Introduction

**Expanded Boolean** definition, firstly claims that the value of proposition cannot be restricted by only 0 and 1. It must be defined as any real numbers **between 0 and 1** to produce more realistic approaches about our world inside digital circuits.

If *p* is a proposition and *x* is its **expanded value**: x € R, 0 <= x <= 1

**Reference Point** is used to define the **truth level** for any expanded value. It can be either static or dynamic. Which is also, the reference point is a real number, between 0 and 1.

Truth Table

Unlike the fundamental approach of Boolean, there are more than two truth levels, as additional to **True** and **False**.

If *p* is a proposition, *x* is the expanded value and the 𝛂 is the reference point, truth table would be:

| **Truth Level** | **Condition** | **Approval at** 𝛂 |
| --- | --- | --- |
| **T** (True) | x = 1 | not important |
| **AT** (Almost True) | x > 𝛂 and 0 < x < 1 | not important |
| **NT** (Neutral-True) | x = 𝛂 | 𝛂 point is accepted as True by the result of **TO** function. |
| **TS** (Tiresias) | x = 𝛂 | 𝛂 point is accepted as neither True or False, there is no equivalent in standard Boolean system. |
| **NF** (Neutral-False) | x = 𝛂 | 𝛂 point is accepted as False by the result of **TO** function. |
| **AF** (Almost False) | x < 𝛂 and 0 < x < 1 | not important |
| **F** (False) | x = 0 | not important |

**Neutral** definition is studied as three parts in Expanded Boolean. The main reason of this to make clear the which behavior must be supplied as Boolean when x equals 𝛂. On the other hand, these levels are totally the same for Expanded Boolean.

Outsider Methods

An **Outsider Method** takes expanded values as parameters and returns a result whose type is Standard Boolean.

Some of the outsider methods are predefined and they are represented by their names by using upper case letters. Also, parameters are placed inside the square brackets. On the other hand, when another outsider method is wanted to be defined, an English alphabet character is used with its lower case form.

# TO

**TO** method takes two parameters as x, the expanded value of the proposition, and 𝛂, the reference point. The method is written with this general form:

TO [ x, 𝛂 ] = z

The result is represented with *z* and its value can be whether 0 or 1. When the change of the method by x and 𝛂 is wanted to make visualized by using 3D coordinate system, *x* in x-axis, *𝛂* in y-axis and *z* in z-axis are represented.

| **Expanded Value** | **Boolean Value** |
| --- | --- |
| T | T |
| AT | T |
| NT | T |
| TS | there is no equivalent |
| NF | F |
| AF | F |
| F | F |

# NOT

**NOT** method works as same as TO with one difference that returns a negative result which is got according to TO. The method is written with this general form:

NOT [ x, 𝛂 ] = z

| **Expanded Value** | **Boolean Value** |
| --- | --- |
| T | F |
| AT | F |
| NT | F |
| TS | there is no equivalent |
| NF | T |
| AF | T |
| F | T |

# COMP

**COMP** method is used to compare magnitudes of two expanded values or spin coefficients. It can be used only when these expanded values or spin coefficients belong to the same situation. There are three types of COMP method as **COMP+**, **COMP-**, **COMPp+**, **COMPp-**, **COMPp0** and **COMP0**. The method is written with this general form:

COMP [ x, y ] = z

COMP+ returns True only if x is greater than y. Which means, if (x / y) is bigger than 1 then the result will be True.

COMP- returns True only if x is smaller than y. Which means, if (x / y) is less than 1 then the result will be True.

COMP0 returns True only if x equals to y. Which means, if (x / y) equals to 1 then the result will be True.

COMPp+ returns True only if x is greater than y and these two propositions have the **same expanded value**. Which means, if (x / y) is bigger than 1 then the result will be True.

COMPp- returns True only if x is less than y and these two propositions have the **same expanded value**. Which means, if (x / y) is smaller than 1 then the result will be True.

COMPp0 returns True only if x is equal to y and these two propositions have the **same expanded value**. Which means, if (x / y) is equal to 1 then the result will be True.

# 

# 

# AND

The AND method takes at least two parameters as expanded values. The general form of method is this:

AND [ x, y, 𝛂 ] = z

If there are more than one reference points for expanded values, they have to be indicated either:

AND [ xA , yB , A , B ] = z

It must be explained that to make everything more clear, AND is a **shortcut** more than a unique outsider method:

p ☰ AND [ xA , yB , A , B ] ☰ TO [ x, A ] ^ TO [ y, B ]

Not only **AND** but also **OR**, **XOR**, **IF**, **ONLY** (if and only if), **NAND**, **NOT**… are known as **shortcut**s.

Insider Methods

An **Insider Method** takes expanded values as parameters and returns a result as an expanded value either.

Unlike outsider methods, the predefined insider methods are written by using only lower case letters. Besides, when another insider method is wanted to be defined, an English alphabet character is used with its upper case form.

# Arithmetic Methods

The arithmetic methods have one common principle: A True proposition cannot be more right than True. Same way, a False proposition cannot be more wrong than False. Which means, no matter what the expanded value can neither go down from zero nor go over from one.

## sum

The general form of sum method is this:

sum [ x, y ] = z

* If x + y >= 1 then z = 1
* If x + y < 1 then z = x + y

## minus

The general form of minus method is this:

minus [ x, y ] = z

* If x - y > 0 then z = x - y
* If x - y <= 0 then z = 0

## times

The general form of times method is this:

times [ x, y ] = z

* If x \* y >= 1 then z = 1
* If x \* y < 1 then z = x \* y

## over

The general form of over method is this:

over [ x, y ] = z

* If x / y >= 1 then z = 1
* If x / y < 1 then z = x / y

## exp

The general form of exp method is this:

exp [ x, n ] = z

* If x ^ n >= 1 then z = 1
* If x ^ n < 1 then z = x ^ n

## root

The general form of root method is this:

root [ x, n ] = z

* If x ^ (1 / n) >= 1 then z = 1
* If x ^ (1 / n) < 1 then z = x ^ (1 / n)

Inner Methods

An inner method takes any real numbers as parameters and returns an expanded value.

Vectorial Truth

A **truth vector** has components as real numbers from 0 to 1. The expanded value of the vector equals the **magnitude** of it.

For example:

If T is a vector then T = < x, y > is a truth vector and its expanded value is || T ||

Polar Truth

The Polar Truth technique is used to compare propositions which have the same expanded value. It does not affect the exact result of expanded value.

If proposition *p* has expanded value as *x* and the **spin coefficient** is *z*, p is pointed on the polar coordinate system at the point ((+ / -)x, z). The spin coefficient can take a value from 0 to 2π (0 <= z <= 2π).

Whenever z is getting bigger, the proposition will be getting more true either.