A Novel, Efficient and Generalised Approach to Boolean Satisfiability

Abstract

One question which has occupied mathematicians, engineers and scientists of this age is the problem of Boolean satisfiability (SAT) which asks whether a given CNF formula has at least a satisfying assignment. The goal of this work is to present a novel, efficient and general method of deciding SAT and finding all the satisfying assignments of CNF formulas. This method is powerful as it employs resolution identities in transforming clauses without a particular variable into clauses with that variable.

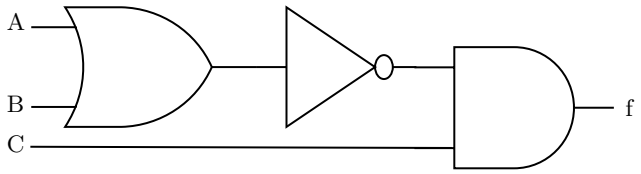
Keywords: SAT, satisfiable, resolution rule, resolution identity, DPLL algorithm

2010 Mathematics Subject Classification: 06E25, 93C85, 03G05.

1 Introduction

Suppose we are given the task of determining whether the output variable f of the digital circuit shown below evaluates to logic 1 for some assignments of logical values 0 and 1 to the input variables A, B, and C of the circuit. Now f is related to the input variables by

$$f = \overline{A}\,\overline{B}C.$$



The question is: *Is there a satisfying assignment of logical values* 0 *and* 1 *to* A, B *and* C, *i.e an assignment which makes* $f = \overline{A}\,\overline{B}C = 1$?. This is what the electronic engineer calls Circuit SAT.

What we commonly call SAT is by many termed Boolean satisfiability problem which imports the question as to whether a Boolean formula, e.g. $f = \overline{A}\,\overline{B}C$, has a satisfying assignment [7]. Boolean satisfiability is a propositional logic problem of interest in multiple fields, e.g., physics, mathematics, engineering and computer science.

It is well-known that the SAT problem requires efficient solution methods in a variety of applications. It is believed to require exponentially growing time for an algorithm to solve for the worst-case instances. [2],[5] [4].

The goal of this paper is to present a novel, efficient and general method of solving SAT and finding all the satisfying assignments of CNF formulas. I have found beneath this emblem some of the richest and most glowing properties of satisfiability of CNF formulas.

The rest of this work is divided into three sections. Section 2 is concerned with the definitions and notations required to understand this work. Section 3 concerns various methods of deciding SAT. Section 4 deals with the new method of solving SAT with ease as it makes use of transformation identities. I sincerely hope that this work may prove, as I intended, valuable to engineers, mathematicians, computer scientists and those involved in the design of SAT solvers.

2 Definitions and Notations

As there may be some of our readers who have little or no knowledge of Boolean Satisfiability, we deem it well to here call attention to the definitions and notations of terms associated with satisfiability.

Boolean Algebra is the algebra of truth values 0 representing *false* and 1 representing *true*. It was lauched by Boole and can be found in his work, the *Laws of Thought*, a gem, that for beauty of style and biilliancy of thought, has never been equalled on the subject of the Algebra of Logic [10] [9], [**?**].

Boolean variables are variables that can have the two possible truth values, 0 and 1. We use the capital letters of the English alphabet to denote variables. Thus the letters $A, B, C \ldots$ represent variables in this work. The next term on board is literal. With the common meaning which authors assigned this term, of course, we have so much to do. Some remarks, however, on the connection in which it stands, seem necessary for a proper understanding of the term. The **literals** of a variable L are its alternative forms, the unnegated L and the negated $\overline{L}$. They exhibit the properties $L\overline{L} = 0$ and $L + \overline{L} = 1$.

A **clause** is a logical sum of literals of different variables or a single literal. For example, $(A + \overline{B} + C)$ is a clause consisting of a the literals A, $\overline{B}$ and C. In arranging the literals in the clauses, we prefer the alphabetical order on account of its simplicity and ease to the reader. For instance, we shall write $(A + \overline{D} + F)$ instead of $(A + F + \overline{D})$. A clause is termed a **k-clause** if its number of literals is k. Thus, the clauses $(A), (A + \overline{B})$, and $(A + \overline{B} + C)$ are respectively 1-clause, 2-clause and 3-clause. A clause evaluates to logic 1 if a least one of its literals has been assigned a logical value of 1. If it evaluates to logic 0, it follows that each of its literals has been assigned a logical value of 0. Thus, if the clause $(A + \overline{B}) = 1$, then $A = 1$ or $\overline{B} = 1$ or both $A = 1$ and $\overline{B} = 1$. If the clause $(A + \overline{B}) = 0$ then $A = 0$ and $\overline{B} = 0$.

A **Boolean formula** is a logical expression defined over Boolean variables. Such a formula is in **conjunctive normal form** (CNF) if it is a logical product of clauses or a single clause. For example, the formula $f = (A + B)(\overline{A} + B + C)(\overline{D})$ is in CNF. A CNF formula is termed k-CNF formula when it contains clauses consisting of at most k literals.

