JOURNAL ACADEMICA

VOLUME 8, NO.1, October 13 2018

N. Abdelwahab

JOURNAL ACADEMICA • NEW YORK

VOLUME 8, NO. 1, OCTOBER 13 2018

EDITOR IN CHIEF

S. Feigenbaum

TABLE OF CONTENT

Theoretical Computer Science

Copyright © 2018 JOURNAL ACADEMICA FOUNDATION • All rights reserved

ISSN 2161-3338 • www.journalacademica.org

Journal Academica Vol. 8(1), pp. 3-88, October 13 2018 - Theoretical Computer Science - ISSN 2161-3338
online edition www.journalacademica.org - Copyright © 2018 Journal Academica Foundation - All rights reserved

Full Length Research Paper

#2SAT is in P

Elnaserledinellah Mahmood Abdelwahab*
Senior Project Manager, makmad.org e.V., Hanover (Germany)

Received January 29 2018, Revised May 15, 2018; Accepted June 27 2018

ABSTRACT

This paper presents a new view of logical variables which helps solving efficiently the #P complete #2SAT problem. Variables are considered to be more than mere place holders of information, namely: Entities exhibiting repetitive patterns of logical truth values. Using this insight, a canonical order between literals and clauses of an arbitrary 2CNF Clause Set S is shown to be always achievable. It is also shown that resolving clauses respecting this order enables the construction of small Free Binary Decision Diagrams (FBDDs) for S with unique node counts in $O(M^4)$ or $O(M^6)$ in case a particular shown Lemma is relaxed, where M is number of clauses. Efficiently counting solutions generated in such FBDDs is then proven to be $O(M^9)$ or $O(M^{13})$ by first running the proposed practical Pattern-Algorithm 2SAT-FGPRA and then the counting Algorithm Count2SATSolutions, so that the overall complexity of counting 2SAT solutions is in P. Relaxing the specific Lemma enables a uniform description of *k*SAT-Pattern-Algorithms in terms of (*k*-1)SAT- ones opening up yet another way for showing the main result. This second way demonstrates that avoiding certain types of copies of sub-trees in FBDDs constructed for arbitrary 1CNF and 2CNF Clause Sets, while uniformly expressing *k*SAT Pattern-Algorithms for any $k>0$, is a *sufficient condition* for an efficient solution of *k*SAT as well. Exponential lower bounds known for the construction of deterministic and non-deterministic FBDDs of some Boolean functions are seen to be inapplicable to the methods described here.

Keywords: Logic, Duality, Variables, Patterns, Container, *k*SAT, #2SAT, FBDD, P=NP

*Corresponding author: elnaser@makmad.org

CONTENT

I INTRODUCTION

The current work aims at applying a new view of logical variables to the solution of #2SAT. This view considers variables to be more than mere place holders of information, namely: Entities exhibiting repetitive patterns of logical truth values. The ideas are materialized in novel Algorithms imposing universally applicable structural criteria on 2CNF Clause Sets, according to which clauses are ordered by their pattern lengths and least literals are always chosen for instantiation without prior trials. This enables efficient construction of small FBDDs upon which simple and equally efficient counting Algorithms can then be applied. To informally illustrate the basic ideas we start first with a concrete example.

Let be $S=\{\{x_0,x_4\}\{x_1,x_2\}\{x_2,x_3\}\}$. w.l.o.g., a monotone 2CNF formula[1] for which we would like to find a validating Truth Assignment by instantiating literals. Our instantiations result ultimately in a decision tree, which may be abstracted into a Binary Decision Diagram (BDD)[2]. Let PR, the used procedure, be described in pseudo-code[3] as follows.

PR:
Inputs: Arbitrary 2CNF Clause Set **S**
Output: BDD
Data Structure: Store of resolved Sets and their BDDs (ST)
Steps:

[1] There is no loss of generality in giving examples from the monotone 2CNF case, because properties of logical variables, relevant for this work, are already applicable in this simplest case.

[2] Formal definitions and illustrations of BDDs are seen below, but can also be found in, e.g., [Wegener 2000].

[3] Functions used in all pseudo-codes given in this work, except those of the theory section, have commonly used meanings and don't need any further specification as the procedures they are embedded in intend to give the reader only a sketch of the ideas under investigation, details and formalizations of which are found only in the Theory section.

1- Select any Literal *x* from a Clause C∈ S
2- Put *x*=TRUE in S forming S'
3- If (S' evaluates to TRUE)
leftResult=TRUE-Node
Else
if (any C'∈ S' Evaluates to FALSE)
leftResult=FALSE-Node
4- Put *x*=FALSE in S forming S''
5- If (S'' evaluates to TRUE)
rightResult=TRUE-Node
Else
if (any C''∈ S'' Evaluates to FALSE)
rightResult=FALSE-Node
6- Search for S' in ST if not TRUE/FALSE
If found
Put leftResult =BDD of S'
Else
- Put leftResult=PR(S')
- Store S' as well as leftResult in ST
7- Search for S'' in ST if not TRUE/FALSE
If found
Put rightResult =BDD of S''
Else
- rightResult=PR(S'')
- Store S'' as well as rightResult in ST
8- Create node Result such that: S is Clause Set of Result and:
a- leftNode(Result)=leftResult
b- rightNode(Result)=rightResult
9- Store S as well as Result in ST
10- Return Result

Algorithm – A1

This procedure *does not* instruct us how to choose literals for instantiation. Such a choice is crucial for the size of resulting BDDs as can be seen in Figures (1-a) and (1-b) in which non-terminal node counts are 5 and 10 respectively.

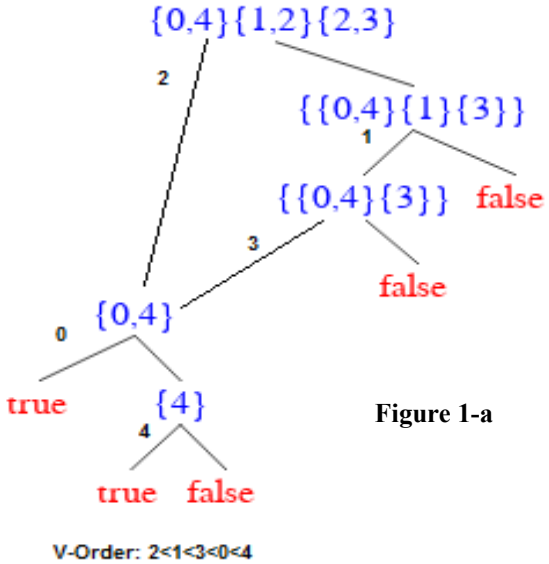


Figure 1-a

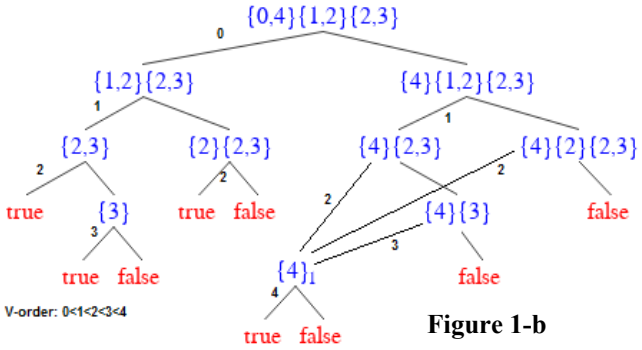


Figure 1-b

Let us call the content of a stack which registers the Literal choices made by PR in step 1 (while solving a problem *p* expressed in a 2CNF Clause Set): A Variable Ordering (to be précised in Section III, Notation: $\prod_p$). (Figure 1-a) shows an ordering $\prod_p$ = 2<1<3<0<4 which makes the number of nodes generated in the final BDD half the number needed if we chose $\prod'_p$ = 0<1<2<3<4 of (Figure 1-b). We call $\prod'_p$ Canonical Ordering (Notation: $\prod_p{}^c$), because it represents the order in which variables are listed from left to right in the Truth Table:

x_0	x_1	x_2	x_3	x_4
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	1
0	0	1	1	0
…..				

Truth Table – **T1**

Since the number of possible orderings may be very large even for a reasonable number of variables: Finding for a problem *p* an optimal ordering $\prod_p$, i.e., one which enables the construction of minimal BDDs, is in general NP-complete (c.f. [Bolling 1996]). The first trivial, but important observation we can make, however, is the following:

Observation-1: *It is possible to change any ordering* $\prod_p$ *to* a *canonical one* $\prod_p{}^c$

by renaming variables in the Truth Table.
In the above example: Renaming $x_2>x_0,x_3>x_2,x_0>x_3$ makes the smaller BDD achievable via a Canonical Ordering for S'=$\{\{x_0,x_1\}\{x_0,x_2\}\{x_3,x_4\}\}$ which is equivalent, via renaming, to S.
An important consequence of Observation-1 is that we can focus our attention on the study of conditions under which a Canonical Ordering produces BDDs with small node counts, instead of searching in all Ordering possibilities for suitable choices. This idea leads to the following central Conjecture:

Conjecture: *If during the resolution process in which PR recursively processes any 2CNF Clause Set S:*

1- It always uses Canonical Orderings to instantiate literals in S

2- It makes sure S that respects the conditions under which Canonical Orderings produce small BDDs, transforming S into an equivalent S' if necessary, Then the BDD produced by PR is small.

Therefore, this work has two main objectives:

a- First understand and then formalize the conditions under which Canonical Orderings produce small BDDs
b- Prove the Conjecture.

To get an intuitive understanding of what those conditions may be, we focus our attention on constructing BDDs for S in the above example only using Canonical Orderings. More particularly: We would like to investigate node counts whenever one single clause is resolved against a BDD constructed for the beginning of a Clause Set[4]. (Figure 1-c) shows two starting alternatives for S: S''=$\{\{x_1,x_2\}\{x_2,x_3\}\}$ and S'''=$\{\{x_0,x_4\}\{x_1,x_2\}\}$. Node counts are clearly different. Remembering that (Figure 1-b) depicts the BDD for the whole S, we have therefore two possibilities of node-count-growth from M=2 to M=3, where M is the number of clauses in S: From 4 (S'') to 10 or from 6 (S''') to 10. In both cases we notice a blow-up of the number of nodes resulting from "copying" almost all of previously constructed nodes.

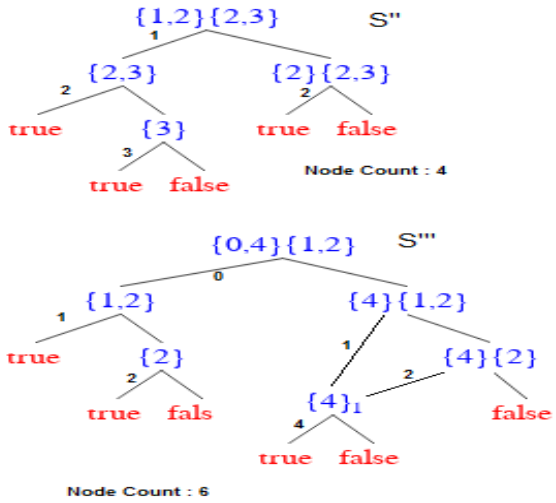


Figure 1-c: Starting alternatives

What about S' ? (Figure 1-d) shows a node-count-growth from 3 to only 5 in the BDDs constructed for S^{iv}=$\{\{x_0,x_1\}\{x_0,x_2\}\}$ and S', respectively.

[4] To do so: PR has to be changed to allow sequential processing of clauses. To avoid unnecessary complication and length: This is only done in the formal part starting with Section III (the 2SAT-GSPRA Procedure of Definition 2). The reader may wish in this section to consider PR capable of sequential processing of clauses and continue reading under this assumption.

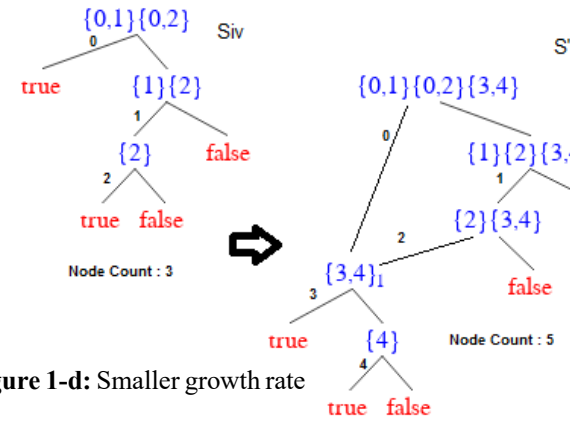


Figure 1-d: Smaller growth rate

Obviously, the nature of growth in the case of S', the formula in which we renamed variables to obtain a Canonical Ordering, is different: The full BDD is constructed from the previous one by just adding two additional nodes to the lowest BDD-level.

How can we explain this?

A second intuitive observation helps in understanding this phenomenon:

Observation-2: *Any variable* x_i *represents in the canonical Truth Table a repetitive pattern of 0s and 1s whose length is* 2^{N-i} *and which is given by the formula:*

$$2^{N-i-1}(0)2^{N-i-1}(1)$$

where N is the total number of variables.

