A Digital Calculation Method for Propositional Logic

Zhou Nongjian

Abstract

This paper presents a novel approach: using a digital calculation method for propositional logical reasoning. The paper demonstrates how to discover the primitive numbers and the digital calculation formulas by analyzing the truth tables. Then it illustrates how to calculate and compare the truth values of various expressions by using the digital calculation method. As an enhanced alternative to existing approaches, the proposed method transforms the statement-based or table-based reasoning into number-based reasoning. Thereby, it eliminates the need for using truth tables, and obviates the need for applying theorems, rewriting statements, and changing symbols. It provides a more streamlined solution for a single reasoning, while demonstrating more efficiency for multiple reasonings in long-term use. It is suitable for manual calculation, large-scale computation, AI and automated reasoning.

Keywords: propositional logic, Boolean logic, truth table, general logic, mathematical logic, symbolic logic

---

There are two main methods that have been used in propositional reasoning.

One is the statement-based reasoning method. The use of this method can be traced back to Aristotle, and in 19th century, it was symbolized by Geoge Boole [1] and others. The techniques employed by Gottlob Frege [2], David Hilbert [3], and Gerhard Gentzen [4] can be mainly classified into this approach category. This method uses symbolic statements line by line to carry out a reasoning. For rewriting each line of statement and changing the symbols contained therein, a theorem or rule must be (explicitly or implicitly) applied to prove the logical rationality. By rewriting the statement line by line, replacing and eliminating symbols, the conclusion is finally obtained.

Another method is the truth table method. Its earliest use can be traced back to Charles Sanders Peirce [5] and Ludwig Wittgenstein [6]. This table-based reasoning method uses a table called a truth table to represent a given proposition and its elements (variables), and then fills in different cells with the input options (T/F) and their corresponding results (called truth values). Finally, the conclusion is drawn by analyzing the truth values.

While the two methods have demonstrated their effectiveness, there remains scope for alternative methodologies.

Feedback and discussion are welcome. Author email: nongjianz@hotmail.com

This paper presents a novel approach: using a digital calculation method for propositional logic. As an enhanced alternative to existing approaches, this method uses primitive numbers and the digital calculation formulas to calculate truth values of propositions. The novel method is functionally equivalent to the conventional methods, but more efficient. It transforms the statement-based or table-based reasoning into number-based reasoning. Thereby, it eliminates the need for using truth tables, and obviates the need for applying theorems, rewriting statements, and changing symbols. The proposed method provides a more streamlined solution for a single reasoning, while demonstrating more efficiency for multiple reasonings in long-term use. It is suitable for manual calculation, large-scale computation, AI and automated reasoning.

1 Primitive Numbers

1.1 Elemental Numbers

Let us obtain the elemental numbers from the truth tables. Suppose there are A, B, and C, let the symbols + and ¬ represent positive state and negative state respectively, then we can have elemental expressions (elemental propositions): +A, ¬A, +B, ¬B, +C, and ¬C. Let 1 and 0 represent True and False respectively. We can make the following truth tables:

Table 1.1

A	+A	¬A
1	1	0
0	0	1

Table 1.2

A	B	+A	¬A	+B	¬B
1	1	1	0	1	0
1	0	1	0	0	1
0	1	0	1	1	0
0	0	0	1	0	1

Table 1.3

A	B	C	+A	¬A	+B	¬B	+C	¬C
1	1	1	1	0	1	0	1	0
1	1	0	1	0	1	0	0	1
1	0	1	1	0	0	1	1	0
1	0	0	1	0	0	1	0	1
0	1	1	0	1	1	0	1	0
0	1	0	0	1	1	0	0	1
0	0	1	0	1	0	1	1	0
0	0	0	0	1	0	1	0	1

In Column +A of Table 1.1, the digits from top to bottom are: 1, 0. We can combine the

two digits into a number 10. By analogy, ¬A has a number 01. These 2-digit numbers are the truth values of the two elemental expressions (the + sign is omitted):

A: 1100 ¬A: 0011

Let's see the second table: Table 1.2. In Column +A, the digits from top to bottom are: 1, 1, 0, 0. We can combine the four digits into a number 1100. By analogy, ¬A has a number 0011. After combining digits of each column, we obtained the four numbers of the four elemental expressions (the + sign is omitted):

A: 1100 ¬A: 0011
B: 1010 ¬B: 0101

We can apply the same method to find out truth values from the third table: Table 1.3. According to the table, each elemental expression has an 8-digit truth value. For example, A is 11110000, that is the combination of all 8 digits from top to bottom in Column +A. For ease of human visual recognition, we can insert a hyphen between the fourth and fifth digits, making it: 1111-0000. However, for machine computation, the hyphen is unnecessary. After combining digits of each column, we obtained six elemental numbers (the + sign is omitted):

A: 1111-0000 ¬A: 0000-1111
B: 1100-1100 ¬B: 0011-0011
C: 1010-1010 ¬C: 0101-0101

We can also apply the same generation method to find out truth values of four-element expressions. The truth table is omitted here, and the results are as follows:

A: 1111-1111-0000-0000 ¬A: 0000-0000-1111-1111
B: 1111-0000-1111-0000 ¬B: 0000-1111-0000-1111
C: 1100-1100-1100-1100 ¬C: 0011-0011-0011-0011
D: 1010-1010-1010-1010 ¬D: 0101-0101-0101-0101

Depending on the number of elements involved, the digits of the numerical values are different. For example, the digital number for A is 10 in one-element (A) expressions, is 1100 in two-element (AB) expressions, is 1111-0000 in three-element (ABC) expressions, and is 1111-1111-0000-0000 in four-element (ABCD) expressions.

1.2 Assignment of State Numbers

Not only can elemental expressions (A, ¬A, B, ¬B and so on) have truth values and can be evaluated by digital numbers, but the positive state and negative state can also have digital numbers. In the above discussion, we use 1 to represent "true", and use 0 to represent "false". It should be noted that, in our system of the digital calculation method, existence, true, and affirmation are equivalent concepts, and they can be denoted by the positive sign + and its digital numbers: 1, 11, 1111, 1111-1111 and 1111-1111-1111-1111. And non-existence, false and negation also are equivalent concepts, and they can be represented by the negative sign ¬ and its digital numbers: 0, 00, 0000, 0000-0000

and 0000-0000-0000-0000. See the following table:

Table 1.4: Assigning Digital Numbers to State Expressions

+	1	11	1111	1111-1111	1111-1111-1111-1111	Existence	True	Affirmation
¬	0	00	0000	0000-0000	0000-0000-0000-0000	Non-existence	False	Negation

1.3 Primitive Number Lookup Table

Propositional expressions can be divided into the following three categories:

1. State expressions +, ¬
2. Elemental expressions A, ¬A, B, ¬B, ...
3. Relational expressions (A ∧ B), (A ∨ ¬B), ...

Now we have state numbers for + and ¬, and have elemental numbers for elemental expressions, we can call them the primitive numbers.

Based on the number of distinct elements contained in an expression, we can categorize expressions into the following categories:

Non-element expressions //+/¬
One-element expressions //A
Two-element expressions //AB
Three-element expressions //ABC
Four-element expressions //ABCD
...

Note that the so-called element refers to an entity object, not an attribute or state (+/¬). Thus, non-element expressions only have positive and negative two options that indicate two states. Although we can call them variables, they are not variables in the sense of substantive objects.

After applying the above number generation method to categories from non-element to four-element, we can obtain the following primitive numbers:

Table 1.5: Primitive Numbers

Expr.	Non-element +/¬	One-element A	Two-element AB	Three-element ABC	Four- element ABCD
+	1	11	1111	1111-1111	1111-1111-1111-1111
¬	0	00	0000	0000-0000	0000-0000-0000-0000
A		10	1100	1111-0000	1111-1111-0000-0000
¬A		01	0011	0000-1111	0000-0000-1111-1111
B			1010	1100-1100	1111-0000-1111-0000
¬B			0101	0011-0011	0000-1111-0000-1111

C				1010-1010	1100-1100-1100-1100
¬C				0101-0101	0011-0011-0011-0011
D					1010-1010-1010-1010
¬D					0101-0101-0101-0101

2 Digital Calculation Formulas and Calculation Method

2.1 Digital Calculation Formulas

A relational expression consists of elemental expressions and a connective (operator). In our notation system, there are eight types of relations or connectives (operators). Let us draw a consolidated truth table for all these eight connectives. Let the symbols $\wedge$, $\vee$, $\rightarrow$, $\oplus$, $\leftrightarrow$, $\uparrow$, $\downarrow$, and $\leftarrow$ represent AND, OR, IMPLIES, XOR, EQUIVALENT, NAND, NOR, and the converse respectively. See the following table:

Table 2.1: Truth Table of Eight Relational Expressions

A	B	(A $\wedge$ B)	(A $\vee$ B)	(A $\rightarrow$ B)	(A $\oplus$ B)	(A $\leftrightarrow$ B)	(A $\uparrow$ B)	(A $\downarrow$ B)	(A $\leftarrow$ B)
1	1	1	1	1	0	1	0	0	1
1	0	0	1	0	1	0	1	0	1
0	1	0	1	1	1	0	1	0	0
0	0	0	0	1	0	1	1	1	1

Applying the same method that we used to find the truth values of elemental expressions (A, ¬A, B, etc), we can find the truth values of relational expressions from the table above. For examples, in the column of the relational expression (A $\wedge$ B), the digits from top to bottom are 1000. This is the truth value of (A $\wedge$ B). By analogy, (A $\vee$ B) has a truth value 1110, and (A $\rightarrow$ B) has a truth value 1011. By using the method shown above, all relational expressions can have digital numbers representing their truth values.

