Journal of Indian Acad. Math.
Vol. 31, No. 2 (2009) pp. 407-444

Narendra S. Chaudhari

COMPUTATIONALLY HARD PROBLEMS: 3-SAT AND ITS POLYNOMIAL SOLVABILITY

Abstract : 3-SAT problem is about the discovery of truth-value assignments of variables that result in the satisfaction of all clauses. To model this high level description of the problem, we first formulate the *conditional* with a clause (in a given 3-SAT formula) as the antecedent, and the literal(s) restricted by this clause as its consequent. We develop a framework for representing and analyzing these conditionals. To begin our discovery of restrictions on the truth values, we represent 3-Clause as a *conditional* having a literal-pair as antecedent, and single literal as consequent. In addition to the exploitation of this conditional, our framework involves the propagation of restricted literals (also called as *stable literals*). Discovery of the restricted literals makes use of the concept of the *logical consequents* of a literal pair, the concept central to our main formulation, and we give an algorithm to determine the same.

The analysis used in our algorithm involves maximum three propagation levels of stable literals. In our formulation, at a particular propagation level, we do not allow the flipping the truth-value of a literal; this feature is important for us to demonstrate the polynomial bound on the number of operations. We give simplified analysis to demonstrate that our algorithm has a polynomial bound of $O(n^{13})$ operations. Since 3-SAT is NP-Complete, our algorithm also implies the polynomial solvability of all NP-Complete problems.

Keywords : Theory of Computation: Complexity Measures and Classes. Complexity Hierarchies. Algorithms: Analysis of Algorithms.

1. Introduction

Various problems like Integer Linear Programming (LP), Subgraph Isomorphism, Traveling Salesman problem (TSP), Graph Coloring problem, etc., have resisted discovery of polynomial algorithms for last few centuries. Computationally, discovery of polynomial algorithm for these algorithms is equivalent to constructing polynomial algorithm any one of the problems that are included in the class of NP-Complete problems [1]. The discovery of polynomial algorithm for an NP-Complete problem has been one of the seven greatest mathematical problems of the Twentieth Century as announced by the Clay Mathematics Institute. 3-SAT has been one of the first known NP-Complete problems [1].

This paper announces the unexpected settlement of the famous question (known as P verses NP) by giving a concrete polynomial algorithm to 3-SAT [1]. 3-SAT is computationally known to be different than 2-SAT. In fact, the approach of collecting the logical consequents to transitively close them to discover contradiction works for 2-SAT [1]. However, for 3-SAT, such an approach does not work. Many existing approaches like DPLL [2] and its variants [3, 4, 5] take care of the propagation of truth-value assignments in the *forward* direction (this propagation is similar to the transitive closure that works for 2-SAT), and they also include some mechanisms to take care of backward direction; however, they remain 'incomplete' for capturing the global effect (of collective clause satisfaction). This paper reports one systematic approach to tack care of this backward direction as well.

Our approach involves (the formulation of) two main concepts. First is the construction of logical consequents of *all* antecedent *literal-pairs* as our analysis progresses. Second is the demonstration that, while the forward propagation of the global effect is captured in our suitably formulated conditional, the (corresponding) *backward* propagation of the global effect is captured by enforcing the satisfaction of the contrapositive of the concerned conditional. In the remaining part of this section, we give a few basic mathematical formulations needed for our construction.

Let $\mathcal{F}$ be 3-SAT formula. It consists of 3-clauses. A 3-clause has an equivalent representation with a literal-pair as an antecedent and the remaining literal as a consequent, *i.e.*,

$$(p \vee q \vee r) \equiv (\bar{p} \wedge \bar{q}) \rightarrow r (\equiv (\bar{q} \wedge \bar{p}) \rightarrow r). \tag{1}$$

Let us call this representation of a clause as ***antecedent literal-pair*** representation. It is depicted in Figure 1.

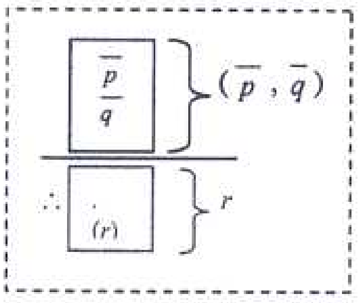

Fig. 1. A representation of 3-clause $(p \vee q \vee r)$: consisting of (i) antecendent literal-pair $(\bar{p}, \bar{q})$ and (ii) the consequent r.

For a clause $(p \vee q \vee r)$, we define the notion of clause consequent of literal-pair. In the clause $(p \vee q \vee r)$, the ***clause consequent*** of literal pair $(\bar{p}, \bar{q})$ is r; we denote this by using the notation $C(\bar{p}, \bar{q}) = r$. We also use an equivalent suffix notation $C_{\bar{p},\bar{q}}$ to denote $C(\bar{p}, \bar{q})$; thus, C, implemented as a two dimensional array (data structure), includes r in its location $(\bar{p}, \bar{q})$. We note that the other representations of $(p \vee q \vee r)$, namely, $(\bar{p} \wedge \bar{r}) \rightarrow q$ and $(\bar{q} \wedge \bar{r}) \rightarrow p$ also involve a literal-pair as the antecedent. For a clause $(p \vee q \vee r)$, we have six antecedent literal-pair representations; they are: (1) $(\bar{P} \wedge \bar{q}) \rightarrow r$, (2) $(\bar{p} \wedge \bar{r}) \rightarrow q$, (3) $(\bar{q} \wedge \bar{r}) \rightarrow p$, (4) $(\bar{q} \wedge \bar{p}) \rightarrow r$, (5) $(\bar{r} \wedge \bar{p}) \rightarrow q$, and, (6) $(\bar{r} \wedge \bar{q}) \rightarrow p$. For our formulation, we find it convenient to represent a clause by generating (and storing) all of them. We also note that $(p \vee q \vee r)$ has a *nested conditional* representation of a 3-clause, *i.e.*,

$$(p \vee q \vee r) \equiv (\bar{p} \rightarrow (\bar{q} \rightarrow r)). \tag{2}$$

This representation is depicted in Figure 2.

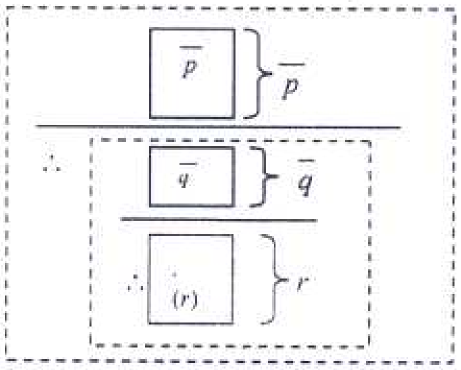

Fig. 2. Nested conditional representation of 3-clause $(p \vee q \vee r)$: consisting of two antecendent, $\bar{p}$ and $\bar{q}$.

We also note that, substitution of the contrapositive of inner conditional in Figure 2 allows us to obtain another representation of $(p \vee q \vee r)$, namely, $(\bar{p} \rightarrow (\bar{r} \rightarrow q))$, which can also be thought of as having the antecedent literal-pair representation $(\bar{p} \wedge \bar{r}) \rightarrow q$ (given in Figure 3).

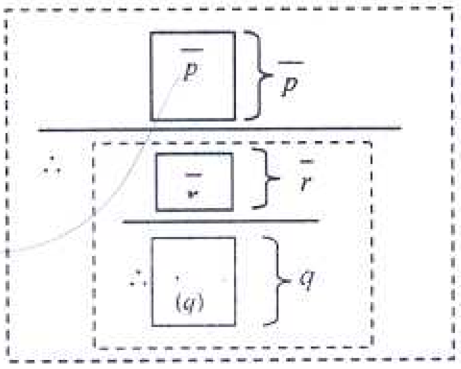

Fig. 3. Other nested conditional representation of 3-clause $(p \vee q \vee r)$: obtained by substituting the inner conditional $(\bar{q} \rightarrow \bar{r})$ in Figure 2 by its contrapositive $(\bar{r} \rightarrow q)$.

We briefly summarize our approach. The 3-SAT problem requires us to satisfy a clause, say C, in the context of other (satisfied) clauses. This constraint may result in the restriction of the truth-value assignment(s). The process of assigning satisfying truth-value assignments hints that we should do the computational analysis to obtain these restrictions, if any. Non-triviality of the 3-SAT problem is this phase (as this phase also discovers the non-satisfiability of $\mathcal{F}$). After taking care of the restriction of the truth-value assignments of variables, the assignment of truth-values can be effected by making sure that progress is made by collecting the satisfying literals.

The demonstration that 3-SAT formula is satisfiable requires us to obtain the truth-value assignments (for all variables), *i.e.*, it requires us to discover the true literals. In the previous paragraph, we already noted that satisfaction of a clause may result in the restriction of the truth-value assignment(s). Developing a systematic approach to investigate this high-level relationship between the clause satisfaction and the restriction of the truth-value assignment (which is the consequence of this clause satisfaction) of a variable is important part of our formulation.

In our analysis, to start with, although we do not know which clause satisfaction would restrict truth-value of which variable, we do know that we can model it by a *conditional*. This conditional forms the basis of our high-level model to start our analysis. Suppose C is a clause in a given 3-SAT formula, and satisfaction of the clause C, in the context of other (satisfied) clauses, results in the restriction of a truth-value assignment. Thus, the effect of satisfaction of clause C in the presence of satisfaction of all other clauses is in restricting the truth-value of some variable, say v_s; let us assume that it restricts the truth value of variable v_s to a literal s. In our analysis, we formulate this restriction as the conditional: $C \rightarrow s$.

We now elaborate some details of the process of formulating the conditional. We note that, in 3-SAT, all clauses can be thought of as having antecedent *literal-pair* representation. For our analysis, let us decide to leave out the clause C, and we collect all other clauses, represented using their antecedent *literal-pair* representations. As an example, the clause $(p \vee q \vee r)$ has six antecedent literal-pair representations. Satisfaction of a 3-clause $(p \vee q \vee r)$ requires that all these antecedent literal-pairs of all other clauses in $\mathcal{F}$ be true. For simplicity of our formulation, we choose to consider all of them.

