Efficient Algorithm and Implementation for Boole Reduction of Large Logic Expressions

Kailin Xuan

The Bishop's School, San Diego, CA 92037
kailinx20@gmail.com

Abstract. Boolean algebra, introduced in the 19th century by the mathematician, philosopher, and logician George Boole, has been foundational in mathematics, logic, computer science, machine proofs, and digital electronics. Recently, a new framework, the "Algebra of Boole", proposed by Professor Norman J. Wildberger, offers a more natural and powerful alternative. To apply this new framework to important areas, such as artificial intelligence, complex machine proofs, and large-scale circuit designs, it is essential to develop a method for automatically reducing complicated logic expressions into a simple standard form. This paper is on developing an efficient algorithm and implementation for Boole Reduction, a process that simplifies large logic and Boole expressions into compact, canonical forms. By utilizing Key Boole Properties from the Algebra of Boole, the algorithm processes arbitrarily complex logic expressions, reducing them iteratively to a canonical form through sophisticated reduction techniques until no further simplifications are possible. Expressions with varying complexity, from simple binary operations to large multi-variable expressions, are used to test the algorithm. Results show a significant improvement in reduction speed and computational efficiency compared to brute-force methods, demonstrating the algorithm's feasibility in handling complex logic expressions. One of the aims of this work is also to stimulate research in the framework of the Algebra of Boole as well as its applications in areas such as propositional logic, AI reasoning, and the optimization of digital circuit designs.

Keywords: Algebra of Boole, Artificial Intelligence, Logic Reduction Algorithm

1 Introduction

The primary objective of this research is to develop a fast, efficient, and scalable algorithm for Boole reduction. By focusing on simplifying expressions using Key Boole Properties, the proposed algorithm is designed to handle arbitrarily large logic or Boole expressions, simplifying them into their canonical form. Central to this approach is the Algebra of Boole, which serves as a structured framework for logical reasoning, artificial intelligence, and circuit analysis. Unlike traditional Boolean algebra, this redefinition involves XOR-based addition, leading to more algebraic calculations and a more powerful treatment of logical relationships,

particularly in applications such as AI reasoning and machine proof. This research contributes to advancing computational logic within the framework of the Algebra of Boole.

1.1 Boolean Algebra & Algebra of Boole

Boolean algebra, or logical algebra, is a field of mathematics introduced and developed by George Boole in the 19th century. Boole took the logic that had been established over the past 2000 years, building on Aristotle's work, and created a mathematical framework to model it [2]. Professor Norman Wildberger, in his series of YouTube public lectures, proposed a new Algebra of Boole system that behaves differently from classical Boolean algebra [12]. In the Algebra of Boole, addition is defined as XOR, where only one element can be true at a time for the statement to be true.

In this paper, we use term "logic expression" to refer to expressions written with logic symbols such as &(AND), |(OR), →(Implication). A "`Boolean expression`" denotes the expression written with Boolean algebra where the addition operator "+" represents logic OR (sometimes expressed as $\oplus$ to avoid confusion with the plus sign in Algebra of Boole). A "`Boole expression`" refers to an expression with the Algebra of Boole where the addition "+" means logic XOR. All these kinds of expressions try to model any Boolean function $f : B_2^n \to B_2$, where $B_2 = \{0, 1\}$ is a set of binary values, and n represents the dimension, or arity, of the function.

Table 1 displays the operations for two variables in the Algebra of Boole, compared to Boolean algebra. In the table, the top row contains the Algebra of Boole representation, while the bottom row is the Boolean algebra representation. The upper part and the lower part together present a complete table, the lower part being the horizontal continuation of the upper part due to space constraints.

For example, the logic OR expression A|B would be expressed as A+B (in Boolean algebra, Boolean expression) and A+B+AB (Algebra of Boole, Boole expression). Variables are written either in capital or lower case.