However, this method does not have an efficiency advantage, because for each expression, we need to draw a truth table to obtain its truth value.

There is a possibility to obtain a truth value of an expression without having to draw a truth-table each time. This is to use a digital calculation method. That is applying a set of calculation formulas to calculate the given primitive numbers to obtain the truth value.

To perform calculations, we need calculation formulas. What are calculation formulas? Where and how can we find them? Let us discover the calculation formulas by analyzing the truth tables.

In the above truth table, let's look at the input columns A and B and the column (A $\wedge$ B), we can find that:

If the input values are 1 for A and 1 for B, then the result is 1.
If the input values are 1 for A and 0 for B, then the result is 0.

If the input values are 0 for A and 1 for B, then the result is 0.
If the input values are 0 for A and 0 for B, then the result is 0.

Let us ignore A and B, but keep the input values 1's and 0's, then from the above four inferences, we can abstract a set of four binary calculation formulas. See below:

$(1 \wedge 1) \leftrightarrow 1$
$(1 \wedge 0) \leftrightarrow 0$
$(0 \wedge 1) \leftrightarrow 0$
$(0 \wedge 0) \leftrightarrow 0$

By analogy, we can abstract other seven sets of binary calculation formulas from the inferences relevant to the other seven columns. See below:

Table 2.2: Eight Sets of Digital Calculation Formulas

$\wedge$	$\vee$	$\rightarrow$	$\oplus$
$(1 \wedge 1) \leftrightarrow 1$	$(1 \vee 1) \leftrightarrow 1$	$(1 \rightarrow 1) \leftrightarrow 1$	$(1 \oplus 1) \leftrightarrow 0$
$(1 \wedge 0) \leftrightarrow 0$	$(1 \vee 0) \leftrightarrow 1$	$(1 \rightarrow 0) \leftrightarrow 0$	$(1 \oplus 0) \leftrightarrow 1$
$(0 \wedge 1) \leftrightarrow 0$	$(0 \vee 1) \leftrightarrow 1$	$(0 \rightarrow 1) \leftrightarrow 1$	$(0 \oplus 1) \leftrightarrow 1$
$(0 \wedge 0) \leftrightarrow 0$	$(0 \vee 0) \leftrightarrow 0$	$(0 \rightarrow 0) \leftrightarrow 1$	$(0 \oplus 0) \leftrightarrow 0$
$\uparrow$	$\downarrow$	$\leftarrow$	$\leftrightarrow$
$(1 \uparrow 1) \leftrightarrow 0$	$(1 \downarrow 1) \leftrightarrow 0$	$(1 \leftarrow 1) \leftrightarrow 1$	$(1 \leftrightarrow 1) \leftrightarrow 1$
$(1 \uparrow 0) \leftrightarrow 1$	$(1 \downarrow 0) \leftrightarrow 0$	$(1 \leftarrow 0) \leftrightarrow 1$	$(1 \leftrightarrow 0) \leftrightarrow 0$
$(0 \uparrow 1) \leftrightarrow 1$	$(0 \downarrow 1) \leftrightarrow 0$	$(0 \leftarrow 1) \leftrightarrow 0$	$(0 \leftrightarrow 1) \leftrightarrow 0$
$(0 \uparrow 0) \leftrightarrow 1$	$(0 \downarrow 0) \leftrightarrow 1$	$(0 \leftarrow 0) \leftrightarrow 1$	$(0 \leftrightarrow 0) \leftrightarrow 1$

The relation NAND ($\uparrow$) can be seen as the opposite of the relation ($\wedge$), the relation NOR ($\downarrow$) can be seen as the opposite of the relation ($\vee$), and the relation ($\leftarrow$) can be seen as the reverse of the relation ($\rightarrow$). For the sake of simplicity, we will not discuss them in this paper. Therefore, the following discussion, including the examples demonstrating the calculations and the lookup tables in the appendix, will be limited to the five relations: $\wedge$, $\vee$, $\rightarrow$, $\oplus$ and $\leftrightarrow$.

2.2 Calculation Method

Let us demonstrate how to perform a calculation using the primitive numbers and the digital calculation formulas that we just discovered.

Suppose there is a two-element relational expression (A $\wedge$ $\neg$B). We can find the primitive numbers of A and $\neg$B in the two-element expression column of the primitive number table: A is 1100, and $\neg$B is 0101. Let's put them into a calculation expression:

$(1100 \wedge 0101) \leftrightarrow ?$

We can transform the above horizontal form to a vertical form:

1100
0101 $\wedge$

?

The above vertical form is different from the classical arithmetic vertical form, the operator $\wedge$ is to the right of the number in the second row. By this way, there is no need to insert spaces for aligning the numbers in the rows.

Here is the method of how to calculate:

1. Arrange the numbers vertically, and line up the digits by column.

2. Draw a horizontal line underneath the bottom row, and place the operator symbol to the right of the number in the bottom row.

3. Applying the calculation formulas according to the operator, and starting from the leftmost column, perform the calculation on digits in two rows of the column.

4. Place the result directly below the horizontal line.

5. Repeat the process in each column moving right until there is no more column to calculate.

Let us illustrate the application of the calculation method. Since the operator is "$\wedge$" in the given expression (A $\wedge$ $\neg$B), we will use the set of the "$\wedge$" calculation formulas:

$(1 \wedge 1) \leftrightarrow 1$
$(1 \wedge 0) \leftrightarrow 0$
$(0 \wedge 1) \leftrightarrow 0$
$(0 \wedge 0) \leftrightarrow 0$

The following is the calculation process column by column using the "$\wedge$" calculation formulas:

First column	Second column	Third column	Fourth column
$(1 \wedge 0) \leftrightarrow 0$	$(1 \wedge 1) \leftrightarrow 1$	$(0 \wedge 0) \leftrightarrow 0$	$(0 \wedge 1) \leftrightarrow 0$
1###	#1##	##0#	###0
0### $\wedge$	#1## $\wedge$	##0# $\wedge$	###1 $\wedge$
0###	#1##	##0#	###0

The result of combining the four digits is: 0100. And this is the truth value of (A $\wedge$ $\neg$B):

$(1100 \wedge 0101) \leftrightarrow 0100$

1100	//A
0101 $\wedge$	//$\neg$B
0100	//(A $\wedge$ $\neg$B)

Note: "//" in the above is a comment symbol.

It should be noted that the digital calculation discussed in this paper is a binary operation without carry. The calculation result of each column does not affect the adjacent columns.

Let's see two more examples:

Given $(A \to \neg B)$
The connective is "$\to$", we will use the "$\to$" calculation formulas:

$(1 \to 1) \leftrightarrow 1$
$(1 \to 0) \leftrightarrow 0$
$(0 \to 1) \leftrightarrow 1$
$(0 \to 0) \leftrightarrow 1$

1100	//A
$\underline{0101 \to}$	//$\neg$B
0111	//$(A \to \neg B)$

Given $(\neg A \vee B)$
The connective is "$\vee$", we will use the "$\vee$" calculation formulas:

$(1 \vee 1) \leftrightarrow 1$
$(1 \vee 0) \leftrightarrow 1$
$(0 \vee 1) \leftrightarrow 1$
$(0 \vee 0) \leftrightarrow 0$

0011	//$\neg$A
$\underline{1010 \vee}$	//B
1011	//$(\neg A \vee B)$

Note that we must choose the calculation formulas according to the operator (connective).

2.3 Calculation Rules of Negative Sign and Positive Sign

It should be noted that, the relation between a state symbol ($\neg$ or $+$) and an expression that the symbol added to is "$\leftrightarrow$". Thus, we must perform the "$\leftrightarrow$" calculation to obtain the correct result. The following are two rules for positive and negative signs:

1. The rule of the negative sign: when a negative sign is added to or removed from an expression, the number of the new expression will be changed to the opposite number.

2. The rule of the positive sign: when a positive sign is added to or removed from an expression, the expression and its number remain the same. Since the addition or removal of the positive sign does not affect the truth value of an expression, the positive sign can be omitted.

Both rules can be demonstrated through the digital calculations. Since the operator is "$\leftrightarrow$", we will use the "$\leftrightarrow$" calculation formulas:

Add a negative sign: $\neg(A \wedge B)$

0000	//$\neg$
$\underline{1000 \leftrightarrow}$	//$(A \wedge B)$
0111	//$\neg(A \wedge B)$

Add a positive sign: $+A$

11	//+
$\underline{10 \leftrightarrow}$	//A
10	//+A

The number 1000 is changed to the opposite 0111	The number 10 remains the same 10

It should be noted that we should choose the primitive numbers based on the number of distinct elements contained in the given expression. Generally, for a one-element expression, we should choose 2-digit primitive numbers (see example above on the right). for a two-element expression, we should choose 4-digit primitive numbers (see example above on the left), for a three-element expression, we should use 8-digit primitive numbers, and for a four-element expression, we should use 16-digit primitive numbers.

2.4 Two Essential Components of the Digital Calculation Method

As demonstrated above, truth values can be obtained through the digital calculation method without using a truth table, achieving the same outcome. In other words, this method achieves equivalent results while eliminating the need for using a truth table.