Although antecedent *literal-pair* representations form the basis for representation of clauses, they are purely local to a clause. The high-level conditional $C \to s$ may restrict the literal s, which may not be locally related to C at all, because it is due to *global effect* of 'accumulation of the satisfaction of other clauses'. However, this formulation raises an important question: what computational analysis would allow us to discover this conditional? To formulate the basis of our computational steps, we consider the following.

The well-known inference rule, Modus Ponens (MP), reminds us that $C \wedge (C \to s) \Rightarrow s$. We briefly review the basis for the formulation of MP in proposition 1.

Proposition 1 : For Boolean variables p, q, we have:

$$p \wedge (p \to q) \text{ is logically equivalent to (denoted by } \Leftrightarrow \text{, or by } \equiv) p \wedge q. \tag{3}$$

Proof : $p \wedge (p \to q) \equiv p \wedge (\vee q) \equiv (p\wedge) \vee (p \wedge q) \equiv \mathbf{F} \vee (p \wedge q) \equiv p \wedge q.$

□

We repeat that, in our formulation, the conditional "$\to$" models the *effect of satisfaction* of C in the presence of *satisfaction of all other clauses*. Substituting p by C and q by s in Eq. (3), we note that $C \wedge (C \to s) \equiv C \wedge s$. Due to the logical equivalence (*i.e.*, the two-way implication), it is possible to capture the effect of application of the conditional "$\to$" by enforcing the (conjunctive) satisfaction of s; in other words, "$\to$" gets its compact representation in terms of "$\wedge$". Informally, we call such a literal s as *stable* literal, stabilized by a clause C (due to the effect of satisfaction of other clauses in $\mathcal{F}$).

Generation of the conditionals "$\to$" directs us to have a formulation of the closure operation. This closure operation is another very important step in our formulation, and we give its details in section III. Exploitation of the compact representation of 'the effect of satisfaction of C in the presence of satisfaction of all other clauses' in terms of stable literal(s) is the main feature in our formulation. Nested conditional representation (involving two levels) of a 3-clause hints us that we need to have a framework to propagate these stable literals to upper levels. This framework, included in section III, uses three levels that are based on the exploitation of the following three (simplifying) equivalences that are stated in proposition 2 below.

Proposition 2 : We have the following equivalences (in propositional logic):

(i) $((p \rightarrow s) \wedge (\bar{p} \rightarrow s)) \equiv s$

(ii) $((p \vee q) \wedge (p \rightarrow s) \wedge (q \rightarrow s)) \equiv (p \vee q) \wedge s$

(iii) $((p \vee q \vee r) \wedge (p \rightarrow s) \wedge (q \rightarrow s) \wedge (r \rightarrow s)) \equiv (p \vee q \vee r) \wedge s$

Proof : We prove (i) as follows:

Consider $((p \rightarrow s) \wedge (\bar{p} \rightarrow s)) \equiv ((\bar{p} \vee s) \wedge (p \vee s)) \equiv (((\bar{p} \wedge p) \vee s)) \equiv ((\mathbf{F} \vee s)) \equiv s$.

We give a sketch for the proof of (ii) below.

Consider $(p \rightarrow s) \wedge (q \rightarrow s) \equiv (\bar{p} \vee s) \wedge (\bar{q} \vee s) \equiv ((\bar{p} \wedge \bar{q}) \vee s) \equiv ((\overline{p \vee q}) \vee s)$.

Hence, $(p \vee q) \wedge (p \rightarrow s) \wedge (q \rightarrow s) \equiv ((p \vee q) \wedge ((\overline{p \vee q}) \vee s)) \equiv (\mathbf{F} \vee ((p \vee q) \wedge s)) \equiv (p \vee q) \wedge s$.

For (iii), we note that, $(p \rightarrow s) \wedge (q \rightarrow s) \wedge (r \rightarrow s) \equiv ((\overline{p \vee q \vee r}) \vee s)$, and rest or the proof is similar to (ii), and hence we omit it.

□

We note that $(p \vee \bar{p}) \equiv \mathbf{T}$ (a tautology); hence $(p \vee \bar{p})$ is a trivial clause in all SAT formulae, and hence we may also rewrite (i) as follows.

Proposition 2 (i') : $((p \vee \bar{p}) \wedge (p \rightarrow s) \wedge (\bar{p} \rightarrow s)) \equiv s$.

When the left hand side (LHS) conditions (for (i), (ii), or (iii) in proposition 3) are satisfied, we formally call literal s as *stable* literal. In the list of LHS conjunctive clauses, the conditionals involved form the basis of expanding our computational analysis. However, this is computational expansion is also *pinned down* to a local clause; this is because we note that, in the LHS conjunction, we also have a clause, e.g., 2-clause $(p \vee q)$ in (ii), or 3-Clause $(p \vee q \vee r)$ in (iii), and a tautology clause $(p \vee \bar{p})$ in (i). It is of special interest to note that, the LHS conditionals involve *only* single literals as their individual antecedents. We base our computational analysis (of stable literals) on suitable formulation and the *collective* analysis of these single-literal conditionals. As our computations proceed, we generate the suitable conditionals algorithmically; we also systematically keep track of their stable literals. In 3-Clause, we have two levels of conditionals (Figure 2); in addition to these two levels, we identify the third (top) level indicating globally restricted literals. Hence, we develop a framework (in section IV) to keep track of stable literals at three different levels. The avoidance of exponential complexity in our formulation is due to our observation that, at a particular level, once the stable literal is generated, our formulation does not permit us to flip its truth-value.

For the discovery of stable literal(s), we also note other variation of Proposition 2(i). We obtain this variation by taking contrapositives of both the conditionals, and it is given below.

Proposition 2 (i") : $((\bar{s} \rightarrow \bar{p}) \wedge (\bar{s} \rightarrow p)) \equiv s$.

This proposition states that, in our analysis, it is possible to discover the stable literal s using the following approach. In the presence of the restriction that literal $\bar{s} =$ **true**, the effect satisfaction of clauses of $\mathfrak{F}$ leads us to the discovery that, the literals $\bar{p}$ as well as (the negation of $\bar{p}$, which is) p are restricted to truth-value **true**. This allows us to conclude that the literal s is stable.

Yet other method to discover the stable literal s is based on the following rewriting Proposition 2(i).

Proposition 2 (i'') : $((\bar{s} \to \bar{p}) \wedge (\bar{p} \to s)) \equiv s$.

This proposition states that, in our analysis, it is possible to discover the stable literal s by considering "logical consequents" of its negation (*i.e.*, logical consequents of $\bar{s}$) and transitively closing them to discover s as one of its consequents. This leads to contradiction of our initial assumption that $\bar{s}$ is **true**, hence we have proof of s. It is well-known that, the approach of collecting the logical consequents to transitively close them to discover contradiction works for 2-SAT [1]. To illustrate this point further, we note the following rewriting of proposition 2(ii).

Proposition 2 (ii') : $((\bar{p} \to q) \wedge \bar{s} \to \bar{p}) \wedge (q \to s)) \equiv (\bar{p} \to q) \wedge s$.

We note that, in Proposition 2(ii'), the formula $(\bar{p} \to q)$ forms the 'intermediate' clause to serve as a *bridge* for the discovery of the contradiction to the initial assumption $\bar{s}$. Thus, assuming $\bar{s}$, when we start with the clause $(\bar{s} \to \bar{p})$, the bridge clause $(\bar{p} \to q)$ allows us to infer the literal q; finally the clause $(q \to s)$ requires the literal s to be restricted to truth-value true.

The non-triviality in the analysis phase is due to the global effect of the truth-value assignments. Our computational analysis relies heavily on Proposition 2(iii). During the progress of our computational analysis, whenever the summarized effect allows, we make use of Propositions 2(ii) and 2(i) as well.

2. Preliminaries and Notations

Let $\mathcal{F}$ be 3-SAT formula involving n variables, say, $x_1, x_2, \ldots, x_n$; m literals, say, $l_1, l_2, \ldots, l_m$, and k clauses, say, $C_1, C_2, \ldots, C_k$. Without loss of generality, we assume that $\mathcal{F}$ has proper 3-Clauses. Thus, we have:

(i) $\mathcal{F}$ does not have any clause which contains a variable in both unnegated and negated form,

(ii) $\mathcal{F}$ does not have any clause which contains two occurrences of the same variable in the same form,

(iii) Each clause of $\mathcal{F}$ has the same (*i.e.*, 3) literals in a clause (hence there is no clause, say C_i, which is a subset of the literals in other clause C_j in $\mathcal{F}$), and,

(iv) $\mathcal{F}$ does not have any clause which contains the truth values **true** or **false**.

Let us consider $m^*(m-1)$ ordered pairs, say of the m literals in $\mathcal{F}$. We note that, the satisfiability of $\mathcal{F}$ may result in more than one consequent literal for a given ordered pair. Hence, for an ordered pair (a, b) of literals, we maintain the collection of consequent literals, denoted by $L_{a,b}$, as a *set* (or as a *list*). We note that we use the words 'set', and, 'list' interchangeably in this paper. We note that $L_{a,b}$ includes $C_{a,b}$; however, in general, $L_{a,b}$ would include the additional literals that are discovered (due to the enforcement of the constraint that, for the satisfaction of formula $\mathcal{F}$, in the presence of satisfaction of this clause, other clauses of $\mathcal{F}$ also need to be satisfied) during (the steps in) our analysis. To distinguish $L_{a,b}$ from clause consequent $C_{a,b}$, we call $L_{a,b}$ as *logical* consequent (or, *formula* consequent) of the pair (a, b). The input length of the problem instance, denoted by $|\mathcal{F}|$, is the number of bits needed to *represent* (in some *reasonable* representation scheme) all clauses of the formula $\mathcal{F}$. While detailed and implementation-oriented approaches may formulate this length (in their chosen *reasonable* representation scheme) in terms of parameters like k (and m), to demonstrate our main result of polynomial solvability, our emphasis is not on such schemes; hence we may consider $|\mathcal{F}|$ as the number of entries needed for representing the array $C_{a,b}$ in terms of n. We note this as $O(n^3)$.