Professor Norman J. Wildberger, in his 31-part YouTube lecture series, demonstrates the better mathematical and algebraic properties of the Algebra of Boole than Boolean algebra and various techniques to simplify expressions using the Algebra of Boole. Drawing from a close reading of George Boole's original writings, he argues that the Algebra of Boole is more natural and powerful than Boolean algebra, and Boole's intended system was more aligned with this algebraic framework, rather than what we now call Boolean algebra. In particular, Wildberger proved that every Boole expression can be reduced to a canonical form. For instance, the canonical form for three variables is _ + _a + _b + _c + _ab + _ac + _bc + _abc, where each underscore represents a coefficient (either 0 or 1). This structure generalizes to any number of variables. He then called for a computer program to perform the reduction of any large logic expressions to canonical form by using Algebra of Boole, which is vital for applications in propositional logic, AI reasoning and circuit designs.

Table 1: Comparison of Algebra of Boole and Boolean Algebra

Algebra of Boole	A	B	A+B	AB	A+AB	B+AB	A+B+AB
	0	0	0	0	0	0	0
	0	1	1	0	0	1	1
	1	0	1	0	1	0	1
	1	1	0	1	0	0	1
Logic Connector			XOR	&(AND)			| (OR)
Boolean Algebra	A	B	$A\overline{B}+\overline{A}B$	AB	$A\overline{B}$	$\overline{A}B$	A+B

Algebra of Boole	1+A	1+B	1+A+B	1+AB	1+A+AB	1+B+AB	1+A+B+AB
	1	1	1	1	1	1	1
	1	0	0	1	1	0	0
	0	1	0	1	0	1	0
	0	0	1	0	1	1	0
Logic Connector	~				→	→	
Boolean Algebra	$\overline{A}$	$\overline{B}$	$AB+\overline{A}\overline{B}$	$\overline{A}+\overline{B}$	$\overline{A}+B$	$A+\overline{B}$	$\overline{A}\overline{B}$

1.2 Circuit Analysis, Computer Science, and AI Applications.

There has been extensive research, program implementation, and application areas using Boolean algebra. Here, we mention a few potential application areas of the Algebra of Boole and Boole Reduction.

In circuit analysis, the goal is often to determine the output based on the given inputs and gate configurations or to simplify the logic for efficient design and implementation [3, 7]. Logic gates, which are devices that lay the basis for digital circuits, implement Boolean functions [11]. There are seven basic logic gates: AND, OR, XOR, NOT, NAND, NOR, and XNOR. Each Boolean operation corresponds to a physical logic gate in a circuit. Simplifying Boole expressions means fewer gates, which saves power, space, and cost in circuit design.

Computer Science deals with computer hardware, computer programming, and efficiency [1, 6]. Boolean algebra has played a fundamental role in this field. The new framework, Algebra of Boole, and logic reduction could find applications in hardware and software verifications, machine proofs, and Boolean satisfiability (SAT) problems [4]. In AI-driven reasoning systems and applications, many inferences are made based on hypotheses and knowledge bases [5, 8, 10]. The ability to reduce complex logic expressions allows AI systems to make faster and more efficient inferences and reason through the possibilities more quickly. AI models also use logic in decision trees and logic-based AI, as it allows for the navigation of complex scenarios. As AI continues to evolve and tackle increasingly

complex problems, the role of Boole Reduction in optimizing logic-based systems will continue to grow in importance, facilitating more effective AI solutions.

2 Reduction and Algorithm in the Algebra of Boole

Though the Algebra of Boole is a better framework than Boolean algebra, an efficient algorithm and implementation for Boole Reduction are necessary to apply the Algebra of Boole in practical application areas. In the first such implementation [9], the logic expression is converted to the Boole expression first. The program then employs expand() function from SymPy, a symbolic mathematics library, to transform the expression to be a sum of monomials(terms) and perform reduction for each monomial. This approach is computationally intensive and is thus referred to as the "Brute-force" method.