This method has two essential components:

1. The primitive numbers.

2. The digital calculation formulas.

By utilizing only primitive numbers and the binary calculation formulas, we can calculate the truth value of a propositional expression, regardless of how many elements or how many digits involved.

3 The Expression-number Lookup Tables

As illustrated above, we can calculate the truth value of a propositional expression by using the digital calculation method. Therefore, we can obtain the truth values of all basic expressions through calculation and store them in lookup tables for future use. The data saved in the tables is reusable. See the relational expression-number lookup tables in the appendix of this paper.

While the digital calculation method has demonstrated the advantages over the conventional methods, it can be even more efficient when used with the pre-calculated expression-number lookup tables. These pre-generated tables can provide two major conveniences:

Reduce Operation Steps

By using the expression-number lookup tables, we can avoid repeated calculations and reduce the number of operation steps. When we need to process a basic relational expression, we can directly find its number of truth value in the lookup table without having to calculate it again.

Let's see an example in which we can obtain the numbers of sub expressions either by calculation from primitive numbers or by looking up the lookup table:

Given: $\neg(A \vee B) \vee (\neg A \wedge B) \leftrightarrow \neg A$

Left-hand side:

Calculation from primitive numbers:

1100	//A
1010 $\vee$	//B
1110	//(A $\vee$ B)
0000 $\leftrightarrow$	//$\neg$
0001	//$\neg$(A $\vee$ B)
0011	//$\neg$A
1010 $\wedge$	//B
0010	//($\neg$A $\wedge$ B)
0001	//$\neg$(A $\vee$ B)
0010 $\vee$	//($\neg$A $\wedge$ B)
0011	//$\neg$(A $\vee$ B) $\vee$ ($\neg$A $\wedge$ B)

Right-hand side:

0011	//$\neg$A

Compare two sides:

0011	//$\neg$(A $\vee$ B) $\vee$ ($\neg$A $\wedge$ B)
0011	//$\neg$A

Conclusion: two sides are equivalent

Left-hand side:

Checking the lookup table, we found:

$\neg$(A $\vee$ B) is 0001
($\neg$A $\vee$ B) is 0010

0001	//$\neg$(A $\vee$ B)
0010 $\vee$	//($\neg$A $\wedge$ B)
0011	//$\neg$(A $\vee$ B) $\vee$ ($\neg$A $\wedge$ B)

Right-hand side:

0011	//$\neg$A

Compare two sides:

0011	//$\neg$(A $\vee$ B) $\vee$ ($\neg$A $\wedge$ B)
0011	//$\neg$A

Conclusion: two sides are equivalent

As we can see that, in the example on the left, we calculate the numbers of the sub expressions from the primitive numbers. But in the example on the right, we obtain the numbers by searching the lookup table. Obviously, looking up a number in a table is simpler and faster than calculating it from scratch.

Directly and Quickly Find Logical Equivalences

From the lookup tables, we can directly and quickly find many equivalences in propositional logic, and no further proof is needed. Because any expressions with the same numerical value are logically equivalent, even though their propositional forms appear different.

For example, in the appendix "Two-element Relational Expression-number Lookup Ta-

ble", we can directly find the following A-B / B-A formulas:

$(A \wedge B) \leftrightarrow (B \wedge A)$
$(A \wedge \neg B) \leftrightarrow (\neg B \wedge A)$
$(\neg A \wedge B) \leftrightarrow (B \wedge \neg A)$
$(\neg A \wedge \neg B) \leftrightarrow (\neg B \wedge \neg A)$

$(A \vee B) \leftrightarrow (B \vee A)$
$(A \vee \neg B) \leftrightarrow (\neg B \vee A)$
$(\neg A \vee B) \leftrightarrow (B \vee \neg A)$
$(\neg A \vee \neg B) \leftrightarrow (\neg B \vee \neg A)$

$(A \rightarrow B) \leftrightarrow (\neg B \rightarrow \neg A)$
$(A \rightarrow \neg B) \leftrightarrow (B \rightarrow \neg A)$
$(\neg A \rightarrow B) \leftrightarrow (\neg B \rightarrow A)$
$(\neg A \rightarrow \neg B) \leftrightarrow (B \rightarrow A)$

$(A \oplus B) \leftrightarrow (B \oplus A)$
$(A \oplus \neg B) \leftrightarrow (\neg B \oplus A)$
$(\neg A \oplus B) \leftrightarrow (B \oplus \neg A)$
$(\neg A \oplus \neg B) \leftrightarrow (\neg B \oplus \neg A)$

$(A \leftrightarrow B) \leftrightarrow (B \leftrightarrow A)$
$(A \leftrightarrow \neg B) \leftrightarrow (\neg B \leftrightarrow A)$
$(\neg A \leftrightarrow B) \leftrightarrow (B \leftrightarrow \neg A)$
$(\neg A \leftrightarrow \neg B) \leftrightarrow (\neg B \leftrightarrow \neg A)$

Due to the length limitation of this paper, we will not list all the hundreds of formulas here.

4 Proof Method and Criteria

4.1 Digitalization of Proof Method

When there is a given expression, it could be difficult to intuitively and immediately tell whether it is true or not. To prove it, if we use the statement-based method, we have to perform complex derivations, applying theorems, changing symbols and statements. If we use the truth table method, we have to make a truth table and perform a tedious filling operation. However, using the digital calculation method, we only need to calculate the numbers to determine whether the formula is true or not. We can call this method a digitalized proof method. The method includes the following three steps:

1. Find the number on the left-hand side;

2. Find the number on the right-hand side;

3. Perform a calculation (based on the operator) using the numbers of the two sides and apply the proof criterion to determine the result.

4.2 Digitalization of Proof Criteria

A proof criterion is a standard and rule used to judge and verify statements. There are three categories of proof criteria: truth criterion, falsity criterion, and criteria of other values. We can digitalize a proof criterion to a digital number and use it for logical proof. Based on the numbers of elements in a relation, the digital number of the proof criterion varies:

Table 4.1: Three Categories of Proof Criteria

Criterion	Non-element	One-element	Two-element	Three-element	Four-element
Truth	1	11	1111	1111-1111	1111-1111-1111-1111
Falsity	0	00	0000	0000-0000	0000-0000-0000-0000
Other Values	n/a	10, 01	1000, ...	1111-0000, ...	1111-1111-0000-0000, ...

For example, if it is a non-element relation, the number of the truth criterion is 1. For a two-element relation, the truth criterion number is 1111. And for a three-element relation, the truth number is 1111-1111.

It should be noted that, the truth-value systems vary depending on how many elements are contained in a relation. For non-element relations, we only need a 2-truth-value system. For one-element relations, a 4-truth-value system (2×2) is required. For two-element relations, a 16-truth-value system (4×4) is required. For three-element relations, there are 256 truth values (16×16). And for four-element relations, there are 65,536 truth values (256×256).

4.3 Examples of Using the Digital Proof Method and Criteria

Prove to be true:

Let's prove a formula using the above proof method and the truth criterion:

Given: $\neg A \rightarrow (A \rightarrow B)$

Left-hand side:

0011 //$\neg$A

Right-hand side:

1100 //A
1011 $\rightarrow$ //B
1011 //(A $\rightarrow$ B)

Compare two sides:

0011 //$\neg$A
1011 $\rightarrow$ //(A $\rightarrow$ B)
1111 //Proved to be true

The result is 1111, which indicates that the formula is proved to be true.

Prove to be false:

Let's see two examples:

$((A \to B) \to (((\neg A \vee \neg B) \to A) \wedge \neg A)$

Left-hand side:

1100	//A
1010 $\to$	//B
1011	//(A $\to$ B)

Right-hand side:

0011	//$\neg$A
0101 $\vee$	//$\neg$B
0111	//($\neg$A $\vee$ $\neg$B)
1100 $\to$	//A
1100	//(($\neg$A $\vee$ $\neg$B) $\to$ A)
0011 $\wedge$	//$\neg$A
0000	//(((($\neg$A $\vee$ $\neg$B) $\to$ A) $\wedge$ $\neg$A)

Compare two sides:

1011	//(A $\to$ B)
0000 $\to$	//(((($\neg$A $\vee$ $\neg$B) $\to$ A) $\wedge$ $\neg$A)
0000	//Proved to be false

$((\neg A \vee B) \oplus \neg A) \leftrightarrow (A \to \neg B)$

Left-hand side:

0011	//$\neg$A
1010 $\vee$	//B
1011	//($\neg$A $\vee$ B)
0011 $\oplus$	//$\neg$A
1000	//(($\neg$A $\vee$ B) $\oplus$ $\neg$A)

Right-hand side:

1100	//A
0101 $\to$	//$\neg$B
0111	//(A $\to$ $\neg$B)

Compare two sides:

1000	//(($\neg$A $\vee$ B) $\oplus$ $\neg$A)
0111 $\leftrightarrow$	//(A $\to$ $\neg$B)
0000	//Proved to be false

Both results are 0000, indicating that the two given expressions are proved to be false.

Prove to be Something Other Than True or False:

In a 16-truth-value system, truth and falsity are two of the 16 values. If a result number is not the number of truth nor falsity, then it falls within the other 14 numbers. The method for proving an expression to be "something else" is similar to the methods of truth proof and falsity proof. It also includes three steps, and the only difference is that, in Step 3: perform a calculation (based on the operator) using the numbers of the two sides. And whatever the outcome is, it proves that the relation between the left-hand side and the right-hand side is the calculation result. Let's see an example.