To fully appreciate this observation: A graph may be drawn in which the x-axis represents rows of a Truth Table and the y-axis Boolean values given for a particular 2CNF formula *f*. We call this graph: Pattern-Domain of *f* (PD$_f$). (Figure 1-e) shows for Truth Table T1 PD$_{\{x0,x4\}}$, PD$_{\{x2,x3\}}$, PD$_{\{x2\}}$, respectively. A Pattern Length Repetition of a variable *v* (PLR$_v$) is the number of times a truth pattern of *v* is repeated within the 2^N rows of the truth table. We call the Pattern Length Repetition of the variable with the least index in a clause C/Clause Set S: Pattern Length Repetition of C/S (PLR$_S$/PLR$_C$).

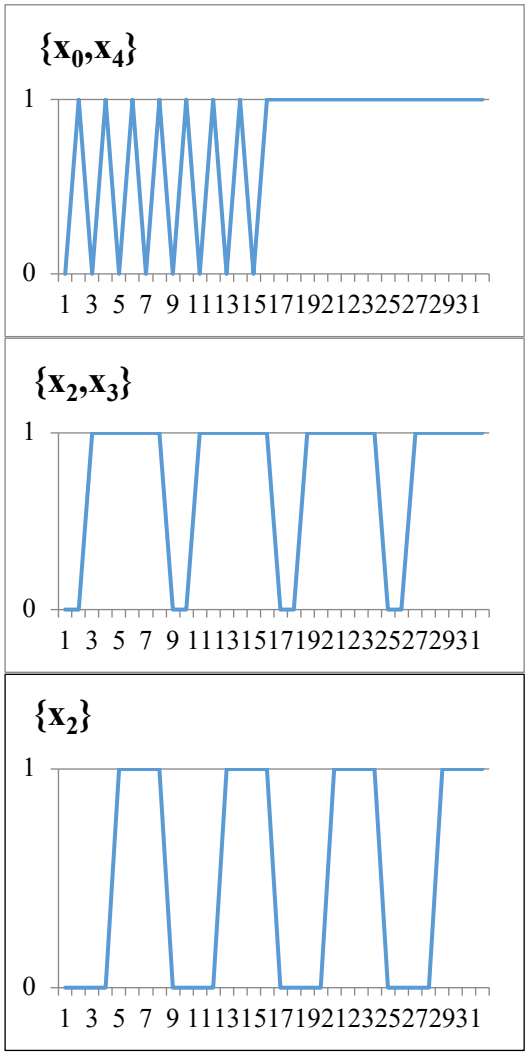


Figure 1-e: Example PDs of different clauses

Using PDs, let's try to explain what happens when we go from the BDD of Clause Set S''={{x_1,x_2}{x_2,x_3}} (Figure 1-c, top) to the one of S={{x_1,x_2}{x_2,x_3}{x_0,x_4}} (Figure 1-b). (Figure 1-f) shows PDs'' and PD$_{\{x0,x4\}}$.

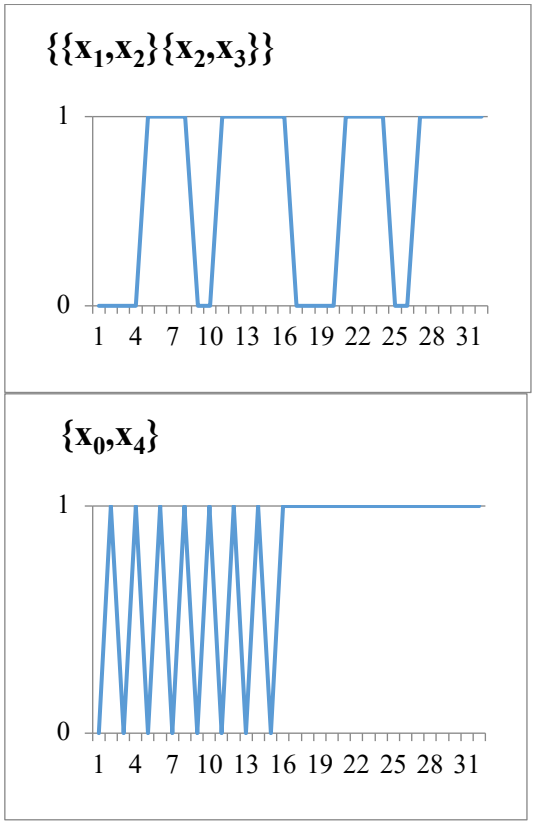


Figure 1-f: PD of an already processed Clause Set S'' (top) compared to the PD of a new clause (bottom)

As seen: $PD_{S''}$ consists of one self-repeating pattern P_1="0000111100111111", where $PLR_{S''}=2$ (i.e., $PD_{S''}=2\ x\ P_1$), P_1 representing the concatenation between sub-patterns for Clause Sets: {2}{2,3}="00001111"& {2,3}="00111111" in (Figure 1-c, top) . When we want to resolve[5] this pattern with $PD_{\{x0,x4\}}=P_2\&P_3$, which has $PLR_{\{x0,x4\}}=1$, where P_2="0101010101010101", P_3="1111111111111111" as seen in (Figure 1-f, bottom), it is clear that we need P_1 to be bit-ANDed against each one of P_2 and P_3. This explains why all nodes of the BDD for S'' had to be copied once as can be seen in (Figure 1-b). Clause {4} is appended there to all copies of such nodes representing the result of bit-AND operation between P_1 and P_2.

Obviously: *Because* $PLR_{\{x0,x4\}} < PLR_{S''}$ *this Copy-Operation (which we also call:* ***Split-Operation*** *or just* ***Split****) was necessary.*
What about PDs of (Figure 1-d)? They are shown in the following (Figure 1-g):

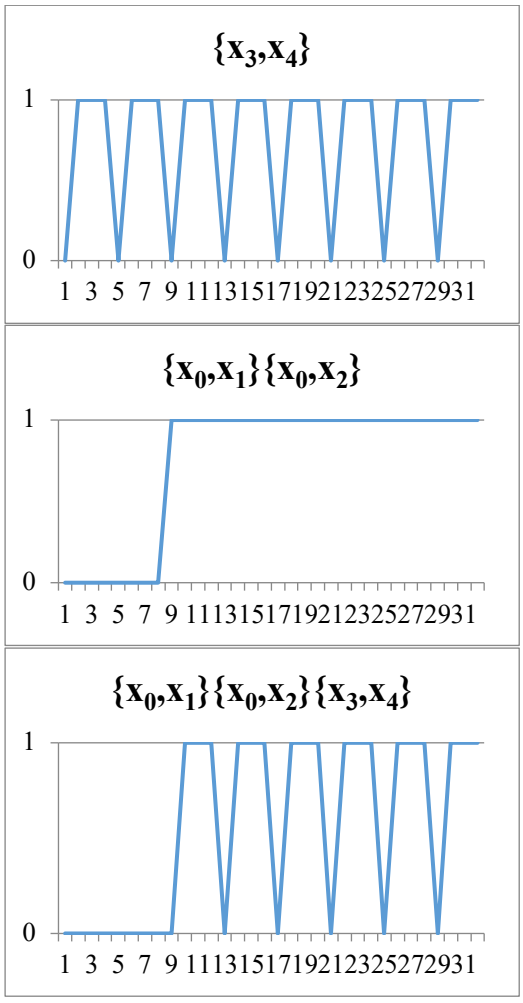


Figure 1-g: PD of an already processed Clause Set {{x_0,x_1}{x_0,x_2}} is bit-ANDed with PD of {x_3,x_4} to form PD of {{x_0,x_1}{x_0,x_2}{x_3,x_4}}

Here the new, to-be-resolved clause C={x_3,x_4} has $PD_C=8x$"0111", $PLR_C=4$

[5] Resolving PD_f with PD_g means: Producing PD_h such that h=AND(f,g).

while $PD_{\{0,1\}\{0,2\}}$=(8*x*"0") & (24*x*"1") is a pattern which repeats itself only once, i.e., $PLR_{\{0,1\}\{0,2\}}$=1. This gives us the opportunity to resolve the new incoming pattern of C with sub-patterns of $PD_{\{0,1\}\{0,2\}}$ *only once* and then refer to the result of this resolution whenever needed. This is reflected in the BDD by including node {3,4} (Figure 1-d, bottom) as a *common sink* between two constructed branches, thus reducing drastically the total amount of unique nodes.

Resuming this motivation example: *We can use the Pattern Domain of* a *2CNF formula f (*PD_f*) to explain blow-ups in the number of nodes generated by sequential resolution procedures which use Canonical Orderings to produce BDDs. It turns out that resolving a clause C with a Clause Set S, where* $PLR_C<PLR_S$ *necessitates Split-Operations. Such Operations are important causes of BDD blow-ups. In the case of S' above we have also seen that sequentially resolving Clause Sets S with* a *clause C does not induce Splits when* $PLR_C>PLR_S$*. We call this condition: Linear Order (l.o.). The core of this work is formally showing that Algorithms observing the l.o. condition always produce small FBDDs.*

Are there any sources of BDD blow-ups other than Split Operations caused by procedures not observing l.o. conditions? An important part of this work is also dedicated to showing that nodes which are sinks between branches (also called: Common Nodes (CNs)) may also cause Splits. Fortunately and precisely because of the l.o. condition: Those Splits are benign, i.e., *they do not cost, for each CN, more than* a *constant number of additional nodes per inductive resolution step.*

II USED METHODS

This work is a second application of ideas presented in [Abdelwahab 2016-1] for solving hard problems, the first being published in [Abdelwahab 2016-2] related to 3SAT. At the core of those two publications is a 3SAT-Solver producing small FBDDs by enforcing l.o. conditions on all resolved Clause Sets. In the present work, this Solver is modified to be applicable to the 2CNF case and may informally be described as per the following high-level pseudo-code and Flowchart of (Figure 1-h)[6]:

PR⁺:
Inputs: Arbitrary 2CNF Clause Set **S**
Output: FBDD
Steps:

1- Use the Renaming and Sorting Algorithm (CRA^+, Definition 9) to convert S to an equivalent l.o. Clause Set S', i.e., Set S'=CRA^+(S).
2- Select the least Literal *x* from the first clause C∈ S'.
3- Instantiate S' using partial Assignments: {*x*=TRUE}, {*x*=FALSE} forming left- and right-Clause Sets S_1, S_2, respectively
4- If either S_1 or S_2 are evaluated to TRUE or FALSE, create left/right TRUE/FALSE-nodes in the respective case.
6- If neither S_1 nor S_2 are TRUE/FALSE and are found in a Resolved-ClauseSet-Store: Call yourself recursively first with S_1, then with S_2, forming leftResult and rightResult, respectively. Otherwise: Call yourself only for the Clause Set is new and not TRUE/FALSE and return the BDD stored for the other.
7- Form the finalResult from Clause Set S', leftResult and rightResult.

Algorithm – A2

(Figure 1-i) shows the actual resolution of F={{x_1,x_3}{x_2,x_4}{x_0,x_2}{x_1,x_4}{x_2,x_4}} using 2SAT-FGPRA after it is converted in to a l.o. Set S=CRA^+(F)).

[6] Algorithm 2SAT-FGPRA in the Theory section is the concrete, detailed counterpart.

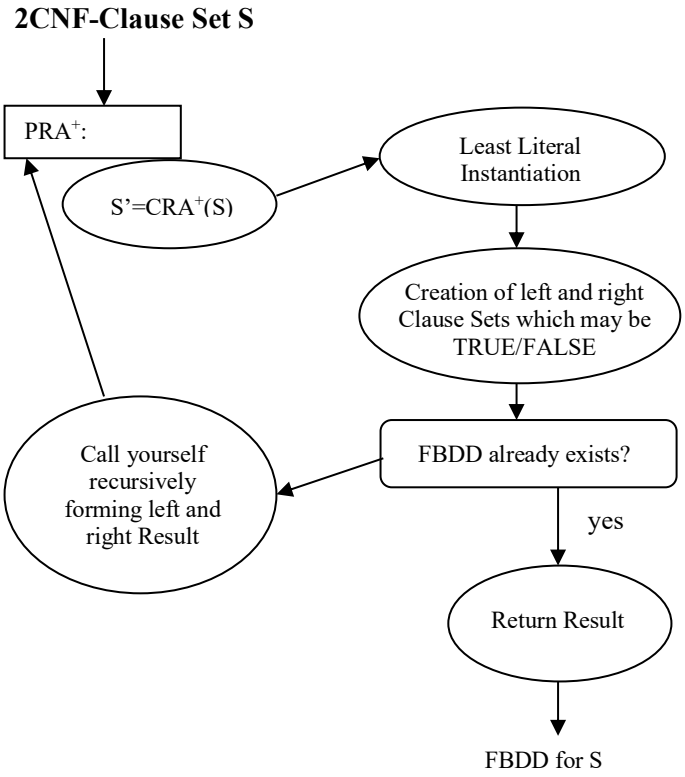


Figure 1-h: Flowchart of Algorithm – A2 (PR^+)

Before going into a discussion of the mentioned publications, showing differences between methods described therein and modifications/adaptations used in this work, known state-of-the-art literature is briefly described. From the vast literature around #2SAT, BDDs/FBDDs and NP-completeness, we have chosen only those research findings which relate to our work or bear possible challenges to our results.

