Completeness of “N”-Input Operators on Propositional Variables

A complete operator has the property of being able to generate all possible truth table output combinations from its given inputs. We may also call a group of 2 or more operators complete if they can be used together to generate all truth outputs. For this discussion, we shall mainly investigate the individual operators which are complete.

Below is a truth table (Table 1a) with the listing of all sixteen possible combinations which a binary operator can have on two propositional variables A and B:

(Note that there are possible combinations since there are two options for each of the four lines in the truth table)

Table 1a

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

For an operator to be complete, it must be able to generate a 1 in the truth table when all its inputs are 0, and it must also be able to generate a 0 in the truth table when all its inputs are 1. This immediately means that combinations 1 through 8, 10, 12, 14, and 16 cannot be complete operators, so we shall eliminate them from our consideration:

Table 1b

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

This leaves us to check combinations 9, 11, 13, and 15 to see if they are complete. It is clear to see that combination 11 is just the NOT of input A, (¬A), and combination 13 is just the NOT of input B, (¬B). It is trivial to show that the NOT operator is incomplete from the following table (Table 2):

Table 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | ¬A | ¬¬A | ¬B | ¬¬B |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 |

Any further repetitions of the NOT operator will simply keep cycling through the same two combinations forever, and as such, will not be complete. So, we may remove combinations 11 and 13 from our consideration. We are left with combinations 9 and 15, which we shall show are complete. For combination 9, we shall do so by creating the binary OR operator and the unary NOT operator, which are complete when used together. For combination 15 we shall create the binary AND operator and the Unary NOT operator, which are also complete when used together.

Let P(A, B) and Q(A, B) be the binary operators defined by the truth tables (Table 3) for combinations 9 and 15 respectively:

Table 3

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | 9 = P(A, B) | 15 = Q(A, B) |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

Table 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | M = P(A, A) | N = P(A, B) = ¬(A ∨ B) | O = P(N,N) |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |

Here the Truth table (Table 4) for M is just the NOT operator (in this case it represents ¬A), while the truth table for N is simply the definition of P(A, B). Finally, the Truth table of O is written in terms of N but is shorthand for saying: P( P(A, B), P(A, B) ). We shall continue using this shorthand notation for ease of reading. We can also see that the truth table for O is just the OR operator (in this case it represents A ∨ B). We have shown a construction of the OR and NOT operators using the definition of combination 9, and so we have proven it is a complete operator. Continuing forward we shall refer to combination 9 as the NOR operator. This is because we can write the operator as the NOT of A OR B:

P(A, B) = ¬ (A ∨ B)

Which is also listed as an equivalent form in the truth table (Table 4) above. We shall now show a similar proof of the completeness of Combination 15 in the below truth table (Table 5):

Table 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | X = Q(A, A) | Y = Q(A, B) = ¬(A ∧ B) | Z = Q(N,N) |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |

Once again we can see that equation X is the truth table as (¬A) and equation Y is simply the definition of combination 15. Finally, equation Z is the same as (A ∧ B), so we have shown that this operator is complete. We shall refer to this operator as NAND since it can be written as the NOT of A AND B:­

Q(A, B) = ¬(A ∧ B)

We have gone through all combinations and found there are exactly two complete binary operators, NAND and NOR. We shall move our investigation into the more complex case of trinary operators.

A trinary operator takes in three inputs (A, B, and C) and generates a truth assignment. There are unique combinations of inputs A, B, and C. As such, there will be 8 lines in our truth table for a total of possible output combinations.

For obvious reasons we shall not list a 256-column truth table, however, there are some slight reductions to this number from the information stated before. If we are only considering complete operators, then we know that when all inputs are 0, we should output a 1, and when all inputs are 1, we should output a 0. If the operator did not have this behavior, then it would never be able to generate a 1 in the first column and a 0 in the last column, making it incomplete. The general form is shown in the table (Table 6) below, where x refers to a truth assignment that can be 0 or 1.

Table 6

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | C | Possible Complete Operators |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | x |
| 0 | 1 | 0 | x |
| 0 | 1 | 1 | x |
| 1 | 0 | 0 | x |
| 1 | 0 | 1 | x |
| 1 | 1 | 0 | x |
| 1 | 1 | 1 | 0 |

From this table, we have two lines with fixed truth assignments, and 6 lines with 2 options each for a total of possible combinations. Once again, we shall not list all 64 combinations, but a reduction in combinations by a factor of ¼ is certainly an improvement.

Continuing from this point onwards we shall delve into conjecture along with some new definitions that may assist in simplifying our work.

The “Dual” of a truth assignment is found by negating all of its assignments, and then “flipping” its assignment column-wise. These descriptions can be seen in the below table (Table 7):

Table 7

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | Assignment C | Negation of C | “Flip” of C | “Dual” of C |
| 0 | 0 | W | ¬W | Z | ¬Z |
| 0 | 1 | X | ¬X | Y | ¬Y |
| 1 | 0 | Y | ¬Y | X | ¬X |
| 1 | 1 | Z | ¬Z | W | ¬W |

Definition: The “Dual” of a truth assignment is found by writing its equation, negating every literal in its equation, and then negating the entire equation. As an example:

Y = A ∨ B Dual(Y) = ¬ ( (¬A) ∨ (¬B) ) = (¬¬A) ∧ (¬¬B) = A ∧ B

Above we can see that the Dual is also the same as switching all ANDS for ORS and vice versa, which can be proven inductively using DeMorgan’s Theorem. We shall now consider the Complete Binary Operators we have discovered:

Table 8

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | NOR= P(A, B) | NAND = Q(A, B) |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

It is easily verified that these operators are duals of each other. Now let us consider the following incomprehensive list of complete trinary operators:

Table 9

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A | B | C | E(A, B, C) | F(A, B, C) | G(A, B, C) | H(A, B, C) |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |

Table 10a

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | M = E(A,B,B) | ¬A = E(A,A,A) | A ∨ B = E(M,M,M) | N = F(A,B,B) | ¬A = F(A,A,A) | A ∧ B = F(N,N,N) |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |

In the above table (Table 10a) we detail that the operator “E” and the operator “F” are both complete since we have shown that the “E” can generate NOT and the 2-input OR, while “F” can generate NOT and the 2-input AND gate.

Notice that the construction and processes for both proofs of completeness are identical (although the actual truth assignments are duals of each other). As such, we may hypothesize that if an operator “E” is complete, then its dual “F” is complete, using the exact same proof of completeness for both. Below (Table 10b) is another complete operator and its complete dual:

Table 10b

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | M = G(A,A,B) | ¬A = G(A,A,A) | A ∧ B = G(M,M,M) | N = H(A,A,B) | ¬A = H(A,A,A) | A ∨ B = H(N,N,N) |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |

Once again, we have shown these operators are complete with generating NOT, and either 2-input AND or 2-input OR. We also begin to see a pattern emerge with these proofs of completeness.

It seems that we can easily generate NOT from any complete “n”-ary operator, by simply making all of its “n” inputs the same propositional variable (i.e. P(A, A, … , A)). Doing this results in two cases, A is 0, or A is 1. If A is 0, then our complete operator must output a 1, and if A is 1, then our complete operator must output a 0. This is the same functionality as ¬A , so we have shown that all complete operators will generate the truth table for NOT, by taking the same propositional variable A into all of its “n” inputs.

We shall ­use the term “NOT-generator” to refer to operators that generate a 1 when all inputs are 0 and generate a 0 when all inputs are 1. We shall also refer to an operator that is its own dual as “self-dual”.

Continuing with our conjecture, we have a few claims that for now will go unproven:

Claim 1: An operator is complete iff its dual is complete

Claim 2: If an operator is self-dual, then it is incomplete

Claim 3: An operator is complete iff it is not self-dual and is a “NOT-generator”

These claims can be checked against all 16 binary operators and (with enough tenacity) against all 256 Ternary operators to be verified. Clearly, that is no proof for our claims when there are 4 or more inputs, however, we shall assume these claims to be true for the moment.

First, we shall develop an equation for determining how many complete operators there must be for “n”-inputs. It is clear to see that for “n” inputs, we shall have rows in our truth table. This is because each input has two possible options, so we have “n” copies of 2 multiplied together, or . Calculating the number of total possible operators is similar, since we have rows and each row has two possible options, so we have “” copies of 2 multiplied together, or possible operators.

of all operators are “NOT-generators”, since they must have a fixed value in the first and last row of the truth table, and each of those fixed values cuts all of our possibilities in half. So, we can derive the equation of the number of “NOT-generators” for “n”-inputs:

Now we need to remove all of the “NOT-generators” that are “self-dual”. For an operator to be “self-dual”, it must have pairs of entries which are opposite in truth value. As an example of this behavior, consider Table 11:

Table 11

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | F(A, B, C) | Entry line |
| 0 | 0 | 0 | 0 | Line 0 |
| 0 | 0 | 1 | 1 | Line 1 |
| 0 | 1 | 0 | 1 | Line 2 |
| 0 | 1 | 1 | 1 | Line 3 |
| 1 | 0 | 0 | 0 | Line 4 |
| 1 | 0 | 1 | 0 | Line 5 |
| 1 | 1 | 0 | 0 | Line 6 |
| 1 | 1 | 1 | 1 | Line 7 |

“Pairs of Entries” are when two entries have exact opposite literal truth values. In the above example, Line 0 and Line 7 are paired, since the former is for the entry ABC = 000, which the latter is for the entry ABC = 111, which are exactly opposite literal values. Similarly, lines 2 and 5 would be paired (Line 2: ABC = 010, Line 5: ABC = 101). Keeping this in mind, we can compute the number of “self-dual” operators.

For “n”-inputs, once again we know there must be rows in our truth table. Now if we are talking about pairs of rows, then there are half as many pairs of rows, which is to say we will have:

pairs of rows. Now for specifically the “self-dual” operators which are “NOT-generators” we must fix the first row and last row to 1 and 0 respectively, so we lose one pair of rows since they are now a fixed value, so there are: unfixed pairs of rows. Finally, each of those pairs has two options, 10 or 01, so in total there are “self-dual, NOT-Generators”. Now we just subtract the “self-dual, NOT-Generators” from the total number of “NOT-Generators”:

And the above equation will calculate the number of compete operators for “n”-inputs. Calculating the number of compete operators using this equation yields the following sequence:

Another interesting sidenote is that the ratio of complete operators to total operators tends to as the number of inputs “n” tends to infinity, as shown below for the total number of operators

­­

With these discoveries depending on the unproven claims listed prior, now all that remains is to get down into the work of developing proofs for those claims.