Given: A $\vee$ ($\neg$A $\wedge$ B)

Left-hand side:

1100	//A

Right-hand side:

0011	//$\neg$A
1010 $\wedge$	//B

0010 //($\neg$A $\wedge$ B)

Compare two sides:

1100 //A
0010 $\vee$ //($\neg$A $\wedge$ B)
1110 //Proved to be something other than true and false

The result is 1110, which indicates that the "$\vee$" relation between A on the left-hand side (LHS) and ($\neg$A $\wedge$ B) on the right-hand side (RHS) is proved to be 1110.

It is well known that, if the calculation formulas are correct, and the calculation is performed properly, then whatever the result of the calculation is, it reflects the relation in the given expression. For example: in arithmetic, 1+4 =5. The result 5 reflects the "+" relation between 1 and 4. It is the same in the digital calculation of logical expressions. Back to the above example:

(A $\vee$ ($\neg$A $\wedge$ B)) $\leftrightarrow$ 1110

The result 1110 reflects the "$\vee$" relation between A and ($\neg$A $\wedge$ B). This result is not "true" (1111) nor "false" (0000), but "something else" (1110). 1110 represents one of the remaining available 14 numerical values other than true (1111) and false (0000).

4.4 Prove All Laws and Theorems of Propositional Logic Using the Digital Method

With the digital calculation method, we can prove all laws and theorems of Boolean logic and propositional logic. Let's see some examples.

Proof of Equivalence:

Let's prove a formula of Boolean algebra using the above proof method:

Given: $\neg$(A $\wedge$ B) $\leftrightarrow$ ($\neg$A $\vee$ $\neg$B)

Left-hand side:

1100 //A
1010 $\wedge$ //B
1000 //(A $\wedge$ B)
0000 $\leftrightarrow$ //$\neg$
0111 //$\neg$(A $\wedge$ B)

Right-hand side:

0011 //$\neg$A
0101 $\vee$ //$\neg$B

0111 //(¬A ∨ ¬B)

Compare two sides:

0111 //¬(A ∧ B)
0111 ↔ //(¬A ∨ ¬B)
1111 //Proved to be true

Proof of Logical Implication:

Let's see how to prove implication expressions. The proof method is the same. Note: since in an implication expression, the operator connecting LHS and RHS is "→", in Step 3, the operation will be the "→" calculation using the numbers of the two sides. If the calculation result in Step 3 is 1111 (or 1111-1111 in three-element relations), then it indicates that the two sides have the implication relation, and the given implication expression is proved to be true.

There are some implication axiom systems, in which, the formulas are using an implication operator to connect LHS and RHS. For examples, the following are two formulas from two implication axiom systems:

(A ∧ B) → A //Rosser J. Barkley's system [7]
A → (B → A) //Gottlob Frege's system [8]

Let's use the proof method to verify them:

(A ∧ B) → A

Left-hand side:

1100 //A
1010 ∧ //B
1000 //(A ∧ B)

Right-hand side:

1100 //A

Compare two sides:

1000 //(A ∧ B)
1100 → //A
1111 //Proved to be true

A → (B → A)

Left-hand side

1100 //A

Right-hand side:

1010 //B
1100 → //A
1101 //(B → A)

Compare two sides:

1100 //A
1101 → //(B → A)
1111 //Proved to be true

Two results are 1111, indicating that the two formulas are proved to be true.

Let's prove the classical syllogism $((A \to B) \land (B \to C)) \to (A \to C)$. Since it involves three elements A, B and C, we will use 8-digit numbers of three-element expressions:

Checking the lookup table, we found:

$(A \to B)$ is 1100-1111
$(B \to C)$ is 1011-1011
$(A \to C)$ is 1010-1111

Left-hand side:

1100-1111	//$(A \to B)$
1011-1011 $\land$	//$(B \to C)$
1000-1011	//$(A \to B) \land (B \to C)$

Right-hand side:

1010-1111	//$(A \to C)$

Compare two sides:

1000-1011	//$(A \to B) \land (B \to C)$
1010-1111 $\to$	//$(A \to C)$
1111-1111	//Proved to be true

Proof of Disjunction Formulas:

In some axiom systems, the operators used for connecting LHS and RHS are not "$\leftrightarrow$" nor "$\to$". For example, in Russell-Bernays axiom system [9], the following formulas are only using operator "$\lor$" for connection:

$\neg(\neg B \lor C) \lor (\neg(A \lor B) \lor (A \lor C))$
$\neg(A \lor B) \lor (B \lor A)$
$\neg A \lor (B \lor A)$
$\neg(A \lor A) \lor A$
$\neg(C \lor A) \lor (C \lor (A \lor B))$
$\neg(C \lor A) \lor ((C \lor B) \lor A)$

We can also use the proof method to verify the above formulas. Note: since the operator connecting LHS and RHS is "$\lor$", in Step 3, the operation will be the "$\lor$" calculation using the numbers of the two sides. If the calculation result in Step 3 is 1111 or 1111-1111 (in three-element relations), then it indicates that the formula is true.

Let's prove one of the above formulas:

$\neg(A \lor B) \lor (B \lor A)$

Checking the lookup table, we found:

Left-hand side: $\neg(A \vee B)$ is 0001
Right-hand side: $(B \vee A)$ is 1110

Compare two sides:

0001	//$\neg(A \vee B)$
1110 $\vee$	//$(B \vee A)$
1111	//Proved to be true

The result is 1111. Thus, the formula is proved to be true.

Using the digital calculation method demonstrated above, we can prove all laws and theorems of Boolean logic and propositional logic, including those theorems of the well-known axiom systems, such as Frege's axiom system [2], and Łukasiewicz's axiom system [10]. Due to the length limitation of this paper, we will not prove all of them one by one here. Interested readers can verify it themselves using the method demonstrated above.

It is important to note that, propositions provable through the statement-based method and table-based method can be proved more efficiently by the new number-based method, while propositions unprovable through the conventional methods are also unprovable using the new method. In other words, the digital calculation method offers an enhanced alternative to existing approaches, rather than addressing the currently unresolved challenges in the field. Therefore, the scope of its application remains consistent with existing methods. The existing challenges related to propositional logic and Boolean logic, which remain unresolved by conventional methods, should not be considered within the scope of this method's objectives.

5 Comparison of Three Reasoning Methods

Compared with the conventional methods, the digital calculation method appears to have efficiency advantages in a single reasoning, while being more efficient when used for multiple reasonings over time.

5.1 Comparative Advantages in a Single Reasoning

We have seen that various techniques have been attempted to prove an equivalence. Despite the methodical diversity, the underlying principle has never changed, that is to find out whether the left-hand side (LHS) and the right-hand side (RHS) are equivalent.

Let's compare the three methods by proving an equivalence.

The Statement-deriving Reasoning Method:

This is a statement-based reasoning method. Let's see an example.

Given: $\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$

1. $\neg(A \wedge B)$	... premise
2. $\neg(\neg A \vee \neg B)$	... assumed [a]
3. $\neg A$	... assumed [b]
4. $\neg A \vee \neg B$	... from 3 by $\vee$-intro
5. $\perp$	... from 2 and 4
6. A	... from 3 and 5 by RAA, discharging [b]
7. $\neg B$	... assumed [c]
8. $\neg A \vee \neg B$	... from 7 by $\vee$-intro
9. $\perp$	... from 2 and 8
10. B	... from 7 and 9 by RAA, discharging [c]
11. $A \wedge B$	... from 6 and 10 by $\wedge$-intro
12. $\perp$	
13. $\neg A \vee \neg B$	... discharging [a]

The final result proves that the left-hand side is equivalent to the right-hand side.

This is a typical statement-deriving reasoning - applying other propositions (axioms, laws, theorems or rules), replacing or eliminating symbols, and adding new lines step by step to reach the conclusion. A proof process like the above requires a high level of comprehension from both proposer and learners. Even for those with higher comprehension abilities, it often takes some time to figure out and understand the reasoning process.

The Truth-table Reasoning Method:

This is a table-based reasoning method. Compared to the statement-deriving reasoning method, the truth-table method has the advantages of being intuitive and standardized. Let's use the same hypothetical example.

Given: $\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$

Table 5.1 Truth-table Reasoning Example

A	B	$(A \wedge B)$	$\neg(A \wedge B)$	$\neg A$	$\neg B$	$(\neg A \vee \neg B)$
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

The column $\neg(A \wedge B)$ and the column $(\neg A \vee \neg B)$ have the same truth value, therefore the two sides are equivalent.

Although the truth table method has demonstrated its advantages over the statement-deriving reasoning method, it still presents some practical challenges. When using this method, how to fill in the table is not arbitrary, but must follow certain rules. For example, to fill the four cells in a column $(A \wedge B)$, we have to follow the rule "$A \wedge B$ = true, only when A = true and B = true". This is similar to applying a theorem in a

statement-deriving reasoning. Although the rule is not explicitly mentioned or marked within the table, it exists implicitly in the user's mind, guiding how to fill in the table. This means that the effectiveness of this method relies on familiarity with the rules, combined with relevant training and experience.

The Digital Calculation Method:

Let's use the digital calculation method to prove the same example.

Given: $\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$

Checking the lookup table, we found:

Left-hand side: $\neg(A \wedge B)$ is 0111
Right-hand side: $(\neg A \vee \neg B)$ is 0111

Compare two sides:

0111	//$\neg(A \wedge B)$
0111 $\leftrightarrow$	//$(\neg A \vee \neg B)$
1111	//Proved to be true

The result is 1111, proving that the two sides are equivalent and the given formula is true. Compared to the two conventional methods, the digital calculation method is much easier to conduct. It does not require to remember and understand those theorems or rules because it does not need to apply them.