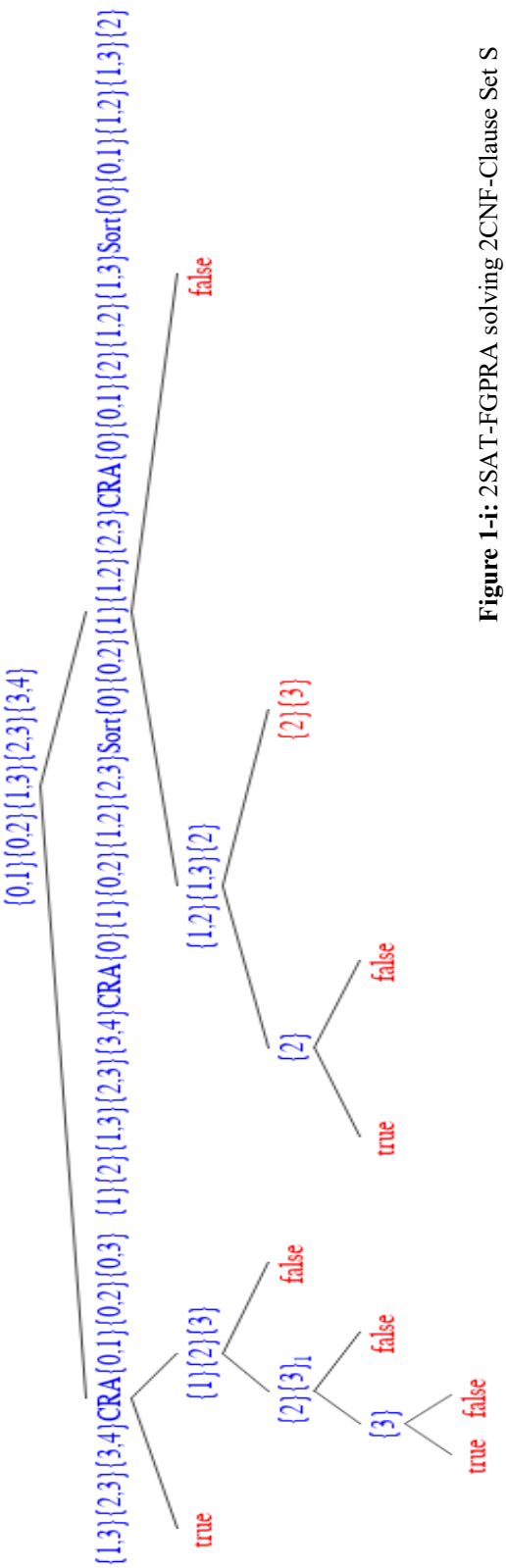


Figure 1-i: 2SAT-FGPRA solving 2CNF-Clause Set S

II-1 Exponential Lower Bounds on FBDD Construction Revisited

Most important BDD/FBDD properties are known since the 80s and 90s of last century and represent well established facts which contributed to the commonly accepted idea that: *Some important Boolean Functions can only possess large BDDs and/or FBDDs and there are no ways to overcome this restriction.* We discuss the seemingly apparent contradiction between our findings and this consensus, despite of the fact that, because of the existence of polynomial reductions, exponential lower bounds proven in literature are targeting mainly Boolean Functions expressible in *k*- or 3CNF while the work here concerns 2CNF formulas.

Exponential lower bounds for BDDs are known for Ordered Binary Decision Diagrams (OBDDs), which are the best studied forms of BDDs and which only need one variable order to govern instantiations of Clause Sets. Alternatively: An FBDD allows the flexibility to choose a different order for each branch. There are many BDD variable ordering heuristics in literature, but the most common way to deal with ordering is to start with something "reasonable" and then swap variables around to improve BDD size. This dynamic variable reordering is called sifting [Rudell 1993]. The overall idea is: Based on a primitive "swap" operation that interchanges x_i and x_{i+1}, pick a variable x_i and move it up and down the order using swaps until the process no longer improves the size. The reader may have noticed that the above procedure PR^+ (Algorithm - A2) *does not perform any Variable/Literal selection-trials and just proceeds, after converting the Clause Set to a l.o. one, by instantiating the least Literal of the first clause.*

The first exponential lower bounds on the size of FBDDs have been proven as early as 1984 by [Zak 1984] and [Wegener 1988]. In his seminal paper *Bryant* also showed [Bryant 1986] that integer multiplication is a function which cannot have a small OBDD irrespective of the variable ordering used. Later, this result was also extended to the FBDD case. A long list of papers, which reported similar results for Boolean functions such as: Hamiltonian-Circuit, Perfect Matching, Clique-Only, Triangle-Parity, Blocking Sets in finite projective planes etc. followed or were published in the same period. In [Wegener 2000] a lower bound technique which is influenced by the algorithmic point of view following [Sieling 1995] is used to explain the methodology behind the majority of results. It turns out that variants of the following observation were constantly used:

***"Lemma:** Let f be* a *Boolean function of n variables. Assume that m is an integer, 1 < m < n, if for m any m-element subset Y of the variables N(f, Y) = 2^m holds*[7]*, then the size of any read-once branching program computing f is at least 2^{m-1}."*

[7] N(*f*,Y) denoting the number of different sub-functions obtained under all possible assignments to Y.

[Abdelwahab 2016-1, Theorem 3] shows that lower bounds related to the construction of FBDDs obtained using the above *Lemma* are bypassed by Sequential Pattern Resolution (SPR)-like Algorithms using 3CNF representations. The direct reason for that being the fact that: While the proof of *Lemma* requires the first *m*-1 levels of any FBDD constructed for such a function to constitute a complete binary tree, SPR-like Algorithms using 3CNF formulations *always* form trees which are bound to reach leaves after at most *k*<=3 instantiation steps in any tree-level (Property 8 [Abdelwahab 2016-2], Section II).

Most of the problems for which lower bounds were proven using this *Lemma* (for example: the "blocking-sets in projective planes" problem shown in [Gal 1997]) are described in *k*CNF formulations which reflect/preserve the exact problem structure, i.e., in the projective planes example: Every plane is exactly one clause and every point is exactly one variable. [Abdelwahab 2016-1] calls such descriptions preserving all properties of decision structures of a problem as well as interrelationships between those structures: *Reserved Descriptions*.

Let *f* be a Boolean Function for which an exponential lower bound LB on the size of the FBDD is obtained, *f'* an equisatisfiable 3CNF formulation of *f*. The reasons why LB isn't applicable to *f'* can be informally summarized in the following points[8]:

1- If *f* has a reserved *k*CNF description, it is sometimes the *only* way to guarantee that, for any *m*-element subset Y of the input variables of *f*, different sub-functions obtained under all possible assignments to Y are truly distinct. For example in the projective planes case we quote the following part of the lower bound proof [Gal 1997], page 15:

"*Proof of the theorem. We show that for every q-element subset A of the variables, N(f_{II}, A) = 2^q holds, i.e., each truth assignment to the variables in A yields* a *different sub-function on the remaining variables.* ***Since each line defines*** a ***clause of the function*** *f_{II}, it follows from the Fact*[9] *that for an arbitrary q-element subset A of the variables there exist q clauses such that each variable from A appears in exactly one of them, and each variable appears in* a *different clause.*"

Obviously: Because *f'* is formalized in 3CNF, a line for projective planes with *q*>3 cannot be represented by only one clause making the above Argument inapplicable.

2- From the logical point of view, *f* and *f'* are *not* equivalent. This means that Deterministic FBDDs constructed for them are not expected to be equivalent[10]. It also

---

[8] Formalizations of the ideas expressed in the points here are not attempted to avoid unnecessary length.

[9] *Lemma* could only be applied to the blocking Sets problem, because of the following combinatory property shown to hold for projective planes [Gal 1997]: *"Fact: Let J={p1,...,pt} be* a *set of t<=m distinct points of the projective plane P, then there exist distinct lines {l1,...lt} such that for i>=1, j <=t we have pi ∈lj iff i=j."*

[10] Let G_f, $G_{f'}$ be FBDDs of *f*,*f'* respectively, then: *f* = *f'* iff $G_f(a) = G_{f'}(a)$ for all $a \in \{0,1\}^n$, where

means: There may be models for f which are not models for f' and vice versa. As f and f' are equisatisfiable, they may disagree for a particular choice of variables. As a matter of fact: A typical equisatisfiable translation from kCNF to 3CNF usually looks like: $(A \vee B \vee x_1) \wedge (\neg x_1 \vee C \vee x_2) \wedge (\neg x_2 \vee D \vee E)$
For a k=5 clause C=$(A \vee B \vee C \vee D \vee E)$ for example. Note that while C has a model in which B=TRUE, x_2=TRUE and all other variables including x_1 are FALSE, this is not a model for the translated 3CNF formula. In such constellations: The number of variables in clauses of f' are strictly larger than the number of variables in clauses of f and consequently: Sub-function properties, necessary for the application of the above *Lemma* are disturbed by the introduction of new variables which have no place in the definition of f and must be treated as *Don't Cares*, i.e., variables whose truth values don't matter for the overall truth-value of the formula. Treating variables as *Don't Cares* makes the FBDD Non-Deterministic, causing all lower bounds for Deterministic FBDDs to be inapplicable[11].

3- Let LB be an exponential lower bound on the size of any Non-Deterministic FBDD[12] constructed for f, as the one given in [Sauerhoff 2003] for example, not necessarily using *Lemma*. Call an efficiently constructed Non-Deterministic FBDD: *pNFBDD* and an efficiently constructed Deterministic FBDD: *pFBDD*, then: For LB to be applicable on procedures using f', something like: "*A pNFBDD exists for f iff* a *pFBDD exists for f'*" must be true[13].

Although starting with a *pFBDD* for f', a *pNFBDD* for f is easily constructed by erasing all markings which represent variables not in f (call a Set containing them: Z), the other way around is not obvious. Starting with a *pNFBDD* for f, in which some nodes are unmarked does not give any clue to how markings must be put such that a procedure produces a *pFBDD* for f'. Correct markings have to be properly "guessed" indicating that this translation may be hard[14].

$G_f(a)$ denotes the leaf node value obtained from G_f for input string a [Wegener 2000].

[11] It must be mentioned here that introducing new variables is known, since the 90s, to disturb exponential lower bounds obtained for multiplication-BDDs for example. In [Burch 1991] a method for using BDDs to verify multipliers while avoiding exponential complexity is shown. Normally the outputs of an n by n bit multiplier circuit are represented by BDDs with $2n$ variables, since the circuit has $2n$ inputs. In the method described there, the outputs of the circuit are represented by a BDD with $2n^2$ variables, instead. The size of this BDD is cubic in n.

[12] Recall: A Deterministic FBDD is a FBDD in which every node is marked with a variable name, while a Non-Deterministic FBDD has some unmarked nodes [Wegener 2000] .

[13] Note that if f and f' are equivalent, agreeing on all used variables, this is trivially true.

[14] To see this: Suppose G_f is a *pNFBDD* for f and suppose there exists an input a, such that $G_f(a)$=TRUE. This means that there is a path P in G_f leading to a TRUE node. P may contain unmarked nodes $\{un_1, un_2, \ldots un_i\}$. If we attempt, using G_f, to construct a *pFBDD*, say $G_{f'}$, for f', we need to mark $\{un_1, un_2, \ldots un_i\}$ with names of variables from Z such that a path P' in $G_{f'}$ (corresponding to P) leads to a TRUE leaf. There are two ways to do so: Either all possibilities of assignments for variables in Z must be explicitly extended creating in the worst case an exponential sub-tree in $G_{f'}$ rather than only one single path, or

II-2 #2SAT Solution Methodologies

There are two types of approaches related to counting problems: Ones which aim at improving known exponential bounds on finding exact solutions and others which seek better approximations. As we are presenting in this work a method for exact counting, we will focus in this section on describing the known state-of-art in this category and underline differences to our proposed method. We discuss also results from parametric complexity which use some notion of 'truth patterns' to reduce the effort needed to bound the number of solutions more tightly.

In exact counting, methods based upon DPLL-style exhaustive search and those based on what is called *knowledge compilation* are distinguished. The method presented here can be classified as a knowledge compilation method, in which a given CNF formula is converted into a FBDD from which the count can be deduced easily, i.e., in time polynomial with regard to the size of the formula. One advantage of this methodology is that once resources have been spent on compiling the formula into this new form, complex queries can potentially be answered quickly.

State-of-the-art methods of this type are best represented by the ones using deterministic, decomposable negation normal forms (d-DNNF) as described in [Darwiche 2002], which are generated by an exhaustive version of the DPLL procedure called c2d. Those forms were created to provide alternatives for BDDs, which could, in principle, be constructed and then "read off" for the solution count by traversing the BDD from the leaf labeled "1" to the root. BDDs are commonly not used for this purpose, because of the consensus regarding exponential lower bounds discussed in the previous section. Compilation of a given CNF formula F into d-DNNF is done via c2d by first constructing a so-called decomposition tree, which is a binary tree whose leaves are tagged with the clauses of F and each of whose non-leaf vertices has a set of variables, called the separator, associated with it. The separator is the set of variables that are shared by the left and right branches of the node, the motivation being that once these variables have been assigned truth values, the two resulting sub-trees will have disjoint sets of variables. The resulting components can then be easily combined using AND nodes [Handbook of Satisfiability 2009]. In [Beame 2013] a special case of d-DNNF formulas, called decision-dDNNF, is shown to be convertible to FBDDs with only a quasi-polynomial increase in representation size in general, leveraging known exponential lower bounds for FBDDs, to exponential lower bounds for decision-DNNFs. The power of decision-DNNFs is separated from d-DNNFs and a generalization of decision-DNNFs known as AND-FBDDs is described as well. This implies exponential lower bounds for natural problems associated with probabilistic databases (c.f. [Beame 2013]).

