Journal of Computer Science and Technology

Completed Review

Reviewer 1

Reviewer Affiliation: University of Massachusetts Amherst, Department of Electrical & Computer Engineering

Manuscript ID:
Manuscript Type:
Keywords:
Speciality: Computer Architecture and Systems
Date Submitted: *blinded*
Manuscript Title:
Date Assigned: 12-Jan-2022
Date Review Returned: 28-Jan-2022

Overview Report	Outstanding	Good	Fair	Poor	Very Poor
Content					
Is the work relevant?		✓			
Is the work original?		✓			
Are the methods/proofs/experiments/etc. sound and convincing?			✓		
Presentation					
Is the abstract an adequate summary of the work?	✓				
Are the background and related work(s) clearly introduced?		✓			
Are the methods/proofs/experiments/etc. properly stated?			✓		
Are the conclusions clear and adequate?			✓		
Are the references adequate?		✓			

req Is the presentation clear to the relevant audience? ✓

req Are the overall organization and length of the manuscript adequate? ✓

req Is the English satisfactory? ✓

Summary Report

req **Quality**

Moderately significant

Priority of Publication

If accepted, should the manuscript be prioritized for publication? No

Accepted as

Regular Paper

req **Confidence of your evaluation**

5(Absolutely Confident)

req **Recommendation**

- Accept without revision
- Accept with minor revision
- ✓ Review again after major revision
- Resubmit after major revision
- Submit to another journal
- Reject

req **Would you be willing to review a revision of this manuscript?**

- ✓ Yes
- No

Comments

Confidential Comments to the Editor

An interesting approach to solving All SAT, but not very well explained. A major revision is necessary.

Comments to the Author

The paper describes an important and interesting problem of finding ALL solutions to a given SAT problem, and proposed an original method based on semi-tensor product matrix representation. The first part (1/3) of the paper is written relatively well (although is not precise enough in terms of definitions and explanation). However the middle part, pages 5-7, suffers from serious problems with the English language. English needs a serious improvement.

Part I, pages 1-4.
- The authors frequently use the term "circuit form" but never define it or illustrate it with an example (is Fig. 4 and example of such a form?). It is clear to an average reader in this area what CNF is, but not what the circuit form; this have to be clearly defined.

- Page 2 and later in Results, p. 11: what do you mean by "Overall, our method can reduce the total number of solutions by 50.6%" ? This is not an optimization problem where you are trying to reduce some function, but you are trying to find ALL solutions. So saying that you are "reducing the total number of solutions" sounds funny, and is incorrect. Please explain this.

- p. 2 Notations: item 20 "R^t is all matrices with dimension t". Is all? Do you mean that it is class of matrices with dimension t?

- Page 3: eq (1), what is S ? It would be very helpful to say that this is how you represent a Boolean variable. Without this, it is not clear how to interpret your formulas that have a matrix multiplied by variables P, Q, etc. It is later mentioned on p. 4 that "S is a logic variable", but this should be done earlier, first.

- Page 3: In Table 1 you should switch the columns representing conjunction and disjunction, to be consistent with the text above the Table, and eq (2); they should appear in the same order.

- Proposition 3 should be illustrated with an example (although this will become more clear after you clearly specify how P is represented as a matrix (like S).

- Page 4, Theorem 4: What is D_k ? Some kind of set, never defined.

- Page 5 - explain the concept of a "cut", it is not clear. You talk about the cut algorithm but never explain the notion of a cut.

- pages 5 + incorrectly use the term "isomorphic". Instead it should be called "homogeneous", in contrast to "heterogeneous" network representation. And you should clearly explain what you mean by this (that this is related to the type of nodes in the graph: either homogeneous, with only AND nodes, like in AIG, or heterogeneous, with OR and AND nodes).

It would be helpful to have one, however simple, example showing how your method finds all solutions, and clearly show those solutions. I have read the paper very carefully and couldn't really grasp how all solutions are generated by your method.

Finally, you should clearly state that your method is applicable only to small circuits, as clearly demonstrated in the results table.

English corrections:

P. 4, Theorem 1: ."...logic variable .." --> "logic variables" (plural)

p. 5, the second paragraph in Section 3.1 has to be rewritten in proper English. (... which are its matrix form ...). In fact this entire section should be rewritten.

p. 8 "computed to a matrix form" --> "converted to a matrix form"
But there are well too many corrections to include in this review. Please see the attached copy with my notes and corrections. Symbol E in circle means English correction is needed.