As we can see that, the digital calculation method is functionally equivalent to the two conventional methods.

Conventional approaches for performing propositional reasoning often present challenges, even for individuals of exceptional intellect. While adherence to these methods theoretically guarantees certainty, their effectiveness and success in practice are contingent upon a comprehensive understanding of the underlying laws and rules, coupled with adequate training, experience, and the ability to apply them. Even with significant effort, some users may still encounter difficulties in achieving desired outcomes.

In contrast, the digital calculation method appears to have the advantages over the conventional methods. Applying this method, all we need are a few primitive numbers and the simple calculation formulas. It eliminates the need for deriving sentences, citing theorems and substituting symbols, and obviates the need for filling truth table cells and analyzing the table patterns. Therefore, it is no longer necessary to memorize and understand those theorems and patterns, and figure out how to use them. The operation is much easier. It simplifies the reasoning and proof, and minimizes the reliance on prior experience and expertise.

5.2 More Efficient in Multiple & Long-term Use with Data Reusability

A significant shortcoming of the conventional reasoning methods is that the results of previous reasonings cannot be saved for later reuse. Every time we make an inference, we have to start from scratch and prove each step. Even if we have proved some of parts or steps many times in the past, we are still required to repeat them. For example, using the truth table method, every time an inference involves (A $\vee$ B), we have to put it in a column and perform the relevant operations, even if we have done this many times in previous inferences. This is obviously repetitive work in terms of multiple reasonings and long-term use.

The digital calculation method provides a solution to save and reuse the previous reasoning results and share them with others. With this method, we can save the data of all basic expressions in the pre-calculated lookup tables and we don't have to repeat the same operations every time by every individual to prove the same basic expressions that we have proved countless time in the past by ourselves or by someone else. Because we can directly find the proof results in the lookup tables. Since the calculation results are stored in reusable and shareable lookup tables, it eliminating the need for everyone to perform the same operations repeatedly. This not only saves time for an individual by avoiding redundant operations, but also benefits the entire user community through reusable, shared results, saving significant time across society.

By using the pre-calculated expression-number lookup tables, we can avoid repeated reasoning operations, and we can also directly and quickly find logical equivalences from the lookup tables, and no further proof is needed.

5.3 Summary of the Comparison

The comparison of the three methods can be summarized as follows:

Table 5.2 Comparison of Three Reasoning Methods

Method	Category	Operation	Use	Simplicity	Ease	Data Reuse
Statement-deriving	Statement-based	Write statements & change symbols	Theorems or rules	Complex	Difficult	No
Truth-table	Table-based	Fill table cells with T/F (1/0)	Theorems or rules	Inter-mediate	Inter-mediate	No
Digital-calculation	Number-based	Calculate digital numbers	Calculation formulas	Simple	Easy	Yes

6 Massive Data Processing and Large-scale Computation

6.1 Quickly Find Valid Formulas Among Large Sets of Combinations

People can arbitrarily propose expressions, and the randomly generated expressions can have countless possibilities. Assuming the symbol * represents any type of the five relations, and that elements A, B, and C have positive and negative states, we can calculate how many possible combinations an expression type contains. For example, each of the

following expression types contains possible combinations ranging from dozens to tens of thousands:

$(A * B) \rightarrow A$	//40 combinations
$(A \rightarrow B) \rightarrow ((A * B) \rightarrow (B * C))$	//256 combinations
$((A * B) * A) \leftrightarrow (A * B)$	//800 combinations
$(A * B) \rightarrow ((A * B) \oplus (A * B))$	//3,800 combinations
$(A * B) \leftrightarrow ((A * B) * (A * B))$	//19,000 combinations

The digital calculation method provides a feasible approach for processing large-scale propositional expressions. To process massive expressions, we need a simple program that performs the following tasks:

1. Generate all possible combinations, calculate their numbers, and store the data in a combination table (lookup table).

2. Identify valid formulas in the combination table using the determination criterion.

The following is an example combination table, containing a total of 256 combinations:

Table 6.1 Part of the Combination Table - $(A \rightarrow B) \rightarrow ((A * B) \rightarrow (B * C))$

Expression	Number
$(A \rightarrow B) \rightarrow ((A \wedge C) \rightarrow (B \wedge C))$	11111111
$(A \rightarrow B) \rightarrow ((A \wedge C) \rightarrow (B \wedge \neg C))$	01111111
...	...
$(A \rightarrow B) \rightarrow ((A \wedge C) \rightarrow (\neg B \wedge \neg C))$	01111111
$(A \rightarrow B) \rightarrow ((A \wedge C) \rightarrow (B \vee C))$	11111111
$(A \rightarrow B) \rightarrow ((A \wedge \neg C) \rightarrow (\neg B \wedge \neg C))$	10111111
...	...

Whether it is through human eye recognition or computer recognition, the valid formulas with the truth criterion number 11111111 can be easily and quickly identified in a table like the above.

In contrast, whether using the statement-based method or the truth-table method, it is a difficult task to generate and verify a large number of expressions. Even if it's technically feasible to generate 256 propositional inferences and 256 truth tables using a computer program, evaluating and comparing dozens to hundreds of pages of output results is a time-consuming and cognitively intensive process.

Since this method is amenable to large-scale computation, it can be used for artificial intelligence and automated reasoning.

6.2 Alternative Approach for Expression Simplification

One of the most practical areas of propositional logic and Boolean logic is how to simplify expressions in circuit design. With the ability to process massive amounts of data,

the digital calculation method offers an alternative approach for expression simplification.

Expression simplification, or logic optimization, is a process of finding a simpler equivalent representation of the given unsimplified expression. It is often used in circuit design to obtain the smallest logic circuit that evaluates to the same values as the unsimplified one. In the 1950s, the K-Map method [11], Petrick's method [12] and Quine-McCluskey method [13] were introduced for simplifying expressions. These approaches all use a grid-like map method to find a simplified expression.

Applying the digital calculation method for expression simplification, the key idea is to find simpler equivalent expressions in the expression combination table based on the number of the given unsimplified expression. The process of the method includes the following steps:

1. Write a computer program script to generate a combination table (lookup table) that contains all possible expression combinations. This table can be saved to the database for future use.

2. Calculate the number of the given unsimplified expression.

3. Use the number obtained in Step 2 as the search criterion to search the combination table generated in Step 1 and find the equivalent number.

4. If an equivalent number is found in the combination table, then the expression that the number is associated with can be determined to be a simplified expression.

Let's see an example. Given:

f(A,B,C) = Σm(0,1,2,5,6,7) = A'B'C' + A'B'C + A'BC' + AB'C + ABC' + ABC

By calculating the given expression, we obtained its number: 11100111. Now we can use this number as the criterion to search the combination table. The following is a part of A-B-C combination table:

Table 6.2: Part of A-B-C Combination Table

Expression	Number
...	...
(A ∨ ¬C)	11110101
...	...
(A ∨ ¬B) ∨ ¬C	11110111
...	...
(A ∧ ¬B) ∨ (¬B ∧ C)	00110010
...	...
(A ∧ B) ∨ (¬B ∧ C) ∨ (¬A ∧ C)	11101010
(A ∧ B) ∨ (¬B ∧ C) ∨ (¬A ∧ ¬C)	11100111
(A ∧ B) ∨ (¬B ∧ ¬C) ∨ (A ∧ C)	11110001

...	...
$(\neg A \wedge \neg B) \vee (B \wedge C) \vee (\neg A \wedge \neg C)$	10001111
$(\neg A \wedge \neg B) \vee (B \wedge \neg C) \vee (A \wedge C)$	11100111
...	...

By searching the table above, we can quickly find that there are two rows (highlighted in gray) containing the same number 11100111:

$(A \wedge B) \vee (\neg B \wedge C) \vee (\neg A \wedge \neg C)$ //AB + B'C + A'C'
$(\neg A \wedge \neg B) \vee (B \wedge \neg C) \vee (A \wedge C)$ //A'B' + BC' + AC

Therefore, the two combinations are the simplified and minimum expressions that are equivalent to the given unsimplified expression. No further action is required.