Algorithms for specifically counting solutions of 2SAT can be found in, e.g., [Fuerer 2007]. The idea is an extension of a research direction focusing on 2SAT problems, where every variable occurs

different assignments of those variables are deterministically tested against f'. Both options don't qualify as 'efficient construction' procedures.

x-times at most, obtaining the best time of $O(1.246069^n)$ for counting models and max-weight models, n number of variables, achieved also in polynomial space. The decisive parameter determining the running time of the proposed Algorithm is the number of degree x=3 nodes. Progress in eliminating those nodes is possible when there are many of them, i.e., when the average degree is higher. In that case: A degree 3 vertex in the constructed graph with a neighbor of degree 3 is found more frequently and they can both be eliminated in the same time. The improved time bounds for degree 3 propagate to formulas of higher degrees, because the average degree has a tendency to shrink during the iterative algorithm's run (c.f. [Fuerer 2007]).

In [DeItaLuna 2012] a method is described where given a formula F, #2SAT(F) can be bounded above by considering a binary pattern analysis over its set of clauses. For each clause $C_i = \{x_j, x_k\}$, A_i is a set of binary strings, called: 'binary pattern', such that the length of each string is n, the number of variables. The values at the j-th and k-th positions of each string, $1 \leq j, k \leq n$ represent the truth value of x_j and x_k that falsifies C_i. E.g., if $x_j \in C_i$ then the j-th element of A_i is set to 0. On the other hand, If $x_j \in C_i$ then the j-th element of A_i is set to 1. The same argument applies to x_k. Using this notion of a 'pattern' it can be shown that for $F = \{C_1, C_2,...,C_m\}$, a 2CNF formula, n variables, $m \geq 2$: The hard cases to answer whether #2SAT(F)=k, are given when $m>n$. *This is one of the rare occasions in the literature of hard problems, where a formalized notion of 'truth patterns' is used to reveal intrinsic properties of logical formulas.*

Before going into the next section, where we distinguish this work from [Abdelwahab 2016-2], we summarize important findings of the previous two sections in the following points, underlining differences between known #SAT solutions and our presented one:

1- Exact counting of solutions can be done using exhaustive knowledge compilation methods which avoid BDD construction, because of the consensus that BDDs possess exponential lower bounds for important Boolean Functions and may thus become large in the worst case.

2- Using an equisatisfiable 3CNF representation f' of a Boolean Function f makes lower bounds obtained for Deterministic-FBDDs of f inapplicable, because of the additional variables in f'. Polynomial Non-Deterministic FBDDs of f fail to capture polynomial Deterministic-FBDDs of f', rendering lower bounds for Non-Deterministic FBDDs of f out of scope as well. This paves the way to the usage of SPR-like methods constructing FBDDs like the ones published in [Abdelwahab 2016-2] to efficiently solve #SAT, especially knowing that conveniently, many of the known reductions between NP-complete problems, including those related to 3SAT, are *parsimonious*, i.e., they preserve the number of solutions during the translation[15].

3- Independent of the above points: The present work is concerned with the construction of FBDDs for 2CNF formulas. *To the best of our knowledge: There are no lower bounds, susceptible to challenge our results, for this special case.*

[15] Note that a Cook-Levin reduction is actually parsimonious. Cook-Levin (Restated): For every language L ∈ NP, there is a parsimonious reduction from L to SAT.

II-3 Similarities and Differences between previous and current work

[Abdelwahab 2016-2] was set up to prove two related assertions:

1- That SPR Algorithms described there (GSPRA$^+$, FGPRA) *always* produce small FBDDs for 3CNF formulas.
2- That they are efficient 2-Approximation Algorithms for MinFBDD, an NP-complete problem.

Although the first point was enough to demonstrate the main theoretical result, it was necessary to provide evidence, that the used Algorithms have practical value as well. Similar to procedure PR$^+$ (Algorithm – A2): GSPRA$^+$ and FGPRA apply, using CRA$^+$, the l.o. condition on *all* Clause Sets generated during resolution. In the same time: Creation of new Clause Sets via instantiation is *solely* done using least literals. The final output being a special form of DAGs we call also here MSRT$_{s.o}$, whose main features are:

a- Nodes contain Clause Sets
b- Variables in a Clause Set may be renamed one or more times in the same branch. Sequences of such renaming operations are called: Variable Space.
c- MSRT$_{s.o}$s can be easily converted to FBDDs by abstracting the least Variable/Literal index of every Clause Set.

The essential difference between this work and [Abdelwahab 2016-2] is the way in which formal concepts are defined, namely: Keeping definitions as close as possible to Set- and Graph-Theory. This facilitates proofs of relevant lemmas and makes them more accessible to readers than their counterparts in [Abdelwahab 2016-2]. New proofs for previously not shown properties of MSRT$_{s.o}$s (like the fact that no N-Splits can exist in such graphs for example) are also important additions.

Table T2 gives an overview of essential formal similarities and differences.

Concept, Algorithm, Proof	**Previous formalization**	**Current formalization**
Linearly Ordered (l.o.), Linearly Ordered, but unsorted (l.o.u.) Clause Sets	Structural property of Clause Sets	Same as before + Var/Literal Index comparison Relation "<" is characterized by the Literal precedence Relation "|" (**Definition 8.6**)
N-Splits, CN-Splits, BigSps	Copies of nodes	Special forms of Clause Sets occurring in a MSRT$_{s.o}$s (**Definition 6**)
MSRT$_{s.o}$	Special form of DAG	+SR-DAG formally defined + Special form of SR-DAG (**Definition 10**)
Variable Space (VS), CN/MSCN, tCN	-Variable Space: Sequence of CRA$^+$ Operations, -CN: Sink node -MSCN: Sink node in a VS -tCN/tMSCN: CNs/MSCNs produced in Symmetric Blocks	Same as before
CRA,CRA$^+$	Properties shown: - Termination - Correctness - Complexity	Same as before + $(x \mid y)$ iff $(x<y)$ **(Lemma 1-b)** + S and CRA$^+$(S) are equisatisfiable + They are also equivalent **(Lemma 2-b,c)**
Lemma 9-a: No BigSps	Shown using "<" Relation and l.o. property of Clause Sets of parent nodes of a CN/MSCN	Shown using "|" Relation and l.o. property of the Base Clause Set (BS)
Lemma 9-b: No N-Splits	Not shown	Shown using new characterization of Splits
Lemma 9-c: No size>1 Splits	Shown using the "<" Relation, CNAL properties	Shown using the "|" Relation, CNAL properties and BS l.o. property
Lemma 14: Counting Solutions	Not in the scope	Shown using DAG properties

Theorem 1: Sufficient conditions which guarantee the efficiency of SPR-like *k*SAT-Algorithms	Not in the scope	Shown using induction on *k* and (**Lemma 11)**
SPR Resolution procedures	GSPRA$^+$,FGPR: -produce optimal Top-Parts -their output is equivalent - FGPRA is efficient, 2-approximative to MinFBDD	2SAT-GSPRA$^+$, 2SAT-FGPRA: 1- Top parts are not shown to be optimal 2- 2SAT-FGPRA simulates 2SAT-GSPRA$^+$ correctly (**Lemma 11**-a) 3- 2SAT-FGPRA is efficient (**Lemma 11-b**)

Overview of differences and similarities to our previous work – T2

II-4 How to read this paper

The *Conjecture* formulated in the introduction of this work includes claims which bear important consequences requiring an extra effort to organize formal concepts and/or proofs thereof in such a way, that the overview is not lost, while readers attempt to check correctness. For this purpose the following tools are made available for use throughout this whole document:

1-All formal Concepts, Algorithms, and Proofs are explained with examples while expressing them as close as possible to Set Theory for formal concepts and concrete near-to-C pseudo-code for Algorithms, highlighting exact formal definitions always in **bold**. Cross-References to definitions are availed to simplify reading.
2- Lemmas, Figures and References are cross-referenced (in pdf-file format).
3- All Acronyms used are highlighted in **blue bold** when they are defined for the first time.
4- All concepts are listed in a comprehensive table in Appendix A, where Acronyms, formalizations, their place in definitions (including page numbers and links), lemmas using them as well as their actual purpose are included.
5- Selected lemmas and their dependencies on formal concepts are listed in Appendix B.
6- A table of content (first page) is provided to facilitate overview as well as referencing of content.
7- (Figure 1-j) below shows interdependencies between lemmas and links them to Theorem 1. Although all lemmas are important, parts marked ***green*** represent the most crucial pieces of information, ***sufficient alone*** to produce the main result one time, followed in importance by ***blue*** marked parts. Coloring parts intends to help readers first find critical flaws in our argumentation more easily and second distinguish between the two presented results in the following way:

i- In a ***first quick scan***, a reader may wish to consider only the ***green*** path, where one can verify the $O(M^6)$ bound of (Lemma 10) on the upper size of the $FBDD/MSRT_{s.o}$, shown to hold under the assumption that (Lemma 9-c) is relaxed, i.e., only N- and BigSps cannot be produced, as follows[16]:

a- Concepts: l.o./l.o.u. 2CNF Clause Sets (Definition 1), (Variable Space) , (CNs/MSCNs) , Splits (N-, as well as

[16]It is commonly known that BDDs admit efficient Algorithms for counting solutions after being built. Therefore: Verifying that node counts cannot exceed $O(M^6)$ for any 2CNF formula is the essential effort a reader may want to do in order to accept the second proof of the main result of this work, i.e., Theorem 1-b.

CN-Splits), (Alignment $MSRT_{s.o}$s) are all well-defined.

b- Algorithms (CRA), (CRA^+), (2SAT-$GSPRA^+$) and (2SAT-FGPRA) are sufficiently detailed and their way of work clearly described.

c- It is *always* possible to convert an arbitrary 2CNF Clause Set to a l.o. one using CRA^+ (*Lemma 2-a*). If necessary, this is done in each recursive step by 2SAT-$GSPRA^+$. CRA^+ delivers Clause Sets which are not only equisatisfiable (*Lemma 2-b*), but also equivalent (*Lemma 2-c*) to the original Clause Set. CRA^+ is also efficient (*Lemma 3*).

d- Mappings produced by CRA are monotone and the Literal precedence Relation '|' is an exact characterization of the trivial Index comparison Relation '>' (*Lemma 1-a, b*). This information is used in the *crucial* proof of (Lemma 9-a).

e- Splits are the actual causes of exponential behavior. While N-Splits are taken care of in the definition of the Canonical Ordering criteria (especially the l.o. condition as has been seen) and thus avoided altogether by 2SAT-$GSPRA^+$ (*Lemma 9-b*), CN-Splits may still occur.

f- CN-Splits cannot occur for nodes of rank>1 (BigSps) (*Lemma 9-a*).

g- 2SAT-$GSPRA^+$ repeats the construction of sub-trees for Clause Sets of sub-problems when they are found to be breaching the l.o. condition. This makes sure that any CN/MSCN at size-level j is only a CN/MSCN at size-level j-1 augmented by a newly resolved clause during re-construction (*Lemma 5-b*), *i.e., the number of CNs/MSCNs is preserved (in the worst case) when they move up the hierarchy of size-levels.*

h- No more than $O(M^2)$ nodes can be created in the lowest j=1 size-level during the whole process of resolution (*Lemma 7*)

i- Rank 1 nodes (i.e., those which have only unit clauses) produce only O(M) new nodes when they split (trivial)

j- All this leads to the $O(M^6)$ upper bound of (Lemma 10, point 4).

k- 2SAT-$GSPRA^+$'s repetitive construction of sub-trees causes redundant operations which are avoided altogether by 2SAT-FGPRA. 2SAT-FGPRA is a practical Algorithm in which all clauses of a Clause Set are instantiated with values of the chosen least Literal in the same time. It simulates 2SAT-$GSPRA^+$ correctly (*Lemma 11-a*).

l- The worst implementation of 2SAT-FGPRA requires comparing all created nodes with each other and always using CRA^+ to rename their Clause Sets, making the overall asymptotic complexity $O(M^{13})$, because Lemma 9-c is relaxed (*Lemma 11-b*).

ii- In a ***second scan*** readers may want to study (Lemma 9-c) (**blue** path), which shows that CN-Splits cannot occur in size-levels j>1. This reduces the upper bound of the nodes count of the FBDD/$MSRT_{s.o}$ to $O(M^4)$:

a- As before: The *only* new nodes added by 2SAT-$GSPRA^+$ to the lowest, size-level j=1 in any step and at any time can't be more than $O(M^2)$ nodes. As 2SAT-$GSPRA^+$ is sequential: Those nodes form, at each step, the basis for size-j-level nodes, j>1, and may be propagated up in the hierarchy of levels, making the maximum amount of nodes in each such level j during the whole resolution process not exceeding the upper bound of nodes at level 1 (*Lemma 5-c*).

b- Either trivial- or rank 1, size 1-CNs can split (*Lemma 8, Lemma 9-b*)

making the maximum amount of nodes added in this way to the lowest level also $O(M^2)$, since one such Split causes, in the worst case, a constant amount of nodes to be created on the size-level it occurs in.

c- The final FBDD has in the worst case a total unique node count of only $O(M^4)$ (*Lemma 10*).

d-To count Assignment possibilities: An Algorithm *Count2SATSolutions* traversing in the worst case all nodes and edges of the FBDD/$MSRT_{s.o}$ is used. As both 2SAT-GSPRA$^+$ and 2SAT-FGPRA are complete, Truth-Table equivalent Algorithms (*Lemma 12*), *Count2SATSolutions* is shown to be counting exact solutions correctly (*Lemma 13*). To do so: It requires $O(M^9)$ or $O(M^{13})$, in case Lemma 9-c is relaxed (*Lemma 14*).

e- One main result, (*Theorem 1-a)* shows *conditions* under which SPR-Algorithms solving *k*SAT-problems become efficient (**green** path). It turns out that avoiding both N- as well as big CN-Splits are *sufficient conditions* for polynomial time performance. In the same time: The uniform way of expressing node counts and time complexity of base case k=2 in terms of base case k=1 makes it possible to demonstrate P=NP by formulating and using the strongest induction hypothesis possible. This is what is gained by relaxing Lemma 9-c.

f- Because *Count2SATSolutions* is in P (both **green** and **blue** paths), P=NP follows also this way (*Theorem 1-b*).