While the Brute-force method is useful in Boole Reduction for logic expression reduction, it can face problems when there are many inference steps in the area such as long AI reasoning and machine proof. For example, with the following type of logic expression:

$$(a \to b)\&(b \to c)(c \to d)(d \to e)(e \to f)\cdots$$

Its corresponding Boole expression would be:

$$(1 + a + ab)(1 + b + bc)(1 + c + cd)(1 + d + de)(1 + e + ef)\cdots$$

Let size n indicate the number of implication ($\to$) symbols or number of steps of inference. The number of terms generated after expansion grows at a rate close to 3^n. In other words, when $n = 18$, there would be around 3^{18}, or 387420489, expanded monomials before reduction. The computation time increases exponentially as $O(3^n)$. Such exponential factor causes the computation time to process large logic expressions to grow rapidly as n increases.

To overcome this rapid increase in computational time, a New Algorithm (**BRP**) is proposed in this paper and implemented in Python. The core idea is to partition the Boole expression into groups of sub-expressions. For example, the Boole expression $(1+a+ab)(1+b+bc)(1+c+cd)(1+d+de)(1+e+ef)(1+f+fg)$ could be partitioned into two groups $(1 + a + ab)(1 + b + bc)(1 + c + cd)$ and $(1+d+de)(1+e+ef)(1+f+fg)$, which are then reduced incrementally. In this way, the number of expanded monomials can usually be kept in a relatively small size. The following outlines the Key Boole Properties used in the algorithm:

Key Boole Properties:

1. Idempotence (Any Boolean variable multiplied by itself is equal to the variable, removing redundancy): $A^2 = A$
2. Complementarity: $A(1 + A) = 0$ or $A + A = 0$

By using Key Boole Properties, any Boole expressions can be reduced to the canonical form. For example, Modus Ponens, which states $(p\&(p \to q)) \to q$, the

correspondent Boole expression and reduction is as follows:

$$\begin{aligned}
&(1+(p((1+p+pq)))+(p((1+p+pq)))q \quad (\texttt{Boole conversion})\\
=&1+p+pp+ppq+(p+pp+ppq)q \quad (\texttt{Algera})\\
=&1+p+p+pq+pq+ppq+ppqq \quad (\texttt{Using Idempotence property})\\
=&1+p+p+pq+pq+pq+pq \quad (\texttt{Using Complementarity})\\
=&1
\end{aligned}$$

Algorithm 1 shows the Boole Reduction with Partition (BRP), the new algorithm for reducing Logic expressions.

Algorithm 1: Boole Reduction with Partition (BRP)

Input: Logic expression E, minimum partition size n_{Minimum}
Output: Canonical Boole expression E_c

1 Parse the logic expression E and convert it to a Boole expression E_b using Table 1 (Comparison of Algebra of Boole and Boolean Algebra).
2 Partition E_b into groups of sub-expressions $E_1, E_2, \ldots, E_n$ such that each E_i has a string length $\geq n_{\text{Minimum}}$ for $i \in [1, n-1]$, ensuring balanced parentheses.
3 Initialize $E_c \leftarrow \emptyset$.
4 **for** $i \leftarrow 1$ **to** n **do**
5 Append E_i to E_c, expanding E_c to a sum of monomials.
6 Apply Boole reduction on the monomials in E_c using Key Boole Properties.
7 Reorder the monomials in E_c to obtain the canonical form.

Remark 1: The algorithm could easily be adapted to perform parallel and recursive reduction of neighboring groups after partition of Boole expressions, potentially bringing more performance gains, especially for very large expressions in AI, SAT solving, and symbolic computation.

Remark 2: All propositional formulas can be converted to Conjunctive Normal Form (CNF), which is a standard format for logic simplification, SAT solving, and digital design. Using De Morgan's laws, implications and negations can be transformed into conjunctions and disjunctions. For instance, $(A \to B)$ becomes $(\sim A|B)$. These rules allow even complex formulas to be rewritten as a conjunction (&) of disjunctions (|). For satisfiability problems, any formula can be further transformed into 3-CNF (clauses with exactly three literals) by introducing auxiliary variables through techniques such as Tseitin Encoding.