Consider two matrices $X \in \mathbb{R}^{m\times n}$ and $Y \in \mathbb{R}^{p\times q}$. For $X \cdot Y$ basic condition is that $n = p$, what is an insurmountable obstacle for many applications. Chen [23] proposed a semi-tensor product, which can conduct matrix multiply in any dimension. Let $\mathcal{A} \in \mathbb{R}^{m\times n}$ and $\mathcal{B} \in \mathbb{R}^{p\times q}$ then semi-tensor product can be defined as:

$$\mathcal{A} \ltimes \mathcal{B} = (\mathcal{A} \otimes I_{t/n})(\mathcal{B} \otimes I_{t/p})$$

where I and t denote identity matrix and the least common multiple of n and p, respectively. For these operations, we can assume

1)when $n = p$, $\mathcal{A} \ltimes \mathcal{B}$ is equivalent to $\mathcal{A} \cdot \mathcal{B}$

2)when $n = tp$, we have multiple relations $\mathcal{A} \succ_t \mathcal{B}$ respectively

Definition 1: *Let X be a row vector of dimension np, and Y be a column vector with dimension p. Then we split X into p equal-size blocks as $X^1, \ldots, X^p$, which are $1 \times n$ rows. Define the left semi-tensor product, denoted by $\ltimes$, as*

$$X \ltimes Y = \sum_{i=1}^{p} X^i y_i \in \mathbb{R}^n.$$

Proposition 1. *Let $A \in M_{p\times q}$ and $B \in M_{m\times n}$. If $q = km$, then*

$$A \ltimes B = A(B \otimes I_k)$$

Proposition 2. *Assume $A \in M_{m\times n}$ is given.*

1.Let $Z \in \mathbb{R}^t$ be a row vector. Then

$$A \ltimes Z = Z \ltimes (I_t \otimes A);$$

2.Let $Z \in \mathbb{R}^t$ be a column vector. Then

$$Z \ltimes A = (I_t \otimes A) \ltimes Z;$$

2.3 Matrix Expression of Logic

In this section, we consider the matrix expression of formulas of logic. Under matrix expression, a general description of logical operators is proposed and used to solve some problems in SAT. By using the semi-tensor product of matrices, the logic synthesis can be simplified a lot. The truth table of conjunction, disjunction, implication, and equivalence is as follows:

Table 1. Truth Table of Binary Operators

P	Q	$P \vee Q$	$P \wedge Q$	$P \to Q$	$P \leftrightarrow Q$
T	T	T	T	T	T
T	F	T	F	F	F
F	T	T	F	T	F
F	F	F	F	T	T

To use matrix expression we denote

$$S = \left\{ T = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, F = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \tag{1}$$

Definition 2: *Consider one fundamental unary operator: Negation, $\neg$, and four fundamental binary operators: Disjunction, $\vee$; Conjunction, $\wedge$; Implication, $\to$: Equivalence, $\leftrightarrow$. Their structure matrices are as follows:*

$$M_n = M_\neg = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix};$$

$$M_c = M_\wedge = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}; M_d = M_\vee = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \tag{2}$$

$$M_i = M_\to = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}; M_e = M_\leftrightarrow = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}.$$

We use P,Q, etc. as logic variables. That is, assume they can take value from S in (1). A straightforward computation shows that

Proposition 3. *1.For unary operator $\neg$, we have*

$$\neg P = M_n P.$$

2.Let σ be a binary operator($\vee, \wedge, \to$, or $\leftrightarrow$). Then

$$P \sigma Q = M_\sigma PQ,$$

where M_σ are defined in (2).

