Hindawi
Security and Communication Networks
Volume 2022, Article ID 7307139, 14 pages
https://doi.org/10.1155/2022/7307139

Research Article

An In-Place Simplification on Mixed Boolean-Arithmetic Expressions

Binbin Liu,[1] **Qilong Zheng**,[1] **Jing Li**,[1] **and Dongpeng Xu**[2]

[1]*School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China*
[2]*College of Engineering and Physical Science, University of New Hampshire, Durham 03824, USA*

Correspondence should be addressed to Binbin Liu; robbertl@mail.ustc.edu.cn

Received 6 June 2022; Revised 21 July 2022; Accepted 30 July 2022; Published 14 September 2022

Academic Editor: Vincenzo Conti

Mixed Boolean-arithmetic (MBA) expression, which involves both bitwise operations (e.g., NOT, AND, and OR) and arithmetic operations (e.g., +, −, and ∗), is a software obfuscation scheme. On the other side, multiple methods have been proposed to simplify MBA expressions. Among them, table-based solutions are the most powerful simplification research. However, a fundamental limitation of the table-based solutions is that the space complexity of the transformation table drastically explodes with the number of variables in the MBA expression. In this study, we propose a novel method to simplify MBA expressions without any precomputed requirements. First, a bitwise expression can be transformed into a unified form, and we provide a mathematical proof to guarantee the correctness of this transformation. Then, the arithmetic reduction is smoothly performed to further simplify the expression and produce a concise result. We implement the proposed scheme as an open-source tool, named MBA-Flatten, and evaluate it on two comprehensive benchmarks. The evaluation results show that MBA-Flatten is a general and effective MBA simplification method. Furthermore, MBA-Flatten can assist malware analysis and boost SMT solvers' performance on solving MBA equations.

1. Introduction

Mixed Boolean-arithmetic (MBA) expression [1, 2] is defined as the expression that mixes the usage of bitwise operations (e.g., ∼, &, |, and ∧) and arithmetic operations (e.g., +, −, and ∗). Several formal methods [1, 3] are designed to generate a new complex MBA expression that is equal to a simple expression. MBA expression, which can be used to replace a simple expression with an equivalent representation that is hard to understand, is an advanced software obfuscation scheme [3–5]. The MBA obfuscation has been adopted by many academic projects and industrial products to protect software [5–9].

The wide practical applications of MBA obfuscation have attracted research on simplifying MBA expression. Recent studies [10, 11] demonstrate that existing computer algebra software has a very limited effect on MBA simplification. Consequently, multiple methods are proposed to simplify MBA expressions, including bit blasting [12], pattern matching [13], program synthesis [14–16], deep learning [17, 18], and table-based solutions [5, 11]. Among them, table-based solutions are the state-of-the-art MBA simplification method. However, one strong limitation is that the complexity of creating and storing the precomputed table is $O(2^{2^t})$, where t is the number of variables in the MBA expression. Thus, it has an overwhelming overhead to produce and store the tables for any $t \geq 5$.

In this study, we propose a novel scheme to simplify an MBA expression without any precomputed requirements. The key idea is that a transformation procedure can be used to reduce a bitwise expression to a unified form, and a mathematical proof is provided to guarantee the correctness of the transformation. Then, the arithmetic reduction is smoothly performed to further simplify the expression and generate the final result. We implement the approach as an open-source tool, named MBA-Flatten. To demonstrate the capability of MBA-Flatten, we evaluate it on two comprehensive MBA benchmarks. The evaluation results show that

MBA-Flatten outperforms existing tools in terms of more solved MBA expressions. Due to the low-cost arithmetic computation, MBA-Flatten is also an effective MBA simplification tool. In addition, the evaluation demonstrates that MBA-Flatten can assist malware analysis and boost SMT solving on MBA equations.

In summary, this study makes the following key contributions:

(1) We find that a bitwise expression can be transformed into a unified form and provide a mathematical proof to support it. To the best of our knowledge, we are the first to prove the existence of the transformation.

(2) The bitwise expression transformation paves the way for our novel in-place MBA simplification method. Our proposed scheme first replaces the bitwise expressions with the corresponding equivalent form. In this way, arithmetic reduction rules can be seamlessly applied to further produce the simplification result.

(3) We have implemented our idea as an open-source tool, called MBA-Flatten, and evaluated it on two comprehensive MBA benchmarks. The evaluation results demonstrate that MBA-Flatten is a general and effective MBA simplification method.

The remainder of this study is structured as follows. Section 2 shows the background of MBA expression. Section 3 illustrates the proposed scheme that can be used to simplify an MBA expression. The proof of Theorem 1 can be found in Section 4. In Section 5, we describe the experimental evaluation of the proposed approach. Section 6 discusses some limitations of our proposed scheme, and Section 7 concludes this study.

2. Related Work

In this section, we first introduce the background of MBA expression and its wide applications. Then, we discuss the existing research on simplifying MBA expressions, pointing out the limitations, which also serve as a motivation in this study.

2.1. MBA Expression. Zhou et al. [1, 2] propose the concept of mixed Boolean-arithmetic (MBA) expression based on Boolean-arithmetic algebra, which mixes the usage of bitwise operators (e.g., NOT, AND, and OR) and arithmetic operations (e.g., +, −, and $*$). MBA expression is specified as linear MBA, polynomial MBA, and non-polynomial MBA [1, 11]. The formal definitions of linear and polynomial MBA expression are denoted as follows, and the linear MBA expression is a subset of polynomial MBA expression [1]. The MBA expression, which fails to satisfy Definition 1, is considered as a non-polynomial MBA expression [11].

Definition 1. (Zhou [1]). *A polynomial MBA expression is of the form*:

$$E_p(x_1, \cdots, x_t) = \sum_{i \in I} a_i * \left(\prod_{j \in J_i} e_{i,j}(x_1, \cdots, x_t) \right), \quad (1)$$

where a_i*is integer constant,*$e_{i,j}$*is bitwise expression of variables*$x_1, \ldots, x_t$*over*B^n, $B = \{0, 1\}$, n, t*are positive integers, and*$I, J_i \subset Z$, $\forall i \in I$.

Definition 2. (Zhou [1]). *A linear MBA expression is a polynomial MBA expression of the form*:

$$E_l(x_1, \cdots, x_t) = \sum_{i \in I} a_i * e_i(x_1, \cdots, x_t), \quad (2)$$

where a_i*is integer constant,*e_i*is bitwise expression of variables*$x_1, \cdots, x_t$*over*B^n, $B = \{0, 1\}$, n, t*are positive integers, and*$I \subset Z$.

Zhou et al. [1] design a generator using truth tables to produce infinite linear MBA equations. Based on existing linear MBA rules, Liu et al. [3] propose several formal methods to generate an unlimited number of polynomial and non-polynomial MBA expressions. Examples of MBA expressions are shown below. In particular, (3) is a linear MBA expression, (4) is a polynomial MBA expression, and (5) is a non-polynomial MBA expression.

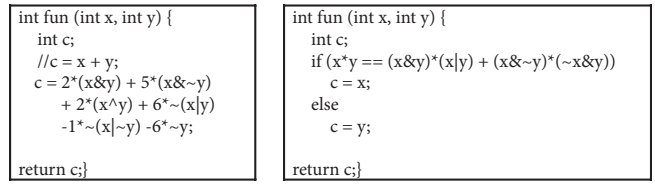
$$2 * (x \wedge y) + y - 3 * (x | y) + 5, \quad (3)$$

$$x * (x \wedge y) - x + 2 * (\sim x | y) * (x^y) - 1, \quad (4)$$

$$(x \wedge (x + y)) * y - 3 * (x | (x + y)) + \sim x + 7. \quad (5)$$

Due to its solid theoretical foundation and simplicity of implementation, MBA expression has been applied in multiple academic tools and industrial products to protect software [5–9]. For example, Cloakware, Irdeto, and Quarkslab apply MBA obfuscation in their commercial products [5, 7]. Tigress [6], an academic C source code obfuscator, encodes simple expressions into complex MBA forms. Blazy et al. [8] develop a C program obfuscator, in which formally verified MBA obfuscation rules are integrated. Ma et al. [9] apply MBA expressions to develop a novel dynamic software watermarking scheme. Figure 1 shows how to use MBA expressions to make software obfuscation [4]. Figure 1(a) demonstrates that the expression $x + y$ is substituted with a complex but equivalent expression. The opaque predicate [19] is shown in Figure 1(b), and the predicate $(x * y == (x \wedge y) * (x | y) + (x \wedge \sim y) * (\sim x \wedge y))$ is actually always true.