To assist us in our reasoning, we shall invite the terms **native clauses** and **foreign clauses** to a formula f to describe respectively *clauses in* f and *clauses not in* f. Thus the native clauses of $f = (A + B)(\overline{A} + B + C)(\overline{D})$ are $(A + B), (\overline{A} + B + C)$, and $(\overline{D})$. The clause $(A + \overline{B})$ is a foreign clause to f because it is not in f.

A **Boolean assignment** to a set of Boolean variables is the set of truth values assigned to the variables or their literals in other to evaluate a Boolean formula. A **satisfying Boolean assignment** for a Boolean formula is an assignment such that the Boolean formula evaluates to logic 1. If the Boolean variables associated with a Boolean formula can be assigned logical values such that the formula turns out to be logic 1, then we say that the formula is *satisfiable*. If it is not possible to assign such values, then we say that the formula is *unsatisfiable*.

The above seem to be the most prominent terms, though not covering entirely the whole ground. There are however deeper terms of SAT that no language can describe, no illustration can reach, and no pencil can paint. And in endeavoring to bring out the valuable features of SAT, I have frequently been made sensible of the inadequacy of terms. For some good remarks on terms associated with SAT, see the paper

3 Methods of Testing SAT

3.1 Truth Table Method

The truth table provides a simple means for testing the satisfiability of a Boolean formula. To test or decide the satisfiability of the Boolean formula, we construct a truth table for the formula. There is one row for each possible truth assignment. For each truth assignment, each of the clauses in F_0 is evaluated. If any clause evaluates to 0, then $F_0 = 0$ implying F_0 as a whole is not satisfied by the truth assignment. If $F_0 = 1$, that is F_0 is satisfiable, then a satisfying truth assignment has been found. If no satisfying truth assignment is found, then F_0 as a whole is unsatisfiable.

As an instance, let us by the truth table method prove that the Boolean formula

$$f = (A + B)(\overline{A} + B)(\overline{A} + \overline{B})$$

is satisfiable. We build a truth table for F_0.

A	B	$A + B$	$\overline{A} + B$	$\overline{A} + \overline{B}$	F_0
0	0	0	1	1	0
0	1	1	1	1	1
1	0	1	0	1	0
1	1	1	1	0	0

The third column in the truth table above represents the clause $(A + B)$ for the two variables A and B. The fourth column represents the clause $(\overline{A} + B)$ and the fifth column represents the clause $(\overline{A} + \overline{B})$. The last column represents the CNF formula F. A 1 is placed in this column when the third, fourth, and fifth columns have a truth value of 1; otherwise 0 is placed in this column. The given Boolean formula is satisfiable because in the second row of truth values, $F_0 = 1$.

Again, by the truth table method, let us decide the satisfiability of the Boolean formula

$$f = (\overline{B})(C)(\overline{A} + B)(A + B + \overline{C}).$$

We build a truth table for F_0.

A	B	C	$\overline{B}C$	$\overline{A} + B$	$A + B + \overline{C}$	F_0
0	0	0	0	1	1	0
0	0	1	1	1	0	0
0	1	0	0	1	1	0
0	1	1	0	1	1	0
1	0	0	0	0	1	0
1	0	1	1	0	1	0
1	1	0	0	1	1	0
1	1	1	0	1	1	0

The seventh column does not have logic 1. It follows that the given Boolean formula F_0 is unsatisfiable.

The truth table approach is said to be complete as every truth assignment is verified. However, the method is not practical for all problem instances. In our first and second examples with 2 and 3 input variables respectively, there are $2^2 = 4$ and $2^3 = 8$ rows respectively. A problem instance with 10 input variables requires $2^{10} = 1024$ rows; this is even too small for a modern computer. But as the number of inputs increases, the number of rows quickly overwhelms even the fastest computers. A more efficient method is therefore required.

3.2 Resolution Algorithm

The **resolution algorithm**, due to Alan Robinson (1965), takes as input a CNF formula f and returns true iff the formula is satisfiable. It does this by performing a sequence of resolution steps, where each step consists of identfying two clausesC_1, C_2 of the form $C_1 = A + X$, and $c_2 = B + X$, and then appending to the CNF formula the *resolvent clause* $c = A + B$, where A and B are sums of literals, and X is a variable, called the *resolved variable.*

The resolution rule is applied to all possible pairs of clauses of a CNF formula, that contain complementary literals. After each aplplication of the resolution rule, the resulting CNF expression is simplified by deleting repeated literals. If a clause contains complementary literals, it is removed because it is equal to logic 1.

The following theorems of resolution will be given without proofs since the proofs can be found in many works on resolution.

Theorem 3.1. *Given a CNF formula f and a clause c which is a resolvent of two clauses of f, then $fc = f$.*

The resolution algorithm will play a paramount role in the rest of this work. The reader who is not acquainted with this algorithm is

Theorem 3.2. *Given a CNF formula f and a clause c which is a resolvent of two clauses of f, then f is satisfiable iff fc is satisfiable.*

This theorem is explained as follows. Any assignment that satisfies the formula f will also satisfy the new formula fc and vice versa.