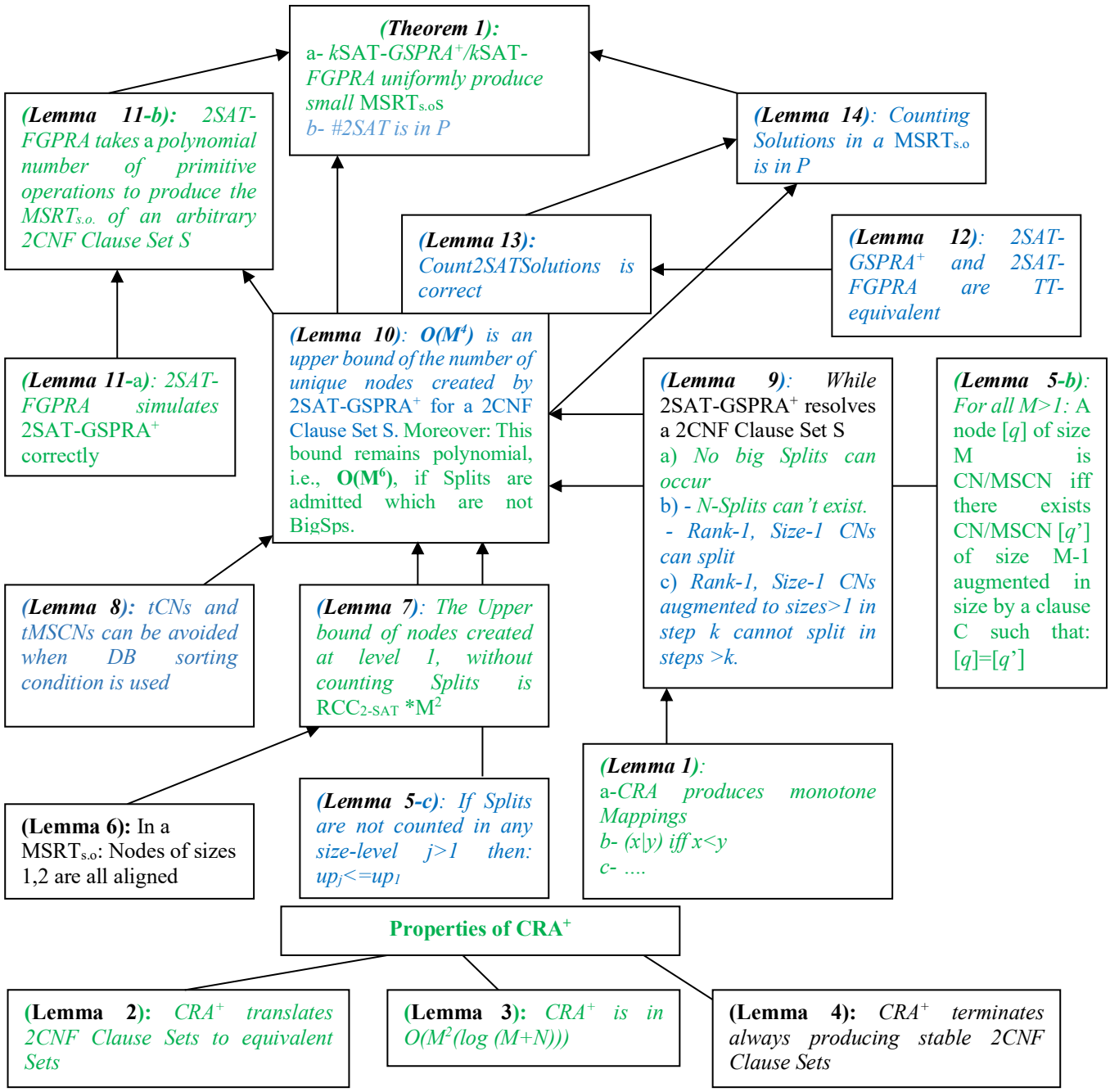


Figure 1-j: Interdependencies of Lemmas

III THEORY

III-1 Definitions

Definition 0: (Nomenclature and Basic): Variable, Literal, Clause, 2CNF Formula/Clause Set, Truth Assignment, Partial Assignment, Restricted Assignment, 2SAT Decision Problem, Graphs, Vertices/Nodes, Edges, adjacent Vertex, Source, Target, reachable, Child, Parent, Base Node, Path, Branch, acyclic, Length of Path/Branch, Directed Acyclic Graph, Source Path of node *n*, Level of node *n* in a DAG, Level of edge *e* in a DAG, Topological Ordering of a DAG, Sequential Resolution DAG, 2CNF Clause Set of a node, Base Clause Set of a node, TRUE-DAG, FALSE-DAG, rankC, rankNode, Size of a node *n*, Size of a 2CNF Clause Set S, Top-Part of a SR-DAG, LeftDAG, RightDAG, literals in a 2CNF Clause Set S, literals of a 2CNF Clause Set S to the left of Literal *x*, SortOrder, Head-Literal, Tail-Literal, Connectivity of a Literal *x* in 2CNF Clause Set S, Permutations of a Clause, Resolution Complexity Coefficient, Instantiations of literals, Satisfiability, Derivations of a Clause, Linear Derivations of a Clause, InstSimple, InstSimpleC, Convert a Clause to SR-DAG, First occurrence of Literal *x* in a 2CNF Clause Set S, Select a Literal *x* of a 2CNF Clause Set S

Definition 0.1: Consider a finite Set of Boolean variables Var=$\{x_1, x_2, \ldots x_n\}$

a- A Literal is either a Boolean variable x_i or its negation $\neg x_i$. Indices deduced from enumerations are also used to stand for Literal names. The relation '***a*<*b*'** expresses the fact, that index a of some Literal is smaller than index ***b*** of another in a given enumeration.

b- A clause is a disjunction of literals. For example, $(x_1 \vee x_2)$ is a clause.

c- A Formula/Clause Set in conjunctive normal form (CNF) is a propositional formula in which clauses are connected using the Boolean AND operation. For example: $(x_1 \vee x_2) \wedge (x_2 \vee \neg x_3) \wedge x_5$ is a CNF formula.

d- A formula ϕ is a 2CNF when every clause has exactly 2 literals. For example $(x_1 \vee x_2) \wedge (x_2 \vee \neg x_3)$ is a 2CNF formula, but $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge (x_5)$ is not.

e- A Truth Assignment is a total Function f:Var =>{0,1}. When f is partial, the assignment is called Partial Assignment. When f is restricted to only one variable it is called Restricted Assignment.

Definition 0.2: 2SAT Decision Problem: Given a 2CNF formula ϕ, is there a Truth Assignment such that ϕ evaluates to true?

Definition 0.3: A graph G = (V,E) consists of a finite set of Vertices/Nodes, V, and a finite set of Edges E.

- Each edge is a pair (v,w) where $v, w \in V$
- A Directed Graph, or Digraph, is a graph in which the edges are ordered pairs: $(v, w) \neq (w, v)$
- In the Digraph: ***b*** is called adjacent to ***a*** when there exists an edge $(a, b) \in E$, also:
  - Node ***a*** is *not* adjacent to node ***b***.
  - Node ***a*** is called predecessor of node ***b***, node ***b*** is a successor of node ***a***
  - The Source of the edge is node ***a***, the Target is node ***b***.
  - Node ***b*** is called reachable from node ***a*** if ***b*** is adjacent to a or there

is a non-empty list $<e_1,e_2,\ldots,e_n>$[17] of edges connecting, indirectly, a to ***b***. Node ***b*** is also called in that case Child of node ***a***, a Parent of ***b***. Boolean Predicates Child(n_1:Node, n_2:Node), Parent(n_1:Node,n_2:Node) are formally used to express this fact

- Base Node (BN) of G is the source of its first edge.
- A Path/Branch is a list of vertices $<w_1, w_2,\ldots w_n>$ such that for all the edges:
  $(w_i, w_{i+1})\in E$, $1 <= i < n$, and each vertex is unique except that the path may start and end on the same vertex if G is cyclic.
- An acyclic Path is a Path where each vertex is unique
- The length of the Path/Branch is the number of edges along the path
- A directed graph which has *no* cyclic paths is called a DAG (Directed Acyclic Graph).
- Source Path of a node ***n*** in a DAG (SP_n) is a list of edges connecting ***n*** to the Base Node: $\mathbf{SP_n=<e_1, e_2,\ldots e_m>}$, $\mathbf{e_i}$**:Edge.** A node may have several non-empty Source Paths and is always reachable from the Source.
- Level of node n (L_n) in a DAG is an integer representing the number of edges in the longest Source Path connecting n to the Base Node. It is given by: $\mathbf{L_n=Max(length(SP_n^1)..length(SP_n^k))}$ where any SP_n^i is a Source Path of n.
- Level of an edge e (L_e) in a DAG: $L_e= L_{Sr}+1$ if Sr is the Source of e.
- A Topological Ordering (TO) of a DAG is an ordering of its nodes such that:
  $\mathbf{\forall e}$**:Edge,** $\mathbf{e=(v_i,v_j)}$, $\mathbf{v_i,v_j\in V: i< j}$.
- A DAG formed for a 2CNF Clause Set BS and whose nodes contain 2CNF Clause Sets is called a Sequential Resolution[18] DAG (SR-DAG or $SR\text{-}DAG_{BS}$), i.e.,$\mathbf{\forall n}$**:Node**$\in d$**:DAG: ∃S, S is 2CNF Clause Set, S is the Clause Set of** n **(**$2CNF_n$**).** BS **is** $2CNF_{BN}$**.**
- A TRUE-DAG is a SR-DAG with only one node labeled TRUE and whose Clause Set is empty. A FALSE-DAG is a SR-DAG whose only node is labeled FALSE and whose Clause Set is empty as well.
- $rank_C$: (C:Clause) => N is a Function returning the number of literals contained in a clause. $rank_{2CNF}$, $rank_{Node}$ are similar Functions returning the maximum number of literals in any clause in the 2CNF Clause Set of a node.
  $\mathbf{\forall n}$**:Node**$\in d$**:SR-DAG:**
  $\mathbf{Rank_S=Rank_n=Max\{rank_C(C_1)..}$
  $\mathbf{rank_C(C_m)\}}$, $\mathbf{C_1,..,C_m\in S}$, **S is** $2CNF_n$
- The size of a node n in a SR-DAG ($Size_n$) is an integer representing the number of clauses in the Clause Set of that node. The same integer is used to denote the size of a Clause Set S ($Size_S$).
  In a SR-DAG of a 2CNF Clause Set S of size M the set of all nodes containing Clause Sets of sizes M or M-1 is called the Top-Part of the SR-DAG. $\mathbf{Top_{d:SR\text{-}DAG}=\{n:Node\in d \mid \exists S, S \text{ is } 2CNF_n, Size_S=M \text{ or } Size_S=M\text{-}1, Size_{BN\in d}=M\}}$

[17] $<obj_1,obj_2,\ldots obj_n>$, where obj_i:**Type** is the notation used to denote lists of Objects of **Type**. **Type** shall be omitted when obvious.

[18] The word "Resolution" and/or any of its declinations are not referring in any place of this work to the classical Resolution procedure used in Logics.

- LeftDAG: (n:Node)=>SR-DAG
  Is a Function which, given any node n of a SR-DAG, returns the SR-DAG of its left Child if existent. RightDAG is defined similarly.
- SubTree: (n:Node)=>SR-DAG
  Is a Function which, given a node n of a SR-DAG, returns the portion of the SR-DAG starting with n.

