. Apart from imposing simulation overhead, it doesn't result in any harm. However, if the implication is part of a more intricate property, the vacuous success could facilitate further evaluation. For instance, if it's utilized as the left operand of an implies statement.

The cases under (5) should be treated as normal. The tool may issue a warning but it should not be considered as non-degenerate. The LRM could include an explanatory note to that effect.

**4.0 Recommendation**:
Change
FROM:
**1800’2023: 16.12.22 Nondegeneracy**
It is possible to define sequences that can never be matched. For example:
(1'b1) **intersect** (1'b1 ##1 1'b1)
It is also possible to define sequences that admit only empty matches. For example:
1'b1[*0]

TO:
**1800’2023: 16.12.22 Nondegeneracy**

It is possible to define sequences that can never be matched. For example:
(1'b1) **intersect** (1'b1 ##1 1'b1) // sequence is degenerate by construction
It is also possible to define sequences that admit only empty matches. For example:
1'b1[*0] // sequence is degenerate by the impossibility of a  match
Note: When a sequence is degenerate due to construction or the impossibility of a match, a tool may issue a warning and proceed with compilation for processing.