3.3 DPLL Algorithm

Davis and Putnam (DP) algorithm for testing satisfiability of Boolean formulas dates back to the 1960's. Two years after this was published Davis, Logemann and Loveland launched a modified version of this algorithm, which is commonly known as DPLL algorithm. Though this algorithm is popular, it has a setback; it suffers mightly from exponential memory use, because it builds up an exponential set of clauses to verify satisfiability.

3.3.1 Rules of DPLL

We apply a given set of rules that preserve satisfiability. The following four rules are applied until they can be applied no more:

1. Unit clause rule
2. Pure-literal rule
3. Split rule

3.3.2 Unit-clause Rule

The unit-clause rule states that unit clause containing a particular literal is removed along with any other clause containing this literal. The negation of this literal is then removed from all clauses. This is also known as unit propagation rule.

3.3.3 Pure-literal Rule

The pure-literal rule states that If there is a pure literal L in F, delete all clauses containing L.

3.3.4 Split Rule

The split rule is stated as follows: From the CNF Boolean formula $F_0 = f_0(A, X)$ where A is a variable and $X = B, C, \ldots$ is a sequence of the other variables of F_0, we choose the variable A and form two new CNF formulas S_1 and s_1 obtained by setting $A = 1$ and $A = 0$ respectively. We then recurse on these.

We furnish an instance of the way in which DPLL algorithm is used to solve SAT. Let it be required to test the satisfiability of

$$F_0 = (\overline{A} + B + C)(\overline{A} + \overline{B} + C)(A + B + \overline{C})(A + B + C).$$

Unit propagation is not possible as there are no unit clauses. Pure literal rule is not applicable as there are no pure literals. We apply the splitting rule by selecting some literal, say A. We put $A = 0$ and propagate. This results in

$$F_{A=0} = (1 + B + C)(0 + \overline{B} + C)(0 + B + \overline{C})(0 + B + C)$$

which becomes

$$F_{A=0} = (\overline{B} + C)(B + \overline{C})(B + C).$$

Unit propagation and pure literal are still not applicable. Apply splitting rule for the next literal B. Set $B = 0$ and propagate:

$$F_{A=0,B=0} = (1 + C)(0 + \overline{C})(0 + C).$$

which becomes

$$F_{A=0,B=0} = (\overline{C})(C).$$

This formula consists of two unit clauses and so it is possible to apply unit propagation, which results in

$$F_{A=0,B=0} = 0.$$

Since $_{A=0,B=0} = 0$, we backtrack, set $B = 1$ and propagate:

$$F_{A=0,B=1} = (0 + C)(1 + \overline{C})(1 + C)$$

which results in

$$F_{A=0,B=1} = (C).$$

We apply unit propagation or the pure literal rule and conclude that this formula and hence the original formula is satisfiable.

The DPLL algorithm depends on the choice of branching literal, which is the literal considered in the backtracking step. As a result, this is not exactly an algorithm, but rather a family of algorithms, one for each possible way of choosing the branching literal. Efficiency is strongly affected by the choice of the branching literal: there exist instances for which the running time is constant or exponential depending on the choice of the branching literals. In reality DPLL is fast – the cases where a wrong choice of the branching literal is the reason for exponential runtime are uncommon. There are, however, formulas, where every strategy for selecting the branching literal will lead to an exponential runtime.

All known algorithms need exponential time to solve the SAT problem for some formulas.

4 A Novel Approach for Solving SAT

It is our purpose to furnish a method for solving SAT, but before turning to the method, it will be necessary to consider two theorems needed in the method.

4.1 Two Special Theorems

Let L be a literal and $X_1, X_2, X_3, \ldots X_n$ be n clauses. Then

$$f = (L + X_1)(L + X_2)(L + X_3)\cdots(L + X_n) = L + X_1X_2X_3, \ldots X_n.$$

Proof.

$$\begin{aligned} f &= (L + X_1)(L + X_2)(L + X_3)\cdots(L + X_n) \\ &= (LL + LX_2 + LX_1 + X_1X_2)(L + X_3)\cdots(L + X_n) \\ &= (L + X_1X_2)(L + X_3)\cdots(L + X_n) \\ &\vdots \\ &= L + X_1X_2X_3, \ldots X_n. \end{aligned}$$

□

There is a beautiful transformation identity that throws its light on SAT. This identity goes thus: Let A, B and C be variables. Then

$$(A + B)(\overline{A} + \overline{B})(B + C) = (A + B)(\overline{A} + \overline{B})(\overline{A} + C).$$

Mark here the transformation of clauses from those of only variable B to those of only variable A. I searched diligently in the most gigantic online libraries for a work on Boolean Satisfiability but I have never met with one embracing the use of this transformation identity. Before we pass on to its application in finding satisfying assignments of 2-CNF formulas, let us first seek the demonstration of its validity. A multitude of proofs might here be presented, but one is sufficient.

Proof. We shall employ the illustrious resolution rule. If we apply this rule to the expression

$$(A + B)(\overline{A} + \overline{B})(B + C)$$

we shall have

$$(A + B)(\overline{A} + \overline{B})(B + C)(\overline{A} + C) \tag{4.1}$$

where the resulting clause $(\overline{A} + C)$ is the resolvent of the parent clauses $(\overline{A} + \overline{B})$ and $(B + C)$.