3 Results and Discussion

The Boole reduction algorithm, with the computer program, successfully simplifies large and complex input logic expressions to their minimal forms by applying the New Algorithm. The following example demonstrates how the program processes and reduces Boole expressions:

One of the most important logical principles in mathematical proof and scientific reasoning is Modus Ponens, which states $(p\&(p \to q)) \to q$. Using the program, this expression simplifies to 1, confirming that the logical inference holds. The following is the Python program's prompt, input, and output.

- Entered logic expression: $(p\&(p \to q)) \to q$
- Boole expression:$(1 + (p((1 + p + pq))) + (p((1 + p + pq)))q)$
- Reduced Boole expression: 1

Logical inference is also a critical part of AI reasoning and knowledge representation. Consider Lewis Carroll's logical puzzle [12]:

i: `it is interesting`	m: `it is modern`	y: `it is your poem`
p: `it is popular`	a: `it is affected`	s: `it is about bubbles`

Given the hypotheses:

$$i \to p, m \to a, y \to s, a \to\sim p, \sim m \to\sim s,$$

can one conclude that "your poem is not interesting" $(y \to\sim i)$? After this inference is inputted into the program, the output is 1, which affirms that "your poem is not interesting," confirming the validity of the conclusion.

- Entered logic expression: $((i \to p)\&(m \to a)\&(y \to s)\&(a \to (\sim p))\&((\sim m) \to (\sim s))) \to (y \to (\sim i))$
- Boole expression: $(1+(((1+i+ip))((1+m+ma))((1+y+ys))((1+a+a((1+p))))((1+((1+m))+((1+m))((1+s)))))+(((1+i+ip))((1+m+ma))((1+y+ys))((1+a+a((1+p))))((1+((1+m))+((1+m))((1+s)))))((1+y+y((1+i)))))$
- Reduced Boole expression: 1

In real-world applications, knowledge bases can consist of hundreds of logical hypotheses,making inference an exceedingly complex task beyond human capability. Traditional methods of reasoning struggle with this scale, but by utilizing the Algebra of Boole and Boole Reduction program, logical inference is transformed into a purely algebraic and computational process. AI systems can perform logic reduction to provide the inference answer within seconds, highlighting how it is a powerful tool for applications in automated reasoning, expert systems, and complex decision-making processes.

Calculation time is incorporated in a complex test case, which contains a large number of terms and reductions throughout the process. Table 2 shows the Brute-Force and new algorithm **BRP** calculation times for Boole reduction, where the size n indicates the number of implication $(\to)$ symbols or number of steps of inference; therefore, the entered expression is $(a \to b)$ when $n = 1$. It follows that when $n = 6$, the logic expression is $(a \to b)\&(b \to c)\&(c \to d)\&(d \to e)\&(e \to f)\&(f \to g)$. This case is particularly important for machine proofs since mathematical proofs often contain hundreds of steps, many of which involve implications. All experiments were conducted on a personal MacBook Pro without GPU acceleration.

Table 2: Comparison of Brute-Force and Optimized Algorithm Calculation Times for Boolean Reduction as Size n Increases

Size n	**Brute-force Time (sec)**	**New Algorithm Time (sec)**
6	0.11	0.068
7	0.25	0.24
8	0.62	0.64
9	1.72	1.11
10	4.81	1.45
11	13.86	2.37
12	39.68	5.10
13	108.74	13.12
18	1803.64	114.19

Table 3: Comparison of Brute-Force and Optimized Algorithm Calculation Times for Boolean Reduction as Number of Clauses Increases