Using the structure matrices and the properties of the semi-tensor product, logical relations can be easily proved. We can also provide a canonical form of the expression. To do this we need some preparations:

Definition 3: *A 2×2^k matrix is called a logic matrix if all its columns are elements in S.*

All logic operation matrices, say M_n, M_c, M_d, M_i, and M_e, are logic matrices. Assume M_1 and M_2 are two logic matrices and S is a logic variable. Then $M_i S$, $i = 1, 2$, and $M_1 M_2$ are also logic matrices. At the same time, we define a matrix, called the power-reducing matrix, which can make $S^2 = M_r S$, as

$$M_r = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

A logic expression may have value-assigned logic variables, free logic variables and they are connected by logic operations. In STP we restrict the operations to $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$. They are complete, which means any logic operators can be expressed by them, so the restriction is non-necessary. But it makes the statement easily. In a logic expression, a logic variable is constant if its value is assigned in advance, it is called a free variable if its value can be arbitrary [24]. Using this concept, we have

Theorem 1: *Any logic expression $L(P_1, P_2, ..., P_s)$ with free logic variables $P_1, P_2, ..., P_s \in D_k$ can be expressed in a canonical form as*

$$L(P_1, P_2, ..., P_s) = M_L P_1 P_2 ... P_s,$$

where M_L is a $k \times k^s$ logic matrix.

Now we use the following example to show the application of `Theorem 1` and define this computation as the basic result seeking (Brs).

Example 1: *Person A said that person B is a liar, person B said person C is a liar, and person C said that both persons A and B are liars. Who is a liar?* [24]

Denote A: person A is not a liar; $\neg$A: person A is a liar. Then the logical expression of the statement is

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

Its matrix form, $L(A, B, C)$, is

$$M_c^2 (M_e A M_n B)(M_e B M_n C)(M_e C M_c M_n A M_n B).$$

Its canonical form can be computed as

$$L(A, B, C) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \end{bmatrix} * ABC.$$

L is true only if

$$A = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, C = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

So the conclusion is that A and C are liars.

2.4 Boolean Satisfiability

Boolean satisfiability problem is to determine whether there is a combination of values of one or more groups of variables through reasoning, which can make the value of the formula true. The algorithms for solving SAT problems are divided into complete algorithms and incomplete algorithms. Complete algorithms can always find all assignments that satisfy the instances or prove the unsatisfiability of the instances. Incomplete algorithm which is the heuristic algorithm improves the efficiency of solving the particular SAT problem with one solution. The boolean propositional formula can be expressed as CNF or circuit information.

The problem-solving method based on SAT can be divided into three steps, namely, encoding, preprocessing, and SAT decision. Encoding means that the problem is represented by propositional logic through some form of model description. Preprocessing is to make some form of transformation or simplification of the original formula in order to reduce the satisfiability judgment time of the formula after preprocessing. SAT decision uses reasoning technology to determine whether the formula is SAT or not.

3 Semi-Tensor Product based AllSAT Solver

In this section, we present the proposed cut algorithm for the STP-based solver. The computation pro-

cess overview is given first, and the algorithm, as well as an example, is demonstrated.

3.1 Overview

The main computation process is shown in this subsection. The input is a CNF or circuit information CI, which is also called the propositional instances ($\mathbb{I}$). The output is a string in input variable order ($\mathbb{S}$) that consists of $\{0, 1, -\}$.

First, we map the all elements in $\mathbb{I}$ to $\mathbb{I}.mf()$, $\mathbb{I}.m()$ and $\mathbb{I}.v()$, which are its matrix form, corresponding matrix and variables, through the parser. We can obtain the prerequisite for the calculation of the semi-tensor product of matrix and the generator of the solution by $\mathbb{I}.mf()$. Selective computation is carried out by determining whether the elements in $\mathbb{I}.m()$ are logic matrices, and assign the values in $\mathbb{I}.m()$ and $\mathbb{I}.v()$ to $\mathbb{I}.result()$ which means the middle solutions. Then get the final solutions $\mathbb{S}$ by mapping the $\mathbb{I}.result()$ to $\mathbb{S}$ from the result seeking. Note that the output string is first assigned $\underbrace{-,...,-}_{s}$. If the value of $\mathbb{I}.result()$ in its corresponding position equals "$-$", the $\mathbb{S}$ in the corresponding position will remain unchanged. If the value of $\mathbb{I}.result()$ in its corresponding position equals "1" and the value of $\mathbb{S}$ equals "1" or "$-$", the $\mathbb{S}$ in the corresponding position is the value of $\mathbb{I}.result()$. If the value of $\mathbb{I}.result()$ in its corresponding position equals "0" and the value of $\mathbb{S}$ equals "0" or "$-$", the $\mathbb{S}$ in the corresponding position is the value of $\mathbb{I}.result()$.