We turn now to the application of the resolution rule to the second member of the identity, namely

$$(A + B)(\overline{A} + \overline{B})(\overline{A} + C).$$

The result is

$$(A + B)(\overline{A} + \overline{B})(\overline{A} + C)(B + C) \tag{4.2}$$

where the resulting daughter clause $(B + C)$ is the resolvent of the parent clauses $(A + B)$ and $(\overline{A} + C)$.

Now the two expressions (4.1) and (4.2) are essentially the same. Hence we reach the conclusion that the transformation identity proposed is valid.

A world of other identities can be derived from the above-mentioned identity. For if we set $A = \overline{A}$ in the identity we shall obtain the new identity

$$(\overline{A} + B)(A + \overline{B})(B + C) = (\overline{A} + B)(A + \overline{B})(A + C).$$

In this new identity, let us set $B = \overline{B}$. We get

$$(A + B)(\overline{A} + \overline{B})(\overline{B} + C) = (A + B)(\overline{A} + \overline{B})(A + C).$$

The following two examples on the testing of satisfiability of 2-CNF formulas will be enriched using the transformation identity.

Example 4.1. *Test the satisfiability of*

$$f = (A + B)(\overline{A} + C)(\overline{B} + \overline{C}).$$

We employ the resolution rule here. This introduced resolvents and f becomes

$$f = (A + B)(\overline{A} + C)(\overline{B} + \overline{C})(B + C)(A + \overline{C})(\overline{A} + \overline{B}).$$

The product $(A + B)(\overline{A} + \overline{B})$ is a transforming product. It transforms $(\overline{B} + \overline{C})$ into $(A + \overline{C})$ and $(B + C)$ into $(\overline{A} + C)$. Hence f is transformed into

$$f = (A + B)(\overline{A} + C)(A + \overline{C})(\overline{A} + C)(\overline{A} + \overline{B}).$$

which, applying Theorem 4.3, becomes

$$f = (A + B\overline{C})(\overline{A} + \overline{B}C)$$

which in turn becomes

$$f = A\overline{B}C + \overline{A}B\overline{C}.$$

Since f does not simplify to 0, we conclude that f is satisfiable and has the satisfying assignment of

$$\{(A = 1, \overline{B} = 1, C = 1), (\overline{A} = 1, B = 1, \overline{C})\}.$$

Let us take up the other instance.

Example 4.2. *Decide the satisfiability of*

$$f = (A + B)(\overline{A} + \overline{B})(A + \overline{C})(\overline{A} + C)(B + \overline{C})(\overline{B} + C).$$

The product $(A + B)(\overline{A} + \overline{B})$ is a transforming product. By it we transform $(B + \overline{C})$ into $(\overline{A} + \overline{C})$ and $(\overline{B} + C)$ into $(A + C)$. Hence, we get

$$\begin{aligned} f &= (A + B)(\overline{A} + \overline{B})(A + \overline{C})(\overline{A} + C)(\overline{A} + \overline{C})(A + C) \\ &= (A + B\overline{C}C)(\overline{A} + \overline{B}C\overline{C}) \\ &= (A)(\overline{A}) \\ &= 0. \end{aligned}$$

Since f is logic 0, we say that f is unsatisfiable.

4.2 A General Method of Solving SAT

Our chief business now is to present a general method of deciding SAT and then finding all the satisfying assignments of CNF formulas. Let $f^{(0)}$ be a given CNF formula comprising clauses with variable A and those without A. Also let $f_a^{(0)}$ be a CNF formula consisting of native clauses of $f^{(0)}$ and foreign clauses of $f^{(0)}$ with variable A such that $f_a^{(0)}$ contains transforming products capable of transforming all the native clauses without A into clauses with A. Let $f_i^{(0)}$ be the CNF expression consisting of all the introduced foreign clauses with A. Let $f^{(1)}$ be a new CNF formula consisting of all the clauses derived from negating $f_i^{(0)}$ and clauses of $f^{(0)}$. The following theorems will help in deciding SAT and finding the satisfying assignments of $f^{(0)}$.

Theorem 4.3. *Let $f^{(0)}$ be a given CNF formula and $f_i^{(0)}$ a CNF formula of foreign clauses to $f^{(0)}$. If*

$$f_a^{(0)} = f_i^{(0)} f^{(0)} \tag{4.3}$$

and

$$f^{(1)} = \overline{f_i^{(0)}} f^{(0)} \tag{4.4}$$

then

$$f_a^{(0)} + f_i^{(0)} = f^{(0)} \tag{4.5}$$

Proof. Adding (4.3) and (4.4) gives

$$\begin{aligned} f_a^{(0)} + f^{(1)} &= f_i^{(0)} f^{(0)} + \overline{f_i^{(0)}} f^{(0)} \\ &= (f_i^{(0)} + \overline{f_i^{(0)}}) f^{(0)} \\ &= f^{(0)}. \end{aligned}$$

□

Theorem 4.4. *Let $f^{(0)}$ be a given CNF formula and $f_i^{(0)}$ a CNF formula of foreign clauses to $f^{(0)}$. If*

$$f_a^{(0)} = f_i^{(0)} f^{(0)} \tag{4.6}$$

and

$$f^{(1)} = \overline{f_i^{(0)}} f^{(0)} \tag{4.7}$$

then $f^{(0)}$ is satisfiable if either or both of $f_a^{(0)}$ and $f^{(1)}$ are satisfiable.

Proof. We start with the equation $f_a^{(0)} + f_i^{(0)} = f^{(0)}$ of Theorem 4.3. If $f^{(0)}$ is satisfiable, then there are assignments to the variables of $f^{(0)}$ which make $f^{(0)=1}$ and hence we get

$$f_a^{(0)} + f_i^{(0)} = 1.$$

For this equation to be valid, these assignments must also make either or both $f_a^{(0)}$ and $f^{(1)}$ equal to logic 1.

□

Theorem 4.5. *Let $f^{(0)}$ be a given CNF formula and $f_i^{(0)}$ a CNF formula of foreign clauses to $f^{(0)}$. If*

$$f_a^{(0)} = f_i^{(0)} f^{(0)} \tag{4.8}$$

and

$$f(1) = \overline{f_i^{(0)}} f^{(0)} \tag{4.9}$$

then $f^{(0)}$ is unsatisfiable iff both $f_a^{(0)}$ and $f^{(1)}$ are unsatisfiable.

Proof. We start with the equation $f_a^{(0)} + f_i^{(0)} = f^{(0)}$ of Theorem 4.3. If $f^{(0)}$ is unsatisfiable, then all assignments to the variables of $f^{(0)}$ make $f^{(0)=0}$ and hence we get

$$f_a^{(0)} + f_i^{(0)} = 0.$$

For this equation to be valid, all these assignments must also make both $f_a^{(0)}$ and $f^{(1)}$ equal to logic 0. □

To obtain all the satisfying assignments it may be necessary to extend the computation to

$$f^{(k)} = \overline{f_i^{(k-1)}} f^{(k-1)}$$

where $k = 2, 3, 4, \ldots$ up to when one arrives at the results $f_a^{(k)} = 0$ and $f^{(k+1)} = 0$ or at a new CNF formula $f^{(k)}$ whose satisfying assignments can be found without introducing foreign clauses.

To render the work still more worthy of the public, I have presented few instances of its application.

4.2.1 Instances of 2-SAT

Example 4.6. *Decide the satisfiability of* $f^{(0)} = (A + B)(B + C)(C + D)$. *If it is satifiable, then find all the satisfying assignments.*

To replace the native clause $(B + C)$ with a clause of A, we append the foreign clause $(\overline{A} + \overline{B})$ to $f^{(0)}$. We employ this foreign clause because it is the alternate clause to the native clause $(A+B)$ and hence its multiplication with the partial CNF expression $(A + B)(B + C)$ furnishes the CNF expression $(\overline{A}+\overline{B})(A+B)(\overline{A}+C)$. We thus have the new formula with the appended foreign clause

$$f_a^{(0)} = (\overline{A} + \overline{B})\ (A + B)(\overline{A} + C)(C + D).$$

We turn to the replacement of the native clause $(C + D)$ with a clause of A. To achieve this, we introduce the foreign clause $(A+\overline{C})$ which is the alternate of the native clause $(\overline{A}+C)$. Multiplying the foreign clause by the partial CNF expression $(\overline{A} + C)(C + D)$ gives the CNF expression $(A + \overline{C})(\overline{A} + C)(A + D)$. Hence the formula with an appended foreign clause is improved to

$$f_a^{(0)} = (A + B)(\overline{A} + \overline{B})(\overline{A} + C)(A + \overline{C})(A + D).$$

This is simplified further to

$$f_a^{(0)} = \overline{A}\, B\overline{C}D + A\overline{B}C.$$

Since $f_a^{(0)}$ is not equal to logic 0, we say that the original CNF formula $f^{(0)}$ is satisfiable. If we set $f^{(0)} = 1$, we obtain three of the satisfying assignments of $f^{(0)}$, namely

$$\{(A = 1, \overline{B} = 1, C = 1, D = \{0, 1\}), (\overline{A} = 1, B = 1, \overline{C} = 1, D = 1)\}$$

To get more satisfying assignments, let us first determine the complement of the CNF formula consisting of the foreign clauses of A used in obtaining $f_a^{(0)}$. The formula is

$$f_i^{(0)} = (\overline{A} + \overline{B})(A + \overline{C}) = \overline{A}\,\overline{C} + A\overline{B}.$$

So

$$\overline{f_i^{(0)}} = (A + C)(\overline{A} + B).$$

Hence,

$$\begin{aligned} f^{(1)} &= (A + C)(A + B)(\overline{A} + B)(B + C)(C + D) \\ &= (B)(A + C)(C + D). \end{aligned}$$

This is a new CNF formula and we recommence the entire process. Thus,

$$\begin{aligned} f_a^{(1)} &= (B)(A + C)(\overline{A} + \overline{C})(\overline{A} + D) \\ &= AB\overline{C}D + \overline{A}BC. \end{aligned}$$

Hence, we obtain the three satisfying assignments

$$\{(A = 1, B = 1, \overline{C} = 1, D = 1), (\overline{A} = 1, B = 1, C = 1, D = \{0, 1\})\}.$$

For more satisfying assignments, we find the complement of the CNF formula consisting of only the foreign clauses used:

$$f_i^{(1)} = (\overline{A} + \overline{C})$$

which becomes

$$\overline{f_i^{(1)}} = (A)(C).$$

Appending this complement to $f^{(1)}$ furnishes the new CNF formula

$$f^{(2)} = (A)(C)(B)(A + C)(C + D)$$

which reduces to

$$f^{(2)} = ABC.$$

We get two more satisfying assignments:

$$\{(A = 1, B = 1, C = 1, D = \{0, 1\})\}.$$

There are altogether eight satisfying assignments of $f^{(0)}$.

Example 4.7. *Determine all the satisfying assignments of*

$$f^{(0)} = (A + C)(B + \overline{C})(\overline{B} + D)(C + \overline{D}).$$

We begin with the CNF formula consisting of the native clauses of A, the foreign clauses of A and the derived clauses of A obtained from the use of the transformation identities,

$$\begin{aligned} f_a^{(0)} &= (A + C)(A + B)(\overline{A} + \overline{B})(\overline{A} + \overline{C})(A + D)(\overline{A} + \overline{D}) \\ &= (A + BCD)(\overline{A} + \overline{B}\,\overline{C}\,\overline{D}) \\ &= A\overline{B}\,\overline{C}\,\overline{D} + \overline{A}BCD. \end{aligned}$$

From this we obtain two satisfying assignments of f^0:

$$\{(A = 1, \overline{B} = 1, \overline{C} = 1, \overline{D} = 1), (\overline{A} = 1, B = 1, C = 1, D = 1)\}.$$

We turn to the CNF formula of foreign clauses.

$$f_i^{(0)} = (A + B)(\overline{A} + \overline{B}) = A\overline{B} + \overline{A}B.$$

So

$$\overline{f^{(0)}}_i = (\overline{A} + B)(A + \overline{B}).$$

We append this to $f^{(0)}$ and get

$$f^{(1)} = (\overline{A} + B)(A + \overline{B})(A + C)(B + \overline{C})(\overline{B} + D)(C + \overline{D}).$$

For this new CNF formula, we employ the entire procedure again. The CNF formula consisting of the native clauses with A, foreign clauses of A and derived clauses obtained by using the transformation identities is as follows:

$$f_a^{(1)} = (\overline{A} + B)(A + \overline{B})(A + C)(A + \overline{C})(\overline{A} + D)(\overline{A} + \overline{C})(\overline{A} + \overline{D}) = 0.$$

We get no satisfying assignment here. We go further to find the CNF formula consisting of the foreign clauses. Thus we get

$$f_i^{(1)} = (\overline{A} + \overline{C})$$

which becomes

$$\overline{f^{(1)}}_i = (A)(C).$$

Hence we have the new CNF formula

$$f^{(2)} = (A)(C)(\overline{A} + B)(A + \overline{B})(A + C)(B + \overline{C})(\overline{B} + D)(C + \overline{D})$$

which simplifies to

$$f^{(2)} = ABCD.$$

We therefore get the satisfying assignments

$$\{(A = 1, B = 1, C = 1, D = 1)\}.$$

4.2.2 Instances of 3-SAT

That the reader may see the great beauty and force of this method, we present instances of 3-SAT.

Example 4.8. *Decide the satisfiability of*

$$f^{(0)} = (A+B+E)(A+C+\overline{E})(A+\overline{B}+\overline{D})(\overline{A}+D+E)(B+C+D)$$
$$(\overline{B}+C+D)(C+D+E)(\overline{C}+\overline{D}+\overline{E})$$

We start with the formula consisting of the native clauses of A, the foreign clauses of A and the derived clauses of A obtined by employing transformation identities:

$$\begin{aligned} f_a^{(0)} &= (A+B)(\overline{A}+\overline{B})(A+C)(\overline{A}+\overline{C})(A+B+E)(A+C+\overline{E})(A+\overline{B}+\overline{D}) \\ &(\overline{A}+D+E)(\overline{A}+C+D)(A+C+D)(\overline{A}+D+E)(A+\overline{D}+\overline{E}) \\ &= A(D+E)(\overline{B})(C+D)(\overline{C})+(\overline{A})(B+E)(C+\overline{E})(\overline{B}+\overline{D})(B)(C+D)(C) \\ &= A\overline{B}\,\overline{C}D+\overline{A}BC\overline{D}. \end{aligned}$$

Since $f_a^{(0)}$ is not equal to 0, we conclude that the original formula is satisfiable and has the satisfying assignments of

$$\{(A=1,\overline{B}=1,\overline{C}=1,D=1),$$
$$(\overline{A}=1,B=1,C=1,\overline{D}=1)\}.$$

In the next instance, I shall throw some light on how all other instances may be handled.

Example 4.9. *Find satisfying assignments of*

$$f^{(0)} = (A+B+F)(\overline{A}+C+\overline{F})(B+C+G)(\overline{B}+D+E)(C+D+\overline{G})(\overline{D}+\overline{E}+\overline{G}).$$

Those clauses without A begin with the variables B, C and D. Hence we introduce the CNF formula consisting of the transforming foreign 2-clauses of A containing these variables, *viz*

$$f_i^{(0)} = (A+B)(\overline{A}+\overline{B})(A+C)(\overline{A}+\overline{C})(A+D)(\overline{A}+\overline{D})(\overline{D}+\overline{E}+\overline{G}).$$

We append this formula to the original formula and get

$$f^{(0)} = (A+B)(\overline{A}+\overline{B})(A+C)(\overline{A}+\overline{C})(A+D)(\overline{A}+\overline{D})(A+B+F)(\overline{A}+C+\overline{F})$$
$$(B+C+G)(\overline{B}+D+E)(C+D+\overline{G})(\overline{D}+\overline{E}+\overline{G}).$$

Here we apply the resolution rule to transform clauses without A to those with A. Thus we get

$$f_a^{(0)} = (A+B)(\overline{A}+\overline{B})(A+C)(\overline{A}+\overline{C})(A+D)(\overline{A}+\overline{D})(A+B+F)(\overline{A}+C+\overline{F})$$
$$(\overline{A}+C+G)(A+D+E)(\overline{A}+D+\overline{G})(A+\overline{E}+\overline{G}).$$

which is simplified to

$$f_a^{(0)} = A\overline{B}C\overline{D}(C+\overline{F})(C+G)(D+\overline{G})+\overline{A}B\overline{C}D(D+E)(D+\overline{G})(\overline{E}+\overline{G})$$

which in turn reduces to

$$f_a^{(0)} = \overline{A}BCD\overline{E}+\overline{A}BCD\overline{G}.$$

Since $f_a^{(0)}$ is not equal to logic 0, we say that the original CNF formula $f^{(0)}$ is satisfiable. Satisfying assignments of $f^{(0)}$ are as follows:

$$\{(\overline{A}=1,B=1,C=1,D=1,\overline{E}=1,F=1,G=1),$$
$$(\overline{A}=1,B=1,C=1,D=1,E=1,F=1,\overline{G}=1)\}.$$

We turn to the complement of the CNF formula of foreign clauses. It is

$$\overline{f_i^{(0)}} = \overline{\overline{A}\overline{B}CD + \overline{A}BCD} \quad = (\overline{A} + B + C + D)(A + \overline{B} + \overline{C} + \overline{D}).$$

We get a new CNF formula and then repeat the procedure:

$$f^{(1)} = (\overline{A} + B + C + D)(A + \overline{B} + \overline{C} + \overline{D})(A + B + F)(\overline{A} + C + \overline{F})(B + C + G)(\overline{B} + D + E)$$
$$(C + D + \overline{G})(\overline{D} + \overline{E} + \overline{G})$$

We remove the complement of the CNF formula of foreign clauses

$$(\overline{A} + B + C + D)(A + \overline{B} + \overline{C} + \overline{D})$$

by appending to $f^{(1)}$ the product $(\overline{A}+B)(A+\overline{B})$ which consists of the subclauses of the complement. Also, we append pairs of clauses capable of transforming clauses without A to those with A. Hence we have

$$f_a^{(1)} = (\overline{A} + B)(A + \overline{B})(\overline{A} + \overline{C})(A + C)(\overline{A} + \overline{D})(A + D)(\overline{A} + B + C + D)(A + \overline{B} + \overline{C} + \overline{D})$$
$$(A + B + F)(\overline{A} + C + \overline{F})(B + C + G)(\overline{B} + D + E)(C + D + \overline{G})(\overline{D} + \overline{E} + \overline{G})$$

which is simplified to

$$f_a^{(1)} = (\overline{A} + B)(A + \overline{B})(\overline{A} + \overline{C})(A + C)(\overline{A} + \overline{D})(A + D)(\overline{A} + B + C + D)(A + B + F)(\overline{A} + C + \overline{F})$$
$$(B + C + G)(\overline{B} + D + E)(C + D + \overline{G})(\overline{D} + \overline{E} + \overline{G})$$

which, employing resolution identity, becomes

$$f_a^{(1)} = (\overline{A} + B)(A + \overline{B})(\overline{A} + \overline{C})(A + C)(\overline{A} + \overline{D})(A + D)(A + B + F)$$
$$(\overline{A} + C + \overline{F})(A + C + G)(\overline{A} + D + E)(\overline{A} + D + \overline{G})(A + \overline{E} + \overline{G}).$$

This is simplified to

$$f_a^{(1)} = AB\overline{C}\,\overline{D}E\overline{F}\,\overline{G} + \overline{A}\,\overline{B}CD\overline{E}F + \overline{A}\,\overline{B}CDF\overline{G}$$

which gives the satisfying assignments of $f^{(0)}$:

$$\{(A = 1, B = 1, \overline{C} = 1, \overline{D} = 1, E = 1, \overline{F} = 1, G = 1),$$
$$(\overline{A} = 1, \overline{B} = 1, C = 1, D = 1, \overline{E} = 1, F = 1, G = \{0, 1\}),$$
$$(\overline{A} = 1, \overline{B} = 1, C = 1, D = 1, E = \{0, 1\}, F = 1, \overline{G} = 1)\}.$$

