UNDER PEER REVIEW

Deciding Satisfiability in the Twinkling of an Eye

Abstract

The question as to whether a CNF Boolean formula is satisfiable is referred to as Boolean satisfiability problem (SAT). For decades now, this problem has attracted a great deal of attention. A well-known algorithm for solving this problem is the DPLL algorithm. However, this algorithm may run in an exponential time. The great plan embraced in this work is the modification of the DPLL algorithm for deciding the satisfiability of CNF Boolean formulas in a relatively short time.

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

2010 Mathematics Subject Classification: 53C25; 83C05; 57N16.

1 Introduction

There are a number of applications where we must consider the following problem: Given a CNF Boolean formula, we need to show whether or not the formula is satisfiable [6]. This problem which is known as Boolean satisfiability problem (SAT) has attracted increasing attention in recent times. A more important and trying problem is that of deciding SAT in polynomial time P. The millennium problem of P versus NP is one of the most significant unresolved problem in computer science. The set of decision problems denoted P (polynomial) is a collection of problems that are resolvable in polynomial time and the set of decision problems symbolized NP (nondeterministic polynomial) is the collection of problems for which a solution can be verified in polynomial time. If a decision problem can be verified in polynomial time, can the problem be solved in polynomial time? This question which was proposed by the illustrious American computer scientists Cook and Levin about forty years ago is famous and well-known as P versus NP.

Cook proved that SAT is NP-complete. SAT was the first problem demonstrated to be NP-complete and opened the door to showing other problems that are members of the set of NP-complete problems [4].

Attempts to show that SAT is in P led to the invention of various techniques. Baker, Gill, and Solovay demonstrated that with respect to some oracles, $P = NP$ whereas with respect to others, $P \neq NP$ [3]. Aaronson and Wigderson demonstrated that algebrizing technique is incapable of resolving the barrier problem of P versus NP. They claimed that the solution to the problem is in opening the Boolean formula wide enough in order to probe the formula in some deeper way for further progress [1].

About 1960, Davis, Putname, Longemann, Loveland, and others investigated the SAT problem, and their basic algorithm which runs in exponential time is called DPLL algorithm [2],[5], which is one of the most commonly used algorithms. Its ideas have been closely imitated. The goal of this paper is to modify the DPLL algorithm to decide SAT in the twinkling of an eye.

This work in its manner of presentation is better adapted to the use of Academics. The rest of it is divided into five sections. Section 2 is concerned with notation and definitions of terms used in the work. Section 3 concerns itself with the modification of the DPLL algorithm to decide SAT quickly. The 4th section introduces a means of transforming sums of CNF formulas into a single CNF formula. Section 5 deals with a new algorithm for deciding SAT. Section 6 provides rules of satisfiability whose application with the new algorithm help satisfiality to be decided in a short time.

2 Definitions, Notations and Laws

Boolean Algebra is that branch of Algebra in which the relations of *truth values* are investigated by representing them by symbols or letters which may be either 0 or 1. It is customary in this Algebra to use the phrase *logical values* as synonymous with *truth values* and this meaning will be attached to the phrase throughout the present work.

Any letter used to represent an unspecified logical value is termed a **Boolean variable**. In Boolean algebra, the logical operations of addition $+$, multiplication $\cdot$ and negation $^{-}$ are performed on the Boolean variables. A Boolean variable or its negation is a **literal**. Any expression built up from Boolean variables, say $A, B, C, \ldots$ or $A_1, A_2, A_3, \ldots$ and the Boolean values 0 and 1 is called a **Boolean expression**. For instance $A + \overline{B}$ is a Boolean expression comprising two variables A and B or two literals A and $\overline{B}$.

The logical assumptions which are taken to be true without proof are called **Boolean axioms**. Theorems used to simplify Boolean expressions are known as **Boolean theorems**. Some special axioms and theorems are stated as follows.

1. **Addition law:**

   A1: $0 + 0 = 0$

   A2: $0 + 1 = 1$

   A3: $1 + 0 = 1$

   A4: $1 + 1 = 1$

2. **Multiplication law:**

   A5: $0 \cdot 0 = 0$

   A6: $0 \cdot 1 = 0$

   A7: $1 \cdot 0 = 0$

   A8: $1 \cdot 1 = 1$

3. **Annulment Law:**

   T1: $1 + A = 1$

   T2: $A \cdot 0 = 0$

4. **Identity Law:**

   T3: $0 + A = A$

   T4: $A \cdot 1 = A$

5. **Idempotent Law:**

   T5: $A + A = A$

T6: $A \cdot A = A$

6. **Double Negation Law:**

   T7: $\overline{\overline{A}} = A$

7. **Complement Law:**

   T8: $\overline{A} + A = 1$

   T9: $A \cdot \overline{A} = 0$

As in ordinary algebra, the following laws hold in Boolean algebra: commutative and associative laws for addition and multiplication, distributive laws both for multiplication over addition and for addition over multiplication.

The logical sum of literals on distinct variables is called a **clause**. A clause with only one literal is referred to as a **unit clause**. The literals of a clause can be written in increasing order as in

$$A_1 + A_4 + A_7$$

or in alphabetical order as in

$$B + C + \overline{F}.$$

A **Boolean formula** is a logical expression defined over Boolean variables. A **Boolean assignment** to a set of Boolean variables is the set of logica values assigned to the variables 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*.

We will be interested in Boolean formulas in a certain special form, the conjunctive normal form; it is the generally accepted norm for SAT solvers because of its simplicity and usefulness. A **conjunctive normal form** CNF is a multiplication of clauses. A K-CNF is a CNF in which every clause contains at most K literals. If the negation of a literal does not appear in a CNF formula, we refer to it as a **pure literal**. The formula

$$d_1 = (A + \overline{B})(\overline{A} + B + C)(A + B + C)$$

is a 3-CNF Boolean formula with three variables

$$A, B, C,$$

five literals

$$A, \overline{A}, B, \overline{B}, C,$$

and three clauses

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

The negation of the literal C does not appear in the formula and so C is a pure literal.

The next term in course is the DPLL algorithm. This algorithm (DPLL) is so familiar that we think it unnecessary to dwell upon it at great length. The algorithm is the most popular complete satisfiability (SAT) solver. While its worst case complexity is exponential, three rules are applied to speed-up the decision process [1].

1. Unit Propagation Rule. This rule states that *one can set the value of the only unassigned literal of a unit clause in such a way that the clause is satisfied.*
2. Pure Literal Rule. The pure literal rule states that *if an unassigned literal appears while its negation does not, we can set the value of the literal to 1.*
3. Splitting Rule. This states that *one should choose an assignment of* 1 *or* 0 *for a Boolean variable in a formula, simplify the formula based on that choice, then recursively check the satisfiability of the simplified formula.* If the simplified formula is satisfiable, the original formula is satisfiable, otherwise, the same recursive check is done assuming the opposite logical value.

For some good remarks on this subject, see the paper [7].

3 Modification of the DPLL

Based on the DPLL splitting rule already mentioned, we choose a literal say A_k from the initial CNF formula d_k consisting of the variables $A_k, A_{k+1}, \ldots, A_n$ and assign the logic value 1 to it. The resulting CNF formula is denoted $[d_k]_{A_k=1}$. Notice the use of the square brackets around d_k. In fact, the notation

$$[d_k]_{A_k=1}$$

indicates that the logic value 1 is to be substituted for the variable A_k in the CNF formula d_k. We check if $[d_k]_{A_k=1}$ is satisfiable; if this is the case, the initial CNF formula d_k is satisfiable; otherwise, we do the same check, assuming the opposite logical value 0. Thus, we see that by the DPLL spliting rule, the initial CNF formula d_k is split into two simpler CNF formulas, $[d_k]_{A_k=1}$ and $[d_k]_{A_k=0}$. Hence, the original or initial formula d_k is satisfiable if either $[d_k]_{A_k=1}$ or $[d_k]_{A_k=1}$ or both are satisfiable.

In set theory, the set that consists of all elements belonging to either set A or set B or both is called the union of A and B, denoted as $A + B$. Thus the statement *either* $[d_k]_{A_k=1}$ *or* $[d_k]_{A_k=1}$ *or both are satisfiable* implies the new Booleam formula

$$d_{k+1} = [d_k]_{A_k=1}^{A_k=0} = [d_k]_{A_k=0} + [d_k]_{A_k=1} \tag{3.1}$$

is satisfiable. It follows that if d_{k+1} is satisfiable, then d_k is satisfiable, and if d_{k+1} is unsatisfiable, then d_k is unsatisfiable. Thus the problem of satisfying d_k is equivalent to the problem of satisfying d_{k+1}. Since the number of variables of d_{k+1} is smaller than that of the variables of d_k by one, deciding the satisfiability of d_{k+1} will be easier than deciding the satisfiability of d_k.

In the following instance we show how to derive the new formula d_2 from the original CNF formula d_1.

Example 3.1. *Given the CNF formula*

$$d_1 = (A_1 + \overline{A_2})(A_1 + \overline{A_3})(\overline{A_2} + A_3)$$

find d_{k+1}.

Letting $A_1 = 1$ gives

$$[d_1]_{A_1=1} = (1 + \overline{A_2})(1 + \overline{A_3})(\overline{A_2} + A_3)$$

which becomes

$$[d_1]_{A_1=1} = (\overline{A_2} + A_3).$$

Setting $A_1 = 0$ gives

$$[d_1]_{A_1=0} = (0 + \overline{A_2})(0 + \overline{A_3})(\overline{A_2} + A_3)$$

which becomes

$$[d_1]_{A_1=0} = (\overline{A_2})(\overline{A_3})(\overline{A_2} + A_3).$$

Hence, we get

$$\begin{aligned} d_2 &= [d_1]_{A_1=1} + [d_1]_{A_1=0} \\ &= (\overline{A_2} + A_3) + \overline{A_2}\,\overline{A_3}(\overline{A_2} + A_3). \end{aligned}$$

The recursive formula d_{k+1} is the sum of two CNF formulas. If it can be transformed to a CNF formula, we will be able to recursively reduce the number of variables of d_k easily and continuously until the satisfiability of d_k becomes decidable, employing the unit propagation and pure literal rules. The next section will be devoted to a method of transforming the sum of two CNF formulas into a single CNF formula.

4 Transforming Sum of CNFs to a Single CNF

I shall here invent a technique for transforming the sum of two CNF formulas to a Single CNF formula. Let $f_1, f_2, \ldots, f_p$ and $g_1, g_2, \ldots, g_n$ be clauses. Then

$$f_1 f_2 \cdots f_p + g_1 g_2 \cdots g_q = (f_1 + g_1)(f_1 + g_2)\cdots(f_1 + g_q)(f_2 + g_1)(f_2 + g_2)\cdots(f_2 + g_q)\cdots(f_p + g_1)(f_m + g_2)\cdots(f_p + g_q).$$

For what purpose were all mathematical theorems before they can be employed in mathematics but to convince, in terms not to be misunderstood, the readers of their soundness. A mathematical proposition then would be vain without the demonstration of its validity. Hence, I shall prove this novel theorem to convince the reader of its truth.

Proof.

$$\begin{aligned} & f_1 f_2 \cdots f_p + g_1 g_2 \cdots g_q \\ &= \overline{\overline{f_1 f_2 \cdots f_p + g_1 g_2 \cdots g_q}} \\ &= \overline{\overline{f_1 f_2 \cdots f_p}\,\overline{g_1 g_2 \cdots g_q}} \\ &= \overline{(\overline{f_1} + \overline{f_2} + \cdots + \overline{f_p})(\overline{g_1} + \overline{g_2} + \cdots + \overline{g_q})} \\ &= \overline{\overline{f_1}(\overline{g_1} + \overline{g_2} + \cdots + \overline{g_q}) + \overline{f_2}(\overline{g_1} + \overline{g_2} + \cdots + \overline{g_q}) + \cdots \overline{f_p}(\overline{g_1} + \overline{g_2} + \cdots + \overline{g_q})} \\ &= \overline{\overline{f_1}\,\overline{g_1} + \overline{f_1}\,\overline{g_2} + \cdots + \overline{f_1}\,\overline{g_q} + \overline{f_2}\,\overline{g_1} + \overline{f_2}\,\overline{g_2} + \cdots + \overline{f_2}\,\overline{g_q} + \cdots + \overline{f_p}\,\overline{g_1} + \overline{f_p}\,\overline{g_2} + \cdots + \overline{f_p}\,\overline{g_q}} \\ &= (f_1 + g_1)(f_1 + g_2)\cdots(f_1 + g_q)(f_2 + g_1)(f_2 + g_2)\cdots(f_2 + g_q)\cdots(f_p + g_1)(f_p + g_2)\cdots(f_p + g_q). \end{aligned}$$

□

With this transformation theorem, the Boolean formula d_{k+1}, the sum of the two CNF formulas, $[d_k]_{A_k=1}$ and $[d_k]_{A_k=0}$, can be transformed into a single CNF formula. We proffer an instance to show how it may be applied.

Example 4.1. *Transform the sum of CNF formulas*

$$d_2 = (A_2 + \overline{A_3})(\overline{A_3} + A_4) + A_2(A_3 + \overline{A_4})(\overline{A_2} + \overline{A_3} + A_4)$$

into a single CNF formula.

By the Transformation Theorem 4, we have

$$\begin{aligned}
d_2 &= (A_2 + \overline{A_3} + A_2)(A_2 + \overline{A_3} + A_3 + \overline{A_4})(A_2 + \overline{A_3} + \overline{A_2} + \overline{A_3} + A_4) + (\overline{A_3} + A_4 + A_2) \\
&\quad (\overline{A_3} + A_4 + A_3 + \overline{A_4})(\overline{A_3} + A_4 + \overline{A_2} + \overline{A_3} + A_4) \\
&= (A_2 + \overline{A_3})(A_2 + \overline{A_3} + A_4)(\overline{A_2} + \overline{A_3} + A_4) \\
&= (A_2 + \overline{A_3})(\overline{A_2} + \overline{A_3} + A_4).
\end{aligned}$$

Now to transform

$$f_1 f_2 \cdots f_p + g_1 g_2 \cdots g_q$$

to a single CNF formula using Theorem 4, the maximum number of possible combinations $f_i + g_j$ for $i = 1$ to p and $j = 1$ to q is pq. Consequently, the Transformation Technique proposed here requires at most $O(n^2)$ operations.

5 New Algorithm for Deciding SAT

We will now look at a new polynomial-time algorithm for solving SAT. This algorithm involves the continuous reduction of the original or initial CNF Boolean formula into a smaller and smaller CNF Boolean formulas until logic 1 or 0 emerges.

Given the CNF Boolean formula

$$d_1 = F(A_1, A_2, \ldots, A_n)$$

take the following steps to decide the satisfiability of d_1.

1. Set $k = 1$.
2. Set $d_{k+1} = [d_k]_{A_k=1} + [d_k]_{A_k=0}$.
3. Express d_{k+1} in CNF using Theorem 4.
4. If $d_{k+1} = 0$, print "d_1 is unsatisfiable"and stop.
5. If $d_{k+1} = 1$, print "d_1 is satisfiable ". Otherwise, go to step 6.
6. Set $k = k + 1$ and return to step 1.

In what follows we proffer instances of the way in which the new procedure proposed can be employed.

Example 5.1. *Decide the satisfiability of the Boolean formula*

$$d_1 = (A_1 + \overline{A_2})(A_1 + \overline{A_3})(\overline{A_2} + A_3).$$

We begin with the recurring formula

$$d_{k+1} = [d_k]_{A_k=1} + [d_k]_{A_k=0}.$$

Putting $k = 1$, we have

$$\begin{aligned}
d_2 &= [d_1]_{A_1=1} + [d_1]_{A_1=0} \\
&= (\overline{A_2} + A_3) + (\overline{A_2})(\overline{A_3})(\overline{A_2} + A_3) \\
&= (\overline{A_2} + A_3 + \overline{A_2})(\overline{A_2} + A_3 + \overline{A_3})(\overline{A_2} + A_3 + \overline{A_2} + A_3) \\
&= (\overline{A_2} + A_3).
\end{aligned}$$

Next, putting $k = 2$, we get

$$\begin{aligned}
d_3 &= [d_2]_{A_2=1} + [d_2]_{A_2=0} \\
&= (A_3) + (1) \\
&= 1.
\end{aligned}$$

The fact that $d_3 = 1$ suggests that d_2 and hence d_1 are satisfiable.

Example 5.2. *Decide the satisfiability of the Boolean formula*

$$d_1 = (\overline{A_1} + \overline{A_3})(A_1 + A_2 + A_3)(A_1 + \overline{A_2} + A_3)(A_1 + \overline{A_3})(\overline{A_1} + A_3).$$

We begin with the recurring formula

$$d_{k+1} = [d_k]_{A_k=1} + [d_k]_{A_k=0}.$$

Setting $k = 1$ gives

$$\begin{aligned} d_2 &= [d_1]_{A_1=1} + [d_1]_{A_1=0} \\ &= (\overline{A_3})(A_3) + (A_2 + A_3)(\overline{A_2} + A_3)(\overline{A_3}) \\ &= (\overline{A_3})(A_2 + A_3)(\overline{A_2} + A_3). \end{aligned}$$

Next, letting $k = 2$, we obtain

$$\begin{aligned} d_3 &= [d_2]_{A_2=1} + [d_2]_{A_2=0} \\ &= (\overline{A_3})(A_3) + (\overline{A_3})(A_3) \\ &= 0. \end{aligned}$$

The logical value of d_3, as we have seen, is 0. This means that the CNF Boolean formula d_3 is unsatisfiable. The implication of this is that the original CNF Boolean formula d_1 is unsatisfiable.

6 Satisfiability Rules

Given a $2-$ CNF formula of n variables, the maximum number of clauses it can have is well-known as $m = 2n(n-1)$. Our method reduces the number of variables for the steps in the order

$$n, n-1, n-2, \ldots.$$

It follows that no step will have clauses the number of which exceed m. Hence $2-$ SAT can be solved in polynomial time.

3-SAT is more difficult to solve than 2-SAT. It will require the introduction of more terms in order to unravel the mysteries surrounding 3-SAT. We have mentioned that the literals of a variable are the alternative forms of it, namely negated and unnegated literals. A negated literal of a variable is one with a bar over the variable. Examples include $\overline{A}, \overline{B}, \overline{A_1}$, etc. An unnegated literal of a variable is one without a bar over the variable. Examples are A, B, A_1, etc.

We shall now speak of three sorts of clauses which are of great significance in deciding $3-$ SAT. These are as follows:

1. Negated-literal clause,
2. Unnegated-literal clause, and
3. Mixed-literal clause.

A clause possessing only negated literals is termed negated-literal clause. Clauses without any negated literal are termed unnegated-literal clauses. A mixed-literal clause is one containing both negated and unnegated literals. Thus, the clauses $(\overline{A} + \overline{D} + \overline{E}), (A + B + D)$ and $(B + \overline{C} + D)$ are respectively negated-literal, unnegated-literal and mixed-literal clauses.

We shall now use these new terms in giving rules of deciding SAT in a short time and of identifying certain satisfiable CNF formulas.

Rule 1: (Unit-propagation rule) Delete from the given CNF formula every unit clause.

Rule 2: (Pure-literal rule) Delete from the given CNF formula every clause containing pure literals.

Rule 3: A CNF formula containing only unnegated-literal clauses is satisfiable. For instance, the CNF formula

$$(A + D + E)(A + E + G)(B + C + E).$$

is satisfiable. This is seen at once when we apply the pure-literal rule.

Rule 4: A CNF formula containing only negated-literal clauses is satisfiable. For instance, the CNF formula

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

is satisfiable. This is reached when we apply the pure-literal rule.

Rule 5: A CNF formula containing only mixed-literal clauses is satisfiable. Two satisfying assignments of such formula are those in which all the variables are assigned the same truth value.

For instance, the CNF formula

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

is satisfiable. Two satisfying assignments of this formula are

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

Rule 6: A CNF formula without any negated-literal clause is satisfiable. A satisfying assignments of such formula is that in which all the variables are assigned truth value of 1.

For instance, the CNF formula

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

is satisfiable and has the satisfying assignment of

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

Rule 7: A CNF formula lacking unnegated-literal clauses is satisfiable. A satisfying assignment of such formula is that in which all the variables are assigned the truth value of 0.

For instance, the CNF formula

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

is satisfiable and has the satisfying assignment of

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

Rule 8: A CNF formula lacking mixed-literal clauses is satisfiable. For instance, the CNF formula

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

is satisfiable and has the satisfying assignment of

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

Rule 9: A CNF formula having negated-literal, unnegated-literal and mixed-literal clauses may be satisfiable or unsatisfiable.

Rule 10: Let f_0 be a given CNF formula and f_L a CNF formula caved from f_0 and consisting of all clauses containing the variable L. Suppose the given formula contains a negated-literal clause containing $\overline{L}$. If there is no mixed-literal clause having the literal L as the only unnegated literal in the clause, then in f_0 replace f_L with $[f_L]_{L=1}^{L=0}$ to form the new CNF formula f_{0L}. This rule is used to eliminate negated-literal clauses from f_0.

Rule 11: Let f_0 be a given CNF formula and f_L a CNF formula caved from f_0 and consisting of all clauses containing the variable L. Suppose the given formula contains an unnegated-literal clause containing L. If there is no mixed-literal clause having the literal $\overline{L}$ as the only negated literal in the clause, then in f_0 replace f_L with $[f_L]_{L=1}^{L=0}$ to form the new CNF formula f_{0L}. This rule is used to eliminate unnegated-literal clauses from f_0.

Example 6.1. *Decide the satisfiability of*

$$f_0 = (A + C + D)(A + \overline{D} + E)(\overline{A} + \overline{E} + \overline{F})(B + \overline{D} + E)(\overline{B} + \overline{C} + F)(C + D + \overline{F}).$$

If f_0 is satisfiable, then it is possible to transform it to a formula without negated-literal clauses. The negated-literal clause in f_0 is

$$(\overline{A} + \overline{E} + \overline{F}).$$

Our goal is to eliminate it from f_0 and to achieve this goal, we begin with the search for clauses in which the literal A of the variable A in $(\overline{A} + \overline{E} + \overline{F})$ is the only unnegated literal. No such clause is found and so we proceed to the elimination of the clause $(\overline{A} + \overline{E} + \overline{F})$ by eliminating the variable A from f_0. Now

$$f_A = (A + C + D)(A + \overline{D} + E)(\overline{A} + \overline{E} + \overline{F})$$

We eliminate A:

$$[f_A]_{A=1}^{A=0} = (C + D)(\overline{D} + E) + (\overline{E} + \overline{F})$$

which becomes

$$[f_A]_{A=1}^{A=0} = (C + D + \overline{E} + \overline{F}).$$

Replacing f_A in f_0 with $[f_A]_{A=1}^{A=0}$ gives rise to the new CNF formula

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

Since f_{0A} lacks negated-literal clauses, we conclude that f_{0A} and hence f_0 are satisfiable.

Let us find a satisfying assignment of f_0. A satisfying assignment of f_{0A} is

$$(B = 1, C = 1, D = 1, E = 1, F = 1).$$

This is a partial satisfying assignment of f_0. Applying this to f_0 gives $f_0 = \overline{A}$. Setting $\overline{A} = 1$ gives a satisfying assignment of

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

Example 6.2. *Decide the satisfiability of*

$$f_0 = (A + B + D)(A + \overline{B} + \overline{E})(\overline{A} + \overline{C} + \overline{E})(\overline{B} + C + \overline{D})(\overline{B} + \overline{D} + E)(C + D + \overline{E}).$$

The formula consists of clauses of the three kinds, namely negated-literal, unnegated-literal and mixed-literal clauses. If this formula is satisfiable, then it will be possible to transform its expression to an expression which lacks negated-literal clauses.The formula consists of only one negated-literal clause, *viz*

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

We consider the first literal $\overline{A}$. We search the clauses of f_0 for the alternative form of this literal, that is the unnegated literal A. In particular we search for a clause which contains A as the only unnegated literal. There exists such clause in f_0 and is

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

We cannot therefore eliminate the variable so as to eliminate $(\overline{A} + \overline{C} + \overline{E})$ since the elimination will lead to another negated-literal clause. We therefore turn to consider the next literal $\overline{C}$ of $(\overline{A} + \overline{C} + \overline{E})$. We cannot eliminate clauses of the variable C because there is a clause in which the literal C is the only unnegated literal, namely $(\overline{B} + C + \overline{D})$. Finally, we look into the things of the last literal $\overline{E}$ in $(\overline{A} + \overline{C} + \overline{E})$. We cannot eliminate the clauses of the variable E because of the presence of $(\overline{B} + \overline{D} + E)$ in which E is the only unnegated literal. In order to solve this problem, let us return to the already mentioned clause $(A+\overline{B}+\overline{E})$ in which A is the only unnegated literal. We wish to eliminate this clause to give room to the elimination of the negated-literal clause

$(\overline{A}+\overline{C}+\overline{E})$ without the emergence of another negated-literal clause. To achieve this we consider the elimination of the variable B in $(A+\overline{B}+\overline{E})$ as this variable is not found in the negated-literal clause. Therefore we compute

$$[f_B]_{B=1}^{B=0} = (A+D)+(A+\overline{E})(C+\overline{D})(\overline{D}+E) = (A+D+\overline{E}).$$

We replace f_B in f_0 with $[f_B]_{B=1}^{B=0}$ and obtain the new CNF formula

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

We notice that $\overline{E}$ is a pure literal and setting it to logic 1 gives $f_{0B}=1$. Hence we conclude that the given formula f_0 is satisfiable.

Let us find a satisfying assignment of f_0 as it is satisfiable. Now

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

is a satisfying assignment of f_1 and hence a partial assignment of f_0. Applying this assignment to f_0 gives

$$f_0 = (1)(1)(1)(1)(\overline{B})(1) = \overline{B}.$$

In the rest instances we shall ignore the step of finding a satisfying assignments.

Example 6.3. *Decide the satisfiability of*

$$f_0 = (A+\overline{C}+\overline{D})(A+E+F)(\overline{A}+\overline{E}+\overline{F})(B+E+F)(\overline{B}+\overline{D}+\overline{F})(C+\overline{D}+E)(\overline{C}+E+F)(D+E+\overline{F}).$$

Searching f_0 for negated-literal clauses, we get two; $(\overline{A}+\overline{E}+\overline{F})$ and $(\overline{B}+\overline{D}+\overline{F})$. The literal that occurs most in these clauses is $\overline{F}$. Now we search f_0 for a clause in which the literal F is the only unnegated literal. There is no such clause and we turn to the elimination of variable F from f_0. Hence

$$\begin{aligned}[f_F]_{F=1}^{F=0} &= (A+E)(B+E)(\overline{C}+E)+(\overline{A}+\overline{E})(\overline{B}+\overline{D})(D+E)\\ &= (A+\overline{B}+\overline{D}+E)(A+D+E)(B+D+E)(\overline{B}+\overline{C}+\overline{D}+E)(\overline{C}+D+E).\end{aligned}$$

We replace f_F in f_0 with $[f_F]_{F=1}^{F=0}$ and get the new CNF formula

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

This formula has no negated-literal clauses and so we reach the conclusion that f_{0F} and hence f_0 are satisfiable.

Example 6.4. *Decide the satisfiability of*

$$\begin{aligned}f_0 = {}&(A+D+E)(\overline{A}+\overline{D}+F)(\overline{A}+C+F)(\overline{A}+E+\overline{F})(B+\overline{E}+\overline{F})(B+C+\overline{D})(\overline{B}+\overline{D}+\overline{E})\\ &(\overline{C}+D+\overline{F})(C+D+\overline{E})(D+E+F)(\overline{D}+\overline{E}+\overline{F}).\end{aligned}$$

The formula contains two negated-literal clauses, $(\overline{B}+\overline{D}+\overline{E})$ and $(\overline{D}+\overline{E}+\overline{F})$. There are clauses wherein B, D, E, F are the only unnegated literals. So we turn to the unnegated-literal clauses in f_0, $(A+D+E)$ and $(D+E+F)$. There are no clauses wherein the literal $\overline{F}$ is the only negated literal. We therefore eliminate $(D+E+F)$ by eliminating variable F. This being done gives the new CNF formula

$$\begin{aligned}f_{0F} = {}&(\overline{A}+\overline{D}+E)(A+\overline{D}+\overline{E})(\overline{A}+C+E)(\overline{A}+B+C+\overline{E})(\overline{A}+C+\overline{D}+\overline{E})(\overline{A}+D+E)\\ &(\overline{C}+D+E)(A+D+E)(B+C+\overline{D})(\overline{B}+\overline{D}+\overline{E})(C+D+\overline{E}).\end{aligned}$$

This is simplified to

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

This formula contains one negated-literal clause, $(\overline{B}+\overline{D}+\overline{E})$, and is satisfiable because the literal B of the variable B in $(\overline{B}+\overline{D}+\overline{E})$ has no clause in the formula in which it is the only unnegatd literal. Hence f_0 is satisfiable.

Example 6.5. *Decide the satisfiability of*

$$f_0 = (A + \overline{B})(A + \overline{C})(\overline{A} + B)(\overline{A} + C)(B + C)(\overline{B} + \overline{C}).$$

There is a negated-literal clause in f_0, $(\overline{B}+\overline{C})$. There are however clauses containing B or C as the only unnegated literal. We turn to the unnegated-literal clause $(B + C)$. There are clauses having $\overline{B}$ or $\overline{C}$ as the only negated literal. We cannot directly eliminate the negated-literal or unnegated-literal clause in f_0. Let us return to the negated-literal clause $(\overline{B} + \overline{C})$. To eliminate it, we have to eliminate the clause in which literal B is the only unnegated literal. This clause is $(\overline{A} + B)$ and to eliminate it we only have to eliminate variable A which is not in the unnegated-literal clause:

$$[f_A]_{A=1}^{A=0} = (B + \overline{C})(\overline{B} + C).$$

Replacement of f_A with $[f_A]_{A=1}^{A=0}$ in f_0 gives

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

It thus follows that f_0 is unsatisfiable.

Example 6.6. *Decide the satisfiability of*

$$f_0 = (A + B)(A + \overline{C})(\overline{A} + \overline{D})(\overline{A} + C)(B + C)(\overline{B} + \overline{C})(B + \overline{D})(\overline{B} + D)(C + D).$$

There are two negated-literal clauses in f_0, $(\overline{A} + \overline{D})$ and $(\overline{B} + \overline{C})$. There are however clauses containing A, B, C and D in which each literal is the only unnegated literal. We turn to the three unnegated-literal clauses $(A + B)$, $(B + C)$ and $(C + D)$. There are clauses having $\overline{A}, \overline{B}, \overline{C}$ or $\overline{D}$ as the only negated literals. We cannot therefore directly eliminate the negated-literal or unnegated-literal clauses from f_0. Let us eliminate $(A + B)$ and $(B + C)$ by eliminating variable B. This gives the new CNF formula

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

The unnegated literal in f_{0B} are $(A + D)$ and $(C + D)$. There is no clause of $\overline{D}$ wherein the literal is the only negated literal. It thus follows that f_0 is satisfiable.

Gladly would we continue this delightful subject, but our general plan forbids more instances.

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] 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. *Mathematics Handbook for Science and Engineering*, Springer, New York, 2006, 5th ed.

[7] Okoh U., A Novel, Efficient and Generalised Approach to Boolean Satisfiability, Journal of Engineering Research and Reports, Volume 26, Issue 3, Page 91-103, 2024; Article no.JERR.110430 ISSN: 2582-2926.