3.2 Cut

Intuitively, solving a sub-problem is much faster than solving the original problem because there are fewer or no decisions needed to solve in sub-problems. Therefore, we propose a cut algorithm to simplify the computation. The pseudo-code of the cut algorithm is shown in Algorithm 1. The *cut* function takes as input an original circuit Oc and returns the final solution of the cut circuit Fs. The cut algorithm can solve more complex problems at the same time and compute solutions as fast as possible. Through this algorithm, we find that the SAT solver should be guided to solve according to the topological order of the circuit.

Algorithm 1 cut algorithm

Input: Original circuit Oc
Output: Final solution Fs

```
1:  cut size (S), number of gates (G), number of output (O) ←Oc
2:  S = 2
3:  Target ←the value of the SAT of the root node
4:  for i = 1 to O do
5:      for j = 1 to G do
6:          if The parent node is Target then
7:              Set.result(i)(j) ←the value of the SAT of all child node
8:              Target ←the value of the SAT of the child or sibling node
9:          end if
10:     end for
11: end for
12: Fs ←sort Set.result()() in the order of input
13: return Fs
```

We first set the original cut size S, where the default is 2, set the number of gates G and outputs O, and assign the value of SAT of the root, which is 1, to the temporary quantity $Target$ (lines 1-3). Then the computation of the SAT is carried out from top to bottom, and according to the S, the SAT result of the parent node in CUT is obtained. And the values of the child or sibling nodes are assigned to a set of results $Set.result()()$ and $Target$, so as to facilitate the next computation (lines 4-11). Finally, the results in $Set.result()()$ are sorted according to the input order to find consistent solutions which are final solutions Fs (line 13).

Before *cut*, for a n-variable problem, the size of the structure matrix is (2×2^n), and the corresponding time complexity $(T(n))$ is equal to $O(2^n)$. When the cut

size is set to default which is 2, the size of the structure matrix to be computed is reduced to (2×4). So T(n) is equal to $O(4n)$. Overall, when the cut size is m ($m \ll n$), our proposed algorithm can effectively reduce $T(n)$ from $O(2^n)$ to $O(n2^m)$.

3.3 CNF Solving

We first define the size of $\mathbb{S}$ (s) and the size of *line* which represents the number of clauses in $\mathbb{I}$. Then traverse rest of $\mathbb{I}$, create its matrix form ($\mathbb{I}.mf()$) and corresponding matrix ($\mathbb{I}.m()$) and its position corresponding to the number ($\mathbb{I}.num()$). For a line i, if $\mathbb{I}.mf(i)$ equals "A", the $\mathbb{I}.m(i)$ will multiply $\mathbb{I}.m(i+1)$. All computation result will be stored in $\mathbb{I}.result()$ which based on `Overview`.

During matrix mapping, we traverse each clause to map the matrix form of the STP. For a variable, we further divide it into two cases that are positive or negative. In terms of a variable is positive, its matrix form is "M_dA". On the contrary, if the variable is negative, its matrix form is "$\overline{M}_nM_dA$". In addition, we define $\mathbb{I}.m()$ as a matrix set. Every matrix set corresponding each line can be computed as one matrix ($Result.M()$). For the basic result seeking (Brs), please refer to `Example 1`. At the same time, we store the absolute value of each variable in a number set as $\mathbb{I}.num()$. As for the cut, we can consider $\mathbb{I}.mf()$ as a circuit and every part before "A" as a smaller circuit. According to every computed result of $Result.M()$, the result seeking can be completed with the number of $\mathbb{I}.num()$.

The input is a set of matrices which is composed of $Result.M()$. The output is a string in input order ($\mathbb{S}$) that consists of $\{0, 1, -\}$. First, we define n as the size of $Result.M()$. Then for a i, each $Result.M(i)$ are computed as basic result seeking (Brs), where the first half of the result is the first variable result and the last of the result is the input to $Result.M(i+1)$. The final result is an intersection of every first half of the $Result.M(i)$.