A'B'C' + A'B'C + A'BC' + AB'C + ABC' + ABC $\leftrightarrow$ 11100111
AB + B'C + A'C' $\leftrightarrow$ 11100111
A'B' + BC' + AC $\leftrightarrow$ 11100111

7 Appendix

7.1 Primitive Number Lookup Table

Table 7.1: Primitive Number Lookup Table

Expr.	Non-element +/¬	One-element A	Two-element AB	Three-element ABC	Four- element ABCD
+	1	11	1111	1111-1111	1111-1111-1111-1111
¬	0	00	0000	0000-0000	0000-0000-0000-0000
A		10	1100	1111-0000	1111-1111-0000-0000
¬A		01	0011	0000-1111	0000-0000-1111-1111
B			1010	1100-1100	1111-0000-1111-0000
¬B			0101	0011-0011	0000-1111-0000-1111
C				1010-1010	1100-1100-1100-1100
¬C				0101-0101	0011-0011-0011-0011
D					1010-1010-1010-1010
¬D					0101-0101-0101-0101

7.2 Digital Calculation Formulas

Table 7.2: Eight Sets of Digital Calculation Formulas

$\wedge$	$\vee$	$\rightarrow$	$\oplus$
$(1 \wedge 1) \leftrightarrow 1$	$(1 \vee 1) \leftrightarrow 1$	$(1 \rightarrow 1) \leftrightarrow 1$	$(1 \oplus 1) \leftrightarrow 0$
$(1 \wedge 0) \leftrightarrow 0$	$(1 \vee 0) \leftrightarrow 1$	$(1 \rightarrow 0) \leftrightarrow 0$	$(1 \oplus 0) \leftrightarrow 1$
$(0 \wedge 1) \leftrightarrow 0$	$(0 \vee 1) \leftrightarrow 1$	$(0 \rightarrow 1) \leftrightarrow 1$	$(0 \oplus 1) \leftrightarrow 1$
$(0 \wedge 0) \leftrightarrow 0$	$(0 \vee 0) \leftrightarrow 0$	$(0 \rightarrow 0) \leftrightarrow 1$	$(0 \oplus 0) \leftrightarrow 0$
$\uparrow$	$\downarrow$	$\leftarrow$	$\leftrightarrow$

$(1 \uparrow 1) \leftrightarrow 0$	$(1 \downarrow 1) \leftrightarrow 0$	$(1 \leftarrow 1) \leftrightarrow 1$	$(1 \leftrightarrow 1) \leftrightarrow 1$
$(1 \uparrow 0) \leftrightarrow 1$	$(1 \downarrow 0) \leftrightarrow 0$	$(1 \leftarrow 0) \leftrightarrow 1$	$(1 \leftrightarrow 0) \leftrightarrow 0$
$(0 \uparrow 1) \leftrightarrow 1$	$(0 \downarrow 1) \leftrightarrow 0$	$(0 \leftarrow 1) \leftrightarrow 0$	$(0 \leftrightarrow 1) \leftrightarrow 0$
$(0 \uparrow 0) \leftrightarrow 1$	$(0 \downarrow 0) \leftrightarrow 1$	$(0 \leftarrow 0) \leftrightarrow 1$	$(0 \leftrightarrow 0) \leftrightarrow 1$

7.3 Relational Expression-number Lookup Tables

Table 7.3: Two-element Relational Expression-number Lookup Table

(*This dataset can be downloaded via https://doi.org/10.7910/DVN/HWRWSR*)

A-B		B-A		¬(A-B)		¬(B-A)	
Expression	Num	Expression	Num	Expression	Num	Expression	Num
$(A \wedge B)$	1000	$(B \wedge A)$	1000	$\neg(A \wedge B)$	0111	$\neg(B \wedge A)$	0111
$(A \wedge \neg B)$	0100	$(\neg B \wedge A)$	0100	$\neg(A \wedge \neg B)$	1011	$\neg(\neg B \wedge A)$	1011
$(\neg A \wedge B)$	0010	$(B \wedge \neg A)$	0010	$\neg(\neg A \wedge B)$	1101	$\neg(B \wedge \neg A)$	1101
$(\neg A \wedge \neg B)$	0001	$(\neg B \wedge \neg A)$	0001	$\neg(\neg A \wedge \neg B)$	1110	$\neg(\neg B \wedge \neg A)$	1110
$(A \vee B)$	1110	$(B \vee A)$	1110	$\neg(A \vee B)$	0001	$\neg(B \vee A)$	0001
$(A \vee \neg B)$	1101	$(\neg B \vee A)$	1101	$\neg(A \vee \neg B)$	0010	$\neg(\neg B \vee A)$	0010
$(\neg A \vee B)$	1011	$(B \vee \neg A)$	1011	$\neg(\neg A \vee B)$	0100	$\neg(B \vee \neg A)$	0100
$(\neg A \vee \neg B)$	0111	$(\neg B \vee \neg A)$	0111	$\neg(\neg A \vee \neg B)$	1000	$\neg(\neg B \vee \neg A)$	1000
$(A \rightarrow B)$	1011	$(B \rightarrow A)$	1101	$\neg(A \rightarrow B)$	0100	$\neg(B \rightarrow A)$	0010
$(A \rightarrow \neg B)$	0111	$(\neg B \rightarrow A)$	1110	$\neg(A \rightarrow \neg B)$	1000	$\neg(\neg B \rightarrow A)$	0001
$(\neg A \rightarrow B)$	1110	$(B \rightarrow \neg A)$	0111	$\neg(\neg A \rightarrow B)$	0001	$\neg(B \rightarrow \neg A)$	1000
$(\neg A \rightarrow \neg B)$	1101	$(\neg B \rightarrow \neg A)$	1011	$\neg(\neg A \rightarrow \neg B)$	0010	$\neg(\neg B \rightarrow \neg A)$	0100
$(A \oplus B)$	0110	$(B \oplus A)$	0110	$\neg(A \oplus B)$	1001	$\neg(B \oplus A)$	1001
$(A \oplus \neg B)$	1001	$(\neg B \oplus A)$	1001	$\neg(A \oplus \neg B)$	0110	$\neg(\neg B \oplus A)$	0110
$(\neg A \oplus B)$	1001	$(B \oplus \neg A)$	1001	$\neg(\neg A \oplus B)$	0110	$\neg(B \oplus \neg A)$	0110
$(\neg A \oplus \neg B)$	0110	$(\neg B \oplus \neg A)$	0110	$\neg(\neg A \oplus \neg B)$	1001	$\neg(\neg B \oplus \neg A)$	1001
$(A \leftrightarrow B)$	1001	$(B \leftrightarrow A)$	1001	$\neg(A \leftrightarrow B)$	0110	$\neg(B \leftrightarrow A)$	0110
$(A \leftrightarrow \neg B)$	0110	$(\neg B \leftrightarrow A)$	0110	$\neg(A \leftrightarrow \neg B)$	1001	$\neg(\neg B \leftrightarrow A)$	1001
$(\neg A \leftrightarrow B)$	0110	$(B \leftrightarrow \neg A)$	0110	$\neg(\neg A \leftrightarrow B)$	1001	$\neg(B \leftrightarrow \neg A)$	1001
$(\neg A \leftrightarrow \neg B)$	1001	$(\neg B \leftrightarrow \neg A)$	1001	$\neg(\neg A \leftrightarrow \neg B)$	0110	$\neg(\neg B \leftrightarrow \neg A)$	0110

Table 7.4: Three-element Relational Expression-number Lookup Table

(*This dataset can be downloaded via https://doi.org/10.7910/DVN/UGRP3U*)

A-B		B-C		(A-C)	
Expression	Number	Expression	Number	Expression	Number
$(A \wedge B)$	1100-0000	$(B \wedge C)$	1000-1000	$(A \wedge C)$	1010-0000
$(A \wedge \neg B)$	0011-0000	$(B \wedge \neg C)$	0100-0100	$(A \wedge \neg C)$	0101-0000
$(\neg A \wedge B)$	0000-1100	$(\neg B \wedge C)$	0010-0010	$(\neg A \wedge C)$	0000-1010
$(\neg A \wedge \neg B)$	0000-0011	$(\neg B \wedge \neg C)$	0001-0001	$(\neg A \wedge \neg C)$	0000-0101
$(A \vee B)$	1111-1100	$(B \vee C)$	1110-1110	$(A \vee C)$	1111-1010
$(A \vee \neg B)$	1111-0011	$(B \vee \neg C)$	1101-1101	$(A \vee \neg C)$	1111-0101
$(\neg A \vee B)$	1100-1111	$(\neg B \vee C)$	1011-1011	$(\neg A \vee C)$	1010-1111