To start with, if $a \wedge b$ is the antecedent literal pair of a clause, its consequent, say c is added to $L_{a,b}$. To construct the context (of a satisfied clause), we put these two things together: thus we call $[(a, b),\ L_{a,b}]$ as our present context. In our computational analysis, we find it useful to keep track of the set of all literals that are discovered as **true**; we use the notation $W_{a,b}$ to denote (set of) such literals (we may use the term '(logically) restricted (worked out) literals' to refer to $W_{a,b}$). For the context $[(a, b), L_{a,b}]$, we note that $W_{a,b} = \{a, b\} \cup L_{a,b}$. We may visualize this context as depicted in Figure 4 below.

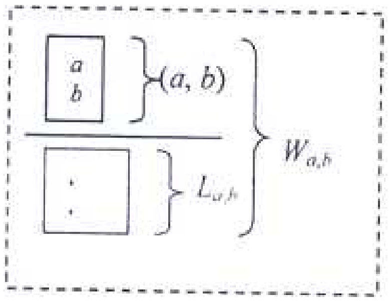

Fig. 4. Context: consisting of (i) antecendent literal-pair (a,b), (ii) (antecedents's) consequent(s) $L_{a,b}$, and (ii) working set of restricted literals $W_{a,b}$.

This context has assigned truth-values to the variables involved in its literals. We also note that each clause of $\mathfrak{F}$ generates such contexts. In our approach, we exploit this generation by constructing all possible contexts (by considering all possible literal pairs as potential antecedents).

A given context needs to be satisfied in the presence of all other contexts. To do this analysis, for a context, we create a collection of additional contexts by appending each of the remaining (variables') literals. From Figures 1, 2 and 4, we note that, this act of appending a literal to the context can be thought of as the act of creating additional level of the nested conditional.

Thus, to analyze the context $[(a, b), L_{a,b}]$, we generate the extended contexts $([(a, b), L_{a,b}], c)$, and obtain the consequents that are constructed by collecting the consequents (including the eventual consequents, by the use of repeated application with newly generated literals as well) of pairing c with every literal in $[(a, b), L_{a,b}]$. Let this list of consequents be denoted by L_c. We give the overall structure of this context analysis in

Figure 5.

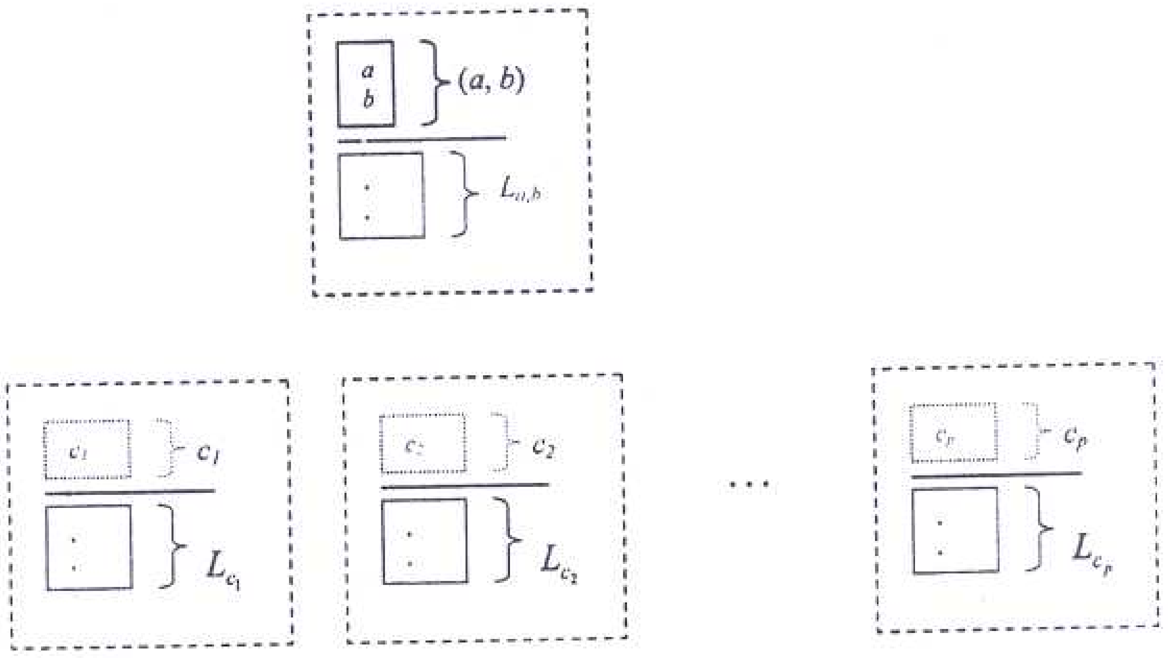

Fig. 5. Context extension by appending p literals.

Our analysis makes use of the contrapositive closure of conditionals. We formally state its formulation in the following proposition.

Proposition 3 : Suppose literal l is in l_{c_i}. Then, $\bar{c}_i$ is in $L_{\bar{l}}$.

Proof : Since $c_i =$ **true** has restricted the literal l to be **true** (in the presence of the global constraint that all clauses in $\mathcal{F}$ are satisfied), we have the conditional $(c_i \rightarrow l)$, which is equivalent to its contrapositive $\bar{l} \rightarrow \bar{c}_i$. Hence, $\bar{c}_i$ is in $L_{\bar{l}}$.

□

This proposition gives us a power of backward reasoning: while forward reasoning proceeds by making (repeated) use of *antecedent literal-pair* representation of clauses, this proposition allows use to combine it with backward reasoning. In our computational analysis, we combine this backward reasoning with the forward reasoning. The forward reasoning is captured by the transitive closure of the conditional. In the next section, after the algorithm C-Closure (A), we state this capability of forward reasoning of our formulation in Proposition 4.

In the next section, we make our exposition precise by giving the formulation of a few of concepts, namely C-closure (and W-Closure, that includes the logical consequents) of a set of literals.

3. Algorithmic Formulation

Let $\mathcal{L}$ be the list of all literals in $\mathcal{F}$. For all literal-pairs (a, b) in $\mathcal{L} \times \mathcal{L}$, obtain the lists $C_{a,b}$ by treating (a, b) as the antecedent literal pair and collecting its consequent(s) (if any), in the corresponding list $C_{a,b}$.

We assume that n variables (namely, $V = \{x_1, x_2, \ldots, x_n\}$) are divided in two parts: the first part, say V_1, contains all variables whose literal are restricted to **true** truth-value. Let A denote the ***nonempty*** set of antecedent literals; hence, to start with, $V_1 =$ (set of variables in the literal set) A. For each literal of a variable in the second part, we maintain the list of (consequent) literals; thus, variable (say y) in the second part of $n - |V_1|$ variables, we maintain two consequent lists: one for the literal y, and other for its negation.

We first informally give several features of the computations involved in C-Closure(A).

(i) The C-Closure(A) constructs a set R of restricted literals.

(ii) During our construction, we do not allow the deletion of literals. As a consequence, the set R of restricted literals may contain a literal as well as its negation. This allows us to conclude that the collection A of literals is unsatisfiable.

(iii) One consequence of our discipline of not permitting the deletion of literals is, we always have $R \supseteq A$; in other words, R would possibly contain additional literals not in A. When R does not contain a literal as well as its negation, the set of additional literals is meaningful; let the set of these additional literals included in R be B.

(iv) During our analysis involved in C-Closure(A), we successively construct consequent lists for all variables not included in R.

(v) We terminate our analysis when we discover that there is no change in the consequent lists for all variables not included in R.

(vi) When we decide to terminate the analysis involved in C-Closure(A), we note that, for each literal of the non-restricted variable (*i.e.*, for each variable whose literal is not included in R), we have the corresponding consequent list.

(vii) We note that, due to (vi), C-Closure(A) also returns p (= 2*number of variables not in R) consequent lists (These lists are potentially useful for discovering the truth-value assignments when the given set of literals A can satisfy $\mathfrak{F}$).

Thus, we note that C-Closure(A) takes, as input, the *non-empty* set of literals A, and it generates: (i) *Satisfy-Flag* (**false** if our computational analysis in C-Closure(A) reveals that set of literals A cannot satisfy $\mathfrak{F}$), (ii) a set B of restricted literals (of some variables in the set of variables, $V - V_1$) , (also the set $R = A \cup B$), and, (iii) p (= 2*number of variables not in R), consequent lists.

Formally, we give the steps of Algorithm: C-Closure (A) below.

```
Algorithm: $C$-Closure ($A$);
    1. Initialize $B = \Phi$; Satisfy-Flag = **true**;
    2. Repeat steps 2.1 to 2.5 below (until there is no update in $B$) {
        2.1 $b$-update = false; $R = A \cup B$; // .. working set of restricted literals
```

```
  2.2 Repeat steps 2.2.1 to 2.2.3 below (until there is no change in $R$) {
    2.2.1 if there exists a literal as well as its negation in $R$, then {
                    Satisfy-Flag = **false**; exit $C$-Closure; }
    2.2.2 else (i.e., if there does not exist a literal as well as its negation in $R$)
          {$B = B \cup C_{m,l}$, for $m, l \in R$; // .. update $B$, $R$ to include
                                                 the literals $n$ such that
          $R = R \cup B$;}                      // .. $n \in C_{m,l}$, for $m, l \in R$
    2.2.3 } until there is no change in $R$;
  2.3 For all literals (of variables not in $R$, say) $c$ ($c$ in $\{c_1, c_2, \ldots, c_p\}$, $p$ = 2*num-
      ber of variables not in $R$), we initialize the $p$ lists, $L_{c1}, L_{c2}, \ldots, L_{cp}$, as follows:
      $L_{ci} = \{n$ a literal of variable not in $R \mid n \in C_{x,ci}$, for some $x \in R\}$ $(i =$
      $1, \ldots, p)$;
  2.4 Repeat steps 2.4.1 - 2.4.4 below (until there is no change in the lists $L_{ci}(i =$
      $1, \ldots, p)$, or no $b$-update) {
    2.4.1 For all lists $L_c$, for all $l$ in $L_c$, include $\bar{c}$ in $L_{\bar{l}}$; // .. include contrapositives
    2.3.2 For all lists $L_c$, update it (i.e., the list $L_c$) to include the literals $n$ (of
          variables not in $R$), such that $n \in C_{m,l}$, for some $m, l \in R \cup L_c$;
    2.4.3 if there exists $k$, such that $L_k$ contains a literal as well as its negation,
          then
          { update $B$ to include $\bar{k}$; $b$-update = **true** };
    2.4.4 } until there is no change in the lists $L_{ci}$ $(i = 1, \ldots, p)$, $\vee$ (not ($b$-update));
  2.5 } Until ((not (Satisfy-Flag $\wedge$ $b$-update));
3. Return *Satisfy-Flag*, $B$, $R$, and $p$ ( = 2*number of variables not in $R$) lists $L_{ci}$ $(1 \leq$
   $i \leq p)$.
```

End of Algorithm C-Closure (A).