3.4 Circuit Form Solving

Common SAT solver is usually designed based on CNF. To solve the SAT problems of the circuit, it is necessary to convert the Gate-level Netlist of the circuit to the corresponding CNF. However, in the process of transformation, the circuit structure information will be missing, especially the topological order between the signals within the circuit will be covered, and all the signals will become the variable of CNF. The circuit-based SAT solver follows the topology of the circuits, so it can solve the SAT problems more effectively.

3.4.1 Basic Solving

The circuit information expression contains complete logical operator which consists of $\{\neg, \wedge, \vee, \leftrightarrow, \rightarrow\}$, where expressed by "!, { }, ( ), [ ] and < >", such as "$!(A![BC])$" means "$M_nM_cAM_nM_eBC$".

We first parse the expression and create its matrix form ($\mathbb{E}.mf()$) and its corresponding matrix ($\mathbb{E}.m()$). Finally, we use the Brs to find all final solutions. When the circuit structure becomes larger, the corresponding number of matrices to be computed becomes larger. Therefore, the basic solving is limited to the computation of small-scale circuits, that is, the circuit computation after the cut.

3.4.2 Bench Solving

The bench file is an input form based on look-up table (LUT), and by analyzing the structure of the bench file, we find that the bench file has a special CUT characteristic. The bench file can be divided into three parts, one is PI, one is PO, and one is LUT. For the LUT part, we can regard it as a sub-circuit. We take the bench file as the input, in fact, the input automatically cut the circuit into multiple sub-circuits and con-

nect each sub-circuit, and the cut size (S) depends on the size of the LUT. After parsing the bench file, the circuit is converted to S-input logical Netlists. Then logical Netlists are provided to our solver as input.

It should be noted that the conversion of the input of the circuit form to the CNF is unique, that is, a circuit can only be converted into the corresponding CNF. but one CNF can be converted into a multi-form circuit structure. Therefore, when converting CNF to circuit form, it is important to choose the optimal circuit structure. At present, all SAT solvers are limited to isomorphic logic networks. In STP, isomorphic networks and heterogeneous networks have the same computing level. But for the number of logic levels and gates of a network, the expression of isomorphic form is much larger than that of heterogeneous form. The advantage of our method is that it can support heterogeneous network computing, and for networks with the same logic, we prefer to use heterogeneous networks as circuits to compute because they are faster.

3.5 Example

In this subsection, we use the MCNC benchmark circuit `c17` as an example to demonstrate the STP-based AllSAT of the proposed method. The open-source logic synthesis framework ALSO[1] maintains several commands for file conversions. Given a well-known AND-Inverter Graph (AIG) as an entry, we can obtain its corresponding CNF and circuit form. The CNF-based method of `c17` can be expressed by 11 variables and 18 clauses. The CNF of `c17` is shown in Fig.2.

```
p cnf 11 18
-2 6 3 0
-2 8 3 0
2 -3 0
2 -6 -8 0
3 -7 9 0
3 -7 8 0
-3 7 0
-3 -8 -9 0
-4 -5 0
-4 7 10 0
4 -7 5 0
4 -10 5 0
5 -8 -9 0
-5 8 0
-5 9 0
-11 0
2 0
4 0
```

Fig.2. c17.cnf

```
INPUT(n1)
INPUT(n2)
INPUT(n3)
INPUT(n4)
INPUT(n5)
OUTPUT(po0)
OUTPUT(po1)
n0 = gnd
n6 = LUT 0x8 (n1, n3)
n7 = LUT 0x8 (n3, n4)
n8 = LUT 0x2 (n2, n7)
n9 = LUT 0x1 (n6, n8)
n10 = LUT 0x2 (n5, n7)
n11 = LUT 0x1 (n8, n10)
po0 = LUT 0x1 (n9)
po1 = LUT 0x1 (n11)
```

Fig.3. c17.bench

Based on the proposed process shown in `Overview` and `CNF Solving`, we find that the matrix form of the fourth line is "$M_d A_1 M_n A_2$", and the corresponding matrix is shown as follows.

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} A_1 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} A_2$$

The $Result.M(1)$ of Brs is "11", "10", and "01". Hence, when A_1 is "1", A_2 can be "1" or "0". when A_1 is "0", A_2 can only be "1". The $Result.M(2)$ of Brs is "0". As a result, the final result is "11", "10" and "00". In all result, the result of fourth line is "-11- - - - - - -", "-10- - - - - - -", "-00- - - - - - -". As for every line in CNF, we can connect them with the cut algorithm by regarding the whole CNF as a parent node and each line as a child node.