$(\neg A \vee \neg B)$	0011-1111	$(\neg B \vee \neg C)$	0111-0111	$(\neg A \vee \neg C)$	0101-1111
$(A \rightarrow B)$	1100-1111	$(B \rightarrow C)$	1011-1011	$(A \rightarrow C)$	1010-1111
$(A \rightarrow \neg B)$	0011-1111	$(B \rightarrow \neg C)$	0111-0111	$(A \rightarrow \neg C)$	0101-1111
$(\neg A \rightarrow B)$	1111-1100	$(\neg B \rightarrow C)$	1110-1110	$(\neg A \rightarrow C)$	1111-1010
$(\neg A \rightarrow \neg B)$	1111-0011	$(\neg B \rightarrow \neg C)$	1101-1101	$(\neg A \rightarrow \neg C)$	1111-0101
$(A \oplus B)$	0011-1100	$(B \oplus C)$	0110-0110	$(A \oplus C)$	0101-1010
$(A \oplus \neg B)$	1100-0011	$(B \oplus \neg C)$	1001-1001	$(A \oplus \neg C)$	1010-0101
$(\neg A \oplus B)$	1100-0011	$(\neg B \oplus C)$	1001-1001	$(\neg A \oplus C)$	1010-0101
$(\neg A \oplus \neg B)$	0011-1100	$(\neg B \oplus \neg C)$	0110-0110	$(\neg A \oplus \neg C)$	0101-1010
$(A \leftrightarrow B)$	1100-0011	$(B \leftrightarrow C)$	1001-1001	$(A \leftrightarrow C)$	1010-0101
$(A \leftrightarrow \neg B)$	0011-1100	$(B \leftrightarrow \neg C)$	0110-0110	$(A \leftrightarrow \neg C)$	0101-1010
$(\neg A \leftrightarrow B)$	0011-1100	$(\neg B \leftrightarrow C)$	0110-0110	$(\neg A \leftrightarrow C)$	0101-1010
$(\neg A \leftrightarrow \neg B)$	1100-0011	$(\neg B \leftrightarrow \neg C)$	1001-1001	$(\neg A \leftrightarrow \neg C)$	1010-0101
B-A		C-B		(C-A)	
Expression	Number	Expression	Number	Expression	Number
$(B \wedge A)$	1100-0000	$(C \wedge B)$	1000-1000	$(C \wedge A)$	1010-0000
$(\neg B \wedge A)$	0011-0000	$(\neg C \wedge B)$	0100-0100	$(\neg C \wedge A)$	0101-0000
$(B \wedge \neg A)$	0000-1100	$(C \wedge \neg B)$	0010-0010	$(C \wedge \neg A)$	0000-1010
$(\neg B \wedge \neg A)$	0000-0011	$(\neg C \wedge \neg B)$	0001-0001	$(\neg C \wedge \neg A)$	0000-0101
$(B \vee A)$	1111-1100	$(C \vee B)$	1110-1110	$(C \vee A)$	1111-1010
$(\neg B \vee A)$	1111-0011	$(\neg C \vee B)$	1101-1101	$(\neg C \vee A)$	1111-0101
$(B \vee \neg A)$	1100-1111	$(C \vee \neg B)$	1011-1011	$(C \vee \neg A)$	1010-1111
$(\neg B \vee \neg A)$	0011-1111	$(\neg C \vee \neg B)$	0111-0111	$(\neg C \vee \neg A)$	0101-1111
$(B \rightarrow A)$	1111-0011	$(C \rightarrow B)$	1101-1101	$(C \rightarrow A)$	1111-0101
$(\neg B \rightarrow A)$	1111-1100	$(\neg C \rightarrow B)$	1110-1110	$(\neg C \rightarrow A)$	1111-1010
$(B \rightarrow \neg A)$	0011-1111	$(C \rightarrow \neg B)$	0111-0111	$(C \rightarrow \neg A)$	0101-1111
$(\neg B \rightarrow \neg A)$	1100-1111	$(\neg C \rightarrow \neg B)$	1011-1011	$(\neg C \rightarrow \neg A)$	1010-1111
$(B \oplus A)$	0011-1100	$(C \oplus B)$	0110-0110	$(C \oplus A)$	0101-1010
$(\neg B \oplus A)$	1100-0011	$(\neg C \oplus B)$	1001-1001	$(\neg C \oplus A)$	1010-0101
$(B \oplus \neg A)$	1100-0011	$(C \oplus \neg B)$	1001-1001	$(C \oplus \neg A)$	1010-0101
$(\neg B \oplus \neg A)$	0011-1100	$(\neg C \oplus \neg B)$	0110-0110	$(\neg C \oplus \neg A)$	0101-1010
$(B \leftrightarrow A)$	1100-0011	$(C \leftrightarrow B)$	1001-1001	$(C \leftrightarrow A)$	1010-0101
$(\neg B \leftrightarrow A)$	0011-1100	$(\neg C \leftrightarrow B)$	0110-0110	$(\neg C \leftrightarrow A)$	0101-1010
$(B \leftrightarrow \neg A)$	0011-1100	$(C \leftrightarrow \neg B)$	0110-0110	$(C \leftrightarrow \neg A)$	0101-1010
$(\neg B \leftrightarrow \neg A)$	1100-0011	$(\neg C \leftrightarrow \neg B)$	1001-1001	$(\neg C \leftrightarrow \neg A)$	1010-0101
¬(A-B)		¬(B-C)		(¬(A-C))	
Expression	Number	Expression	Number	Expression	Number
$\neg(A \wedge B)$	0011-1111	$\neg(B \wedge C)$	0111-0111	$\neg(A \wedge C)$	0101-1111
$\neg(A \wedge \neg B)$	1100-1111	$\neg(B \wedge \neg C)$	1011-1011	$\neg(A \wedge \neg C)$	1010-1111
$\neg(\neg A \wedge B)$	1111-0011	$\neg(\neg B \wedge C)$	1101-1101	$\neg(\neg A \wedge C)$	1111-0101
$\neg(\neg A \wedge \neg B)$	1111-1100	$\neg(\neg B \wedge \neg C)$	1110-1110	$\neg(\neg A \wedge \neg C)$	1111-1010
$\neg(A \vee B)$	0000-0011	$\neg(B \vee C)$	0001-0001	$\neg(A \vee C)$	0000-0101
$\neg(A \vee \neg B)$	0000-1100	$\neg(B \vee \neg C)$	0010-0010	$\neg(A \vee \neg C)$	0000-1010
$\neg(\neg A \vee B)$	0011-0000	$\neg(\neg B \vee C)$	0100-0100	$\neg(\neg A \vee C)$	0101-0000
$\neg(\neg A \vee \neg B)$	1100-0000	$\neg(\neg B \vee \neg C)$	1000-1000	$\neg(\neg A \vee \neg C)$	1010-0000
$\neg(A \rightarrow B)$	0011-0000	$\neg(B \rightarrow C)$	0100-0100	$\neg(A \rightarrow C)$	0101-0000

¬(A → ¬B)	1100-0000	¬(B → ¬C)	1000-1000	¬(A → ¬C)	1010-0000
¬(¬A → B)	0000-0011	¬(¬B → C)	0001-0001	¬(¬A → C)	0000-0101
¬(¬A → ¬B)	0000-1100	¬(¬B → ¬C)	0010-0010	¬(¬A → ¬C)	0000-1010
¬(A ⊕ B)	1100-0011	¬(B ⊕ C)	1001-1001	¬(A ⊕ C)	1010-0101
¬(A ⊕ ¬B)	0011-1100	¬(B ⊕ ¬C)	0110-0110	¬(A ⊕ ¬C)	0101-1010
¬(¬A ⊕ B)	0011-1100	¬(¬B ⊕ C)	0110-0110	¬(¬A ⊕ C)	0101-1010
¬(¬A ⊕ ¬B)	1100-0011	¬(¬B ⊕ ¬C)	1001-1001	¬(¬A ⊕ ¬C)	1010-0101
¬(A ↔ B)	0011-1100	¬(B ↔ C)	0110-0110	¬(A ↔ C)	0101-1010
¬(A ↔ ¬B)	1100-0011	¬(B ↔ ¬C)	1001-1001	¬(A ↔ ¬C)	1010-0101
¬(¬A ↔ B)	1100-0011	¬(¬B ↔ C)	1001-1001	¬(¬A ↔ C)	1010-0101
¬(¬A ↔ ¬B)	0011-1100	¬(¬B ↔ ¬C)	0110-0110	¬(¬A ↔ ¬C)	0101-1010
¬(B-A)		¬(C-B)		(¬(C-A)	
Expression	Number	Expression	Number	Expression	Number
¬(B ∧ A)	0011-1111	¬(C ∧ B)	0111-0111	¬(C ∧ A)	0101-1111
¬(¬B ∧ A)	1100-1111	¬(¬C ∧ B)	1011-1011	¬(¬C ∧ A)	1010-1111
¬(B ∧ ¬A)	1111-0011	¬(C ∧ ¬B)	1101-1101	¬(C ∧ ¬A)	1111-0101
¬(¬B ∧ ¬A)	1111-1100	¬(¬C ∧ ¬B)	1110-1110	¬(¬C ∧ ¬A)	1111-1010
¬(B ∨ A)	0000-0011	¬(C ∨ B)	0001-0001	¬(C ∨ A)	0000-0101
¬(¬B ∨ A)	0000-1100	¬(¬C ∨ B)	0010-0010	¬(¬C ∨ A)	0000-1010
¬(B ∨ ¬A)	0011-0000	¬(C ∨ ¬B)	0100-0100	¬(C ∨ ¬A)	0101-0000
¬(¬B ∨ ¬A)	1100-0000	¬(¬C ∨ ¬B)	1000-1000	¬(¬C ∨ ¬A)	1010-0000
¬(B → A)	0000-1100	¬(C → B)	0010-0010	¬(C → A)	0000-1010
¬(¬B → A)	0000-0011	¬(¬C → B)	0001-0001	¬(¬C → A)	0000-0101
¬(B → ¬A)	1100-0000	¬(C → ¬B)	1000-1000	¬(C → ¬A)	1010-0000
¬(¬B → ¬A)	0011-0000	¬(¬C → ¬B)	0100-0100	¬(¬C → ¬A)	0101-0000
¬(B ⊕ A)	1100-0011	¬(C ⊕ B)	1001-1001	¬(C ⊕ A)	1010-0101
¬(¬B ⊕ A)	0011-1100	¬(¬C ⊕ B)	0110-0110	¬(¬C ⊕ A)	0101-1010
¬(B ⊕ ¬A)	0011-1100	¬(C ⊕ ¬B)	0110-0110	¬(C ⊕ ¬A)	0101-1010
¬(¬B ⊕ ¬A)	1100-0011	¬(¬C ⊕ ¬B)	1001-1001	¬(¬C ⊕ ¬A)	1010-0101
¬(B ↔ A)	0011-1100	¬(C ↔ B)	0110-0110	¬(C ↔ A)	0101-1010
¬(¬B ↔ A)	1100-0011	¬(¬C ↔ B)	1001-1001	¬(¬C ↔ A)	1010-0101
¬(B ↔ ¬A)	1100-0011	¬(C ↔ ¬B)	1001-1001	¬(C ↔ ¬A)	1010-0101
¬(¬B ↔ ¬A)	0011-1100	¬(¬C ↔ ¬B)	0110-0110	¬(¬C ↔ ¬A)	0101-1010