2.2. MBA Expression Simplification. The wide practical application of MBA obfuscation has encouraged research on simplifying MBA expressions. Eyrolles' PhD thesis [10] shows that popular symbol software (Maple, SageMath, Wolfram Mathematica, and Z3 [20]) fails to simplify MBA expressions. The root cause is that existing reduction rules cannot reduce expressions that mix the usage of bitwise and arithmetic operators [11]. Researchers have developed multiple solutions to simplify MBA expressions, including bit blasting [12], pattern matching [13], program synthesis



(a) (b)

Figure 1: Applications of MBA expression implemented in C programming language. (a) Data flow obfuscation. (b) Opaque predicate.

[14–16], and deep learning-based [17, 18]. While promising, these simplification methods are still in their infancy: they either suffer from high-performance penalties, or they produce many false simplification cases.

To effectively reduce MBA expression, researchers investigate the MBA mechanism and propose table-based solutions. Liu et al. [5] prove a two-way feature in the MBA transformation and design a two-variable transformation table to simplify MBA expression. Xu et al. [11] create multiple semantic-preserving transformation tables, which enumerate all bitwise expressions and the corresponding simplified forms. Using these transformation tables, MBA-Solver can effectively simplify an MBA expression.

So far, table-based solutions are the state-of-the-art MBA simplification methods. However, the space complexity of the transformation table is $O(2^{2^t})$ and t is the number of variables in the MBA expression. Therefore, table-based solutions are not scalable to reduce an MBA expression involving five or more variables. Here, (6) is a linear MBA expression with five variables, and table-based solutions fail to simplify it. Note that multiple methods are proposed to generate an unlimited number of MBA expressions [1, 3], and thus, an emerging challenge for MBA simplification is the MBA expression with five or more variables.

$$\begin{aligned} f(x,y,z,t,a) = &-((\sim(x \vee y \vee t)) \wedge (\sim a)) \\ &-((\sim(x|y|\sim t)) \wedge (\sim a)) - (\sim(x|y|z|t) \wedge a) \\ &-((\sim x \wedge \sim y \wedge (z \vee t)) \wedge (a)) \\ &-2 * y - 1 + (x \wedge y). \end{aligned} \tag{6}$$

3. The Proposed Scheme

To reduce MBA expressions, we first present an existing finding: a bitwise expression can be transformed into a unified form. This finding paves the way for our novel in-place MBA simplification scheme, MBA-Flatten.

3.1. Bitwise Expression Transformation. A bitwise expression is denoted as $e(x_1, \cdots, x_t)$ of variables $x_k \in B^n$, $k = 1, \cdots, t$. The transformation T is defined as follows:

$$\begin{aligned} &\sim x_i \mapsto -x_i + 1, \\ &x_i \wedge x_j \mapsto x_i * x_j, \\ &x_i | x_j \mapsto x_i + x_j - (x_i * x_j), \\ &x_i \wedge x_j \mapsto x_i + x_j - 2 * (x_i * x_j), \\ &x_i * \cdots * x_i \mapsto x_i, \end{aligned} \tag{7}$$

where $x_i, x_j \in B^n$. Equation (7) can be recursively applied to transform a bitwise expression $e(x_1, \cdots, x_t)$ into an arithmetic expression denoted as $T(e)$, which is shown as follows:

$$\begin{aligned} T(e) = &\sum_{i_1=1}^{t} a_{i_1} x_{i_1} + \sum_{1 \le i_1 < i_2 \le t} a_{i_1 i_2} (x_{i_1} * x_{i_2}) \\ &+ \cdots + a_{1 \cdots t} (x_1 * \cdots * x_t) + a_e, \end{aligned} \tag{8}$$

where $a_{i_1 \cdots i_k}$ and a_e are integers determined by e. After replacing all $(x_{i_j} * \cdots * x_{i_k})$ in $T(e)$ with $(x_{i_j} \wedge \cdots \wedge x_{i_k})$, (8) will be reduced as follows:

$$\begin{aligned} R(e) = &\sum_{i_1=1}^{t} a_{i_1} x_{i_1} + \sum_{1 \le i_1 < i_2 \le t} a_{i_1 i_2} (x_{i_1} \wedge x_{i_2}) \\ &+ \cdots + a_{1 \cdots t} (x_1 \wedge \cdots \wedge x_t) + a_e. \end{aligned} \tag{9}$$

An instance of the above transformation procedures is shown in Example 1. One interesting observation is that there is a gap between e and $R(e)$, because $\sim x_1 | x_2$ is equal to the expression $-x_1 + (x_1 \wedge x_2) - 1$ rather than $-x_1 + (x_1 \wedge x_2) + 1$.

Example 1. For a bitwise expression $e = \sim x_1 | x_2$, we have

$$\begin{aligned} T(e) &= (-x_1 + 1) + x_2 - (-x_1 + 1) * x_2 \\ &= -x_1 + x_1 * x_2 + 1, \\ R(e) &= -x_1 + (x_1 \wedge x_2) + 1. \end{aligned} \tag{10}$$

Moreover, Theorem 1 shows that the gap between a bitwise expression e and the corresponding $R(e)$ is actually a constant value, 0 or −2. In other words, a bitwise expression e can be successfully reduced to a unified form, Equation (10). Theorem 1 can be proved by induction on the number of bitwise operators in the bitwise expression $e(x_1, \cdots, x_t)$. For detailed proof of the theorem, refer to Section 4 of this study.

Theorem 1. *Let n, t be positive integers, $e(x_1, \cdots, x_t)$ be a bitwise expression of variables $x_k \in B^n$, $k = 1, \cdots, t$, and $F(e) = R(e) - 2 * a_e$ with the form of*

$$\begin{aligned} F(e) = &\sum_{i_1=1}^{t} a_{i_1} x_{i_1} + \sum_{1 \le i_1 < i_2 \le t} a_{i_1 i_2} (x_{i_1} \wedge x_{i_2}) \\ &+ \cdots + a_{1 \cdots t} (x_1 \wedge \cdots \wedge x_t) - a_e. \end{aligned} \tag{11}$$

Then, $F(e) \equiv e$ with $a_e = 0 or 1$.

By this theorem, Example 2 shows that a bitwise expression $\sim(x| \sim y)$ is reduced to $(y - (x \wedge y))$.

Example 2. For a bitwise expression $e = \sim (x| \sim y)$, we have

$$\begin{aligned} T(e) &= -(x + (-y + 1) - x * (-y + 1)) + 1, \\ &= y - x * y, \\ R(e) &= y - (x \wedge y), \\ F(e) &= y - (x \wedge y). \end{aligned} \tag{12}$$

The above procedures introduced so far are integrated into Algorithm 1. The algorithm takes a bitwise expression e as the input and outputs the transformation result (e). Algorithm 1 applies arithmetic computation to transform a bitwise expression, so it does not introduce extra memory cost to maintain the heap or precomputed tables.

3.2. Simplifying MBA Expression. As noted above, Algorithm 1 can transform a bitwise expression e into a unified form. Using Algorithm 1, we will discuss how to simplify linear, polynomial, and non-polynomial MBA expressions.

We first introduce how to simplify a linear MBA expression. According to Equation (2), a linear MBA expression is essentially a linear combination of bitwise expressions. Using Algorithm 1, the bitwise expressions in (2) are first substituted with the corresponding transformation result. After combining like terms, (2) will be reduced to the following simple form:

$$\begin{aligned} E_l(x_1, \cdots, x_t) &= \sum_{i=1}^{2^t} A_i * \tilde{e}_i(x_1, \cdots, x_t) \\ &= \sum_{i_1=1}^{t} A_{i_1} x_{i_1} + \sum_{1 \le i_1 < i_2 \le t} A_{i_1 i_2}\left(x_{i_1} \wedge x_{i_2}\right) \\ &\quad + \cdots + A_{1 \cdots t}\left(x_1 \wedge \cdots \wedge x_t\right) + A_{E_l}, \end{aligned} \tag{13}$$

where A_i is integer, $\tilde{e}_i(x_{\ 1}, \cdots, x_t) \in \{x_1, x_2, \cdots, x_1 \wedge x_2, x_2 \wedge x_3, \cdots, x_1 \wedge \cdots \wedge x_t, -1\}$. (13) indicates that a linear MBA expression can be simplified to the concise form including at most 2^t terms and t is the number of variables in the MBA expression. Example 3 shows that a complex linear MBA expression can be reduced to a simple result $(x + y)$.

Example 3. For the MBA expression in Figure 1(a), we have

$$\begin{aligned} &2 * (x \wedge y) + 5 * (x \wedge \sim y) + 2 * (x \wedge y) + 6 * \sim (x|y) - 1 * \\ &\quad \sim (x| \sim y) - 6 * \sim y \\ &= 2 * (x \wedge y) + 5 * (x - (x \wedge y)) + 2 * (x + y - 2 * (x \wedge y)) \\ &\quad + 6 * (-x - y + (x \wedge y) - 1) - 1 * (y - (x \wedge y)) \\ &\quad - 6 * (-y - 1) \\ &= x + y. \end{aligned} \tag{14}$$

Enlighten by the above simplification procedure, using Algorithm 1, (1) will be transformed to an equivalent form shown as follows:

$$E_p(x_1, \cdots, x_t) = \sum_{i \in I} A_i * \left(\prod_{j \in J_i} \tilde{e}_{i,j}(x_1, \cdots, x_t) \right), \tag{15}$$

where A_i are integers, $\tilde{e}_{i,j}(x_1, \cdots, \ x_t) \in \{x_1, x_2, \cdots, x_1 \wedge x_2, x_2 \wedge x_3, \cdots, x_1 \wedge \cdots \wedge x_t, -1\}$, and $I, J_i \subset Z$, $\forall i \in I$. The following example shows how to simplify a polynomial MBA expression. First, every bitwise expression is substituted with the equivalent form; e.g., $(x \wedge \sim y)$ is replaced with $(x - (x \wedge y))$. Then, arithmetic reduction rules are performed to produce the simplification result $(x * y)$. Note that the linear MBA expression is also polynomial, so the polynomial MBA simplification method can reduce a linear MBA expression.

Example 4. For the MBA expression in Figure 1(b), we have

$$\begin{aligned} &(x \wedge y) * (x|y) + (x \wedge \sim y) * (\sim x \wedge y) \\ &= (x - (x \wedge y)) * (y - (x \wedge y)) + (x \wedge y) * (x + y - (x \wedge y)) \\ &= x * y - x * (x \wedge y) - (x \wedge y) * y + (x \wedge y) * (x \wedge y) + (x \wedge y) * x \\ &\quad + (x \wedge y) * y - (x \wedge y) * (x \wedge y) \\ &= x * y. \end{aligned} \tag{16}$$

For a non-polynomial MBA expression, we notice that it includes multiple sub-expressions obfuscated by polynomial MBA rules. This finding inspires us to use the polynomial MBA simplification procedure to reduce a non-polynomial MBA expression. In particular, we first simplify the inner sub-expression (polynomial MBA expression), and the simplification result of the inner sub-expression is treated as a temporary variable to expose further reduction opportunities. An instance is shown in Example 5. During the simplification procedure, the inner polynomial MBA expressions are reduced to the simplified form, such as $(x \wedge y) + 2 * (x \wedge y)$, which is reduced to $(x + y)$. By replacing $(x + y)$ with an intermediate variable t_1, the expression can be further reduced to $(t_1 + x)$. At the last step, all temporary variables t_i are substituted back to produce the final result $(2 * x + y)$.

Example 5. For the non-polynomial MBA expression $(((x \wedge y) + 2 * (x \wedge y))|x) + ((2 * (x|y) - (x \wedge y)) \wedge x)$, we have

$$
\begin{aligned}
&(((x\wedge y)+2*(x\wedge y))|x)+((2*(x|y)-(x\wedge y))\wedge x)\\
&=((x+y-2*(x\wedge y))|x)+((2*(x+y-(x\wedge y))\\
&\quad-(x+y-2*(x\wedge y)))\wedge x)\\
&=((x+y)|x)+((x+y)\wedge x)t_1=(x+y)\\
&=(t_1|x)+(t_1|x)\\
&=t_1+x\\
&=2*x+y.
\end{aligned}
\tag{17}
$$

3.3. Algorithm and Implementation. The MBA simplification scheme we have described above is illustrated in Algorithm 2. The algorithm takes an MBA expression E as input and outputs its concise form. First, it checks whether the MBA expression is a polynomial MBA or not. For polynomial MBA, the algorithm applies Algorithm 1 to simplify the bitwise expressions. Then, an arithmetic reduction is performed to return the simplification result. For non-polynomial MBA, the algorithm applies the polynomial MBA simplification procedure to recursively reduce each inner sub-expression (polynomial MBA) and replace it with the simplified result. At last, the algorithm performs the arithmetic reduction to generate the final result. Note that Algorithm 2 applies Algorithm 1 and arithmetic computation to simplify an MBA expression, so it does not introduce any additional tables or manage extra heap memory.

We implement Algorithm 2 as an open-source tool, named MBA-Flatten. It accepts a complex MBA expression as the input and outputs the corresponding simplification result. An overview of MBA-Flatten's architecture is shown in Figure 2. The whole framework is written in around 1,800 lines of *Python* code. The parser and AST traversal components are coded based on the *Python* AST library. Moreover, we leverage the *Python* SymPy library for arithmetic reduction.

Inside MBA-Flatten, the main program consists of three major components. First, a parser receives the MBA expression and translates it to abstract syntax tree (AST) for the remaining process. Then, MBA-Flatten reduces the expression to a concise form. For polynomial MBA expression, the program uses the transformation procedure to reduce a bitwise expression, and a math reduction module is adopted to further simplify the expression. The math reduction module also includes the optimization function to generate an optimal result for some expressions; e.g., $x+y-2*(x\wedge y)$ can be further reduced to $(x\wedge y)$. For non-polynomial MBA expression, MBA-Flatten traverses the AST bottom-up and simplifies every inner subtree (polynomial MBA expression). After reducing each sub-expression, the simplified expression is replaced with the temporary variable. At last, arithmetic reduction rules are further performed to reduce the expression and return the final simplification result. MBA-Flatten also includes utilities for measuring the complexity metrics of MBA expressions, such as counting the number of nodes in the directed acyclic graph (DAG) representation of an MBA expression, and we will discuss the complexity measurement of MBA expressions further in Section 5.1.

4. Proof of Theorem 1

To prove Theorem 1, we first present that the transformation T is well defined. The definitions of value and form equivalence between two MBA expressions are shown as follows.

Definition 3. Suppose two MBA expressions $E_1(x_1,\cdots,x_t)$, $E_2(x_1,\cdots,x_t)$ *of variables* $x_k\in B^n, k=1,\cdots,t$. $E_1(x_1,\cdots,x_t)=^V E_2(x_1,\cdots,x_t)$ if $E_1(a_1,\cdots,a_t)\equiv E_2(a_1,\cdots,a_t)$ for all $a_k\in B^n E_1(x_1,\cdots,x_t)=^F E_2(x_1,\cdots,x_t)$ if $E_1(x_1,\cdots,x_t)$ and $E_2(x_1,\cdots,x_t)$ are of the same form

The maps in Equation (7) are identical in one-bit space. In other words, the bitwise expression e is equivalent to $T(e)$ with $x_k\in B$, which is shown as follows:

$$
T\left(e^1(x_1,\cdots,x_t)\right)=^V e^1(x_1,\cdots,x_t), x_k\in B, k=1,\cdots,t.
\tag{18}
$$

Proposition 1 shows that the transformation T is well defined, and one instance is shown in Example 6.

Proposition 1. *Let* e^n *be the bitwise expression of variables* $x_k\in B^n, k=1,\cdots,t$. *Given two bitwise expressions* e_1^n *and* e_2^n, *if* $e_1^n=^V e_2^n$, *then* $T(e_1^n)=^F T(e_2^n)$.

Proof. $e_1^n=^V e_2^n$ induces $e_1^1=^V e_2^1$. According to Equation (18), there is $T(e_1^1)=^V T(e_2^1)$. Note the uniqueness of $T(e)$, and then, $T(e_1^1)=^F T(e_2^1)$. Since $T(e^1)=^F T(e^n)$, we have $T(e_1^n)=^F T(e_2^n)$. □

Example 6. For the bitwise expressions $e_1=\sim(x_1\wedge x_2)$, $e_2=(x_1\wedge x_2)|(\sim x_1\wedge\sim x_2)$, and $e_1=e_2$. We have

$$
\begin{aligned}
T(e_1)&=-(x_1+x_2-2*(x_1*x_2))+1\\
&=-x_1-x_2+2*(x_1*x_2)+1,\\
T(e_2)&=x_1*x_2+(-x_1+1)*(-x_2+1)\\
&\quad-(x_1*x_2)*((-x_1+1)*(-x_2+1))\\
&=-x_1-x_2+2*(x_1*x_2)+1.
\end{aligned}
\tag{19}
$$

Thus, $T(e_1)=T(e_2)$.

Next, we present the concept of the signature vector shown as follows. The signature vector of a linear MBA expression is a vector with 2^t dimensions, where t is the number of variables in the expression.