Lists $L_{c1}, L_{c2}, \ldots, L_{cp}$, that are returned by C-Closure (A), satisfy the following property.

Proposition 4 : Suppose $l \in L_c$. Then, $L_l \subseteq L_c$, and $L_{\bar{c}} \subseteq L_{\bar{l}}$.

Proof : Step 2.3 constructs the list L_l by including all literals n_1 such that $n_1 \in C_{m,l}$, for some $m \in R$. In other words,

$$L_l = \{n_1 \mid n_1 \in C_{m,l}, \text{ for some } m \in R\}. \tag{4}$$

Next, we are given that $l \in L_c$; so step 2.4.2 guarantees that all the literals n, such that $n \in C_{m,l}$, for some $m \in R \cup L_c$ are included in L_c. In other words,

$$L_c = \{n \mid n \in C_{m,l}, \text{ for some } m \in R \cup L_c\}. \tag{5}$$

Consider a literal n_1 in L_l. From (4), it is given that $n_1 \in C_{m,l}$, for some $m \in R$. However, since $m \in R \Rightarrow m \in R \cup L_c$, this also implies that $n_1 \in C_{m,l}$, for some $m \in R \cup L_c$. Hence, using (5), we also have n_1 is a literal in L_c. Hence, $L_l \subseteq L_c$.

In step 2.4, the change (if any) in the lists is made. Specifically, due to step 2.4.1, we also have $L_{\bar{l}}$ includes $\bar{c}$, i.e., $\bar{c} \in L_{\bar{l}}$. Using the argument similar to (4) and (5) above, we conclude that $L_{\bar{c}} \subseteq L_{\bar{l}}$.

We emphasize that, in Closure(A), we create the conditionals for the analysis needed by the Proposition 2. Formally, we state our observation about the effect of stable literals in Theorems 5 and 6 below.

Theorem 5 : Let $(p \vee q \vee r)$ be a clause in $\mathcal{F}$. Then, the C-Closure(A) guarantees that there is no literal s that is (common) in all the three lists L_p, L_q, and L_r.

Proof : Assume the contrary; thus assume that there exists a literal s that is in all the three lists L_p, L_q, and L_r. Due to step 2.4.1, in $L_{\bar{s}}$, we have three literals $\bar{p}$, $\bar{q}$, and $\bar{r}$.

Further, due to step 2.4.2, we have, $C(\bar{p}, \bar{q}) = r$ is included in $L_{\bar{s}}$. Step 2.4.3 discovers that $L_{\bar{s}}$ has both literal r and its negation $\bar{r}$; so it includes s in B, and the steps 2.1 to 2.5 are repeated; in particular, in step 2.3, for all literals not in $A \cup B$, fresh lists are created. This guarantees the non-existence of such a literal s in the lists L_p, L_q, and L_r.

$\square$

Theorem 6 : Let $(p \vee q \vee r)$ be one of the clauses in $\mathcal{F}$. Further, suppose that a literal s in the two lists L_p, L_q. Then, the C-Closure(A) guarantees that: (i) $L_{\bar{r}}$ contains s, and, (ii) $L_{\bar{s}}$ contains r.

Proof : Given that a literal s is in the two lists L_p and L_q. Due to step 2.4.1, in $L_{\bar{s}}$, we have two literals $\bar{p}$ and $\bar{q}$. Later, due to step 2.4.2, we have, $C(\bar{p}, \bar{q}) = r$ is included in $L_{\bar{s}}$ (this completes (ii)). Since there is change in $L_{\bar{s}}$, to complete step 2.4, the substep 2.4.1 is repeated which guarantees that $L_{\bar{r}}$ contains s, thereby guaranteeing (i).

$\square$

Proposition 7 : Let $(p \vee q \vee r)$ is one of the clauses in $\mathcal{F}$. Further, suppose that a literal s is in the two lists L_p, L_q. Then, (i) $L_{\bar{p}} \subseteq L_{\bar{s}}$, (ii) $L_{\bar{q}} \subseteq L_{\bar{s}}$, and (iii) $L_r \subseteq L_{\bar{s}}$.

Proof : Follows from Theorem 6(ii) and Proposition 4.

$\square$

We construct the list $L_{\bar{s}}$ as an upper-bound on the lists (sets) $L_{\bar{p}}$, $L_{\bar{q}}$, and L_r. In the process of construction of our such hierarchical set (collection) of literals, we ensure that we have taken care of satisfaction of the clause $(p \vee q \vee r)$, which did not involve the variable v_s. This observation allows us to formulate the computational step(s) for establishing the high level relationship between the clause (here, as an example, it is $(p \vee q \vee r)$) and the restriction of the truth-value assignment (here, it is the literal s). We stated that such a modeling is our goal in Section I, where we briefly summarized our approach.

We introduced the notion of C-Closure(A) as one step to algorithmically define computational structure needed to capture the context extension (Figure 5). The following formula captures the conditionals generated in algorithm C-Closure(A):

$$((\wedge a, \forall a \in A) \rightarrow ((\wedge b, \forall b \in B) \rightarrow (\wedge(c \rightarrow (\wedge l_c, \forall l_c \in L_c), \forall c, c \text{ is not a literal involving in } A \cup B))) \quad (6)$$

We depict the lists of conditionals generated in C-Closure(A) in Figure 6.

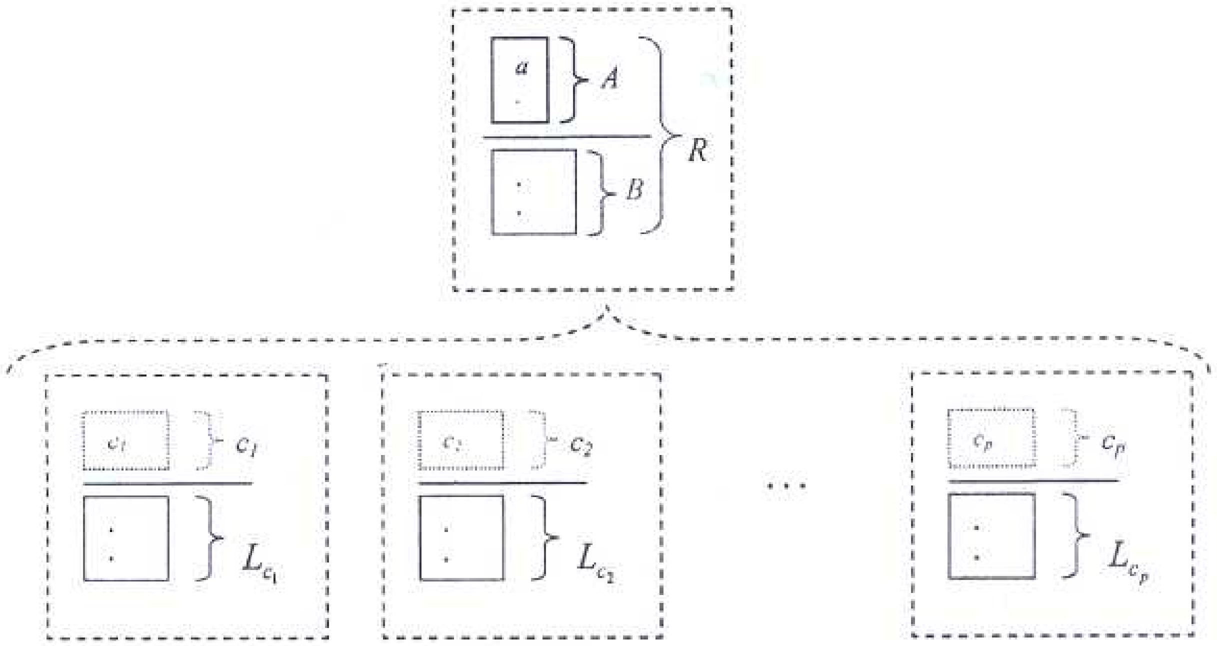

Fig. 6. Context extension (by appending p literals) for a given literal set A.

Proposition 8 : Suppose that C-Closure(A) returns R that contains a literal as well as its negation. Then the discovery of such a variable is in the same iteration of step 2.2.

Proof : Assume that all literals in A are true. Further, assume the contrary:

hence assume that in our algorithm C-Closure(A), we have the literal $r \in R$ discovered in the earlier iteration of step 2.2, and now the most recent step in the computational analysis in C-Closure(A) has discovered that $\bar{r}$ needs to be restricted to be assigned as **true** truth-value. We first claim that this most recent step has to be step 2.2.2. This is because, in steps 2.3, and 2.4, only variables involved are the ones not in R, and hence these steps cannot fix the literal $\bar{r} = \mathbf{true}$.

This recent step in 2.2.2, $\bar{r} = n \in C_{m,l}$, for some $m, l \in R$, is due to the presence of a clause $(\bar{m} \vee \bar{l} \vee \bar{r}) \in \mathcal{F}$.

Next our claim is: even the discovery of the restriction of the literal $r \in R$has to be in the same iteration of the step 2.2. Again, suppose the contrary.

Let us assume that the literal r is discovered to be true in the earlier R-update. To discover $\bar{r} = \mathbf{true}$, we need to have some clause in which $\bar{r}$ is in it; let us assume that it is the clause $(\bar{m} \vee \bar{l} \vee \bar{r}) \in \mathcal{F}$. Now, since the literal $r \in R$ was already discovered earlier, in the presence of this literal, we also have its literal-pair representation as $C_{r,l}$ includes $\bar{m}$, as well as, $C_{r,m}$ includes $\bar{l}$; in other words, in the presence of r, we need to satisfy the two-clause $(\bar{l} \vee \bar{m})$, with its representation in terms of conditional as $l \rightarrow \bar{m}$ (and its contrapositive: $m \rightarrow \bar{l}$). From amongst the literal pairs $m, l \in R$, one of them (without loss of generality, let it be l) must have been discovered in the earlier R-update, and the step 2.2.2 of that earlier update must have forced the negation of other literal to be true ($\bar{m} = \mathbf{true}$). This process continues backwards till we hit all the literals in A. Tracing these computations in the forward direction, under the assumption that all intermediate clauses in $\mathcal{F}$ are satisfied, we get satisfying truth-value assignment with literal r ; hence our assumption that, satisfaction of some clause in $\mathcal{F}$ requires that we need $\bar{r}$ as restricted literal is incorrect.

This leaves the choice of discovery of restricted literals r and $\bar{r}$ to be within the same iteration of step 2.2.

□

Proposition 8 highlights a useful property: we need not attempt to collect all stable literals at lower levels to propagate them at the upper level(s). In case there is a need for such multiple propagations, the upper level includes computational steps to detect the same. This observation is also interesting from the point of view of saving the number of operations.

While the analysis based on C-Closure may discover the unsatifiability (by setting *Satisfy-Flag* to **false**), unfortunately, it falls short to discover the satisfiability. Thus, when C-Closure(A) returns *Satisfy-Flag* as **true**, we are not guaranteed that there exists a satisfying truth-value assignment for F with all literals in A being **true**. The reason for this inadequacy is due to the fact that, the consequent lists generated in C-Closure made use of antecedent literal-pair clause consequent ($C_{a,b}$), and not $L_{a,b}$.

While the lists $L_{c1}, L_{c2}, \ldots, L_{cp}$, that are returned by C-Closure (A), would guide us for the discovery of truth-value assignments, they are not sufficient to guarantee that the decisions about the selection of literals that we make are correct in the sense that there would be no need to choose the negation of the selected literal. To close this gap is very important aspect in our formulation. The approach we choose to take to close this gap is to substitute the use of the antecedent literal-pair representation of a clause (i.e., $C_{a,b}$) by a global logical consequent representation, that have chosen to denote by $W_{a,b}$. We recall that, in Figure 4, we distinguished the clause consequent $C_{a,b}$, from the formula consequent $L_{a,b}$, as well as the working set of restricted literals, denoted by $W_{a,b}$.

Our analysis for 3-SAT-satisfiability progresses by updating $W_{a,b}$. To start with, we initialize $W_{a,b}$ to $C_{a,b}$. To capture the global effect of satisfaction of other clauses, we make use of this updated (list, or the collection of) restricted literals$W_{a,b}$ (to replace the local construct of clause consequent $C_{a,b}$). Hence, we define W-Closure(A) in exactly similar way as C-Closure(A), but by substituting all occurrences of clause consequents (i.e., $C_{a,b}$) by the logical consequents (by $W_{a,b}$); it is given below.

Algorithm: W-Closure (A);

```
1. Initialize $B = \Phi$; Satisfy-Flag = **true**;
2. Repeat steps 2.1 to 2.5 below (until there is no update in $B$) {
    2.1 $b$-update = false; $R = A \cup B$; // working set of restricted literals
    2.2 Repeat steps 2.2.1 to 2.2.3 below (until there is no change in $R$) {
        2.2.1 if there exists a literal as well as its negation in $R$, then {
                                  Satisfy-Flag = **false**; exit $W$-Closure; }
        2.2.2 else ($i.e.$, if there does not exist a literal as well as its negation in $R$)
              {$B = B \cup W_{m,l}$, for $m, l \in R$; // .. update $B$, $R$ to include
                                                   the literals $n$ such that
              $R = R \cup B$;}                     // .. $n \in W_{m,l}$, for $m, l \in R$
        2.2.3 } until there is no change in $R$;
    2.3 For all literals $c$ ($c$ in { $c_1, c_2, \ldots, c_p$}, $p$ = 2*number of variables not in $R$),
        we initialize the $p$ lists, $L_{c1}, L_{c2}, \ldots, L_{cp}$, as follows:
        $L_{ci} = \{n$ a literal of variable not in $R \mid n \in W_{x,ci}$, for some $x \in R\}$ $(i =$
        $1, \ldots, p)$;
    2.4 Repeat steps 2.4.1 – 2.4.4 below (until there is no change in the lists $L_{ci}(i =$
        $1, \ldots, p)$, or no $b$-update) {
        2.4.1 For all lists $L_c$, for all $l$ in $L_c$, include $\bar{c}$ in $L_{\bar{l}}$; // .. include contrapositives
        2.3.2 For all lists $L_c$, update it (i.e., the list $L_c$) to include the literals $n$ (of
              variables not in $R$), such that $n \in W_{m,l}$, for some $m, l \in R \cup L_c$;
        2.4.3 if there exists $k$, such that $L_k$ contains a literal as well as its negation,
              then
              { update $B$ to include $\bar{k}$; $b$-update = **true** };
        2.4.4 } until there is no change in the lists $L_{ci}$ $(i = 1, \ldots, p)$, $\vee$ (not ($b$-update));
    2.5 } Until ((not (Satisfy-Flag $\wedge$ $b$-update));

  3. Return Satisfy-Flag, $B$, $R$, and $p$ ( = 2*number of variables not in $R$) lists $L_{ci}$ ($1 \leq i \leq p$).
```

End of Algorithm W-Closure (A).

In Figure 2, we noted that, the antecedent literal-pair has alternate representation in terms of nested conditionals. Using it with the contrapositivity (Proposition 3) closure allows us to algorithmically treat the propagation of restricted literals all the way to the top level.

We find it convenient to work with two-dimensional array $W_{x,y}$ with its indices x, y being literals (*i.e.*, $x, y \in \mathcal{L}$), and the entry $W_{x,y}$ holds a set of (restricted) literals. We find it convenient to use the additional two-dimensional array $F_{x,y}$ (with its indices x, y as literals) to keep *summarized* information about whether $W_{x,y}$ includes a literal as well as its negation.

We start our computational analysis with the assumption that $W_{x,y}$ does not include a literal as well as its negation; to denote this state, we initialize $F_{x,y}$ to **true**. As our computational analysis progresses, during the course of our computations, as an *accumulated effect* of satisfaction of clauses in $\mathcal{F}$, when we discover that $W_{x,y}$ needs to include a literal as well as its negation, we set $F_{x,y}$ to **false**. As our computations progress, we note that, we permit only this change (*i.e.*, change from **true** to **false**) for the entries in this flag array. Further, we note that, as our computations progress, the array $W_{x,y}$ only accumulates additional literals (hence the operation is *only* additive, and we do not allow deletion of any literal from the literal set $W_{x,y}$). The *summarized effect* of our discovery that $W_{x,y}$ has included a literal as well as its negation guides our progress of computations by setting $F_{x,y}$ to **false**.

We formalize this construction in the 3-SAT satisfiability algorithm below.

Algorithm: 3-SAT-Satisfiability ($\mathfrak{F}$);

```
I. For all literal-pairs $(a, b)$ in $\mathcal{L} \times \mathcal{L}$, let $C_{a,b}$ denote the list of clause consequents of the
   antecedent literal pair $(a, b)$. Initialize $W_{a,b}$ to $C_{a,b} \cup \{a, b\}$. In this initialization,
   we note that $W_{a,a} = \{a\}$. For all literal pairs $(a, b)$, initialize $F_{a,b}$ to truth-value
   true.

II. Declare (and initialize) global variables needed for $W$-Closure( ). Thus, declare
    and initialize $B$, set of literals, to null-set $\Phi$. Initialize Satisfy-Flag to true. For
    all literals $a$ in $\mathcal{L}$, initialize lists $L_a$ to null-set $\Phi$.

III. Let $T$ be list of (top level) restricted literals; initialize $T$ to null-set $\Phi$. Initialize
     Satisfiable = true.

IV. Repeat steps 1, .., 5 (until there is no change, update in $T$) { Change = false;

 1. For all literal pairs $(x, y)$ in $\mathcal{L} \times \mathcal{L}$, not in $T$, { // ... literal-pair analysis

    1.1. initialize (the lists, or sets) $W_{x,y} = T \cup W_{x,x} \cup W_{y,y} \cup W_{x,y}$;

    1.2. invoke $W$-Closure($W_{x,y}$); // .. results in Satisfy-Flag,
                                        // .. and $B$, set of restricted literals

    1.3. if (Satisfy-Flag) then $W_{x,y} = W_{x,y} \cup B$ // .. include restricted literals

    1.4. else {

       1.4.1. $F_{x,y} =$ false; // .. $x \wedge y =$ false $\Leftrightarrow \overline{x \wedge y} =$ true;

       1.4.2. if $\bar{y} \notin W_{x,x}$ then { $W_{x,x} = W_{x,x} \cup \{\bar{y}\}$; Change = true; }

       1.4.3. if $\bar{x} \notin W_{y,y}$ then { $W_{y,y} = W_{y,y} \cup \{\bar{x}\}$; Change = true; }

       1.4.4. }; // .. end of else in step 1.4

    1.5. }; // .. end of For loop in step 1

 2. For all literals $x$ in $\mathcal{L}$, {                // .. Top level restricted literals
```

```
    2.1. if $\bar{x} \notin T$ and $W_{x,x}$ includes a literal as well as its negation {
    2.2. $F_{x,x}$ = **false**;
    2.3. $T = T \cup \{\bar{x}\}$;
    2.4. invoke $W$-Closure($T$); // .. results in *Satisfy-Flag,* and $B$, set of literals
    2.5. $T = T \cup B$ ; // .. include restricted literals
    2.6. if ( (not *Satisfy-Flag*) ) then {
        2.6.1. *Satisfiable* = **false**;
        2.6.2. Exit For loop in step 2 & loop in step IV
        2.6.3. }; // .. end of if of step 2.6
    2.7. *Change* = **true**; }; // .. end of if in step 2.1
    2.8. } ; // .. end of For loop in step 2
3. For all literals $x$in $L$, {                         // .. Contrapositive closure
    3.1. if ( ($\bar{x} \notin W_{\bar{a},\bar{a}}$) and ($a \in W_{x,x}$) ) then {
    3.2. $W_{\bar{a},\bar{a}} = W_{\bar{a},\bar{a}} \cup \{\bar{x}\}$;
    3.3. invoke $W$-Closure($W_{bara,\bar{a}}$); // .. results in *Satisfy-Flag,* and $B$ literals
    3.4. if (*Satisfy-Flag*) then $W_{\bar{a},\bar{a}} = W_{\bar{a},\bar{a}} \cup B$ ; // .. include restricted literals
    3.5. if ( (not *Satisfy-Flag*) ) then {
        3.5.1. $F'_{\bar{a},\bar{a}}$ = **false**;
        3.5.2. $T = T \cup \{a\}$;
        3.5.3. invoke $W$-Closure($T$); // .. results in *Satisfy-Flag,* and $B$
        3.5.4. $T = T \cup B$ ;
        3.5.5. if ( (not *Satisfy-Flag*) ) then {
            3.5.5.1. *Satisfiable* = **false**;
            3.5.5.2. Exit For loop in step 3 & loop in step IV
        3.5.5.3. }; // .. end of if in step 3.5.5
    3.6. }; // .. end of if of step 3.5
    3.7. *Change* = **true**; }; // .. end of if in step 3.1
    3.8. } ; // .. end of For loop in step 3
4. For all literals $x$in $L$, {                    // .. Transitive closure
    4.1. For all literals $a \in W_{x,x}$ such that $W_{a,a} \not\subset W_{x,x}$ {
    4.2. $W_{x,x} = W_{x,x} \cup W_{a,a}$; // .. transitivity closure for the case $W_{a,a} \not\subset W_{x,x}$
    4.3. invoke $W$-Closure($W_{x,x}$); // .. results in *Satisfy-Flag*, and $B$, set of literals
    4.4. if (*Satisfy-Flag*) then $W_{x,x} = W_{x,x} \cup B$; // .. include restricted literals
    4.5. if ( (not *Satisfy-Flag*) ) then {
        4.5.1. $F_{x,x}$= **false**;
        4.5.2. if $\bar{x} \notin T$ then {
        4.5.3. $T = T \cup \{\bar{x}\}$;
        4.5.4. invoke $W$-Closure($T$); // .. results in *Satisfy-Flag*, and $B$
        4.5.5. $T = T \cup B$;
        4.5.6. if ( (not *Satisfy-Flag*) ) then {
            4.5.6.1. *Satisfiable* = **false**;
            4.5.6.2. Exit For loop in step 4 & loop in step IV
            4.5.6.3. }; // .. end of if in step 4.5.6
        4.5.7. }; // .. end of if of 4.5
    4.6. *Change* = **true**; }; // .. end of For loop in step 4.1
    4.7. } ; // .. end of For loop in step 4
5. Until (not (*Satisfiable* $\wedge$ *Change*) ); // .. end of IV
```

V. If *Satisfiable* then T contains list of top-level restricted literals, $W_{x,x}$ contains set of restricted literals when $x =$ **true** and $W_{x,y}$ contains set of restricted literals when the pair of literals $(x, y) =$ (**true, true**).

End of Algorithm 3-SAT-Satisfiability ($\mathcal{F}$).

We note that, in sub-step 1.4 of step IV, we note that $F_{x,y} =$ **false** is used in our computations to conclude that $x \wedge y =$ **false** $\Leftrightarrow \overline{x \wedge y} =$ **true**; hence we have discovered the two-clause $(\bar{x} \vee \bar{y})$.

The use of $W_{x,y}$ (also, in its summarized form, $F_{x,y}$) instead of $C_{x,y}$, coupled with the propagation of discovered restrictions on literals at upper levels, as done in Algorithm 3-SAT-Satisfiability ($\mathcal{F}$), guarantees the completeness of the discovery of restrictions on literals due to clause satisfactions. Formally, we summarize this observation in the following propositions.

Proposition 9 : Suppose Algorithm 3-SAT-Satisfiability ($\mathcal{F}$) returns *Satisfiable* to be **true**. Further suppose any literal of variable v is not in T and let a be a literal of the variable v. Then $F_{a,a} =$ **true**, and $F_{\bar{a},\bar{a}} =$ **true**.

Proof : Without loss of generality, suppose $F_{a,a} =$ **false**. Then the constructions in: case (i): steps 2.2, 2.3, or case (ii): steps 3.5.1, 3.5.2, or case (iii): steps 4.5.1, 4.5.2, guarantee us that T would have included the literal $\bar{a}$; so variable v would have already included its literal $\bar{a}$ in T, and hence our assumption that it was a variable not in T is **false**. Hence the result.

□

Proposition 10 : Suppose Algorithm 3-SAT-Satisfiability ($\mathcal{F}$) returns *Satisfiable* to be **true**. Further suppose any literal of variables u, v is not included in T and let a be a literal of the variable u and b be a literal of the variable v, with $F_{a,b} =$ **false**. Then $W_{a,a}$ includes $\bar{b}$, and $W_{b,b}$ includes $\bar{a}$.

Proof : Follows due to the construction in step 1.4.

□

Proposition 11 : Suppose Algorithm 3-SAT-Satisfiability ($\mathfrak{F}$) returns *Satisfiable* to be **true**. Further suppose any literal of variables u, v is not included not in T. Let a be a literal of a variable u and b be a literal of v, with $F_{a,b}$ = **true**. Then $W_{a,b}$ does not include any literal pair, say (p, q), such that $F_{p,q}$ = **false**.

Proof : Assume that $F_{a,b}$ = **true**, and $W_{a,b}$ includes a literal pair, say p, q such that $F_{p,q}$ = **false**. We note that $F_{p,q}$ is assigned a **false** value due to $W_{p,q}$ containing a literal as well as its negation in our construction. Since $W_{a,b}$ includes a literal pair, (p, q), due to the W-Closure ($W_{a,b}$) operation that we perform in all steps (sub-steps 1,2,3,4) of step IV of Algorithm 3-SAT Satisfiability($\mathfrak{F}$), and due to the step 2.2.2 of our algorithm W-Closure (A), our construction $W_{a,b}$ includes all literals in $W_{p,q}$.

In other words, $W_{a,b}$ includes a literal pair, $(p, q) \Rightarrow W_{a,b} \supseteq W_{p,q}$; hence $W_{a,b}$ contains a literal as well as its negation, thereby making $F_{a,b}$ = **false**. This contradicts our assumption that $F_{a,b}$ = **true**. Hence the result.

□

Proposition 11 suggests that, in our computations, for each $F_{a,b}$ = **true**, its corresponding $W_{a,b}$ contains all literal-pairs (p, q), such that $F_{p,q}$ = **true**. In other words, on the set of literals, when $F_{a,b}$ = **true**, the subset of literals $W_{a,b}$ forms a clique with the binary relation defined by $F_{p,q}$.

We note that, our computations proceed by considering the conditional, $a \rightarrow b$, as well as its contrapositive $\bar{b} \rightarrow \bar{a}$. For the purpose of construction of the literal sets, its significance is as follows. When $W_{a,a}$ includes a literal b, we have $W_{a,a} \supseteq W_{b,b}$; however, for the "negated" (dual) literals, we also simultaneously require, in our analysis, $W_{\bar{b},\bar{b}} \supseteq W_{\bar{a},\bar{a}}$. Thus, the contrapositive closure (proposition 3) operation translates to the following. In the process of expansion of our literal sets, we simultaneously perform the expansion (of the literal set) both *original* as well as *dual* (negated) space of literals.

When there are no changes in this process of expanding the literal sets, our analysis terminates. At this termination, if *Satisfiable* = **true**, then satisfying truth-value assignment exists. While we formally state this result in the next section, we express this observation in the following.

Proposition 12 : Suppose Algorithm 3-SAT-Satisfiability($\mathcal{F}$) returns *Satisfiable* to be true. Then there exists a truth-value assignment that does not include any literal pair, say (p, q), such that $F_{p,q}$ = **false.**

Proof : Since *Satisfiable* is true, T does not include a literal as well as its negation. Next, suppose any literal of variables u, v (u, v not necessarily distinct) is not included not in T. Let a be a literal of a variable u and b be a literal of v, with $F_{a,b}$ = **true**. We note that the W-Closure($W_{a,b}$) returns the lists L_c, where c is a literal of non-restricted variables. Non-conflicting (satisfying) truth-value assignment exists and it can be obtained by choosing a (choice) literal say c, and including all literals in L_c to be in R. From amongst the remaining variables, again we repeat the choice literal and 'chase all its consequents' till all literals are exhausted. This process of *chasing the consequents* never conflicts with earlier choices made and this is guaranteed due to our contrapositive closure and transitive closure properties in Algorithm: 3-SAT Satisfiability(F). Using these lists created by$W$-Closure($W_{a,b}$), non-conflicting truth-value assignment is guaranteed to exist in the sense that this truth-value assignment does not include any literal pair, say p, q such that $F_{p,q}$ = **false**. We specially note that, if literal p is assigned **true** in this process of truth-value assignment, in our sub-step 1 (of step IV) in Algorithm: 3-SAT Satisfiability($\mathcal{F}$), we have already constructed the list $W_{p,p}$ to include $\bar{q}$; other alternative is to allow the literal q to be assigned truth-value **true** in this process; we also constructed the list $W_{q,q}$ to include $\bar{p}$. We also note that list-updates for lists $W_{x,y}$ for all literal pairs (x, y) are done till there is no more addition of literal, it has taken into account all possible changes to all lists $W_{x,y}$; this guarantees that we have captured all 'global consequent' literals in the process of 'chasing down the consequents'.

□

We note that the collection of $2 \times n$ sets (or lists) $W_{x,x}$ (for each x in $\mathcal{L}$ such that $F_{x,x} = \textbf{true}$, *i.e.*, $W_{x,x}$ does not contain a literal as well as its negation) possesses the properties of the contrapositive closure as well as the transitive closure (we stated these properties for the collection of lists L_{ci} in Propositions 3 and 4 respectively). In our construction, steps 3, 4 of step IV of Algorithm 3-SAT-Satisfiability($\mathcal{F}$) ensure these properties. Formally we state them again in the following propositions.

Proposition 13 : Suppose literal a is in $W_{x,x}$ (for x in $\mathcal{L}$) such that $F_{x,x} = \textbf{true}$ (*i.e.*, $W_{x,x}$ does not contain a literal as well as its negation). Then, if $F_{\bar{a},\bar{a}} = \textbf{true}$, then $\bar{x}$ is in $W_{\bar{a},\bar{a}}$.

Proof : Follows due to step 3 of step IV of Algorithm 3-SAT-Satisfiability($\mathcal{F}$).

□

Proposition 14 : Suppose literal a is in $W_{x,x}$ (for x in $\mathcal{L}$) such that $F_{x,x} = \textbf{true}$ (*i.e.*, $W_{x,x}$ does not contain a literal as well as its negation). Then, $W_{x,x} \supseteq W_{a,a}$.

Proof : Follows due to step 4 of step IV of Algorithm 3-SAT-Satisfiability($\mathcal{F}$).

□

4. Correctness

Algorithm 3-SAT-Satisfiability($\mathcal{F}$) makes use of this context extension, and creates top level structure of restricted literals (collected in T as list), and the next level of conditionally restricted literal lists, with the antecedent (of each list) consisting of single literal (with its variable is not included in T). The following formula captures the conditionals generated in the algorithm:

$$(\mathcal{F} \text{ satisfiable } \rightarrow ((\wedge t, \forall t \in T) \rightarrow (\wedge(a \rightarrow (\wedge r_a, \forall r_a \in W_{a,a})\forall a \in \mathcal{L} - T \cup \bar{T})))) \quad (7)$$

We note that Figure 5 gives diagrammatic representation of the context extension for literal pair (a, b). We depict this structure (of T, and each $W_{a,a}$) in a similar diagram, given in Figure 7.

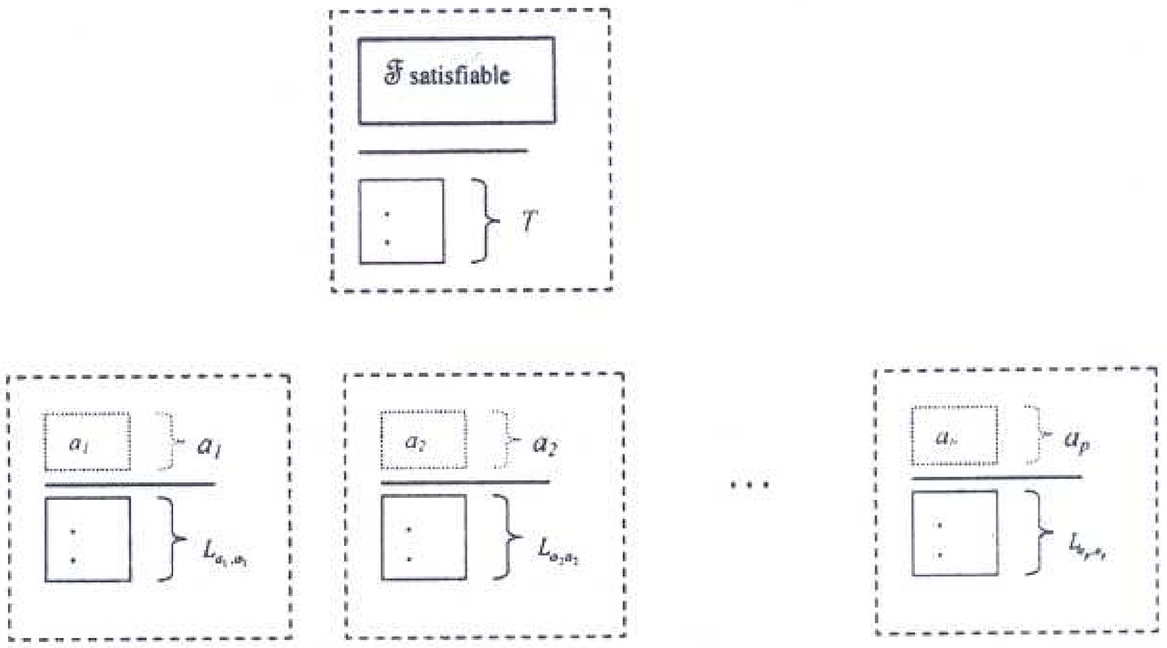

Fig. 7. Top level restricted literals (T) and, for each literal in the remaining variables (i.e., for each literal in $\mathcal{L} - T \cup \bar{T}$) the lists of $p = |\mathcal{L} - T \cup \bar{T}|$ conditionally restricted literals.

After completing the computations in 3-SAT-Satisfiability ($\mathfrak{F}$), our main claim is, If T does not contain a literal as well as its negation, then the formula is satisfiable. To prove this claim, we first give the scheme of satisfying truth-value assignments in the following algorithm Truth-Value Assignments($\mathfrak{F}$).

Algorithm: Truth-Value Assignments($\mathfrak{F}$);

1. Execute algorithm 3-SAT-Satisfiability($\mathfrak{F}$) to obtain (i) Boolean variable *Satisfiable*,

(ii) T, set of top level restricted literals, (iii) $W_{a,a}$, for each literal a and, (iv) $W_{a,b}$ (and, $F_{a,b}$) and for each literal pair (a, b).

2. if *Satisfiable*, then we start by assigning all literals in T the truth-value **true**. We collect these literals in R by performing the operation $R = T$.

3. Update $R = R \cup W_{m,l}$, for all $m, l \in R$.

4. From amongst the variables not assigned any truth-value, choose one variable, say v_c, and choose one of its literals, say c. Assign all literals in $W_{c,c}$ the truth-value **true**. We collect these literals in R by performing the operation $R = R \cup W_{c,c}$.

5. Repeat steps 3 and 4 until we have exhausted all the n variables.

End of Algorithm: Truth-Value Assignments($\mathcal{F}$).

Theorem 15 : When *Satisfiable* is **true** as a result of executing Algorithm: 3-SAT-Satisfiability($\mathcal{F}$), in steps 3 and 4 of the algorithm: Truth-Value Assignments($\mathcal{F}$), the literals that are added in R are guaranteed not to result into the conflicting truth-value assignments.

Proof : The repeated application of steps 3 and 4 generates the sequence of literal sets, say $R_1, R_2, \ldots, R_m$ (since every time at least one literal is added, $m < n$). At some intermediate stage, say ith stage, let R_i be the literal set without conflicting truth-value assignments. Suppose the literal sequence chosen in step 4 is $a_1, a_2, \ldots, a_i$. We note that the effect of the repeated union in steps 3 and 4 allows us to express:

$$R_i = T \cup W(a_1, a_1) \cup W(a_2, a_2) \cup \ldots \cup W(a_i, a_i) \cup W(m, l)(\text{with } m, l \in R_i). \quad (8)$$

Suppose, if possible, at the $(i + 1)$th stage, the conflicting truth-value assignment arises. This situation is possible due to the following three cases:

(1) when the currently chosen literal of a non-restricted variable at the $(i + 1)$th stage, say $a_{i+1} = d$, has in its $W_{d,d}$, one of the literals, say n, whose negation $\bar{n}$ is already

included in one of the earlier sets, *i.e.*, in one of the sets $R_0, R_1, R_2, \ldots, R_i$. Let these sets be denoted as $S_0, S_1, S_2, \ldots, S_i$, and their union be denoted by $(\sum_i S)$. With this notation, we have, $n \in W(d, d)$, and, $\bar{n} \in (\sum_i S)$.

(2) when the currently chosen literal of a non-restricted variable at the $(i+1)$th stage, say $a_{i+1} = d$, results in inclusion of some literal, say e. such that, with earlier included literals, say s, and $\bar{n}$, in $(\sum_i S)$, has $n \in W(s, e)$.

(3) when the currently chosen literal of a non-restricted variable at the $(i+1)$th stage, say $a_{i+1} = d$, results in inclusion of two literals, say e, and f, such that, with earlier included literal, say $\bar{n}$, in $(\sum_i S)$, we have, $n \in W(f, e)$.

We now argue that all these cases cannot arise when Algorithm: 3-SAT-Satisfiability($\mathcal{F}$) has resulted in *Satisfiable* flag to be **true**.

Case (1): Since $n \in W(d, d)$, we have $d \to n$. Due to contrapositive closure step in Algorithm: 3-SAT-Satisfiability($\mathcal{F}$), we have, $\bar{n} \to \bar{d}$. Further, since $\bar{n} \in (\sum_i S)$, due to transitivity, $W(\bar{n}, \bar{n}) \supseteq W(\bar{d}, \bar{d})$. This means, when $\bar{n} \in (\sum_i S)$, $\bar{d}$ is also included in $W(\bar{n}, \bar{n})$ and hence in $(\sum_i S)$ in first i stages. So our assumption that the variable involving literal d was free for making choice at $(i+1)$th stage is not correct. Hence, this case cannot arise.

Case (2): Since $n \in W(s, e)$, we have the 3-clause (either directly given or logically implied due to the result of the satisfaction of other given clauses) $(\bar{s} \vee \bar{e} \vee n)$. Hence, $W(s, \bar{n})$ includes $\bar{e}$. Hence, $W(s, \bar{n}) \supseteq W(\bar{e}, \bar{e})$. Further, we have given that $d \to e$, which also is equivalent to its contrapositive as $\bar{e} \to \bar{d}$. Hence, we have $W(\bar{e}, \bar{e}) \supseteq W(\bar{d}, \bar{d})$. Due to transitivity of $\supseteq$, we get $W(s, \bar{n}) \supseteq W(\bar{d}, \bar{d})$. This means, when $s, \bar{n} \in (\sum_i S)$, $\bar{d}$ is also included in $W(s, \bar{n})$ and hence in $(\sum_i S)$ in first i stages. So our assumption that the variable involving literal d was free for making choice at $(i+1)$th stage is not correct. Hence, this case cannot arise.

Case (3): Since $n \in W(f,e)$, we have $(f \wedge e) \rightarrow n$; hence, we have the 3-clause (either directly given or logically implied due to the result of the satisfaction of other given clauses) $(\bar{f} \vee \bar{e} \vee n)$. Thus, we have $(\bar{n} \rightarrow (\bar{f} \vee \bar{e}))$. Further, since $d \rightarrow e$, and $d \rightarrow f$, we have $(\bar{f} \vee \bar{e}) \rightarrow \bar{d}$. Hence, $W(\bar{n}, \bar{n})$ includes $\bar{d}$. So our assumption that the variable involving literal d was free for making choice at $(i+1)$th stage is not correct. Hence, this case cannot arise.

Thus our assumption that the currently chosen literal at $(i+1)$th stage, we had the nonrestricted variable with the choice of its literal $a_{i+1} = d$ being available is incorrect (as earlier "contrapositive" and "transitive" closure operations guarantee us that our earlier choices must have restricted it to $\bar{d}$). Hence the result.

□

5. Polynomial Complexity

In this section, we demonstrate the polynomial complexity of our algorithm 3-SAT-Satisfiability($\mathcal{F}$).

Theorem 16 : Number of operations needed by C-Closure(A) (or, W-Closure(A)) have a bound of $O(n^5)$.

Proof : In Step 2.4, the steps 2.4.1 to 2.4.3 are repeated till there is no change in the lists. During this repetition, we note that literal(s) are only added (and never deleted), and each list can grow to include maximum p = 2*number of variables not in R) number of literals. Since we focus on demonstration of time complexity, we only consider an upper-bound on the length of list as $O(n)$. In step 2.4, we maintain p = 2*number of variables not in R) number of lists. again we may treat this expression as $O(n)$ lists.

In step 2.4.2, for each list, we need to consider pairs of literals; since the list size is $O(n)$, the number of ordered pairs is $O(n^2)$. For each ordered pair, we need to retrieve $C_{m,l}$ (or $W_{m,l}$, in case of W-Closure), whose size has an upperbound of $O(n)$. This makes a count of $O(n^3)$ per list. In step 2.4, we maintain $O(n)$ lists, and hence the count of the number of operations is $O(n^4)$ for all $O(n)$ lists.

We note that, from amongst the steps 2.1 to 2.5 that are repeated for completing step 2, the costliest step is 2.4, and it requires $O(n^4)$ operations.

Steps 2.1 to 2.5 are repeated for each possible change in B; there are $O(n)$ changes possible in B. Hence the total number of operations required for completing step 2 are $O(n^5)$.

Since Step 2 requires maximum number of operations in the algorithm C-Closure(A) (or, W-Closure(A)), we conclude that the number of operations needed by C-Closure(A) (or, W-Closure(A)) have a bound of $O(n^5)$.

Theorem 17 : The algorithm 3-SAT-Satisfiability ($\mathcal{F}$) has a bound of $O(n^{13})$.

Proof : W-Closure(A) is invoked in Step IV's substep 1.2 of Algorithm: 3-SAT-Satisfiability ($\mathcal{F}$). The step 1 involves construction of multiple lists, one for each the literal-pair (a, b). An upperbound the number of these literal pairs is $O(n^2)$.

In Step IV, the steps 1 to 5 are repeated for each change. We have already noted that the changes for the each entry $W_{a,b}$ are only additive in the sense that the literals can only get added; in fact this requirement necessitated our formulation to allow the possibility of adding a literal as well as its negation (and retaining it, after the addition) in $W_{a,b}$. We also found it convenient to keep this *summarized effect* in additional Boolean array$F_{a,b}$. This change permitted is from initial **true** truth-value assignment to **false** (when we discover that its corresponding$W_{a,b}$ entry needs to include a literal as well as its negation). In particular, once $F_{a,b}$ has been discovered to be **false**, we never flip its truth-value to **true**.

We note that each entry$W_{a,b}$, includes maximum $O(n)$ literals; thus there are $O(n^3)$ possible changes. Let us denote these possible changes by m. In our algorithm, for such a every change, since this change is in only one direction, *i.e.*, there is a repetition on smaller number of changes possible. However $O(n^3)$ serves as a simple upper-bound on the number of repetitions needed for each such a change. Thus, we have a bound of $m \times O(n^3) = O(n^3) \times O(n^3) = O(n^6)$ for these repetitions of steps 1 to 5.

During every such a change, we execute sub-step 1 of step IV of Algorithm: 3-SAT-Satisfiability ($\mathfrak{F}$).

In sub-step 1, for each literal pair (a, b), we invoke W-Closure($W_{a,b}$). In theorem 16, we noted that each invocation of W-Closure($W_{a,b}$) has a bound of $O(n^5)$ operations. Hence, step 1 has a bound of $O(n^7)$ operations.

As we noted, the bound on the number of repetitions of substep 1 is $O(n^6)$. This combined with the requirement of $O(n^7)$ operations per repetition gives us a bound of $O(n^{13})$ operations. Hence the bound on the total number of operations needed by step IV is $O(n^{13})$. Since Step IV is the step requiring maximum number of operations for Algorithm: 3-SAT-Satisfiability ($\mathfrak{F}$), this serves as a bound on our Algorithm: 3-SAT-Satisfiability ($\mathfrak{F}$) as well.

We note that, our main claim is to demonstrate polynomial solvability of 3-SAT. Hence, in our formulations, we refrained from analyzing the dependencies involved in the successive updates. In particular, we note that our analysis used the simple and well-known argument that is commonly used for the demonstration of quadratic time complexity of sorting algorithms like bubble-sort; other refinements are not of useful interest for demonstrating our main claim of polynomial complexity.

We also note that, as our input size, we have chosen to use the representation of clauses as $C_{a,b}$, having the input size of $O(n^3)$. Hence, our complexity bound of $O(n^{13})$ is not too bad for having the immediate practical implications of our formulation.

6. Concluding Remarks

Let us reiterate a few significant ideas introduced in our formulation. To formalize the unexpected result that we have collected all global consequent literals, we needed to consider *all* antecedent *literal-pairs* as our analysis progresses. We have formalized this concept as W-closure, and we have given its formulation in Section III. Our formulation in Algorithm: 3-SAT-Satisfiability(F) allows only addition of literals in $W_{p,q}$; this may result in the possibility that $W_{p,q}$ contains literal as well as its negation. We needed to treat this situation as a special case, because it allows us to infer the negation of the literal pair $p \wedge q$, *i.e.*, $(\bar{p} \vee \bar{q})$ as a two clause. In the formalization of 3-SAT-Satisfiability($\mathcal{F}$), (given in Section III), we captured such an inference by giving the formulation of the (flag) array $F_{p,q}$ whose each entry is initialized as "true" in the beginning; it may change to "false" value (when we discover that $W_{p,q}$ contains literal as well as its negation), but our formulation does not allow $F_{p,q}$ to change from '**false**' to '**true**'. Thus, this change allowed in the truth value of this flag array is unidirectional. This is another significant idea in our formulation. We may consider this idea as an adaptation of Chaitin's idea used in his famous register allocation algorithm (based on Graph coloring Heuristic); however, it is used here in very different context.

Let us now give the wider consequences of our work. Intelligent search has been the heart of all NP-Complete problems. One of the first NP-Complete problems is 3-SAT, and it retains all interesting features needed for intelligent search. For several thousands of NP-Complete problems, several attempts for developing the systematic technique for performing intelligent search had resisted repeated attempts in the last forty years. Hence, we believe our contribution has several interesting and far-reaching consequences. In this context, we reiterate a few important aspects of our contribution below.

In 3-SAT, the satisfiability problem is stated in terms of collection of clauses. It is known to be one of the first NP-Complete problems; it is well-known that it retains the salient features for the efficient algorithmic formulation of many interesting combinatorial and optimization problems. Satisfiability problem requires an intelligent search of truth-value assignment of variables with the constraint that each clause to be satisfied (i.e., it is evaluated as true). Our approach for intelligent search involved several interesting points:

(i) Formation and demonstration of the use of the concept of literal(s) stabilized by the restriction of clause satisfaction,

(ii) Use of contrapositives of suitable conditional(s), specially to propagate the partial restrictions our computations have discovered,

(iii) Systematic approach to transform *local* clause literal-pair representation (given in (1), as $C_{a,b}$) to *global* logical consequent representation (denoted by $W_{a,b}$ in our Algorithm: 3-SAT-Satisfiability ($\mathcal{F}$)).

(iv) Use of structural features of the problem to formulate the search space exploration. We note that, in our algorithm 3-SAT-Satisfiability ($\mathcal{F}$), we needed 3 levels of propagation.

(v) To do the search space exploration, we constructed $W_{a,b}$: a two dimensional array whose indices are literals. During our computations, the changes for the each entry $W_{a,b}$ are only additive in the sense that the literals can only get added. This requirement allows us get the polynomial bound on the number of operations.

We expect that the polynomial solutions to wide variety of NP-Complete problems would involve many similar features. Unification of the study of such features is expected to have insightful results, with the potential to have interesting computational consequences as well.

REFERENCES

1. Hopcroft, J. E., Motwani, R., and Ullman, J. D. : (2007), Automata, Languages, and Computation, third ed., (Exercise 10.3.4), pp. 458, Addison Wesley, N.Y.

2. Wikipedia: http://en.wikipedia.org/wiki/DPLL_algorithm (Retrieved 09 May 2009: version updated as on 01 April 2009).

3. Goldberg, E. : (2008), A decision-making procedure for resolution-based SAT-solvers, *in* "Theory and Applications of Satisfiability Testing: *SAT-2008* (11th International Conference)" (H.K. Buning, and X. Zhao, Eds.), Guangzhou, China, May 12-15, 2008, Lecture Notes in Computer Science (*LNCS*), **4996**, pp. 119-132, Springer Verlag, Berlin, Germany.

4. Mahajan Y., Fu Z ., and Malik, S. : (2004), Zchaff2004: An Efficient SAT Solver, *in* "Theory and Applications of Satisfiability Testing: *SAT-2004* (7th International Conference)" (H.H. Hoos, and D.G. Mitchell, Eds.), Vancouver, BC, Canada, May 10-13, 2004, Lecture Notes in Computer Science (*LNCS*), **3542**, pp. 360-375, Springer Verlag, Berlin, Germany.

5. Marques-Silva, J. P., and Sakallah, K. A. : (1999, May), GRASP: a search algorithm for propositional satisfiability, *IEEE Trans. Computers*, **48** (5), 506-521.

Professor,
Deptt. of Computer Science and Engineering (CSE),
Indian Institute of Technology (IIT),
Indore 452017 (M.P.)
E-mail: primary: nsc183@gmail.com

(*Received* : Aug. 9, 2009)
(*Revised* : Nov. 20, 2009)