Number of Clauses	**Expression Used**	**Brute-force Time (sec)**	**New Algorithm Time (sec)**
6	(~a) & (b | (~c)) & (d | e | (~f)) & ((~g) | h | i) & (j | (~b)) & (c | d | (~e))	1.00	0.57
7	((~a) | b | c) & (d | (~e)) & ((~f) | g) & (h | i | (~j)) & ((~b) | a) & (c | d | (~f)) & ((~g) | h)	7.56	1.26
8	(a | (~b)) & (c | d | (~e)) & ((~f) | g | h) & (i | j) & ((~b) | a | (~c)) & ((~d) | e) & (f | g | (~h)) & ((~i) | j)	16.64	1.47
9	((~a) | b | c) & ((~d) | e) & (f | g | (~h)) & ((~i) | j) & (a | (~b)) & ((~c) | d | e) & (f | (~g)) & ((~h) | i) & (j | a | (~b))	100.89	2.29
10	(~a) & (b | (~c)) & (d | e | (~f)) & ((~g) | h | i | j) & (a | (~b)) & (c | d | (~e)) & ((~h) | i) & (f | (~g) | (~h)) & (j | (~a) | b | (~c)) & ((~i) | (~j))	208.68	6.30

The new algorithm **BRP** shows a more controlled and gradual increase in computation time. In many cases, the number of intermediate terms in the reduction iteration is much smaller compared to the expansion terms, which leads to the speedup being significant.

To further test its performance, we use a second case involving horizontal clause expansion, with expressions in conjunctive normal form (CNF) randomly generated by large language models. Each expression is a conjunction of clauses built from variables “a” to “j” using various logical operators. This structure reflects typical Boolean conditions in software, SAT solving, and circuit verification. As shown in Table 3, unlike the brute-force method, **BRP** scales efficiently as the number of clauses increases from 6 to 10.

4 Conclusion and Future Research

The fast algorithm for Boole reduction in the framework of the Algebra of Boole has been successfully developed and implemented in Python. The program simplifies complex logic expressions to canonical forms, demonstrating its effectiveness. Specifically, the New Algorithm showed magnitudes of speedups in Boole reduction than the Brute-force approach. Potential applications include propositional logic, AI reasoning and integrated circuit designs.

Based on the success of this research, we hope to stimulate more research and algorithm innovation in the Algebra of Boole framework. Future developments could incorporate Boole Reduction into machine learning models, which could augment AI systems' reasoning capabilities and automate logic analysis.

Acknowledgements

I would like to thank my mentor, Dr. Lenore Blum, for valuable advice and discussions during the research process. I also appreciate Yueqing Luo for sharing his method of implementation and Dr. Marcus Jaiclin for encouraging me.

References

1. Blum, L., Cucker, F., Shub, M., & Smale, S. Complexity and Real Computation. New York, NY: Springer-Verlag (1998).
2. Boole, G. The mathematical analysis of logic. Macmillan (1847).
3. Brayton, R. K., Hachtel, G. D., McMullen, C., & Sangiovanni-Vincentelli, A. Logic minimization algorithms for VLSI synthesis (Vol. 2). Springer Science & Business Media (1984).
4. Cook, S. A. The complexity of theorem-proving procedures. Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC), 151-158 (1971).
5. Darwiche, A. Three Modern Roles for Logic in AI. In Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'20), Association for Computing Machinery (2020).
6. Knuth, D. E. Chapter 7.2.2.2: Satisfiability. In The art of computer programming (Vol. 4B, Combinatorial algorithms, Part 2). Addison-Wesley Professional. pp. 185–369 (2022).
7. Mano, M. M., & Kime, C. R. Logic and computer design fundamentals (4th new international ed.). Pearson Education Limited. pp. 54 (2014).
8. McCarthy, J. Programs with common sense. In Proceedings of the Teddington Conference on the Mechanization of Thought Processes (1959).
9. Luo, Y. Algebra of Boole. (2024). https://github.com/shanyoumu461/Algebra-of-Boole.
10. Russell, S. and Norvig, P. Artificial Intelligence: A Modern Approach. 3rd Edition, Prentice-Hall, Upper Saddle River (2010)
11. Shannon, C. E. A Symbolic Analysis of Relay and Switching Circuits. Transactions of the American Institute of Electrical Engineers, 57, 713-723 (1938).
12. Wildberger, N. J. Boole reduction: A challenge for programmers | MathFoundations 269 [Video]. (2019). YouTube. https://youtu.be/AJr44JgbKho?feature=shared