Definition 4. (Xu [11]). *Let* $E=\sum_{i=1}^{s}a_ie_i$ *be a linear MBA expression, where* a_i *is integers and* e_i *is bitwise expressions. Let* M *be the* 2^t*s *Boolean matrix representing the truth table of* $e_1,\cdots e_s$, $\overrightarrow{v}=[a_1,\cdots,a_s]^T$. *The signature vector* $s(E)$ *is the product of the MBA truth table matrix* M *and the coefficient vector* $\overrightarrow{v}$.

$$
s(E)=M\overrightarrow{v}.
\tag{20}
$$

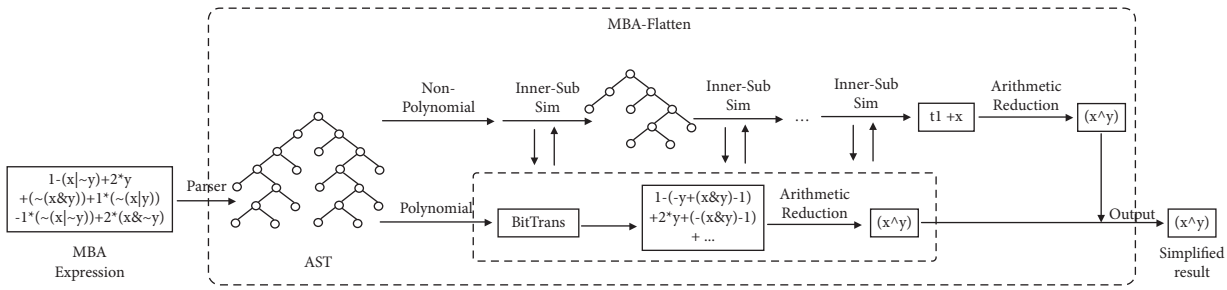


Figure 2: An overview of MBA-Flatten's architecture.

(i) Input: a bitwise expression e.
(ii) Output: the simplification result of e.
(1) Function BitTrans (e)
(2) Recursively apply the transformation T to transform e into $T(e)$.
(3) Replace all $(x_{i_j} * \cdots * x_{i_k})$ in $T(e)$ with $(x_{i_j} \& \cdots \& x_{i_k})$ to get $R(e)$.
(4) $F(e) = R(e) - 2 * a_e$.
(5) Return $F(e)$.
(6) End function

Algorithm 1: Transformation procedure of a bitwise expression.

Table 1 shows the truth table of multiple 2-variable bitwise expressions, and the column with all "1" is encoded as "−1" [1, 11]. Using Table 1, Example 7 presents the procedure of calculating the signature vector for expression $-x_1 - x_2 + 2 * (x_1 \wedge x_2) - 1$. The signature vector of a bitwise expression is actually to treat its corresponding truth table as a column vector, such as $s(x_1 \wedge x_2) = [0, 0, 0, 1]^T$.

Example 7. For a linear MBA expression $E = -x_1 - x_2 + 2 * (x_1 \wedge x_2) - 1$, using Table 1, we have

$$M = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad \overrightarrow{v} = \begin{bmatrix} -1 \\ -1 \\ 2 \\ 1 \end{bmatrix}, \quad s(E) = M\overrightarrow{v} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{21}$$

Then, we introduce the following lemma.

Lemma 1 *(Xu [11]). Given two linear MBA expressions E_1 and E_2, $E_1 = E_2$, if and only if $s(E_1) = s(E_2)$.*

Using Proposition 1 and Lemma 1, Theorem 1 can be proved as below.

Proof. Let $s(e)_j$ be the jth element of $s(e)$, $j = 1, \cdots, 2^t$. Note that Equation (11) is a linear MBA expression, $s(-1)_j = 1$, and $s(e)_j = 0$ or 1.

We prove $F(e) \equiv e$ using mathematical induction on the number of bitwise operators in the expression $e(x_1, \cdots, x_t)$ of variables $x_k \in B^n$.

Base step: the basis is the bitwise expression $e(x_1, \cdots, x_t)$ with a single bitwise operator, which is one of the following four cases:

$$\sim x, x \wedge y, x|y, \text{and } x \wedge y, \tag{22}$$

where $x, y \in \{x_1, \cdots, x_t\}$. □

Case 1. Suppose $e = \sim x$, we have $T(e) = -x + 1$, and then, $a_e = 1$; thus $F(e) = -x - 1$.

If $s(x)_j = 0$, then $s(e)_j = 1$ and $s(F(e))_j = -s(x)_j + s(-1)_j = 1$

If $s(x)_j = 1$, then $s(e)_j = 0$ and $s(F(e))_j = -s(x)_j + s(-1)_j = 0$

Therefore, $s(e)_j \equiv s(F(e))_j$.

Case 2. Suppose $e = x \wedge y$, we have $T(e) = x * y$, and then, $a_e = 0$; thus, $F(e) = x \wedge y$. It is plainly correct that $s(e)_j \equiv s(F(e))_j$.

Case 3. Suppose $e = x|y$, we have $T(e) = x + y - x * y$, and then, $a_e = 0$; thus $F(e) = x + y - (x \wedge y)$.

If $s(x)_j = 0$ and $s(y)_j = 0$, then $s(e)_j = 0$ and $s(F(e))_j = s(x)_j + s(y)_j - s(x \wedge y)_j = 0$

If $s(x)_j = 0$ and $s(y)_j = 0$, then $s(e)_j = 1$ and $s(F(e))_j = s(x)_j + s(y)_j - s(x \wedge y)_j = 1$

If $s(x)_j = 1$ and $s(y)_j = 0$, then $s(e)_j = 1$ and $s(F(e))_j = s(x)_j + s(y)_j - s(x \wedge y)_j = 1$

If $s(x)_j = 1$ and $s(y)_j = 1$, then $s(e)_j = 1$ and $s(F(e))_j = s(x)_j + s(y)_j - s(x \wedge y)_j = 1$

Therefore, $s(e)_j \equiv s(F(e))_j$.

Case 4. Suppose $e = x \wedge y$, proven as above.

The above four cases led to $a_e = 0$ or 1 and $s(e)_j \equiv s(F(e))_j$ that implies $s(e) \equiv s(F(e))$. By Lemma 1, $e \equiv F(e)$ holds where variables $x_k \in B^n$.

Induction step: assume $F(e) \equiv e$ holds with r bitwise operators ($r \geq 1$) in e. Performing one more bitwise operator to e, the new expression $\overline{e}(x_1, \cdots, x_t)$ is one of the following forms:

$$\begin{aligned} &\sim e, \\ &e \wedge x, x \wedge e, \\ &e|x, x|e, \\ &e \wedge x, x \wedge e, \end{aligned} \tag{23}$$

where $x \in \{x_1, \cdots, x_t\}$. Due to the commutative law of bitwise operators $\wedge, |, \wedge$ and the following equations:

$$\begin{aligned} e \wedge (\sim x) &= \sim (\sim e|x), \\ e|(\sim x) &= \sim (\sim e \wedge x), \\ e \wedge (\sim x) &= \sim (\sim e \wedge x), \end{aligned} \tag{24}$$

we only need to show that $F(\overline{e}) \equiv \overline{e}$ holds on the following four cases with $r + 1$ bitwise operators:

$$\sim e, e \wedge x, e|x, e \wedge x. \tag{25}$$

Assume $F(e) \equiv e$ with $a_e = 0$, and we get $s(e) \equiv s(F(e))$ and the following inductive hypothesis:

$$\begin{aligned} T(e) = &\sum_{i_1=1}^{t} a_{i_1} x_{i_1} + \sum_{1 \leq i_1 < i_2 \leq t} a_{i_1 i_2} \left(x_{i_1} * x_{i_2}\right) \\ &+ \cdots + a_{1 \cdots t} \left(x_1 * \cdots * x_t\right), \end{aligned} \tag{26}$$

$$\begin{aligned} F(e) = &\sum_{i_1=1}^{t} a_{i_1} x_{i_1} + \sum_{1 \leq i_1 < i_2 \leq t} a_{i_1 i_2} \left(x_{i_1} \wedge x_{i_2}\right) + \cdots \\ &+ a_{1 \cdots t} \left(x_1 \wedge \cdots \wedge x_t\right). \end{aligned} \tag{27}$$

Case 5. Suppose $\overline{e} = \sim e$; from the inductive hypothesis (Equation (26)), we have

$$\begin{aligned} T(e) = &\sum_{i_1=1}^{t} a_{i_1} x_{i_1} + \sum_{1 \leq i_1 < i_2 \leq t} a_{i_1 i_2} \left(x_{i_1} * x_{i_2}\right) \\ &- \cdots - a_{1 \cdots t} \left(x_1 * \cdots * x_t\right) + 1. \end{aligned} \tag{28}$$

Then, $a_{\overline{e}} = 1$; thus,

$$\begin{aligned} F(\overline{e}) = &\sum_{i_1=1}^{t} a_{i_1} x_{i_1} - \sum_{1 \leq i_1 < i_2 \leq t} a_{i_1 i_2} \left(x_{i_1} \wedge x_{i_2}\right) \\ &- \cdots - a_{1 \cdots t} \left(x_1 \wedge \cdots \wedge x_t\right) - 1. \end{aligned} \tag{29}$$

According to (27), we get $F(\overline{e}) = -F(e) - 1$.

If $s(e)_j = 0$, then $s(\overline{e})_j = 1$ and $s(F(\overline{e}))_j = -s(F(e))_j + s(-1)_j = 1$

If $s(e)_j = 1$, then $s(\overline{e})_j = 0$ and $s(F(\overline{e}))_j = -s(F(e))_j + s(-1)_j = 0$

Therefore, $s(e)_j \equiv s(F(e))_j$.