The circuit-based method of `c17` is shown in Fig.4. The basic solving's matrix form of `c17` can be expressed as follows, respectively, where is the original logic network and the heterogeneous logic network, in which $1, 2, 3, 4, 5$ are input and $po0, po1$ are original output, $po2, po3$ are heterogeneous output.

[1] Chu Z. ALSO: Advanced logic synthesis and optimization tool. https://github.com/nbulsi/also

Consider two matrices $X \in \mathbb{R}^{m\times n}$ and $Y \in \mathbb{R}^{p\times q}$. For $X \cdot Y$ basic condition is that $n = p$, what is an insurmountable obstacle for many applications. Chen [23] proposed a semi-tensor product, which can conduct matrix multiply in any dimension. Let $\mathcal{A} \in \mathbb{R}^{m\times n}$ and $\mathcal{B} \in \mathbb{R}^{p\times q}$ then semi-tensor product can be defined as:

$$\mathcal{A} \ltimes \mathcal{B} = (\mathcal{A} \otimes I_{t/n})(\mathcal{B} \otimes I_{t/p})$$

where I and t denote identity matrix and the least common multiple of n and p, respectively. For these operations, we can assume

1)when $n = p$, $\mathcal{A} \ltimes \mathcal{B}$ is equivalent to $\mathcal{A} \cdot \mathcal{B}$

2)when $n = tp$, we have multiple relations $\mathcal{A} \succ_t \mathcal{B}$ respectively

Definition 1: *Let X be a row vector of dimension np, and Y be a column vector with dimension p. Then we split X into p equal-size blocks as $X^1, \ldots, X^p$, which are $1 \times n$ rows. Define the left semi-tensor product, denoted by $\ltimes$, as*

$$X \ltimes Y = \sum_{i=1}^{p} X^i y_i \in \mathbb{R}^n.$$

Proposition 1. *Let $A \in M_{p\times q}$ and $B \in M_{m\times n}$. If $q = km$, then*

$$A \ltimes B = A(B \otimes I_k)$$

Proposition 2. *Assume $A \in M_{m\times n}$ is given.*

1.Let $Z \in \mathbb{R}^t$ be a row vector. Then

$$A \ltimes Z = Z \ltimes (I_t \otimes A);$$

2.Let $Z \in \mathbb{R}^t$ be a column vector. Then

$$Z \ltimes A = (I_t \otimes A) \ltimes Z;$$

2.3 Matrix Expression of Logic

In this section, we consider the matrix expression of formulas of logic. Under matrix expression, a general description of logical operators is proposed and used to solve some problems in SAT. By using the semi-tensor product of matrices, the logic synthesis can be simplified a lot. The truth table of conjunction, disjunction, implication, and equivalence is as follows:

Table 1. Truth Table of Binary Operators

P	Q	$P \vee Q$	$P \wedge Q$	$P \to Q$	$P \leftrightarrow Q$
T	T	T	T	T	T
T	F	T	F	F	F
F	T	T	F	T	F
F	F	F	F	T	T

To use matrix expression we denote

$$S = \left\{ T = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, F = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \tag{1}$$

Definition 2: *Consider one fundamental unary operator: Negation, $\neg$, and four fundamental binary operators: Disjunction, $\vee$; Conjunction, $\wedge$; Implication, $\to$; Equivalence, $\leftrightarrow$. Their structure matrices are as follows:*

$$M_n = M_\neg = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix};$$

$$M_c = M_\wedge = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}; M_d = M_\vee = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \tag{2}$$

$$M_i = M_\to = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}; M_e = M_\leftrightarrow = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}.$$

We use P,Q, etc. as logic variables. That is, assume they can take value from S in (1). A straightforward computation shows that

Proposition 3. *1.For unary operator $\neg$, we have*

$$\neg P = M_n P.$$

2.Let σ be a binary operator($\vee, \wedge, \to$, or $\leftrightarrow$). Then

$$P \sigma Q = M_\sigma PQ,$$

where M_σ are defined in (2).

Using the structure matrices and the properties of the semi-tensor product, logical relations can be easily proved. We can also provide a canonical form of the expression. To do this we need some preparations: