

# Three complete deterministic polynomial algorithms for 3SAT

charles sauerbier

**Abstract** – Three algorithms are presented that determine the existence of satisfying assignments for 3SAT Boolean satisfiability expressions. One algorithm is presented for determining an instance of a satisfying assignment, where such exists. The algorithms are each deterministic and of polynomial complexity. The algorithms determining existence are complete as each produces a certificate of non-satisfiability, for instances where no satisfying assignment exists, and of satisfiability for such assignment does exist.

**Key Terms** – Algorithms, Boolean algebra, Boolean propositional logic, satisfiability, complexity, computation theory, set theory.

**Note** – This paper stands as a revision, correction, expansion and mea culpa of [5].

## 1 Introduction

The problem is what by convention is identified as 3SAT. It is the problem of determining the ‘existence’ and ‘instance’ of a satisfying assignment to a Boolean propositional logic expression of form that is by convention referred to as *Conjunctive Normal Form (CNF)*.

All approaches evolve from two basic operations on the set of clauses of any given satisfiability expression: resolution and propagation. Both operations are well described in [1]. Conventional approaches for solvers are also given in the cited reference and can be found in numerous papers on the subject. [2], [3] provide compendium type references for methods and representative references.

The algorithms presented here are based on constraint propagation. The algorithms differ from solvers conventional approaches, well documented in available literature, in their performing the operation over specialized structures. In one case, the structure represents the complete set of all possible assignments to subsets of the set of variables on which clauses of the satisfiability expression are defined. In the other case, the structure represents the set of all possible constraints on the subsets of the set of variables. The first structure, representing assignments, is equivalent to a conjunction on subsets of *disjunctive normal form (DNF)* expressions; for brevity referred to as *C/DNF*. The second structure, representing constraints, is equivalent to a conjunction on subsets of conjunctive normal form (CNF) expressions; for brevity referred to as *C/CNF*.

The C/DNF and C/CNF expressions are transformational equivalents. The binary string representation of an assignment to and a constraint on any single subset of variables are negations of the respective binary string representations. The C/CNF expression corresponding

to any C/DNF expression contains all corresponding constraints for which an assignment is absent in the C/DNF (e.g., assignment ‘100’ being absent would cause constraint ‘011’ to be present.)

It happens that for any C/CNF instance there exists a ‘basis’ set of constraints from which all constraints in the C/CNF expression can be obtained by constraint propagation. It happens that the conventional CNF SAT expression is operative in context of the corresponding C/CNF expression in manner equivalent to a ‘basis’ set of constraints. The set of all satisfying assignment constraints can be obtained by constraint propagation to the respective C/CNF form. The C/CNF form transformed to equivalent C/DNF form provides operative representation for algorithm present for obtaining an instance of an assignment.

On termination of constraint propagation, C/DNF and C/CNF serve as certificates of satisfiability or non-satisfiability.

The C/DNF and C/CNF representations are product of work performed in early 1970’s in digital electronic applications. The solution of problems presented then was obtained, by commonly known means and methods, by reduction to sets of DNF Boolean expressions; those sets then combined into a single expression by conjunction.

[4] and related works respective complexity of computation based on SAT expressions were discovered in the literature at a later time. The work in regards computational complexity differed in the form of Boolean expression, as discussed above. The most significant difference in the respective considerations existed in the dependence on *a priori* knowledge that the expression represents a satisfying set of assignments in author’s work during the 1970’s. The underlying question of complexity theory thus never rose to consideration in application. The questions of relevance and relation were raised where it was observed that systems exploiting C/DNF expressions, and corollary C/CNF expressions, at that time (over course of the 1970’s) appeared to routinely resolve all instances of NP and NP-complete problems presented. The principal problem arising in the work of that time was the question of paths and circuits of specific lengths in graphs, to include Hamiltonian paths and circuits.

## 2 Approach

One algorithm for determining existence is based on reduction of an *a priori* known set of binary sequences. The set, represented using C/DNF expression, is reduced by imposition of constraints defined by clauses of the 3SAT expression. The reduction (1) removes assignments, to subsets of the set of variables, inconsistent with the constraints imposed by the clauses, and; (2) removes any assignments inconsistent with any derived constraints resulting from imposition of clausal constraints of 3SAT expression. The algorithm is complete in that it can produce a ‘certificate’ (proof) of non-satisfiability or satisfiability. The algorithm for determining

a satisfying assignment using the resulting reduced assignments set to derive an assignment, where such exists.

The second algorithm for determining existence, a corrected version of algorithm in [5], produces the C/CNF expression equivalent to the C/DNF expression of first algorithm by propagation of constraints defined by clauses of the given 3SAT expression. The first algorithm was motivation of the second. Aspects of algorithms, being related to other work, were subject to non-disclosure for a period of time. The result of second algorithm can be used to obtain a satisfying assignment by transform to assignment structure form, then applying the algorithm presented that operates on that structure to produce the assignment.

The third algorithm for determining existence executes on the set P of subsets of the set of variables in the 3SAT expression, where P is defined as the set of all subsets p such that  $|p| = 3$  for all p. It is made obvious by the execution of the second algorithm to produce an equivalent result. The algorithm can be extended to include all subsets of P such that  $|p| \leq 3$  for all p. The algorithm may run to termination condition more quickly for non-satisfiability cases.

### 3 Definitions

- Define V to be a set of Boolean (binary) variables.
- Define C to be a set of disjunctive clauses where each such clause is defined on 3 variables of V.
- Define  $E(V_E, C_E)$  to be a 3SAT CNF expression where  $V = V_E$ .
- Define K to be a set of integers where:  $|K| = |V|$ , and; 0 (zero) is the least value of K, and; there exists a bijection from K to V. Let K index the set V.
- Define P to be a set where: each element is a subset of V such that  $|p| = 3$  for all p, and;  $p_i = (v_a, v_b, v_c) = p_j$  if and only if  $i = j$ .
- Define D to be a set of elements  $d = (p, m)$  where: for each element p in P there exists an element in d that contains p, and; m is a set of disjunctive clauses on the variables of p component. Let elements of D be a representation of subsets of the clauses of  $C_E$ , and D thus represent the CNF expression  $E(V_E, C_E)$ .
- Define A to be a set of elements  $a = (p, q)$  where: for each element p in P there exists an element in 'a' that contains p, and; q is a set of conjunctive clauses on the variables of p component. Let the elements of A be a representation of subsets of the assignments to variables of V, thus  $V_E$ , and A thus represent a C/DNF expression.

Representation of components m and q may be had by using a single byte where elements of ordered set {000, 100, 010, 110, 001, 101, 011, 111} of binary strings are assigned a representative bit; the obvious and simplest mapping is the corresponding decimal value of each string, indexing the bits of a byte with base 0.

The constraints represented by m component in elements of D can be transformed to admitted assignments in q component of corresponding elements in an instance of A by removal of the negation of the binary string representation of the clauses as represented in D.

**Example:**

*Let  $(v_a, \neg v_b, \neg v_c) = '100'$  be a clause in  $C_E$ . The only assignment to the variable set  $(v_a, v_b, v_c)$  that will result in false on evaluation of the clause is '011', the negation of '100'; which is readily observed to be the value  $7 - 1 = 6$  in decimal numbers.*

*If the given clause were the only clause in its respective element d then one would have  $m = '01000000'$ , and  $q = '11111101'$  in the corresponding element of A.*

#### 4 Discussion

The C/DNF expression led to development at the time and since of means to perform various operations on the representation. Such led to development of the algorithm of [5] and the corrected versions presented here.

The 'hole' in the algorithm in [5] arises of a flaw in reasoning. D is a representational equivalent of E where D contains those constraints corresponding to the clauses of E. While C/DNF and C/CNF expressions are transformational equivalents, the C/CNF expression is the constraints on the set of strings. The C/DNF expression is a representation of a set of strings so constrained. Applying operations developed for application on C/DNF and C/CNF representations in context of previous noted work, to an instance of D containing only the set of constraints corresponding to clauses of the CNF expression will fail.

The reasoning and arguments of [5] were predicated on a processing model to which a central premise was that the set of strings represented is known *a priori*. The set of strings to be reduced needs to be known *a priori* application of constraints that reduce that set. The set is maintained consistent by constraint propagation over assignments representing the set of binary strings that was subject to imposition of the clausal constraints of the CNF expression.

The structure A is the representation of a set of strings that is used by the assignment reduction algorithm. Where  $q = '11111111'$  (1-string) for all elements of some instance of A, the latter is the set of all strings on the set V of variables. A subset of strings on V is then defined by the assignments in q components of A that are absent.

The correctness of the representation follows directly from the well-known theory on Boolean expressions and their calculus, Boolean algebra, and that of complexity theory as relates to binary propositional expressions.

The problem of determining a satisfying assignment to a given  $E(V_E, C_E)$  expression using an instance of A requires more than simply eliminating assignments in A corresponding to those

that fail to satisfy clauses of E. The correctness of an instance of A is predicated on an *a priori* known set of strings and consistency in the assignments to variables in all elements of A. Removing any assignments potentially produces inconsistencies in the assignments in the elements of A. Inconsistencies arise where removal of an assignment in some element of A produces a constraint on any single or multiple variables in the p component. The resulting ‘derived’ constraint(s) must then be propagated through all elements of A. Any derivative constraints produced in the process of constraint propagation must be similarly propagated. The process of constraint propagation produces a strictly monotonic decreasing sequence in the number of assignments in A. The constraint propagation process is therefore finitely bounded.

As no satisfying assignment of a given instance of  $E(V_E, C_E)$  can contain an assignment that is the negation of a clause’s binary string representation, representation of a satisfying assignment set in an instance of A must not contain the respective assignment and be consistent in assignments to variables in all elements of A. It is shown by simple contradiction that any such set A where the constraints of D are imposed and A is by constraint propagation made consistent in assignment to all variables in all elements of A that A is the satisfying set of assignments to the respective CNF expression of which D is representative. If one assumes A to contain an assignment to V that is not satisfying of the respective instance of E then A must admit assignment to at least one variable in contradiction of the constraints in  $C_E$ , thus elements of D. That contradicts removal of all assignments disallowed by the constraints of D and assignments to variables in A having been made consistent with the constraints.

It is obvious then that success hangs on the consistency of assignments to variables post elimination of an assignment by imposition of a constraint in D.

If with imposition of each constraint in D to an instance of A we propagate, recursively, the resulting constraints on individual and multiple variables until no additional constraints arise then the resulting instance of A must be consistent. If one assumes the resulting instance of A is inconsistent then there exists at least one variable v in A such that there exists some  $a_i$  and  $a_j$  containing v, and either  $a_i$  or  $a_j$  constrains assignments to v so as to disallow an assignment present in the other; contradicting the constraint propagation process terminating condition.

A structure D derived from an instance of A in which clauses of E have been inserted and all derivative constraints resolved reflects the complete set of clauses determining the set of strings to which E restricts assignment of the variables of V.

That same instance of D can also be obtained, without use of A, by constraint propagation over an instance of D after clauses of E have been inserted, by propagating derived constraints within D. The latter approach was the intent of the algorithm of prior versions, in which restriction to just the set of elements in D to which clauses were entered gave rise to failure of

the algorithm. This approach of constraint propagation in D then differs from methods found in the literature, in 2002 and present, in propagation over all feasible subsets of V of order 3.

The specific problematic structure that produced failure in previous algorithm is presented in appendix A, with worked example in appendices B and C, respectively, for both assignment reduction in an instance of A and constraint propagation in an instance of D methods.

## 5 Algorithms

### 5.1 Algorithms Determining Existence of Satisfying Assignment

#### 5.1.1 By Assignment Reduction in A

The method for producing the set of satisfying assignments, where such exist, for any given 3SAT expression,  $E(V, C)$ , by assignment reduction in an instance of A, is to iteratively reduce the set of all assignments to the variables of the expression by imposition of the constraints defined by clauses of the expression.

##### 5.1.1.1 Process

Given a CNF expression  $E(V_E, C_E)$  as by convention defined to be 3SAT:

1. *Create the corresponding structure D, and;*
2. *Create an instance of A where  $q = '11111111'$  (1-string) in each element based on variables of E.*
3. *For each element of D in indexed order: For each constraint in m component of the respective element of D: Remove from the corresponding element of A the assignment disallowed by the constraint, and; perform constraint propagation as defined by reduce operation, as latter is defined below (5.3). Halt if  $q = '00000000'$  in any element of A.*
4. *If  $q = '00000000'$  for any element of A then E has no satisfying assignment, else; if there exists in A no element where  $q = '00000000'$  then E has at least one satisfying assignment and A contains the set of satisfying assignments of E.*

##### 5.1.1.2 Runtime

The structure D requires creation of elements with a maximum upper bound of  $|V|^3$ . If one creates an instance of D that contains all elements admitted by P and inserts clauses of E into the corresponding element of D then such a process can complete within  $O(|V|^3)$  time.

The structure A being of the same number of elements as P in all cases, creating an instance of A and removing constrained assignments can be completed within  $O(|V|^3)$  time, as well.

If one chooses to first create an instance of P then use such to create instance of D and A the creation of P can similarly be completed within  $O(|V|^3)$  time.

Each of the structures are created sequentially resulting an upper bound on runtime for creating the structures that is  $O(|V|^3)$ . The space requirement bound above by  $O(|V|^3)$  as well.

Step-3 is a single *for-loop* over elements of D. Bounded by the number of elements in D and the number of clauses contained in each element of D, iteration of the *for-loop* is constrained above by  $(8 \times |V|^3)$ . The reduce operator, as derived in analysis of it, has a worst case upper bound on runtime of  $O(|V|^9)$ , giving a worst case upper bound for step-3 of  $O(|V|^{12})$ .

### 5.1.1.3 Correctness

The correctness of result stands on the arguments by contradiction in the discussion section. If one assumes that A contains a binary string that is not a satisfying assignment then such implies that there exists at least one element in A that contains an assignment,  $a_j$ , not satisfying of at least one clause in E. Since all assignments disallowed by assignments to any 3 variables on which a clause in E is defined the assignment at issue must be excluded by a derived constraint. The existence of such an assignment implies that there exists at least one assignment  $a_i$  disallowed by clause or clauses of E, and;  $a_i$  constrains variables in common with  $a_j$  such that  $a_j$  is inconsistent with such constraint. Existence of such an instance of  $a_j$  contradicts the terminating condition for constraint propagation.

Similarly, assume there to exist a satisfying assignment to E that is not in the set of binary strings represented by A at completion of processing. Such implies that there exists at least one element in A that does not contain an assignment,  $a_j$ , that should be present. The absence of  $a_j$  implies it was removed either because it was non-satisfying of one or more clauses of E, or;  $a_j$  was inconsistent with at least one constraint on 1 or 2 variables derived from satisfying assignments to all clauses of E. The first case contradicts removal from A of only assignments to 3 variables that are not satisfying of corresponding clauses of E. The second case contradicts definition of derived constraints and their propagation.

## 5.1.2 By Constraint Propagation in D

The method for producing the set of satisfying assignments, where such exist, for any given 3SAT expression,  $E(V, C)$ , by constraint propagation in an instance of D, is to insert into an initial instance of D representation of corresponding clauses of E, and; to then iteratively derive and propagate all derivative constraints, arising in consequence of presence of the clauses of E, over all elements of the respective instance of D; not just those elements containing clausal constraints of C, as was case in algorithm of [5].

### 5.1.2.1 Process

Given a CNF expression  $E(V_E, C_E)$  as by convention defined to be 3SAT:

1. Create an instance of P for the variables of V where  $|p|=3$  for all p in P, and;
2. Create an instance of D in which m='00000000' (0-string) for all elements, and;

3. For each clause  $c$  in  $C_E$  insert  $c$  into the corresponding element of  $D$ , and;
4. For each element  $d_i$  in  $D$ , into which clauses were inserted, derive the set of constraints to all subsets of variables of the corresponding  $p$  component of  $d_i$ .
5. For each element  $d_j$  in  $D$  impose those constraints derived in step-7 where  $d_j$  contains the respective variables in  $p$ .
6. If any element of  $D$  contains  $q='11111111'$  (1-string) then halt,  $E$  has no satisfying assignments and  $D$  is a certificate of non-satisfiability, else; if no constraints were added to any element of  $D$  then halt,  $E$  has satisfying assignments and  $D$  is a certificate of satisfiability, else; continue to step-7.
7. For each element  $d_i$  in  $D$ : derive the set of constraints to all subsets of variables of the corresponding  $p$  component of  $d_i$ .
8. For each element  $d_j$  in  $D$ : impose those constraints derived in step-7 where  $d_j$  contains the respective variables in  $p$ .
9. Continue at step-6.

### 5.1.2.2 Runtime

Steps 1 through 5, 7 and 8, are executed sequentially, and each has as upper bound the number of 3 variable subsets of variable set  $V$ , which is  $O(|V|^3)$ .

Step-9 effectively implements a *while-loop* with step-6 over steps 7 and 8. The while-loop process produces a strictly monotonic increasing sequence in the numbers of constraints present in the respective instance of  $D$ , and a strictly monotonic decreasing sequence in the number of absent constraints. Both sequences are constrained by upper bound on the total number of constraints that can be present in the respective instance of  $D$ , which is a constant times  $|V|^3$ . Step-9 iterating over steps 7 and 8 thus has a runtime bounded above by  $O(|V|^6)$ .

The process worst case runtime is dominated by the runtime of the while-loop construct of steps 6 through 9, making the worst case runtime for the process  $O(|V|^6)$ .

### 5.1.2.3 Correctness

The correctness of the process follows from the correctness of assignment reduction process. Any instance of  $D$  produced by constraint propagation can be transformed to an instance of  $A$ . If one assumes the resulting  $A$  to contain an assignment to variables of  $V$  that is not satisfying of  $E$  then  $A$  must contain an assignment in some element that is not satisfying of at least one clause in  $E$  on the corresponding variables, contradicting the resolution of derived constraints across all elements of  $D$ . If one assumes an assignment to variables of  $V$  exists that is satisfying of  $E$  where  $A$  does not contain the assignment then there exists at least one element of  $A$  for which an assignment is absent. The latter implies existence of a constraint not admitted by either directly by clausal constraints or by derived constraints on single variables and variable

pairs, contradicting D containing only clausal constraints and 1 and 2 variable constraints derived from the clausal constraints.

### 5.1.3 By Iteration on P

Iteration on P is simply another form of constraint propagation. The each individual assignment to an element p of P when evaluated against the constraints present in elements of D is either consistent with constraints to the intersection of variables of p and the respective element of D or is inconsistent. If the assignment to any given p under evaluation is inconsistent then the assignment must be removed from A or its constraint added to D, depending on which is being used for accounting. An empty set of assignments or full set of constraints is certificate of non-satisfiability. If algorithm runs to completion without producing an empty set of assignments or full set of constraints in any element of the respective structure then the 3SAT expression has a satisfying assignment.

The method is simply a minimalist derivation of reduction and extension methods of two algorithms presented above.

#### 5.1.3.1 Process

*Note: This process can use either the A or D structure to account for derived assignment constraints. The algorithm as presented uses D to account for derived constraints.*

Given a CNF expression  $E(V_E, C_E)$  as by convention defined to be 3SAT:

1. Create an instance of P for the variables of V where  $|p|=3$  for all p in P, and;
2. Create an instance of D in which m='00000000' (0-string) for all elements, and;
3. Insert into D the clauses of E, and;
4. For each p element in P, in index order: For each admissible assignment s to p: For each d in D where d is contains one or more variables in p:
  - a. evaluate<sup>1</sup> if s is consistent with assignment constraints imposed by d on the respective subset of p, and;
  - b. if s is inconsistent with the assignment constraints imposed by d then insert the constraint corresponding to s into D, and;
  - c. if m = '11111111' (1-string) in any element of D then halt, E has no satisfying assignment.
5. If there exists in D an element where m = '11111111' (1-string) then E has no satisfying assignment, and D is a certificate of non-satisfiability, else; if a derived constraint was

---

<sup>1</sup> Each element of D admits some set of assignments that determine assignment constraints. For example if an element d is satisfied by '001' and '011' then such constrain the respective variables to assignments {0}, {0,1}, {1}, {00, 01}, {01}, {01,11}, respectively. Any assignment evaluated against subsets of the respective variables must be consistent with such assignment constraints, as opposed to evaluation against clausal constraints.

*inserted into D then continue at step-4, else; the instance of A corresponding to D contains the representation of the set of assignments satisfying of E.*

### 5.1.3.2 Runtime

The algorithm iterates over elements of the set P which has a maximum upper bound on its size of  $|V|^3$ . In each such iteration it iterates over the set of feasible assignments to elements of P, which are constant in relation to  $|V|$ , and; for each feasible assignment iterates over elements of the elements of D. The elements of D have as maximum upper bound  $|V|^3$ . The runtime of the process for step-3 therefore has a worst case upper bound of  $O(|V|^6)$ . The effective *do-while* loop of conditional at step-5 results in step-4 executing at most  $|V|^3$  time. The algorithm thus has a maximum upper bound of  $O(|V|^9)$  on its runtime.

The algorithm can be made to run to termination quicker, particularly for instances of E that have no satisfying assignments, by including in P subsets of 1 and 2 variables, and including in D the corresponding elements. Assignment constraints on 1 and 2 variable elements propagate to those elements and may propagate earlier across elements of D, if such do not cause there to exist an element with 1 or 2 variables that has an m component containing a 1-string earlier in processing.

### 5.1.3.3 Correctness

The process propagates constraints across D by the evaluation of assignment constraints on subsets of variables in each element of D that contains the respective subset. The correctness follows directly from the definition of satisfiability of the 3SAT expression. If there does not exist a satisfying assignment to some subset of V then there is no assignment satisfying of E.

## 5.2 Algorithm Determining a Satisfying Assignment

It is important to note that absent A containing an *a priori* known set of Boolean sequences (binary strings) and consistent constraints on the variables the algorithm described here is not assured to reliably produce a correct result.

### 5.2.1 Process

Given an instance of A:

- 1) *For each element  $a_i$  of A in indexed order: For one assignment in  $a_i$ ; Eliminate from  $a_i$  all other assignments, and; perform constraint propagation as defined by reduce operation, as latter is defined below (5.3).*

### 5.2.2 Runtime

The process is a single *for-loop* iterating over elements of an instance of A, executing the reduce operator once in each iteration. The process consequently has a worst case upper bound of  $O(|V|^{12})$ .

### 5.2.3 Correctness

The process is an abbreviated version of assignment reduction in which the assignments in each element of an instance of A are reduced in sorted order by elimination of all but one assignment, with reduction of assignments in remaining elements of A. The correctness of results follows directly from correctness of assignment reduction.

## 5.3 Algorithm of Reduce Operator

The *reduce* operator recursively propagates constraints within elements of an instance of A on individual and multiple variables to all elements of the instance of A to derive consistency of assignment constraints within the instance of A. Note that the operation as described here is not optimized.

### 5.3.1 Process

Given an instance of A:

- 1) For each element  $a_i$  in A in indexed order: For each variable in  $p_{ai}$  and pair of variables in  $p_{ai}$  determine respective constraints:
  - a. For each element  $a_j$  in A in indexed order,  $j$  not equal to  $i$ , where  $a_j$  contains a variable or pair of variables, remove from  $a_j$  any assignments inconsistent with constraints obtained from  $a_i$ .
- 2) If any elements of A contain a q='00000000' (0-string), return an empty instance of A, else; if any assignment was removed in step 2 then continue at step 1, else; if no assignment was removed then return A.

### 5.3.2 Runtime

Step-1 consists of three nested *for-loops*. The outer loop iterates over the elements of an instance of A, the first nested for-loop iterates over the set of assignments in each element of the instance of A, and the inner for-loop iterates over the elements of the instance of A

The outer and inner *for-loop* iterations have a maximum upper bound of  $|V|^3$ , the upper bound on number of elements in A. The first nested *for-loop* will iterate at most 8 times for each element in A.

Step-1 thus has a runtime upper bound of  $(8 \times (|V|^3)^2) \rightarrow O(|V|^6)$  in each execution.

Step-2 then determines the runtime of the reduce operation, as it controls iterative execution of step-1 in what is an effective *do-until* loop construct. This loop construct produces a strictly monotonic decreasing sequence of the total number of assignments in all elements of the respective instance of A. It can therefore cause the execution of step-1 at most  $(8 \times (|V|^3))$  times, in the worst case.

The reduce operator, as given, therefore has a worst case runtime upper bound of  $O(|V|^9)$ .

### 5.3.3 Correctness

The reduction process propagates assignment constraints within each element of an instance of A to all other elements of A, reducing the set of assignments in the instance of A to a consistent state or empty state. The correctness of the process is established by the same argument by contradictions by which assignment reduction process correction is argued correct.

## 6 Concluding Comments

A satisfying assignment to a SAT expression,  $E(V,C)$  is determined by the constraints to which all subsets of variables V are subject. The set of clauses in C does not have to contain all of such constraints. The set of constraints can be obtained by treating the set of clauses as defining of a ‘basis’ like set of constraints from which, by constraint propagation, the set of constraints to all subsets of variables can be obtained. The resulting set of constraints, or the corresponding set of admissible assignments, determines if E has or does not have a satisfying assignment. The set of constraints or assignments stands as certificate of satisfiability or non-satisfiability, determined by the presence or absence of empty set of assignments or full set of constraints to any element of the representative structure.

The algorithms presented are suboptimal sequential processing variant of what was performed by a parallel processing system of 8-bit nodes, with nodes for single and pairs of variables in the set P. The network of nodes executed in the manner of a Cellular Automata with all nodes communicating with adjacent nodes at the same time. The parallel configuration allowed the communication of constraints on single and pairs of variables from all elements of A, as defined for the presented algorithm, to the respective node for the single or pair of variables in one step, and subsequent retrieval of resulting constraints in one step. Space in parallel nodes can be traded off against runtime by virtualization of nodes on an appropriate cellular computing fabric.

## 7 References

- [1] “Boolean Satisfiability Solvers: Techniques and extensions”; G. Weissenbacher, S. Malik; Software Safety and Security – Tools for Analysis and Verification, ppg 205-253; Nipkow, Grumberg, Hauptmann Editors; IOS Press, 2012.
- [2] “A survey of SAT Solver”; Weiwei Gong, Xu Zhou; AIP Conference Proceedings, 1836(1); AIP Publishing, 2017.
- [3] “Handbook of Satisfiability”; Armin Biere, et al, editors; IOS Press, 2009.
- [4] “The Complexity of Theorem-Proving Procedures”; Stephen Cook; Proceedings 3rd Annual ACM Symposium on the Theory of Computing, ppg 151-198; Association for Computing Machinery, New York, 1971.

[5] “A polynomial time (heuristic) SAT algorithm”; C Sauerbier;  
<https://arxiv.org/abs/cs/0205064v3>; 2002.

## 8 Appendices

### A. Example Expression

Define  $V = \{ v_0, v_1, v_2, v_3, v_4 \}$

Define  $C =$

```
{  
  (-v0, -v1, -v4), (v0, v1, -v4), (-v0, -v1, v4), (v0, -v1, v4), (-v0, v1, v4), (v0, v1, v4),  
  (-v0, -v2, -v3), (v0, -v2, -v3), (-v0, v2, -v3), (v0, v2, -v3), (v0, -v2, v3), (-v0, v2, v3),  
  (-v1, -v2, -v3), (v1, -v2, -v3), (-v1, v2, -v3), (v1, v2, -v3), (v1, -v2, v3), (-v1, v2, v3),  
  (-v2, -v3, -v4), (v2, -v3, -v4), (-v2, -v3, v4), (v2, -v3, v4), (-v2, v3, v4), (v2, v3, v4)  
}
```

Define  $E(V, C)$  to be an instance of 3SAT.

In the case above the respective admitted assignments to the subsets of variables are  $\{ (-v_0, v_1, v_4), (v_0, -v_1, v_4), (v_0, v_2, -v_3), (-v_0, -v_2, -v_3), (-v_1, -v_2, -v_3), (v_1, v_2, -v_3), (-v_2, -v_3, -v_4), (v_2, v_3, -v_4) \}$ .

3SAT instances such as the above produce a set of assignments that produce a false support in the subset of elements of A corresponding to the subset of elements of D in which clauses of E are defined; that is to exclude all elements of D and of A for which no clause in E is defined on the respective variables of p component. The false support arises from a chain of resolution extending from any one allowed assignment in an element of the subset of A to the other assignment in the same element. Such a chain of resolution arises from a cross-over occurring in one or more of the corresponding elements of the subset of A. Constraint propagation on just the subset of D for which clauses in E are defined on the respective variables of the p component will result in no addition of derivative constraints.

Variations on the construct can be defined to produce reduction in the assignments, and addition of constraints, in the restricted subset of elements of structures A and D, respectively. Application of the processes to the restricted subset of elements will settle to a steady state that fails to produce an empty set of assignments in A or complete set of constraints in D.

Determination of non-satisfiability, for the example above and instances containing similar structure producing false support, produce the respective certificate of non-satisfiability in consequence of elimination of all assignments in an element of A or insertion of all constraints

into an element of D other than one in the restricted subset of elements corresponding to defined clauses of E.

## B. Assignment Reduction on Instance of A

The case is correctly reduced by assignment reduction process, as presented in Section 5, as the constraints in combination result in elements of the corresponding A structure containing q='00000000' (0-string). A 0-string in q component of any element of A is representative of an empty set of assignments to the corresponding subset of variables.

*Figure B-1*, below, presents an initial instance of structure A for the example expression of Appendix A.

		A										
		p	012	013	014	023	024	034	123	124	134	234
q	000	x	x	x	x	x	x	x	x	x	x	
	100	x	x	x	x	x	x	x	x	x	x	
	010	x	x	x	x	x	x	x	x	x	x	
	110	x	x	x	x	x	x	x	x	x	x	
	001	x	x	x	x	x	x	x	x	x	x	
	101	x	x	x	x	x	x	x	x	x	x	
	011	x	x	x	x	x	x	x	x	x	x	
	111	x	x	x	x	x	x	x	x	x	x	

*Figure B-1: 'A' structure for example expression*

*Figure B-2*, below presents the structure A subsequent to imposition of constraints of CE on variables ( $v_0, v_1, v_4$ ) in (a) and the structure subsequent to resolution of derivative constraints by propagation in (b). The derived constraints are  $(v_4) = \{1\}$ ,  $(v_0, v_1) = \{01, 10\}$ ,  $(v_0, v_4) = \{01, 11\}$ ,  $(v_1, v_4) = \{01, 11\}$ .

		A										
		p	012	013	014	023	024	034	123	124	134	234
q	000	x	x		x	x	x	x	x	x	x	
	100	x	x		x	x	x	x	x	x	x	
	010	x	x		x	x	x	x	x	x	x	
	110	x	x		x	x	x	x	x	x	x	
	001	x	x		x	x	x	x	x	x	x	
	101	x	x	x	x	x	x	x	x	x	x	
	011	x	x	x	x	x	x	x	x	x	x	
	111	x	x	x	x	x	x	x	x	x	x	

(a)  $(v_0, v_1, v_4)$  Clausal constraints imposed
(b)  $(v_0, v_1, v_4)$  Derived constraints resolved

*Figure B-2: 'A' structure with  $(v_0, v_1, v_4)$  constraints*

*Figure B-3*, below presents the structure A subsequent to imposition of constraints of CE on variables ( $v_0, v_2, v_3$ ) in (a) and the structure subsequent to resolution of derivative constraints

by propagation in (b). The derived constraints are  $(v_3) = \{0\}$ ,  $(v_0, v_2) = \{00, 11\}$ ,  $(v_0, v_3) = \{00, 10\}$ ,  $(v_2, v_3) = \{00, 10\}$ .

		A											
		p	012	013	014	023	024	034	123	124	134	234	
q	000				x			x					
	100	x	x		x			x					
	010	x	x			x		x					
	110			x		x		x					
	001				x	x	x	x	x	x	x	x	
	101	x	x	x	x	x	x	x	x	x	x	x	
	011	x	x	x	x	x	x	x	x	x	x	x	
	111				x	x	x	x	x	x	x	x	

(a)  $(v_0, v_2, v_3)$  Clausal constraints imposed
(b)  $(v_0, v_2, v_3)$  Derived constraints resolved

Figure B-3: 'A' structure with  $(v_0, v_1, v_4)$  and  $(v_0, v_2, v_3)$  constraints

Figure B-4, below presents the structure A subsequent to imposition of constraints of CE on variables  $(v_1, v_2, v_3)$  in (a) and the structure subsequent to resolution of derivative constraints by propagation in (b). The derived constraints are  $(v_3) = \{0\}$ ,  $(v_1, v_2) = \{00, 11\}$ ,  $(v_1, v_3) = \{00, 10\}$ ,  $(v_2, v_3) = \{00, 10\}$ .

It is worth noting the consistency of derived constraints to 1 and 2 variable subsets as each set of clauses of the expression is imposed on the respective element of A. The subset of variables that is unique in the derived constraints extending the reduction of assignments.

		A											
		p	012	013	014	023	024	034	123	124	134	234	
q	000				x			x					
	100	x			x								
	010	x	x		x			x					
	110			x		x		x					
	001				x	x	x	x	x	x	x	x	
	101	x		x	x	x	x	x	x	x	x	x	
	011		x	x	x	x	x	x	x	x	x	x	
	111				x			x					

(a)  $(v_1, v_2, v_3)$  Clausal constraints imposed
(b)  $(v_1, v_2, v_3)$  Derived constraints resolved

Figure B-4: 'A' structure with  $(v_0, v_1, v_4)$ ,  $(v_0, v_2, v_3)$  and  $(v_1, v_2, v_3)$  constraints

The imposition and resolution of constraints for  $(v_0, v_1, v_4)$ ,  $(v_0, v_2, v_3)$  and  $(v_1, v_2, v_3)$  in the context of all elements of A results in the element for variables  $(v_0, v_1, v_2)$  containing an empty assignment set, as observable in Figure B-4(b), producing a certificate of non-satisfiability.

### C. Constraint Propagation on Instance of D

Figure C-1, below, provides an illustration of derivative constraints for 1 and 2 variable subsets for clauses of the example expression. Figure C-2, below, illustrates in (a) the initial D structure for the example expression, and in (b) insertion of clausal constraints of example expression E.

	D																													
p	012			013			014			023			024			034			123			124			134					
	0	1	2	0	1	3	0	1	4	0	2	3	0	2	4	0	3	4	1	2	3	1	2	4	1	3	4			
q	000	-	-	-	-	-	0	0	0	0	0	0	-	-	-	0	0	0	-	-	-	-	-	-	0	0	0			
	100	-	-	-	-	-	1	0	0	1	0	0	-	-	-	1	0	0	-	-	-	-	-	-	1	0	0			
	010	-	-	-	-	-	0	1	0	0	1	0	-	-	-	0	1	0	-	-	-	-	-	-	0	1	0			
	110	-	-	-	-	-	1	1	0	1	1	0	-	-	-	1	1	0	-	-	-	-	-	-	1	1	0			
	001	-	-	-	-	-	0	0	1	0	0	1	-	-	-	0	0	1	-	-	-	-	-	-	0	0	1			
	101	-	-	-	-	-	1	0	1	1	0	1	-	-	-	1	0	1	-	-	-	-	-	-	1	0	1			
	011	-	-	-	-	-	0	1	1	0	1	1	-	-	-	0	1	1	-	-	-	-	-	-	0	1	1			
	111	-	-	-	-	-	1	1	1	1	1	1	-	-	-	1	1	1	-	-	-	-	-	-	1	1	1			
(a) 1 variable subsets																														
	D	012			013			014			023			024			034													
p	01	02	12	01	03	13	01	04	14	02	03	23	02	04	24	03	04	34	12	13	23	12	14	24	13	14	34	23	24	34
	000	-	-	-	-	-	-	00	00	00	00	00	00	-	-	-	00	00	00	-	-	-	-	-	-	00	00	00	00	00
q	100	-	-	-	-	-	10	10	00	10	10	00	-	-	-	10	10	00	-	-	-	-	-	-	10	10	00	10	10	00
	010	-	-	-	-	-	01	00	10	01	00	10	-	-	-	01	00	10	-	-	-	-	-	-	01	00	10	11	10	10
	110	-	-	-	-	-	11	10	10	11	10	10	-	-	-	11	10	10	-	-	-	-	-	-	11	10	10	11	11	10
	001	-	-	-	-	-	00	01	01	00	01	01	-	-	-	00	01	01	-	-	-	-	-	-	00	01	01	01	01	01
	101	-	-	-	-	-	10	11	01	10	11	01	-	-	-	10	11	01	-	-	-	-	-	-	10	11	01	11	11	01
	011	-	-	-	-	-	01	01	11	01	01	11	-	-	-	01	01	11	-	-	-	-	-	-	01	01	11	11	11	11
	111	-	-	-	-	-	11	11	11	11	11	11	-	-	-	11	11	11	-	-	-	-	-	-	11	11	11	11	11	11
	(b) 2 variable subsets																													
<i>Figure C-1: Derived constraints illustration</i>																														

	D											
m	p	012	013	014	023	024	034	123	124	134	234	
	000	-	-	-	-	-	-	-	-	-	-	
100	-	-	-	-	-	-	-	-	-	-	-	
010	-	-	-	-	-	-	-	-	-	-	-	
110	-	-	-	-	-	-	-	-	-	-	-	
001	-	-	-	-	-	-	-	-	-	-	-	
101	-	-	-	-	-	-	-	-	-	-	-	
011	-	-	-	-	-	-	-	-	-	-	-	
111	-	-	-	-	-	-	-	-	-	-	-	
(a) Initial D structure												
	D											
m	p	012	013	014	023	024	034	123	124	134	234	
	000	-	-	-	x	x	-	-	x	-	x	
100	-	-	-	-	-	-	-	-	x	-	x	
010	-	-	-	-	-	-	-	-	x	-	x	
110	-	-	-	x	x	-	-	-	x	-	-	
001	-	-	-	-	x	-	-	-	-	-	x	
101	-	-	-	x	x	-	-	-	x	-	x	
011	-	-	-	x	x	-	-	-	x	-	x	
111	-	-	-	x	-	-	-	-	-	-	x	
(b) Clauses of E inserted into D structure												
<i>Figure C-2: 'D' structure for example expression</i>												

Figure C-3, below, illustrates D with clausal constraints inserted with propagation of 1 variable derived constraints in (a), and; with propagation of 2 variable derived constraints in (b).

		D										
		p	012	013	014	023	024	034	123	124	134	234
m	000		x	x	x		x		x	x		
	100		x		x		x		x	x		
	010		x		x		x		x	x		
	110		x	x	x		x		x	x		
	001		x		x	x		x	x	x	x	
	101		x	x	x	x	x	x	x	x	x	
	011		x	x	x	x	x	x	x	x	x	
	111		x		x	x		x	x	x	x	

  

		D										
		p	012	013	014	023	024	034	123	124	134	234
m	000		x	x	x	x		x	x	x	x	
	100		x	x	x	x		x	x	x	x	
	010		x	x	x	x		x	x	x	x	
	110		x	x	x	x		x	x	x	x	
	001		x	x	x	x		x	x	x	x	
	101		x	x	x	x		x	x	x	x	
	011		x	x	x	x		x	x	x	x	
	111		x	x	x	x		x	x	x	x	

(a) 1 variable constraints propagated

(b) 1 and 2 variable constraints propagated

Figure C-3: Derived constraints propagated

For the example expression the propagation of derived constraints on 1 variable was sufficient to produce a complete set of constraints in element for variables ( $v_0, v_3, v_4$ ). The subsequent propagation of derived constraints on 2 variables produced a further complete set of constraints on variables ( $v_0, v_1, v_2$ ), consistent with the assignment reduction method for example expression presented in [Appendix B](#).

Both [Figure C-3 \(a\)](#) and [\(b\)](#) are certificates of non-satisfiability for the example expression.

#### D. Iteration on P

The set of assignments is correctly reduced, or the set of constraints correctly extended, by the process of iteration over P and corresponding set of assignments to each element of P. The detail of evaluation of an assignment s to an element p is presented here.

[Figure D-1\(a\)](#), below, is the structure D containing clausal constraints of example expression E. It is copied from [Figure C-2\(b\)](#), above.

Consider the variable set ( $v_0, v_1, v_2$ ). Starting with assignment  $s = '000'$  one can observe that constraints of the element corresponding to ( $v_0, v_1, v_4$ ) can be satisfied subject to the constraint  $v_4 = 0$  or  $v_4 = 1$ , and not both. The absent constraints {'100', '010'} however dictate that the satisfying assignment be constrained to {'011', '101'}; thus, constraining  $v_4 = 1$  and ( $v_0, v_1$ ) an element of {'10', '01'}. The assignment  $s = '000'$  cannot therefore satisfy constraints on ( $v_0, v_1, v_4$ ). The implementation has the option of working on structure A, eliminating '000' from q for element corresponding to ( $v_0, v_1, v_2$ ), or; operating on structure D, adding derived constraint '111' to the element corresponding to ( $v_0, v_1, v_2$ ).

One also has the option to perform accounting disallowed assignments or derived constraints on 1 and 2 variable subsets, in which case the assignment that  $v_4 = '0'$  would be removed or constraint  $v_4 = '1'$  added to the respective structure. Alternatively one simply removes any assignment inconsistent with the derived constraint on  $v_4$  in A, or adds any derived constraint in D.

The remaining assignments of  $(v_0, v_1, v_2)$  will process similarly with those inconsistent with assignment to  $(v_0, v_1)$  from {'10', '01'} failing to satisfy constraints of the respective element for  $(v_0, v_1, v_4)$  in D.

Consider the variable set  $(v_0, v_1, v_2)$ . Starting with first assignment  $s = '100'$  one can observe that constraints of the element corresponding to  $(v_0, v_2, v_3)$  can be satisfied subject to the constraint  $v_3 = 0$  or  $v_3 = 1$ , and not both. The absent constraints {'001', '111'} however dictate that the satisfying assignment be constrained to {'110', '000'}; thus, constraining  $v_3 = 0$  and  $(v_0, v_2)$  to an element of {'00', '11'}. Assignment  $s = '100'$  cannot therefore satisfy constraints on  $(v_0, v_2, v_3)$ . Evaluating assignments {'010', '101', '011'} to  $(v_0, v_1, v_2)$  on  $(v_0, v_2, v_3)$  results in derived constraints {'011', '101', '010', '100'}, respectively, being imposed on  $(v_0, v_2, v_3)$ .

The result is a full set of constraints,  $m = '11111111'$ , on element corresponding to  $(v_0, v_1, v_2)$  in D, or; an empty set,  $q = '00000000'$ , of assignments in corresponding A.

Figure D-1(b), below, illustrates structure D with derived constraints resulting from evaluation of assignments for  $(v_0, v_1, v_2)$  on  $(v_0, v_1, v_4)$  and  $(v_0, v_2, v_3)$  inserted. The element of D corresponding to  $(v_0, v_1, v_2)$  has a complete set of constraints and D serves as a certificate of non-satisfiability of E.

		D											
		p	012	013	014	023	024	034	123	124	134	234	
m	000			x	x			x			x		
	100			x			x			x			
	010			x			x						
	110		x	x			x						
	001	x							x				
	101	x	x			x			x				
	011	x	x			x			x				
	111	x							x				

  

		D											
		p	012	013	014	023	024	034	123	124	134	234	
m	000	x	x	x	x			x	x	x	x		
	100	x	x			x		x	x	x	x		
	010	x	x			x			x				
	110	x	x	x	x				x				
	001	x			x		x	x	x	x	x		
	101	x		x	x	x	x	x	x	x	x		
	011	x		x	x	x	x	x	x	x	x		
	111	x			x		x	x	x	x	x		

(a) Clauses of E inserted into D structure

(b)  $(v_0, v_1, v_2)$  on  $(v_0, v_1, v_4)$  and  $(v_0, v_2, v_3)$

Figure D-1: Iteration on P for example expression, accounting derived constraints