Case 6. Suppose $\overline{e} = e \wedge x$; from the inductive hypothesis (Equation (26)), we have

$$\begin{aligned} T(\overline{e}) = &\sum_{i_1=1}^{t} a_{i_1} \left(x_{i_1} * x\right) + \sum_{1 \leq i_1 < i_2 \leq t} a_{i_1 i_2} \left(x_{i_1} * x_{i_2} * x\right) \\ &+ \cdots + a_{1 \cdots t} \left(x_1 * \cdots * x_t * x\right). \end{aligned} \tag{30}$$

Then, $a_{\overline{e}} = 0$; thus,

$$\begin{aligned} F(\overline{e}) = &\sum_{i_1=1}^{t} a_{i_1} \left(x_{i_1} \wedge x\right) + \sum_{1 \leq i_1 < i_2 \leq t} a_{i_1 i_2} \left(x_{i_1} \wedge x_{i_2} \wedge x\right) \\ &+ \cdots + a_{1 \cdots t} \left(x_1 \wedge \cdots \wedge x_t \wedge x\right). \end{aligned} \tag{31}$$

If $s(x)_j = 0$, then $s(\overline{e})_j = 0$ and

$$\begin{aligned} s(F(\overline{e}))_j = &\sum_{i_1=1}^{t} a_{i_1} * s\left(x_{i_1} \wedge x\right)_j + \sum_{1 \leq i_1 < i_2 \leq t} a_{i_1 i_2} * s\left(x_{i_1} \wedge x_{i_2} \wedge x\right)_j \\ &+ \cdots \\ &+ a_{1 \cdots t} * s\left(x_1 \wedge \cdots \wedge x_t \wedge x\right)_j. \\ = &\sum_{i_1=1}^{t} a_{i_1} * 0 + \sum_{1 \leq i_1 < i_2 \leq t} a_{i_1 i_2} * 0 + \cdots + a_{1 \cdots t} * 0 = 0. \end{aligned} \tag{32}$$

If $s(x)_j = 1$, then $s(\overline{e})_j = s(e)_j$ and

$$\begin{aligned} s(F(\overline{e}))_j = &\sum_{i_1=1}^{t} a_{i_1} * s\left(x_{i_1}\right)_j + \sum_{1 \leq i_1 < i_2 \leq t} a_{i_1 i_2} * s\left(x_{i_1} \wedge x_{i_2}\right)_j c \\ &+ \cdots + a_{1 \cdots t} * s\left(x_1 \wedge \cdots \wedge x_t\right)_j \\ = &\, s(F(e))_j. \end{aligned} \tag{33}$$

Therefore, $s(e)_j \equiv s(F(e))_j$.

Case 7. Suppose $\overline{e} = e|x$; from the inductive hypothesis (Equation (26)), we have

$$\begin{aligned} T(\overline{e}) = &\sum_{i_1=1}^{t} a_{i_1} x_{i_1} + \sum_{1 \leq i_1 < i_2 \leq t} a_{i_1 i_2} \left(x_{i_1} * x_{i_2}\right) \\ &+ \cdots + a_{1 \cdots t} \left(x_1 * \cdots * x_t\right) + x \\ &- \sum_{i_1=1}^{t} a_{i_1} \left(x_{i_1} * x\right) + \sum_{1 \leq i_1 < i_2 \leq t} a_{i_1 i_2} \left(x_{i_1} * x_{i_2} * x\right) \\ &- \cdots - a_{1 \cdots t} \left(x_1 * \cdots * x_t * x\right). \end{aligned} \tag{34}$$

Then, $a_{\overline{e}} = 0$; thus,

$$F(\overline{e}) = \sum_{i_1=1}^{t} a_{i_1} x_{i_1} + \sum_{1 \le i_1 < i_2 \le t} a_{i_1 i_2}\left(x_{i_1} \wedge x_{i_2}\right) + \cdots + a_{1\cdots t}\left(x_1 \wedge \cdots \wedge x_t\right) + x - \sum_{i_1=1}^{t} a_{i_1}\left(x_{i_1} \wedge x\right) + \sum_{1 \le i_1 < i_2 \le t} a_{i_1 i_2}\left(x_{i_1} \wedge x_{i_2} \wedge x\right) - \cdots - a_{1\cdots t}\left(x_1 \wedge \cdots \wedge x_t \wedge x\right). \tag{35}$$

According to (27) and (31), we get $F(\overline{e}) = F(e) + x - F(e \wedge x)$.

If $s(x)_j = 0$, then $s(\overline{e})_j = s(e)_j$ and $s(F(\overline{e}))_j = s(F(e))_j + s(x)_j - s(F(e \wedge x))_j = s(F(e))_j$

If $s(x)_j = 1$, then $s(\overline{e})_j = 1$ and $s(F(\overline{e}))_j = s(F(e))_j + s(x)_j - s(F(e \wedge x))_j = s(F(e))_j + 1 - s(e)_j = 1$

Therefore, $s(e)_j \equiv s(F(e))_j$.

Case 8. Suppose $\overline{e} = e \wedge x$, proven as above.

The above four cases led to $a_{\overline{e}} = 0$ or 1 and $s(\overline{e})_j \equiv s(F(\overline{e}))_j$ that implies $s(\overline{e}) \equiv s(F(\overline{e}))$. By Lemma 1, $\overline{e} \equiv F(\overline{e})$ holds where variables $x_k \in B^n$.

Assume $F(e) \equiv e$ with $a_e = 1$; from the similar discussion as above, we have

$$a_{\overline{e}} = \begin{cases} 0, & \text{if} \overline{e} = \sim e \text{ore} \wedge x, \\ 1, & \text{if} \overline{e} = e | x \text{ore} \wedge x, \end{cases} \tag{36}$$

and $\overline{e} \equiv F(\overline{e})$ with variables $x_k \in B^n$.

As discussed above, the induction is completed. Thus, we have $F(e) \equiv e$ with variables $x_k \in B^n$ and $a_e = 0$ or 1 determined by e.

5. Experimental Results

In this section, a set of experiments are conducted to evaluate the MBA simplification scheme, MBA-Flatten. We first run MBA-Flatten and existing peer tools on two comprehensive MBA benchmarks. Z3 SMT solver [19] is used to check whether the simplified result is equivalent to the original MBA expression. The corresponding simplification results are discussed in Section 5.2–5.4. As reported in Section 5.5 and 5.6, MBA-Flatten can assist humans in analyzing software. At last, Section 5.7 studies MBA-Flatten's performance data, such as running time and memory footprint.

5.1. Experimental Setup

5.1.1. Peer Tools for Comparison. We collect and check existing state-of-the-art MBA simplification tools: MBA-Blast [5] and MBA-Solver [11]. MBA-Blast is a Python tool for simplifying MBA expressions via a two-variable transformation table. MBA-Solver produces multiple precomputed transformation tables, which enumerate all bitwise expressions and corresponding concise forms. Then, MBA-Solver uses these tables to simplify an MBA expression. For a more thorough evaluation, we also check other MBA simplification tools: GraphMR [18], SSPAM [13], and Syntia [14]. GraphMR is a neural network-based solution to reduce an MBA expression. SSPAM (symbolic simplification with pattern matching) is a pattern matching method that detects and reduces MBA expressions by multiple known MBA rules. Syntia is a program synthesis framework for approximating the semantics of expressions. It uses a set of input-output samples from the expression, learns the semantics of the samples, and synthesizes a simpler expression that is equal to the original expression.

5.1.2. Benchmarks. To fully expose the capability of diverse methods on simplifying MBA expressions, a large scale of MBA expressions is required for evaluation. Therefore, we consider two comprehensive MBA benchmarks: Dataset 1 [14] and Dataset 2 [11]. Dataset 1 comprises 500 MBA samples generated by Tigress [6] with up to three variables. Dataset 2 collects 3,000 MBA equations with up to four variables, which contains 2,000 polynomial MBA (1,000 linear MBA) and 1,000 non-polynomial MBA expressions. Every sample in datasets is a 2-tuple: (E_c, E_g). E_c is the complex MBA expression, and E_g is the related equivalent simple form. Multiple samples in benchmarks are shown in Table 2.

5.1.3. MBA Complexity Metrics. We use the following metrics to measure MBA complexity: number of DAG nodes and MBA alternation. For example, the expression $\sim (x \wedge y) + 3 * (x | y)$, whose DAG representation is shown in Figure 3, has 8 nodes and an MBA alternation (a red arrow means one MBA alternation) of 2. The larger a metric's value, the more complex an MBA expression. We expect the metrics' values will be reduced after simplification.

(1) Number of DAG Nodes. An MBA expression is transformed into a directed acyclic graph (DAG) representation in which the nodes are operators, variables, and constants. The number of nodes in the DAG is defined as a complexity metric for an MBA expression.