Table 7.5: A-A, A-1, A-0 and 1-0 Relational Expression-number Lookup Table

(*This dataset can be downloaded via https://doi.org/10.7910/DVN/OCN14M*)

A-A					A-1				
Expression	Number				Expression	Number			
(A ∧ A)		10	1100	1111-0000	(A ∧ 1)		10	1100	1111-0000
(A ∧ ¬A)	0	00	0000	0000-0000	(1 ∧ A)		10	1100	1111-0000
(¬A ∧ A)	0	00	0000	0000-0000	(¬A ∧ 1)		01	0011	0000-1111
(¬A ∧ ¬A)		01	0011	0000-1111	(1 ∧ ¬A)		01	0011	0000-1111
(A ∨ A)		10	1100	1111-0000	(A ∨ 1)	1	11	1111	1111-1111
(A ∨ ¬A)	1	11	1111	1111-1111	(1 ∨ A)	1	11	1111	1111-1111

$(\neg A \vee A)$	1	11	1111	1111-1111	$(\neg A \vee 1)$	1	11	1111	1111-1111
$(\neg A \vee \neg A)$		01	0011	0000-1111	$(1 \vee \neg A)$	1	11	1111	1111-1111
$(A \rightarrow A)$	1	11	1111	1111-1111	$(A \rightarrow 1)$	1	11	1111	1111-1111
$(A \rightarrow \neg A)$		01	0011	0000-1111	$(1 \rightarrow A)$		10	1100	1111-0000
$(\neg A \rightarrow A)$		10	1100	1111-0000	$(\neg A \rightarrow 1)$	1	11	1111	1111-1111
$(\neg A \rightarrow \neg A)$	1	11	1111	1111-1111	$(1 \rightarrow \neg A)$		01	0011	0000-1111
$(A \oplus A)$	0	00	0000	0000-0000	$(A \oplus 1)$		01	0011	0000-1111
$(A \oplus \neg A)$	1	11	1111	1111-1111	$(1 \oplus A)$		01	0011	0000-1111
$(\neg A \oplus A)$	1	11	1111	1111-1111	$(\neg A \oplus 1)$		10	1100	1111-0000
$(\neg A \oplus \neg A)$	0	00	0000	0000-0000	$(1 \oplus \neg A)$		10	1100	1111-0000
$(A \leftrightarrow A)$	1	11	1111	1111-1111	$(A \leftrightarrow 1)$		10	1100	1111-0000
$(A \leftrightarrow \neg A)$	0	00	0000	0000-0000	$(1 \leftrightarrow A)$		10	1100	1111-0000
$(\neg A \leftrightarrow A)$	0	00	0000	0000-0000	$(\neg A \leftrightarrow 1)$		01	0011	0000-1111
$(\neg A \leftrightarrow \neg A)$	1	11	1111	1111-1111	$(1 \leftrightarrow \neg A)$		01	0011	0000-1111

A-0					1-0				
Expression	Number				Expression	Number			
$(A \wedge 0)$	0	00	0000	0000-0000	$(1 \wedge 1)$	1	11	1111	1111-1111
$(0 \wedge A)$	0	00	0000	0000-0000	$(1 \wedge 0)$	0	00	0000	0000-0000
$(\neg A \wedge 0)$	0	00	0000	0000-0000	$(0 \wedge 1)$	0	00	0000	0000-0000
$(0 \wedge \neg A)$	0	00	0000	0000-0000	$(0 \wedge 0)$	0	00	0000	0000-0000
$(A \vee 0)$		10	1100	1111-0000	$(1 \vee 1)$	1	11	1111	1111-1111
$(0 \vee A)$		10	1100	1111-0000	$(1 \vee 0)$	1	11	1111	1111-1111
$(\neg A \vee 0)$		01	0011	0000-1111	$(0 \vee 1)$	1	11	1111	1111-1111
$(0 \vee \neg A)$		01	0011	0000-1111	$(0 \vee 0)$	0	00	0000	0000-0000
$(A \rightarrow 0)$		01	0011	0000-1111	$(1 \rightarrow 1)$	1	11	1111	1111-1111
$(0 \rightarrow A)$	1	11	1111	1111-1111	$(1 \rightarrow 0)$	0	00	0000	0000-0000
$(\neg A \rightarrow 0)$		10	1100	1111-0000	$(0 \rightarrow 1)$	1	11	1111	1111-1111
$(0 \rightarrow \neg A)$	1	11	1111	1111-1111	$(0 \rightarrow 0)$	1	11	1111	1111-1111
$(A \oplus 0)$		10	1100	1111-0000	$(1 \oplus 1)$	0	00	0000	0000-0000
$(0 \oplus A)$		10	1100	1111-0000	$(1 \oplus 0)$	1	11	1111	1111-1111
$(\neg A \oplus 0)$		01	0011	0000-1111	$(0 \oplus 1)$	1	11	1111	1111-1111
$(0 \oplus \neg A)$		01	0011	0000-1111	$(0 \oplus 0)$	0	00	0000	0000-0000
$(A \leftrightarrow 0)$		01	0011	0000-1111	$(1 \leftrightarrow 1)$	1	11	1111	1111-1111
$(0 \leftrightarrow A)$		01	0011	0000-1111	$(1 \leftrightarrow 0)$	0	00	0000	0000-0000
$(\neg A \leftrightarrow 0)$		10	1100	1111-0000	$(0 \leftrightarrow 1)$	0	00	0000	0000-0000
$(0 \leftrightarrow \neg A)$		10	1100	1111-0000	$(0 \leftrightarrow 0)$	1	11	1111	1111-1111

8 Conclusion

This paper presents a novel approach: using a digital calculation method for propositional logic. The innovation of this method comes from an idea: every propositional expression can be evaluated to a truth value and every truth value can be represented by a digital number. By calculating and comparing these numbers, we can find and prove propositional equivalences and other propositional expressions without using conventional methods.

The proposed method demonstrates notable improvements over conventional approaches in single reasonings. It transforms propositional reasoning from the statement-based operation or the table-based operation to the number-based operation, changing the mechanism and form of the operation. The new method requires only a few primitive numbers and the calculation formulas, thereby eliminating the need for using truth tables, and obviates the need for deriving sentences, citing theorems and substituting symbols. The process is simplified to binary calculation, substantially reducing operational complexity.

Furthermore, the method exhibits more efficiency in multiple reasonings in long-term use. Through the implementation of expression-number data in lookup tables, it facilitates data reusability and shareability, thereby eliminating repetitive operations in long-term applications.

Additionally, the method is suitable for manual calculation, large-scale computation, AI and automated reasoning. It provides an efficient mechanism for quickly identifying a large number of valid formulas while offering an alternative approach for expression simplification.

It is important to note that this method offers an enhanced alternative to existing approaches, rather than addressing the currently unresolved challenges in the field. Therefore, the scope of its application remains consistent with existing methods: tasks solvable through conventional methods can be accomplished more efficiently by the new method, while the existing challenges related to propositional logic and Boolean logic, that remain unresolved by conventional methods, should not be considered within the scope of this method's objectives. Nevertheless, the novel approach presented here may provide valuable insights for future research directions.

Data Availability Statement

The author confirms that all data generated or analysed during this study are included in this article. Furthermore, all sources and data supporting the findings of this study were all publicly available at the time of submission.

References

[1] Nahin, Paul J. (2012). *The Logician and the Engineer: How George Boole and Claude Shannon Created the Information Age*. Princeton University Press.

[2] Mendelsohn, Richard L. (2005). *The Philosophy of Gottlob Frege*. Cambridge University Press. p. 185-201.

[3] Bruinier, Jan H. (2008). *Hilbert modular forms and their applications*, Retrieved from: *https://arxiv.org/pdf/math/0609763*

[4] Jan von Plato. (2014). *Elements of Logical Reasoning*, Cambridge University Press. p. 23.

[5] Anellis, Irving H. (2012). Peirce's Truth-Functional Analysis and the Origin of the Truth Table. *History and Philosophy of Logic*. 33 (1): 87–97.

[6] Wittgenstein, Ludwig. (1922) *Tractatus Logico-Philosophicus*. First English-language edition by Routledge & Kegan Paul LTD, 1922. p. 98-99.

[7] Rosser J. Barkley (1953). *Logic for Mathematicians*. McGraw-Hill Press. p. 26.

[8] Edward N. Zalta (1998). Frege's Theorem and Foundations for Arithmetic. p. 5. *Stanford Encyclopedia of Philosophy*. Retrieved from: *https://plato.stanford.edu/ENTRIES/frege-theorem/*

[9] Hart, Anthony (2011). Derive the Lukasiewicz axioms from the Russell-Bernays Axioms. *Theorem List - Metamath Proof Explorer*. p. 17. huihoo.com: *https://docs.huihoo.com/proof/metamath.org/mpegif/mmtheorems17.html*

[10] Łukasiewicz, Jan (1970). *Jan Lukasiewicz: Selected Works*. North-Holland. p. 136.

[11] Karnaugh, Maurice (1953). The Map Method for Synthesis of Combinational Logic Circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*. 72 (5): 593–599.

[12] Svoboda, Antonín; White, Donnamaie E. (2016). 9.9. The Petrick function solution for the minimal ΣΠ-form of y. *Advanced Logical Circuit Design Techniques*.

[13] Coudert, Olivier (October 1994). Two-level logic minimization: an overview *Integration, the VLSI Journal*. 17–2 (2): 97–140.