We append the complement of the product of the introduced foreign clauses to $f^{(1)}$ and get the new CNF formula

$$f^{(2)} = (\overline{A} + \overline{B} + C + D)(A + B + \overline{C} + \overline{D})(\overline{A} + B + C + D)(A + \overline{B} + \overline{C} + \overline{D})(A + B + F)$$
$$(\overline{A} + C + \overline{F})(B + C + G)(\overline{B} + D + E)(C + D + \overline{G})(\overline{D} + \overline{E} + \overline{G})$$

which is reduced to

$$f^{(2)} = (\overline{A} + C + D)(A + \overline{C} + \overline{D})(A + B + F)(\overline{A} + C + \overline{F})(B + C + G)(\overline{B} + D + E)(C + D + \overline{G})$$
$$(\overline{D} + \overline{E} + \overline{G}).$$

We recommence the procedure. To remove the product of clauses

$$(\overline{A}+C+D)(A+\overline{C}+\overline{D})$$

from the formula, we append the product of their subclauses

$$(\overline{A}+C)(A+\overline{C}).$$

We also append the pairs of clauses which can transform clauses without A to those with A. Thus we get the formula

$$f_a^{(2)} = (A+B)(\overline{A}+\overline{B})(\overline{A}+C)(A+\overline{C})(\overline{A}+\overline{D})(A+D)(A+B+F)(\overline{A}+C+\overline{F})(B+C+G)$$
$$(\overline{B}+D+E)(C+D+\overline{G})(\overline{D}+\overline{E}+\overline{G})$$

which, applying resolution identity, furnishes

$$f_a^{(2)} = (A+B)(\overline{A}+\overline{B})(\overline{A}+C)(A+\overline{C})(\overline{A}+\overline{D})(A+D)(A+B+F)(\overline{A}+C+\overline{F})(B+C+G)$$
$$(\overline{B}+D+E)(C+D+\overline{G})(\overline{D}+\overline{E}+\overline{G}).$$

This simplifies to

$$f_a^{(2)} = A\overline{B}C\overline{D}+\overline{A}B\overline{C}D\overline{E}+\overline{A}B\overline{C}D\overline{G}.$$

which gives the satisfying assignments of $f^{(0)}$:

$$\{(A=1,\overline{B}=1,C=1,\overline{D}=1,E=\{0,1\},F=\{0,1\},G=\{0,1\}),$$
$$(\overline{A}=1,B=1,\overline{C}=1,D=1,\overline{E}=1,F=\{0,1\},G=\{0,1\}),$$
$$(\overline{A}=1,B=1,\overline{C}=1,D=1,E=\{0,1\},F=\{0,1\},\overline{G}=1)\}.$$

Other satisfying assignments can be found if we continue the computation following similar procedure.

What a glorious method is presented to our view! But we must stop here, for our limit reminds us that we must be brief.

5 Conclusion

This paper presented a novel approach to solving SAT. This was achieved by the introduction a foreign clauses which could transform a given CNF formula into one with clauses of a particular variable. Instances of the method were provided.

6 Acknowledgement

I am very thankful to Mr. Agun Ikhile for his persistent financial support. My thanks also go to the editors and reviewers of this paper for their helpful suggestions.

References

[1] Aaronson S. and Wigderson A., Algebrization: a new barrier in complexity theory, in STOC, 2008, pp. 731–740

[2] Aho, Alfred V.; Hopcroft, John E.; Ullman, Jeffrey D. (1974). The Design and Analysis of Computer Algorithms. Addison-Wesley. Theorem 10.4.

[3] Baker T. P., Gill J, and Solovay R., Relativizations of the P=?NP question, SIAM Journal on Computing 4:4,431-442, 1975

[4] Cook S. A., The complexity of theorem proving procedures, Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, 151-158, 1971.

[5] Davis M., Logemann G. , and Loveland D., Communications of the ACM 5, 394–397 (1962).

[6] Henryk Greniewski, Krystyn Bochenek and Romuald Marczyński, Application of Bi-Elemental Boolean Algebra to Electronic Circuits, Studia Logica: An International Journal for Symbolic Logic T. 2 (1955), pp. 7-76 (71 pages)

[7] Mironov, Ilya; Zhang, Lintao (2006). Biere, Armin; Gomes, Carla P. (eds.). "Applications of SAT Solvers to Cryptanalysis of Hash Functions". Theory and Applications of Satisfiability Testing — SAT 2006. Lecture Notes in Computer Science. Springer.

[8] Rajaraman, Radhakrishnan: Introduction To Digital Computer Design. PHI Learning Pvt. Ltd. p. 65.

[9] Rajendra P.: Fundamentals of Electrical Engineering. Prentice-Hall of India.

[10] Okoh U.: Boolean Subtraction and Division with Application in the Design of Digital Circuits. Journal of Engineering Research and Reports 20 (5), 95-117 DOI: 10.9734/JERR/2021/v20i517316