(2) MBA Alternation. The MBA complexity mainly comes from mixing bitwise operations and arithmetic operations. We adopt "MBA alternation" to measure the number of edges linking different types of operations in the DAG representation of an MBA expression.

5.1.4. Machine Configuration. All of our experiments are performed on a server with Intel Core i9 3.00 GHz CPU, 64 GB DDR4 RAM, 2 TB SSD Hard Drive, and running Ubuntu 20.04 OS.

5.2. Simplification on Dataset 1. In the first experiment, we run MBA-Flatten and other peer tools on Dataset 1. The evaluation result in Table 3 shows that only MBA-Flatten successfully produces verifiable simplification outputs for all

(i) Input: an MBA expression E.
(ii) Output: the simplification result of E.
(1) Function MBA-Flatten (E)
(2) If E is a polynomial MBA expression then
(3) Return PolySim (E).
(4) Else
(5) For inner sub-expression E_i is a polynomial MBA expression do
(6) $E_i' \leftarrow$ PolySim (E_i).
(7) Replace E_i with E_i'.
(8) Replace E_i' with temp variable t_i.
(9) End for
(10) Replace all t_i with E_i'.
(11) Arithmetic reduction on E.
(12) Return E.
(13) End if
(14) End function
(15) Function PolySim (E)
(16) For every bitwise expression $e \in E$ do
(17) $e' \leftarrow$ BitTrans (e).
(18) Replace e with e' in E.
(19) End for
(20) Arithmetic reduction on E.
(21) Return E.
(22) End function

Algorithm 2: Simplification procedure of an MBA expression.

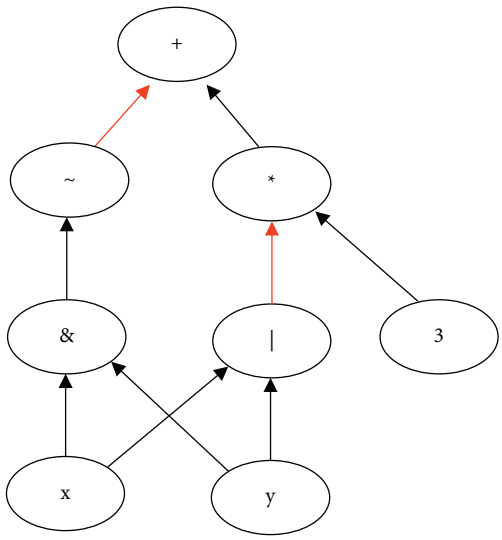


Figure 3: DAG representation of an MBA expression $\sim (x\&y) + 3 * (x|y)$.

MBA expressions with negligible overhead (within 0.1 seconds).

We first study the *correctness* that means an expression before and after simplification is semantically equivalent. Z3 solver [19] is adopted to check whether the output of a simplification tool is equivalent to the input. The solver may not return the solving result due to the MBA's complexity, so we set 1 hour as a practical threshold for this and the following experiments.

Table 1: Truth table of multiple bitwise expressions with 2 variables.

x_2	x_1	$\sim x_1$	$\sim x_2$	$x_1\&x_2$	$x_1\|x_2$	-1
0	0	1	1	0	0	1
0	1	0	1	0	1	1
1	0	1	0	0	1	1
1	1	0	0	1	1	1

Table 3 presents the number of MBA expressions that can be reduced by simplification tools. GraphMR is trained on the linear MBA dataset, so it can only simplify 137 of 500 MBA expressions. SSPAM outputs 168 wrong simplification results because of the limited number of MBA rules in the pattern library. Syntia uses stochastic program synthesis to generate a simple expression, which successfully synthesizes 369 simplification results. MBA-Blast performs well on simplifying 2-variable MBA expressions rather than three or more variables, and therefore, it generates 416 simplification results. MBA-Solver can successfully simplify the majority of the MBA expressions (454 of 500), but it cannot process several special cases, e.g., the non-polynomial MBA expression including sub-expression $\sim (x - 1)$. In contrast to MBA-Solver, MBA-Flatten can successfully simplify all 500 MBA samples, and it reduces $\sim (x - 1)$ to the expression $-x$.

Next, we investigate the *effectiveness* that reflects how much complexity is reduced by the simplification methods. Table 4 reports the expression complexity before and after simplification. Two quantitative metrics are used to measure expression complexity: the number of DAG nodes and MBA alternation. Table 4 shows that all simplification tools

Table 2: MBA samples in the benchmarks.

E_c	E_g
$(a^{\wedge} \sim a) + 2 * (a \mid a) + 1$	$a + a$
$2 * (\sim (x \wedge y)) + 3 * (\sim x \& y) + 3 * (x \& \sim y) - 2 * (\sim (x \& y))$	$x + y$
$\sim (((x \& y) * (x \mid y) + (x \& \sim y) * (\sim x \& y)) - 1)$	$-(x * y)$

Table 3: Simplification results using Dataset 1.

Tools	✓	✗	Timeout	Ratio (%)	Average time (s)
GraphMR	137	363	0	27.4	0.01
SSPAM	332	168	0	66.4	1.45
Syntia	369	131	0	73.8	38.5
MBA-Blast	416	0	84	83.2	0.02
MBA-Solver	454	0	46	90.8	0.02
MBA-Flatten	500	0	0	100.0	0.02

Table 4: Complexity metrics of the results on correctly simplified samples in Dataset 1 before and after simplification.

Tools	Average # of nodes			Average MBA alternation		
	Before	After	A/B (%)	Before	After	A/B (%)
GraphMR	9.7	3.6	37.1	3.6	0.2	5.6
SSPAM	6.2	4.0	64.5	2.4	1.5	62.5
Syntia	9.4	3.4	36.2	3.7	0.2	5.4
MBA-Blast	9.0	3.5	38.9	3.3	0.2	6.1
MBA-Solver	9.6	3.6	37.5	3.6	0.2	5.6
MBA-Flatten	9.8	3.7	37.8	3.7	0.2	5.4

(except SSPAM) can considerably reduce the complexity measurement of the solved MBA expressions. SSPAM cannot effectively reduce a complex MBA expression to a simpler form due to the limited known MBA rules used in the software.

5.3. Simplification on Dataset 2. As the second experiment, we run MBA-Flatten and other baseline tools on Dataset 2. As shown in Table 5, MBA-Flatten can successfully simplify 2,943 of 3,000 MBA expressions, and its average processing time is less than 0.2 seconds.

Considering the MBA expression in Dataset 2 is more complex and diverse than the one in Dataset 1, this experiment exposes more detailed findings. GraphMR and Syntia have limited effect on simplifying complex MBA expression, which can only correctly simplify less than 450 MBA samples. SSPAM cannot generate a simpler expression, so nearly 2/3 (1,975 of 3,000) of the simplified results cannot be checked by the Z3 solver within the time threshold. Compared with MBA-Blast (1,763 simplified samples), MBA-Solver can reduce more MBA expressions with three or four variables, and it successfully simplifies 2,899 MBA samples. MBA-Flatten can reduce 2,943 MBA samples, but it fails to simplify several special cases. One exception is the non-polynomial MBA expression $(\sim (x - 1) \wedge y) * (\sim (x - 1) \mid y) + (\sim (x - 1) \wedge \sim y) * (\sim (\sim (x - 1)) \wedge y)$. Table 6 reports that all solutions (except SSPAM) can generate a simpler equivalent expression. Overall, MBA-Flatten presents its advanced capability by successfully simplifying 98.1% of MBA samples.

Table 5: Simplification results using Dataset 2.

Tools	✓	✗	Timeout	Ratio (%)	Average time (s)
GraphMR	379	2621	0	12.6	0.01
SSPAM	705	320	1975	23.5	143.1
Syntia	437	2563	0	14.6	38.9
MBA-Blast	1763	0	1237	58.8	0.05
MBA-Solver	2899	0	101	96.7	0.10
MBA-Flatten	2943	0	57	98.1	0.16

Furthermore, we compare the average solving time of simplification tools on the two benchmarks. From Tables 3 and 5, the simplification time of GraphMR and Syntia is almost not increased, but SSPAM takes much more time when it simplifies a more complex MBA expression. MBA-Blast takes less than 0.1 seconds to simplify a two-variable MBA expression. Compared with MBA-Solver, MBA-Flatten takes slightly more time to simplify an MBA expression. The main reason is that MBA-Solver directly gets the bitwise expression simplification results from the transformation tables, rather than reduces it by multiple simplification procedures.

Table 6: Complexity metrics of the results on correctly simplified samples in Dataset 2 before and after simplification.

Tools	Average # of nodes			Average MBA alternation		
	Before	After	A/B (%)	Before	After	A/B (%)
GraphMR	32.1	6.9	21.5	6.8	0.9	13.2
SSPAM	35.3	30.4	86.1	7.5	6.5	86.7
Syntia	30.9	5.4	17.5	6.2	0.7	11.3
MBA-Blast	37.4	10.4	27.9	11.6	1.5	12.9
MBA-Solver	45.4	13.2	29.1	12.2	2.1	17.2
MBA-Flatten	45.3	11.2	24.7	12.3	1.5	12.2

5.4. Case Study. The evaluation results in Tables 3 and 5 show that MBA-Solver and MBA-Flatten are the most powerful MBA simplification tools. Throughout this case study, we demonstrate the strengths and weaknesses between MBA-Solver and MBA-Flatten.

We manually check the MBA expressions solved by MBA-Solver or MBA-Flatten, and one interesting observation is that MBA-Flatten can reduce all polynomial MBA expressions in the datasets, as MBA-Solver does. Does this scenario mean that MBA-Solver and MBA-Flatten can be substituted for each other? The answer is relevant to the number of variables in an MBA expression: as described in Section 2.2, MBA-Solver can successfully simplify a polynomial MBA expression with up to four variables; compared with MBA-Flatten, MBA-Solver is more efficient when it reduces an MBA expression. However, MBA-Flatten can simplify a polynomial MBA expression with an arbitrary number of variables, and the form of simplification result is shown in Equation (15). The following example shows how to apply MBA-Flatten to simplify Equation (6), which is an MBA expression with five variables.

Example 8. For Equation (6), we have

$$
\begin{aligned}
A &= -(\sim(x|y|t))\wedge(\sim a)\\
&= -\begin{pmatrix}(a\wedge t\wedge x\wedge y)-(a\wedge t\wedge x)-(a\wedge t\wedge y)+(a\wedge t)-(a\wedge x\wedge y)+(a\wedge x)\\ +(a\wedge y)-a-(t\wedge x\wedge y)+(t\wedge x)+(t\wedge y)-t+(x\wedge y)-x-y-1\end{pmatrix},\\
B &= -((\sim(x|y|\sim t))\wedge(\sim a))\\
&= -(-(a\wedge t\wedge x\wedge y)+(a\wedge t\wedge x)+(a\wedge t\wedge y)-(a\wedge t)+(t\wedge x\wedge y)-(t\wedge x)-(t\wedge y)+t),\\
C &= -(\sim(x|y|z|t)\wedge a)\\
&= -\begin{pmatrix}(a\wedge t\wedge x\wedge y\wedge z)-(a\wedge t\wedge x\wedge y)-(a\wedge t\wedge x\wedge z)\\ +(a\wedge t\wedge x)+a-(a\wedge t\wedge y\wedge z)+(a\wedge t\wedge y)+(a\wedge t\wedge z)\\ -(a\wedge t)-(a\wedge x\wedge y\wedge z)+(a\wedge x\wedge y)+(a\wedge x\wedge z)-(a\wedge x)\\ +(a\wedge y\wedge z)-(a\wedge y)-(a\wedge z),\end{pmatrix},\\
D &= -((\sim x\wedge\sim y\wedge(z|t))\wedge(a))\\
&= -\begin{pmatrix}-(a\wedge t\wedge x\wedge y\wedge z)+(a\wedge t\wedge x\wedge y)+(a\wedge t\wedge x\wedge z)\\ -(a\wedge t\wedge x)+(a\wedge t\wedge y\wedge z)-(a\wedge t\wedge y)-(a\wedge t\wedge z)+(a\wedge t)\\ +(a\wedge x\wedge y\wedge z)-(a\wedge x\wedge z)-(a\wedge y\wedge z)+(a\wedge z)\end{pmatrix},
\end{aligned}
\tag{37}
$$

and thus,

$$
f(x,y,z,t,a)=A+B+C+D-2*y-1+(x\wedge y)=x-y. \tag{38}
$$

The other observation is that MBA-Flatten can simplify all non-polynomial MBA expressions solved by MBA-Solver, but not vice versa. It is because that MBA-Solver treats the common sub-expression as an intermediate variable, rather than a sub-expression itself. Therefore,

Table 7: Experiment result of SMT solving on Dataset 2.

	Boolector			STP			Z3		
	Before	MS	MF	Before	MS	MF	Before	MS	MF
Polynomial	468	2000	2000	70	2000	2000	56	2000	2000
Non-polynomial	28	899	943	28	899	943	28	899	943
Total	496	2899	2943	98	2899	2943	84	2899	2943

Table 8: MBA-Flatten's performance on MBA expressions with different complexity.

# of nodes	Time (s)	Memory (MB)
10	0.02	0.2
50	0.18	1.1
100	0.53	4.3
150	0.91	7.6

MBA-Flatten can simplify more special cases that cannot be simplified by MBA-Solver. Moreover, MBA-Flatten can reduce a non-polynomial MBA expression with five or more variables.

From the description above, MBA-Flatten is a general MBA simplification method.

5.5. MBA-Powered Malware Deobfuscation. MBA expression is always used to obfuscate code, so malware developer also adopts the MBA expression to complicate the program. Liu et al. [5] report that MBA expressions are used in a ransomware sample to protect the encryption key, and they also observe that MBA rules are integrated into the software obfuscator VMProtect, which is widely used by malware developers.

In this experiment, we demonstrate that MBA-Flatten can assist in reverse-engineering the malware obfuscated by MBA expressions. We collect all MBA expressions used in malware from existing work [5]. Then, MBA-Flatten is applied to simplify the expressions, and the Z3 solver is used to check the correctness of the simplified result. The evaluation result shows that MBA-Flatten can successfully simplify all MBA expressions collected from existing malware samples. One simplification procedure is shown as follows, and MBA-Flatten produces the final result $(x - y)$.

$$\begin{aligned} &\sim (\sim x + y) \wedge \sim (\sim x + y) \\ &= \sim (-x - 1 + y) \wedge \sim (-x - 1 + y) t_1 = (-x - 1 + y) \\ &= \sim t_1 \wedge \sim t_1 \\ &= -t_1 - 1 \\ &= x - y. \end{aligned} \tag{39}$$

Furthermore, we replace the MBA expressions used in malware with new MBA expressions involving five or more variables and produce 130 variants, such as the above expression $\sim (\sim x + y) \wedge \sim (\sim x + y)$, which is replaced with Equation (6). We apply MBA-Blast and MBA-Flatten to simplify the new MBA expressions. Unfortunately, MBA-Blast fails to simplify them. In contrast, MBA-Flatten can successfully simplify all new MBA expressions. Therefore, this experiment shows that MBA-Flatten can simplify the MBA expressions used in existing malware and the complex MBA expression with five or more variables.

5.6. Boosting SMT Solving MBA Equations. Satisfiability modulo theory (SMT) solvers have been widely applied in diverse software engineering areas, such as software analysis [21, 22], symbolic execution [23, 24], and test generation [25]. Existing work [10, 11] has presented that SMT solvers are hard to solve MBA equations. However, the MBA simplification method, MBA-Solver, can be used to boost the SMT solver's performance on solving MBA equations (11).

In this experiment, we report that MBA-Flatten (denoted as MF) can assist SMT solvers in solving MBA equations. We consider the benchmark from work [11] and test three popular SMT solvers: Boolector [26], STP [27], and Z3 [20]. The benchmark is actually considered as Dataset 2 in this study, and MBA-Solver (denoted as MS) is considered as the baseline. MBA-Flatten and MBA-Solver are used to simplify all MBA equations in the benchmark, and then, the simplification results are output to the three SMT solvers.

The evaluation result is shown in Table 7, and the solving time threshold is set as 1 hour. Before simplification, all three SMT solvers can only solve a small portion (Boolector 496 (16.5%), STP 98 (3.3%), Z3 84 (2.8%)) of the MBA equations within the time threshold, but after simplification, all three solvers can solve over 96% of MBA equations. Compared with MBA-Solver, all SMT solvers can solve more MBA equations after MBA-Flatten's simplification. This is because MBA-Flatten can successfully simplify more MBA expressions than MBA-solver, as shown in Table 5. After MBA-Flatten's simplification, all SMT solvers can solve 2,943 of 3,000 MBA equations, which means that the distinction between solvers' performance on solving MBA expressions becomes insignificant. These results indicate that MBA-Flatten is a generic method to boost SMT solver's performance on solving MBA expressions.

5.7. Performance. This section reports MBA-Flatten's performance data. Table 8 shows the time and memory cost when MBA-Flatten processes an MBA expression with different complexity measured by the number of nodes. For every complexity measurement, 100 different MBA expressions are generated to do the test. As some of the timings are small, we repeat every test 100 times. MBA-Flatten is effective because it only performs low-cost arithmetic

computation. Our implementation adopts the *Python* SymPy library to efficiently perform the arithmetic reduction. Overall, MBA-Flatten is an effective tool for simplifying MBA expressions.

6. Discussion

MBA-Flatten has demonstrated the feasibility of automatically reducing MBA expressions. However, we also note some potential enhancements for future improvement.

As introduced in Section 5.3, MBA-Flatten cannot simplify the non-polynomial MBA expression $(\sim(x-1)\wedge y)*(\sim(x-1)|y)+(\sim(x-1)\wedge\sim y)*(\sim(\sim(x-1))\wedge y)$. We further investigate how to reduce it, and the simplification procedure is shown below. During the simplification procedure, the sub-expression $\sim(x-1)$ is treated as an intermediate variable rather than the expression $(x-1)$. However, it is hard for an automatic tool to precisely detect and identify the sub-expression, such as the sub-expression $\sim(x-1)$. To mitigate this problem, one possible solution is to integrate multiple heuristic rules into MBA-Flatten. Therefore, MBA-Flatten can explore diverse reduction directions to generate a simpler result.

$$\begin{aligned}
&(\sim(x-1)\wedge y)*(\sim(x-1)|y)+(\sim(x-1)\wedge\sim y)\\
&\quad*(\sim(\sim(x-1))\wedge y)\\
&=(t_1\wedge y)*(t_1|y)+(t_1\wedge\sim y)*(\sim(t_1)\wedge y)t_1=\sim(x-1)\\
&=t_1*y\\
&=\sim(x-1)*y\\
&=\sim t_2*yt_2=x-1\\
&=(-t_2-1)*y\\
&=(-x)*y.
\end{aligned}\tag{40}$$

It is possible that an adversary attacks MBA-Flatten by combining MBA obfuscation with other obfuscation techniques to generate an expression that does not satisfy the MBA definition in this study. Note that MBA-Flatten is designed for simplifying MBA expressions, so it may correctly handle the certain MBA sub-expression, but cannot solve the remaining non-MBA part. It is interesting to further investigate whether MBA-Flatten can interact with other analysis techniques (e.g., symbolic execution) to produce a better result.

7. Conclusion

Existing work performs well on simplifying MBA expression with very few variables. However, the state-of-the-art methods are hard to simplify a multivariable MBA expression. We investigate it and address this challenge using an in-place simplification method. A transformation procedure is proposed to transform a bitwise expression into a unified form, and we provide a mathematical proof to guarantee the correctness of this transformation. Then, the arithmetic reduction is used to further simplify the expression and produce a simplified result. Our large-scale experiments show that MBA-Flatten is a general and effective MBA simplification method. Furthermore, developing MBA-Flatten not only advances automated malware analysis but also boosts SMT solving on the MBA equations.

Data Availability

The data and codes presented in this study are available at https://tinyurl.com/y5l948pu.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this study.

Acknowledgments

The authors would like to thank team members from Anhui Province Key Laboratory of High Performance Computing for their valuable suggestions. The authors appreciate Jingyao Ke for proofreading the paper. This work was supported by the Core Electronic Devices, High-End Generic Chips and Basic Software of National Science and Technology Major Projects of China, under Grant no. 2012ZX01034-001-001.

References

[1] Y. Zhou, A. Main, Y. X. Gu, and H. Johnson, "Information Hiding in Software with Mixed Boolean-Arithmetic Transforms," in *Proceedings of the International Workshop on Information Security Applications*, pp. 61–75, Jeju Island, Republic of Korea, August 2007.

[2] Y. Zhou and A. Main, "Diversity via Code Transformations: A Solution for NGNA Renewable Security," in *Proceedings of the NCTA-The National Show*, Atlanta, GA, USA, April 2006.

[3] B. Liu, W. Feng, Q. Zheng, J. Li, and D. Xu, "Software Obfuscation with Non-linear Mixed Boolean-Arithmetic Expressions," in *Proceedings of the International Conference on Information and Communications Security*, pp. 276–292, Chongqing, China, September 2021.

[4] S. Schrittwieser, S. Katzenbeisser, J. Kinder, G. Merzdovnik, and E. Weippl, "Protecting software through obfuscation: can it keep pace with progress in code analysis?" *ACM Computing Surveys*, vol. 49, no. 1, pp. 1–37, 2016.

[5] B. Liu, J. Shen, J. Ming, Q. Zheng, J. Li, and D. Xu, "MBA-blast: unveiling and simplifying mixed boolean-arithmetic obfuscation," in *Proceedings of the 30th USENIX Security Symposium*, pp. 1701–1718, August 2021.

[6] C. Collberg, S. Martin, J. Myers, and J. Nagra, "Distributed application tamper detection via continuous software updates," in *Proceedings of the 28th Annual Computer Security Applications Conference*, pp. 319–328, Orlando FL USA, December 2012.

[7] C. Liem, Y. X. Gu, and H. Johnson, "A compiler-based infrastructure for software-protection," in *Proceedings of the Third ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, pp. 33–44, Tucson AZ USA, June 2008.

[8] S. Blazy and R. Hutin, "Formal verification of a program obfuscation based on mixed boolean-arithmetic expressions," in *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pp. 196–208, Cascais Portugal, January 2019.

[9] H. Ma, C. Jia, S. Li, W. Zheng, and D. Wu, "Xmark: dynamic software watermarking using Collatz conjecture," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 11, pp. 2859–2874, 2019.

[10] N. Eyrolles, *Obfuscation with Mixed Boolean-Arithmetic Expressions: Reconstruction, Analysis and Simplification Tools*, Université Paris-Saclay, Gif-sur-Yvette, France, 2017.

[11] D. Xu, B. Liu, W. Feng et al., "Boosting SMT solver performance on mixed-bitwise-arithmetic expressions," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 651–664, Canada, June 2021.

[12] A. Guinet, N. Eyrolles, and M. Videau, "Arybo: manipulation, canonicalization and identification of mixed boolean-arithmetic symbolic expressions," in *Proceedings of the GreHack*, Grenoble, France, November 2016.

[13] N. Eyrolles, L. Goubin, and M. Videau, "Defeating mba-based obfuscation," in *Proceedings of the 2016 ACM Workshop on Software PROtection*, pp. 27–38, Vienna Austria, October 2016.

[14] T. Blazytko, M. Contag, C. Aschermann, and T. Holz, "Syntia: synthesizing the semantics of obfuscated code," in *Proceedings of the 6th USENIX Security Symposium*, pp. 643–659, Vancouver, BC, CANADA, August 2017.

[15] R. David, L. Coniglio, and M. Ceccato, "Qsynth-a Program Synthesis Based Approach for Binary Code Deobfuscation," in *Proceedings of the BAR 2020 Workshop*, San Diego, CA, USA, February 2020.

[16] G. Menguy, S. Bardin, R. Bonichon, and C. D. S. Lima, "Search-based local black-box deobfuscation: understand, improve and mitigate," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2513–2525, Republic of Korea, Novembe 2021.

[17] W. Feng, B. Liu, D. Xu, Q. Zheng, and Y. Xu, "Neureduce: reducing mixed boolean-arithmetic expressions by recurrent neural network," in *Proceedings of the Findings of the Association for Computational Linguistics: Empirical Methods in Natural Language Processing*, pp. 635–644, Dominican Republic, November 2020.

[18] W. Feng, B. Liu, D. Xu, Q. Zheng, and Y. Xu, "GraphMR: Graph neural network for mathematical reasoning," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3395–3404, Punta Cana, Dominican Republic, November 2021.

[19] C. Collberg, C. Thomborson, and D. Low, "Manufacturing cheap, resilient, and stealthy opaque constructs," in *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 184–196, Boston , MA , USA, January 1998.

[20] L. D. Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *Proceedings of the International Conference on Tools And Algorithms For the Construction And Analysis Of Systems*, pp. 337–340, Budapest, Hungary, March 2008.

[21] J. Chen and F. He, "Control flow-guided SMT solving for program verification," in *Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering*, pp. 351–361, Montpellier France, September 2018.

[22] L. Cordeiro and B. Fischer, "Verifying multi-threaded software using SMT-based context-bounded model checking," in *Proceedings of the 33rd International Conference on Software Engineering*, pp. 331–340, Honolulu HI USA, May 2011.

[23] C. Cadar, D. Dunbar, and D. R. Engler, "Klee: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs," in *Proceedings of the 8th USENIX Symposium On Operating Systems Design And Implementation*, pp. 209–224, San Diego, CA, USA, December 2008.

[24] X. Li, Y. Liang, H. Qian et al., "Symbolic execution of complex program driven by machine learning based constraint solving," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pp. 554–559, Singapore, September 2016.

[25] E. Bounimova, P. Godefroid, and D. Molnar, "Billions and billions of constraints: whitebox fuzz testing in production," in *Proceedings of the 35th International Conference on Software Engineering*, pp. 122–131, San Francisco, CA, USA, May 2013.

[26] R. Brummayer and A. Biere, "Boolector: an efficient SMT solver for bit-vectors and arrays," in *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 174–177, York UK, March 2009.

[27] V. Ganesh and D. L. Dill, "A decision procedure for bit-vectors and arrays," in *Proceedings of the International Conference on Computer Aided Verification*, pp. 519–531, Berlin Germany, July 2007.