Definition 0.4: For a 2CNF Clause Set S of the form:
$\{\{a_1,b_{11}\}\{a_1,b_{12}\}..\{a_1,b_{1i}\}$
$\{a_2,b_{21}\}\{a_2,b_{22}\}..\{a_2,b_{2j}\}\ldots$
$\{a_m,b_{m1}\}\{a_m,b_{m2}\}....\{a_m,b_{mk}\}\}$[19]

a) LIT: (S) => Var
   Is a Function mapping S to the Set of all unique Literal Names/Indices in S
b) LEFT: (x:Literal ∈ C, C:Clause∈ S) => Var
   Is a Function mapping Literal x, and clause C∈ S to the Set of all variable Names/Indices occurring in the string representation of S to the left of Literal x in clause C. Right(x,C) is defined similarly.
c) SortOrder:(C:Clause∈ S,S)=>int
   Is a Function mapping clause C∈ S and S to an integer number representing the position of C within S.
d) First Literal in any clause is called Head-, last ones is called Tail-literal (HL, TL).
   HL={L:Literal | C∈ S,C={L, *t*}, *t*:Literal}
   TL={L:Literal | C∈ S,C={*t*, L}, *t*:Literal}
   Connectivity:(x:Literal∈ S,S)=>int
   Is a Function mapping a Literal x in a Clause Set S (also: $Connect_{x,S}$) to the number of clauses of S in which the Literal x appears
e) For any clause C∈ S the cardinality of the Set of all clauses which contain permutations of literals in C ($perm_C$) is called Resolution Complexity Coefficient (RCC). Both are formally defined as follows:
   - $perm_C$={C∈ S | C={*a, b*} or C={*b, a*} or C={*a*} or C={*b*}, *a, b*:Literal∈C}
   -$RCC_{k\text{-}SAT}={}^kP_k+{}^kP_{k-1}+{}^kP_{k-2}....+{}^kP_1$
   i.e., for 2SAT
   $RCC_{2\text{-}SAT}={}^2P_2+{}^2P_1=4$[20]
f) Instantiations of literals,
   Inst:(A:Assignment, S) => 2CNF Clause Set are Functions using Total, Partial or Restricted Truth Assignments to create new 2CNF Clause Sets. They substitute the literals in Clause Sets by Boolean values given in the Assignments. The clause resulting from applying an instantiation on any C∈ S is called a Derivation of C. It is called Linear Derivation if consecutive instantiations respect the linear order of literals in C[21]. If consecutive instantiations result in a clause containing only truth values and no literals, the derivation is called: Empty Derivation. Derivations containing one TRUE value are called Positive Derivations, those containing only FALSE values are called Negative Derivations. Empty, Positive and Negative Derivations can be directly evaluated to TRUE or FALSE. In this work we assume that this evaluation is embedded in the Inst function. If this evaluation results in the TRUE, S is said to be *satisfiable* by A. When Partial Assignments used by Inst are related to only one variable, Inst is

[19] AND and OR connectives are omitted as per known convention.
[20] Recall that ${}^nP_r=n!/(n-r)!$
[21] Examples of derivations of clause C={x, y } for any ordered indices x, y are {x} and {y} respectively of which only the latter is a linear derivation if the order is given by: $x<y$.

called InstSimple. InstSimple can also be restricted to only one clause.

Formally:

1) **InstSimple$_C$:(A:Assignment,C:Clause) => Clause**
2) **Derivation of a Clause C is$\in${C':Clause | C'$\in$ perm$_C$}.**
3) **Linear Derivation of C is $\in$ {C':Clause| C'={*a,b*} or C'={*b*} , *a*, *b*:Literal$\in$C, *a*<*b*}**
4) **Empty Derivation of C is $\in$ {C':Clause |C'={TRUE} or {FALSE} or {TRUE,FALSE} or {FALSE,TRUE} or {FALSE,FALSE} or {TRUE,TRUE}}**
5) **Positive Derivation of C is $\in$ {C':Clause| TRUE $\in$C'}**
6) **Negative Derivation of C is $\in$ {C':Clause| C'={FALSE,FALSE} or C'={FALSE}}**
7) **Every Derivation of C is $\in${C':Clause| C'$\in$ perm$_C$ or C'$\in$ Empty Derivation of C}**

g) Convert(C:Clause$\in$S)=>SR-DAG. Is a Function mapping a 2CNF clause C={a_1,b_{11}} to a SR-DAG by substituting in two subsequent simple instantiation steps first a_1 with TRUE and FALSE creating Clause Sets and placing them in the respective left- and right-nodes of the SR-DAG and then doing the same for b_{11} as in below (Figure 2):

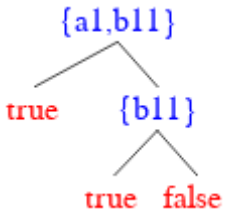


Figure 2

h) FIRST(L:Literal, S)=>int
Is a Function mapping a Literal to its integer position (starting from the left) in the string representation of S. FIRST$_C$ is the version of this function which returns the index of the clause in which L appears for the first time in the current enumeration of clauses.

i) SELECT(S)=>int
Is a Function selecting a Literal from LIT(S).

Definition 1: Almost Arbitrary-, Linearly Ordered-, Linearly Ordered, but Unsorted Clause Sets, Block, Block-Sequence, Block Literal, Symmetric Block, Dissymmetric Block, DB Sorting Condition

For a 2CNF formula S of the above form, S is called linearly ordered (l.o.) **if** the following Conditions hold:

a) **$\forall a_i,b_{ij}\in C_{i,j}$: $a_i<b_{ij}$,** i.e., Literal Names/Indices are sorted in ascending order within clauses[22].
b) **S is sorted by a_i & b_{ij} in ascending order** taking into consideration negation signs[23]. **Formally: $\forall i,j,x,y$: if $i<j$ then $L_2\in C_{j,x}$ >= $L_1\in C_{i,y}$, where L_2 is HL of $C_{j,x}$ and L_1 HL of $C_{i,y}$ *SortOrder*($C_{j,x}$,S)> *SortOrder*($C_{i,y}$,S)**
c) **$\forall x\in$LIT(S),$\forall C_{i,j}\in$S: if $x\notin$ LEFT(x,$C_{i,j}$) then $\forall y\in$ LEFT(x,$C_{i,j}$): $x>y$**
(all new Names/Indices of literals occurring for the first time in any clause of S are strictly greater than all the Literal Names/Indices occurring before them in S).
d) **S is a Set, i.e., clauses appear only once in S.**

[22] Alternatively: Clauses in S can be enumerated from left to right. In that case subscripts *i,j* are omitted and only one index is used. This is the way clauses are referred to in the rest of this paper.

[23] i.e., {1,2} comes before {1,3} or {¬1,3} and {¬1,2} before {1,2} or vice versa.

If S fulfills Conditions a), c), d), but not b) it is called linearly ordered, but unsorted (abbreviated l.o.u.). If S fulfills Conditions a), d) only it is called almost arbitrary (a.a.). Clause Sets of the form: S={{a_x,b_{x1}}{a_x,b_{x2}} .. {a_x,b_{xi},}} are called Blocks and are referred to by the name of the leading Literal (in this case S is called a_x-Block). Clause Sets of the form: S={B_a…B_n} are called Block-Sequences (B_{seq}). a_x is called Block-Literal. Clauses having a_x as leading Literal are said to belong to the a_x-Block. A Block B_x is called Symmetric Block (SB) **if ∃A: Assignment such that: instSimple(A:{*X*=TRUE},B_x}= instSimple(A:{*X* =FALSE}, B_x}** i.e., *-ve* and/or *+ve* instantiations of Block Literal *x* result in the same Clause Set. It is called Dissymmetric Block (DB) **if ∃A:Assignment such that: instSimple(A:{*X*=TRUE},B_x}=S_1, instSimple(A:{*X* =FALSE}, B_x}=S_2 and either $S_1 \subseteq S_2$ or $S_2 \subseteq S_1$.** i.e., *-ve* and/or *+ve* instantiations of Block Literal *x* result in Sets S_1, S_2 and one of them is included in the other. If a DB B_x is sorted such that all clauses containing *–ve* instances of Literal *x* are placed before all those containing *+ve* instances or vice versa, this condition is called: DB Sorting Condition.

Definition 2: 2SAT-GSPRA Procedure, Align Procedure, Name Literal, Least Literal Rule, Edge Literal, Branch Literal, Base Clause Set, Variable Ordering, Canonical Ordering

The 2SAT-Generic Sequential Patterns Resolution Algorithm (2SAT-GSPRA) applied to an arbitrary Set S of 2CNF clauses consists of the following procedure:

2SAT-GSPRA:
Inputs: Arbitrary 2CNF Clause Set **S** of size M
Output: SR-DAG
Steps: -
1- convert arbitrary clauses in S to a.a. ones (only sorting literals inside each clause).
2- choose a clause $C_0 \in S$
3- convert C_0 to a SR-DAG using Convert(C_0)
4- set IRT (**Intermediate Resolution Tree**) = SR-DAG produced in 3
5- ∀ $C_i \in S$ (one by one)
IRT=Align(IRT , C_i)
6- return IRT

Align (SR-DAG, C):
Inputs: An SR-DAG with base-node *n* and S 2CNF_n, an a.a. 2CNF clause C
Outputs: SR-DAG
Steps: -
If (SR-DAG=FALSE-DAG)
Return FALSE-DAG
else
If (SR-DAG=TRUE-DAG)
Return Convert (C)
else
{ <bracket-1>
Update S in node *n* with C: S=S ∪ C
X=SELECT(S) such that X is the least Literal of S
leftC=instSimple$_C$({X=TRUE},C)
rightC= instSimple$_C$({X=FALSE},C)
if (leftC=empty)
(i..e. C evaluated to TRUE via InstSimple)
leftResult=LeftDAG(*n*)
else
If (leftC=Nil)
(i..e., C evaluated to FALSE via InstSimple)
leftResult=FALSE-DAG
else

{<bracket-2>
leftResult=
align(LeftDAG(*n*), leftC)
}<bracket-2>
if (rightC=empty)
rightResult=RightDAG(*n*)
else
If (rightC=Nil)
rightResult=FALSE-DAG
else
{<bracket-2>
rightResult=
align(RightDAG(*n*), rightC)
}<bracket-2>
Result=SR-DAG formed from node *n*, left- and rightResult
Return SubTree(Result)
}<bracket-1>

1. A node in a SR-DAG is symbolized by [x] if the lead clause in its Clause Set is headed by a least-Literal x. Moreover: x is called the Name Literal (NL) of this Clause Set/node.
2. Edges going out of a SR-DAG node [x] are marked with x and represent instantiations of the NL x of the Clause Set S of that node (this fact is called the Least-Literal/Head-Clause-rule of S or just Least-Literal Rule of S, LLR_S). Formally:
   NL=LLR_S={*i*:Literal |∃BS: 2CNF Clause Set, ∃*n*:Node∈$SR\text{-}DAG_{BS}$, S is $2CNF_n$, SELECT(S)=*i* and ∀*x*∈LIT(S): *i*<*x*}
3. Literals on edges of branches leading indirectly to a node ***n*** are called branch-literals of ***n*** while literals on edges connected directly to ***n*** are called edge-literals of ***n***. Every edge-Literal is a branch-Literal, but not vice versa.
4. A variable ordering of a problem p ($\prod_p$) expressed as a 2CNF Clause Set S and resolved by any resolution procedure PR is a list of integers representing indices of Literal/variable names indicating priorities of instantiations of literals/variables of S used in PR. Formally: **$\prod_p$=<*i,j,k,…*> where *i,j,k,…*∈Var such that *i*<*j*<*k*<….**
5. If $\prod_p$ represents the canonical, truth table ordering of variables the following notation is used: $\prod^c_p$. As the 2SAT-GSPRA procedure described above always uses LLR_S to instantiate Clause Sets S, it obviously uses $\prod^c_p$

   The following example shows for 2CNF Clause Set BS={{0,1}{2,3}{1,2}} the first steps of 2SAT-GSPRA(BS):

A) C_0={0,1} is converted to a SR-DAG identical with (Figure 2) (replace $a1$ by 0 and $b11$ by 1) using Convert(C_0), where node n_0 contains Clause Set {{0,1}}, n_1 is TRUE-DAG, n_2 contains Clause Set {{1}}, n_3 is TRUE-DAG and n_4 is FALSE-DAG

B) **Align($SR\text{-}DAG_{\{0,1\}}$,{2,3}):**

a) S={{0,1}} ∪ {2,3}={{0,1}{2,3}}
b) least Literal x=0
c) leftC={2,3}
d) rightC={2,3}
e) IRT=leftDAG(n_0)=TRUE-DAG (the DAG of node n_1)
f) **leftResult=Align(TRUE-DAG,{2,3})**

- Return Convert({2,3})

g) IRT=rightDAG(n_0)
h) **rightResult=Align(IRT,{2,3}), node=n_2**

i) S={{1}} ∪ {2,3}={{1}{2,3}}
ii) least Literal x=1
iii) leftC={2,3}
iv) rightC={2,3}
v) IRT=leftDAG(n_2)=TRUE-DAG (The DAG of node n_3)
vi) **leftResult=Align(TRUE-DAG,{2,3})**

- Return Convert({2,3})

vii) IRT=rightDAG(n_2)=FALSE-DAG
viii) **rightResult=Align(FALSE-DAG,{2,3})**

- Return FALSE-DAG

ix) Result=SR-DAG formed from node n_2, left- and rightResult
x) Return SubTree(Result)

i) Result=SR-DAG formed from node n_0, left- and rightResult
j) Return SubTree(Result)

Definition 3: Sequentially Ordered SR-DAG, Strongly Ordered-, Loosely ordered 2CNF Clause Sets

An SR-DAG of a Set S of 2CNF clauses is called sequentially-ordered **if ∀S, *n* ∈SR-DAG, S is 2CNF$_n$: S={C$_i$,C$_j$,…C$_M$} for some *i*<*j*<….<M', M'<=M. M number of clauses in S, C$_x$'s are clauses or derivations of clauses enumerated from left to right in S**.
An SR-DAG of a Set S of 2CNF clauses is called strongly ordered (s.o.) **if ∀S, *n* ∈SR-DAG, S is 2CNF$_n$: S is linearly ordered (l.o.)** (Figure 3, right). In such case the Set S is also called strongly ordered. Strongly ordered Sets are always linearly ordered, the inverse is not always the case, i.e., some l.o. Sets may have Clause Sets in their SR-DAGs which are not l.o. If a Set S has a base Clause Set which is l.o. while some other Clause Sets in its generated SR-DAG are l.o.u., then S as well as its SR-DAG is called loosely ordered (lo.o., Figure 3, left), e.g.: **Loosely Ordered SR-DAG: ∀S, *n* ∈SR-DAG, S is 2CNF$_n$: S is either l.o. or l.o.u.**

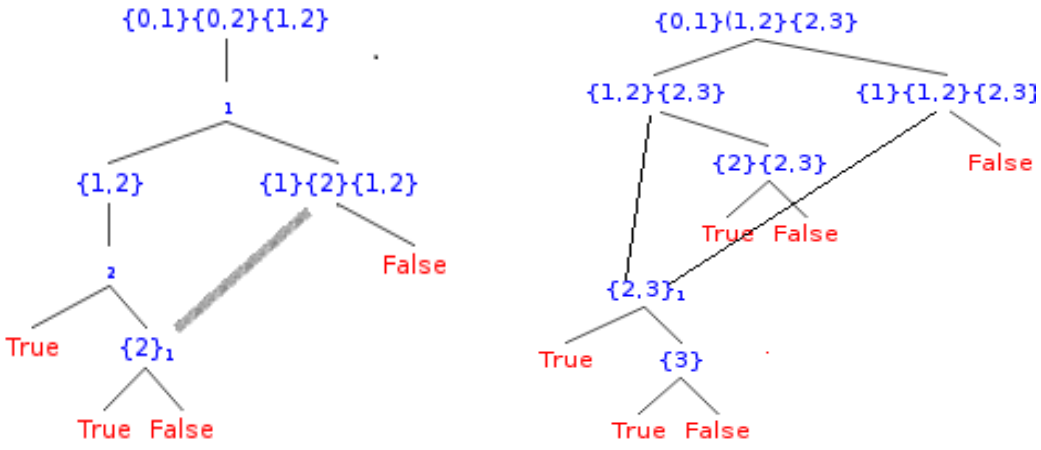


Figure 3: lo.o. and s.o. Trees

Definition 4: Common Node, Head-CN, Tail-CN, Trivial-CN, Supported CN, Supporting Parent, Direct Parent, Direct Child, Double-Sided CN from the perspective of *x*, Distinguished Literal, Single-Sided CN from the perspective of *x*, Non-Distinguished Literal, CN-Augmenting Literal

A node [*q*] is called Common Node (CN) in a SR-DAG of a Set of 2CNF clauses S **if $\exists n_1,n_2 \in$SR-DAG: [*q*] adjacent to both n_1 and n_2,** i.e., [*q*] becomes (in step *k* of the resolution procedure) a common child to two or more nodes [*x*], [*y*], [*z*], … (Figure 4). This happens when *x*,*y*,*z*,… literals are replaced by TRUE or FALSE in their respective Clause Sets. The common-node [*q*] contains the first appearance of its name Literal (NL) *q* in all branches of the SRT containing [*x*],[*y*],[*z*],..

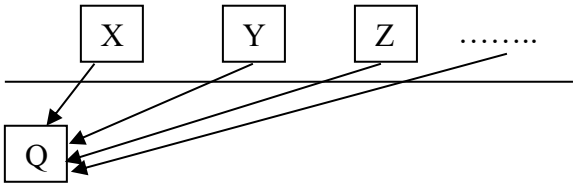


Figure 4: Common-node generated in <=*k*.

Types of common-nodes for 2CNF clauses are Head- and Tail Common-nodes (HCNs, TCNs).

More precisely:

- A CN [*q*] is called HCN **if** its Clause Set has a leading/head clause C∈S, NL *q* is HL of C
- A CN [*q*] is called TCN **if** its Clause Set has a leading/head clause C' which is a derivation of a clause C∈ S, NL *q* is TL of C

(Figure 7, upper part) shows nodes n_1,n_2 **not** connected. They both get instantiated through their least-literals *a*,*b* to different directions in the SR-DAG. Any further clause {*x*,*y*} in steps >*k* will keep this situation intact, since ***a*** and ***b*** remain the least-literals in their respective Clause Sets and cannot be bypassed by clause {*x*,*y*} in the new tree.

(Figure 7, lower part) shows a situation where both nodes are merged in steps >*k* (right) as the new clause {*i*,*a*} belongs to a block B_i parents of both nodes were instantiating in steps <=*k*. The added clause makes N1 equivalent to N2 as seen. We call those types of CNs: Trivial Common Nodes (tCNs). They are formed in SBs and are included in the Properties/Lemmas dealing with the generation of CNs. Formally: A node [*q*] ∈SR-DAG is called Trivial Common Node (tCN) **if $\exists n \in$ SR-DAG, S is 2CNF$_n$, S is SB, Child([*q*],*n*)=TRUE**

A CN [*q*] ∈SR-DAG with S=2CNF$_{[q]}$, produced in steps <=*k*, is called supported in a step l>*k* **if ∃C:Clause, C∈B_x such that: S=S ∪ C in step l>*k* while in steps <=*k*: $\exists n \in$SR-DAG, Parent(*n*,[*q*])=TRUE, S' is 2CNF$_n$, S' is B_{seq} and $B_x \notin$ S',**

i.e., its Clause Set S gets clauses appended to its head in step l>*k* which don't belong to any Block instantiated in steps <=*k* by **one or more** of its parents. A parent-set of such a CN is called supporting. In (Figure 5) an example is shown for the CN {*b*} which is supported by clause {*c*,*d*} not belonging to block B_a. If a head-clause of a CN is also a clause of one of the Clause Sets of its parents, then this parent is called direct parent of the CN. The CN itself is called direct child of this parent (Figure 6):

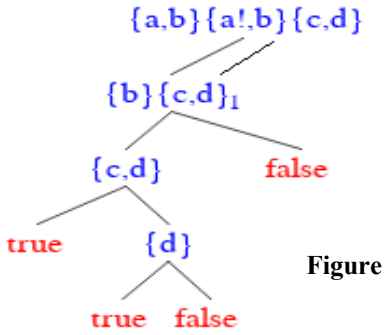


Figure 5

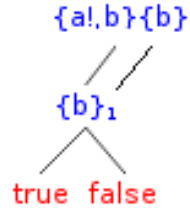


Figure 6

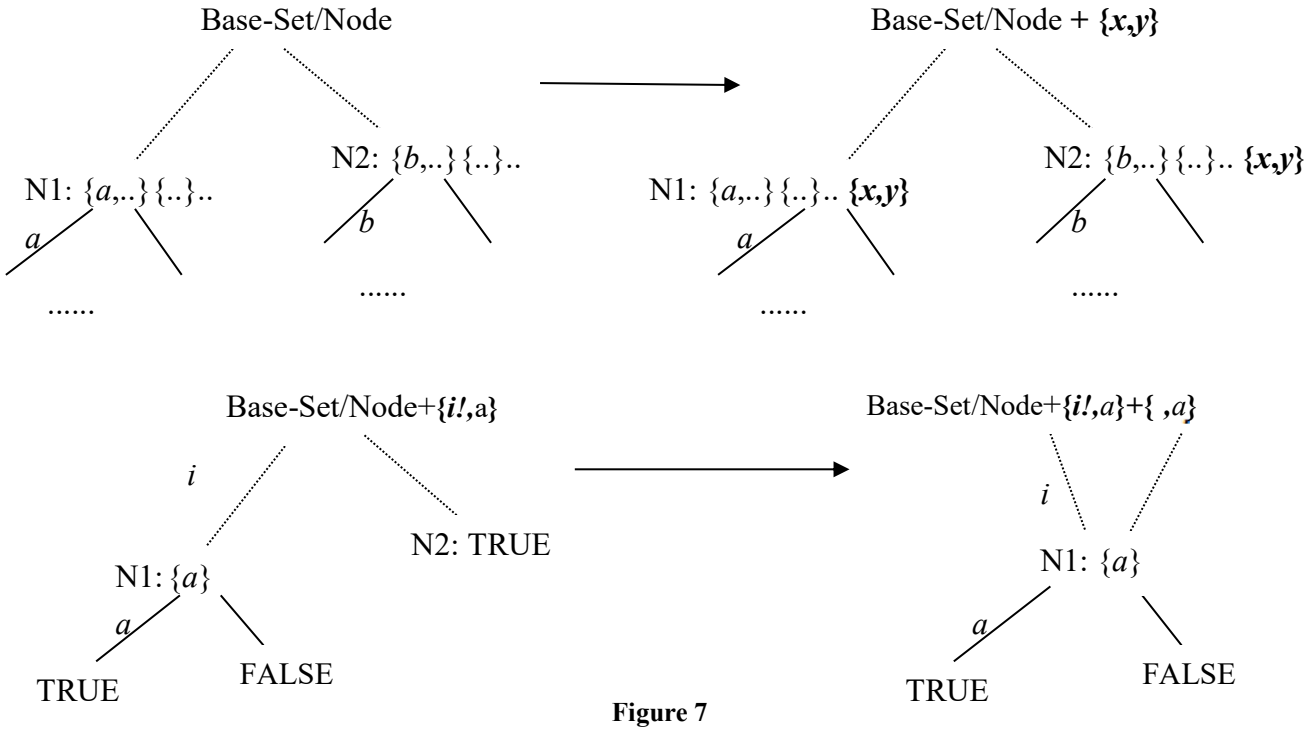


Figure 7

A CN [q] formed within a Block B_x through $+ve$ as well as $-ve$ edge- or branch-literals x is called: Double-Sided CN from the perspective of x, $DSCN_x$. Such a x is called distinguished Literal for [q]. A CN [q] formed within a Block B_x through only $+ve$ or only $-ve$ edge- or branch-literals x is called: Single-Sided CN from the perspective of x, $SSCN_x$, x is called non-distinguished Literal for [q]. Formally:

- CN [q] $\in$SR-DAG$_{BS}$ is called DSCN$_x$ if $\exists n_1,n_2$:Node$\in$SR-DAG$_{BS}$, x,y:Literal, S_1 2CNF$_{n1}$, S_2 2CNF$_{n2}$ such that: LLR$_{S1}$=x, LLR$_{S2}$=y, x=$\neg y$, Parent(n_1,[q])=TRUE, Parent(n_2,[q])=TRUE.

- CN [q] $\in$SR-DAG$_{BS}$ is called SSCN$_x$ if $\exists n_1,n_2$:Node$\in$SR-DAG$_{BS}$, x,y:Literal, S_1 2CNF$_{n1}$, S_2 2CNF$_{n2}$ such that: LLR$_{S1}$=LLR$_{S2}$=x, Parent(n_1,[q])=TRUE, Parent(n_2,[q])=TRUE.

If for a CN [q] there is no distinguished Literal x such that the CN is $DSCN_x$, then [q] is called simply SSCN. If a non-distinguished Literal x for a CN [q] formed in steps <=k is used to augment the size of [q] in step l>k, i.e., x is instantiated in a clause whose derivation is added to the clauses of [q] in l, then x is called: CN-Augmenting Literal (CNAL) for [q].

CNAL={L:Literal$\in$C:Clause, [q] is CN$\in$SR-DAG$_{BS}$ formed in steps<=k, L is non-distinguished for [q] | Size$_{|q|}$ is augmented in steps>k through invocations: InstSimple$_C$ ({L=TRUE},C) or InstSimple$_C$ ({L=FALSE},C) }

Concepts defined here are used mainly in (Lemma 9-a), (Lemma 9-b) and (Lemma 9-c).

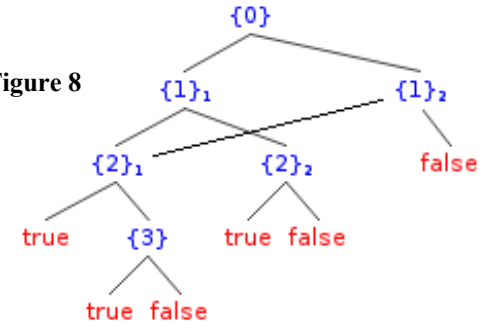
Definition 5: Dependency Graph, Leaves of Dependency Graphs, Free Binary Decision Diagrams

A dependency graph (DG) of a 2CNF Clause Set S is a directed, acyclic graph <V,E> where V is the Set of all NLs, E the Set of ordered pairs $<v_1,v_2>$, $v_1,v_2 \in V$ representing instantiations of NLs produced during resolution. DGs can be deduced from SR-DAGs in a canonical, straightforward way[24] and used as practical alternatives for truth tables. They are equivalent to Free Binary Decision Diagrams (FBDDs)[25] as shown in [Abdelwahab 2016-2] . The following two properties define a DG:

1. Each NL can appear only once in a branch.
2. Branches can have different Literal/variable orderings $\prod_p$ depending on the sub-problem p they belong to[26].

A leaf of a DG is a node whose value is TRUE or FALSE. Positive leaves have the value TRUE. (Figure 8) shows an example of a DG for the exemplary s.o. tree in Definition 3 (Figure 3).

Figure 8



Definition 6: Splits, N-Splits, CN-Splits, Split Node, Big-Splits

An SR-DAG is said to possess a Split **if ∃S':2CNF Clause Set such that: For some n_1,n_2:Node∈SR-DAG$_{BS}$, S_1 is 2CNF$_{n1}$, S_2 is 2CNF$_{n2}$, $n_1 \neq n_2$: $S' \subseteq S_1$, $S' \subseteq S_2$,∄n:Child(n,n_1)=Child(n,n_2)=TRUE** (i.e., n_1,n_2 possess common sub-formulas, but don't possess common sub-trees). CN-Splits are characterized on top of that by the existence of different Derivations of the same clause in the non-common parts of the Clause Sets of both nodes. Formally: **Splits are called CN-Splits, if, in addition to the condition above: ∃q:Node, ∃C:Clause∈ BS: S' is 2CNF$_{[q]}$, [q] is CN/MSCN in step k and C is resolved in steps >k such that: $C_1 \subseteq S_1$, $C_2 \subseteq S_2$, $C_1,C_2 \notin S'$, $C_1,C_2 \in$Every Derivation of C, $C_1 \neq C_2$.** If a Split is not a CN-Split, it is called N-Split.

Splits are thus formed when **either** node n containing Clause Set S constructed in step k is duplicated one or more times in steps >k together with all or parts of its nodes or sub-trees, the cause of this duplication being that S is resolved with a clause whose least-Literal was new in that step and had an index strictly less than all or any indices of head-literals in S as seen in the introduction (N-Split) **Or** a CN [q] constructed in step k and/or any of its nodes or sub-trees are duplicated

[24] By abstracting in each resolution-step for each node of the SR-DAG and Clause Set S the least-literal of the head-clause used in LLRs and building out of it a corresponding node in the DG.

[25] FBDDs are normally generated by recording - on top of resolution-procedures - variable assignment decisions encountered while trying to find a solution. The methods described here as well as in in [Abdelwahab 2016-2] produce a canonically ordered FBDD(=DG) representing existent variable alignments in the used clauses.

[26] In contrast to the more common OBDDs in which one Literal/variable-ordering is governing the whole graph.

with variations[27] one or more times in steps >*k* (CN-Split). We focus on CN-Splits in furtherance, since N-Splits are already covered in l.o.u. and l.o. conditions imposed by our main Algorithms below which both require condition c of Definition 1 prohibiting the use of new names/indices which are < indices of already resolved clauses.

Example of a CN-Split:

The reason why different CN-Splits occur is generally that different derivations of C get resolved with a CN through different branches of the SR-DAG linked to this CN. New nodes [*q*]'=[*q*]+C' are formed where C' is a possible derivation. [*q*'] is called: Split-Node. **If $rank_{[q]}=rank_{BN}$ this form of Splits is called** Big-Split (plural: BigSps) This situation is illustrated in below (Figure 9) as well as the concrete example of (Figure 10). BigSps are causes of exponential behavior of 2SAT-GSPRA when it is applied to a.a. or l.o.u. Clause Sets.

Concepts defined here are used mainly in (Lemma 9-a), (Lemma 9-b) and (Lemma 9-c).

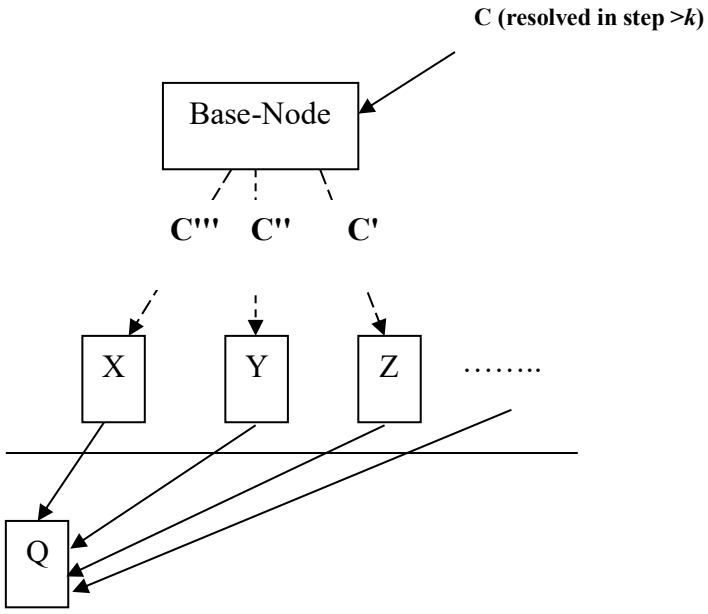


Figure 9

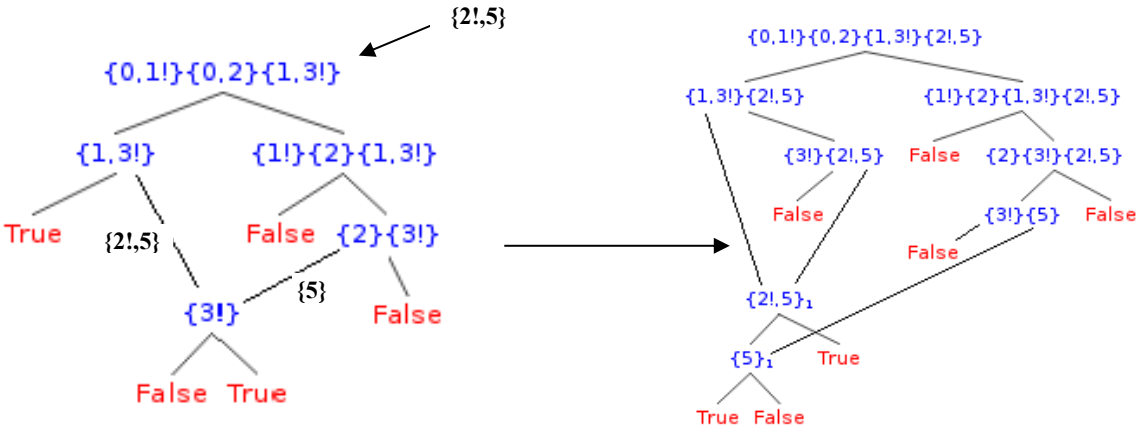


Figure 10: A concrete example for the sequential resolution of the ordinary 2-SAT case showing a new clause {2!,5} traversing in step *k* a IRT produced in steps < *k*. CN {3!} (left) is seen to split (right) to form nodes {{3!}{2!,5}}} and {{3!}{5}} respectively. This Split is **not** a BigSps.

[27] Different variations of the duplicated CN correspond to the resolution of different derivations of a newly resolved clause C with the CN.

Definition 7: Clauses Renaming Algorithm, Connection Matrix, Renaming Precedence Condition

The Clauses Renaming Algorithm (**CRA**) is a procedure which takes an arbitrary Clause Set S as input, renames its literals yielding a new, logically equivalent S' as output which is guaranteed to be l.o.u. This procedure consists of the following steps:

CRA:
Inputs: Arbitrary 2CNF Clause Set **S** of size M
Output: Clause Set S'
Steps: -

1. Enumerate clauses in S (starting with 0) in ascending order.
2. For each clause C_i:
   a) Arrange literals in ascending order within C_i such that literals which were not renamed before and appear more often in other clauses become HLs before those which appear less often or which only appear in C_i. This condition shall hereafter be called: Renaming Precedence Condition (**RPC**). RPC uses $Connect_{x,S}$ of Definition 0.4.
   b) Create a matrix whose rows represent variable/Literal names/indices while columns represent clauses. This matrix is called: **Connection Matrix**.
3. For all clauses C_i and all literals in C_i:
   - Create a new row and write column values TRUE or FALSE according to whether the Literal appears in the corresponding clause or not.
4. Rename all variables in the Connection Matrix in ascending order.
5. Reconstruct the clauses again using the new variable names. This reconstruction may be done by simply substituting each Literal in the original Clause Set with its new Literal name/index.

Example: If S = {{0,5} {0,2} {1,3} {1,4} {2,3}}, then the Connection Matrix of S is:

	C_0	C_1	C_2	C_3	C_4
0	**True**	**True**	False	False	False
5	**True**	False	False	False	False
2	False	**True**	False	False	**True**
1	False	False	**True**	**True**	False
3	False	False	**True**	False	**True**
4	False	False	False	**True**	False

Transformed (via step 4 of CRA) to:

	C_0	C_1	C_2	C_3	C_4
0	**True**	**True**	False	False	False
1	**True**	False	False	False	False
2	False	**True**	False	False	**True**
3	False	False	**True**	**True**	False
4	False	False	**True**	False	**True**
5	False	False	False	**True**	False

The new clause list for the above reads S: S' = {{0,1}{0,2}{3,4}{3,5}{2,4}}. Note that S' is l.o.u. Note also that if we would want to convert S' to a l.o. Set by sorting clauses via their least-literals (as required by Condition b) in Definition 1) we would get: S'' = {{0,1} {0,2} {2,4} {3,4} {3,5}} which is not fulfilling Condition c) because of Literal 3 (i.e., S'' is neither l.o. nor even l.o.u.). To convert an arbitrary Clause Set to a l.o. Clause Set, an extension to CRA is needed, introduced hereafter with some definitions:

Definition 8: Mapping, Image, Variable Space, Node in space-i, Apply, Inverse Apply, Equivalence via Mapping, trivial Mapping, Stable Set, Stable Clause, Stable Clause Set, Mixed Space Node, Single Space Node, Mixed Space SR-DAG/Tree, Single Space SR-DAG/Tree, Literal in space-i, Assignment in space-i, Literal *x* proceeds *y* in space-i, Mapping in space-i, monotone Mapping

Definition 8.1: Mapping: (N) => N is a bijective function giving a Literal Name/Index in a 2CNF Clause Set S its new Name/Index after a renaming operation using CRA. The new Name/Index is also called: Image of the Literal. New Names of literals forming single clauses or Clause Sets are called

Images of original clauses or Clause Sets. Subsequent application of mappings starting from a BS is called a Variable Space (VS). To express that a Clause Set is formed in a space-i the notation: $S=\{\{..\}\ldots\{..\}\}_{space\text{-}i}$ is used. To express that a node is formed in a space-i the notation: $Node_{space\text{-}i}$ is used.

Definition 8.2: Apply: (M:Mapping, S: 2CNF Clause Set) => Clause Set
Is a function which replaces occurrences of literals in a Clause Set S with their Names/Indices given by the mapping M. InvApply is similarly defined, but applies to S: M^{-1} instead of M.

Definition 8.3: Two 2CNF Clause Sets S_1, S_2 are said to be Equivalent via Mapping (Notation: $S_1 \Leftrightarrow_M S_2$) **if $\exists M_1, M_2$:Mapping such that: $Apply(M_1,S_1)=Apply(M_2,S_2)=S'$.** S' is called: Syntactic Image of both S_1, S_2.

Definition 8.4: If $\exists$M:Mapping, S 2CNF Clause Set, $\forall x \in LIT(S)$: $M(x)=x$, i.e., each Literal index is given itself after a renaming operation using CRA, M is called trivial Mapping (**tMapping**).
If $\exists$M:Mapping produced in step k such that: $\forall x \in Sub$, $Sub \subseteq Lit(S)$: $M(x)=x$ in any step $>k$, i.e., a subset of Literal indices is mapped to itself via CRA in step k and remains always mapped to itself for any step>k, Sub is called a Stable Set of literals. **If $\forall x$:*Literal*$\in C_i \in S$, $x \in Sub \subseteq Lit(S)$, Sub is stable, then:** C_i is called Stable Clause. **If $\forall C_i \in S$, C_i is stable, then: S is** a Stable Clause Set.

Definition 8.5: If S_1, S_2 are 2CNF Clause Sets of nodes $n_1, n_2 \in$SR-DAG, respectively, $S_1 \neq S_2$, but $n_1=n_2=n$, then: n is called Mixed-Space Node (MSN) as opposed to Single-Space Nodes (SSN).

Definition 8.6: SR-DAGs with MSN nodes are called Mixed-Space Trees (MSTs). SR-DAGs with only SSNs are called Single-Space Trees (SSTs). A Literal index subscribed by space-i ($L_{space\text{-}i}$) refers to the name L given by a mapping M in space-i. An Assignment giving literals in space-i truth values is called space-i-Assignment ($A_{space\text{-}i}$)
If $\exists$space-i:VS such that: $S_{space\text{-}i}$ is a 2CNF Clause Set where: $FIRST_C(x,S_{space\text{-}i}) < FIRST_C(y, S_{space\text{-}i})$, then: x proceeds y in $S_{space\text{-}i}$ or, if S is known from the context, just: x proceeds y in space-i (Notation: $(x \mid y)_{space\text{-}i}$)
i.e., within space-i the first occurrence of Literal x in Clause Set S comes before the first occurrence of Literal y. When space-i is known, its subscript is omitted. Mappings subscribed by space-i: ($M_{space\text{-}i}$) refer to the mapping created by a CRA operation within space-i.

Example:
For S = {{0,5}{0,2}{1,3}{1,4}{2,3}} and S' = Apply (M,S) = {{0,1}{0,2}{3,4}{3,5}{2,4}} in the example of Definition 7, Mapping M is: {{0,0}{5,1}{2,2}{1,3}{3,4}{4,5}}, Stable-Set = {0,2}

Definition 8.7: A mapping $M_{space\text{-}i}$ is called monotone Mapping in space-i ($mM_{space\text{-}i}$), **when** $\forall x,y \in LIT(S_{space\text{-}i})$: **if** $(x \mid y)_{space\text{-}i}$ **then also** $M_{space\text{-}i}(x) < M_{space\text{-}i}(y)$

Definition 9: Clauses Renaming & Ordering Algorithm, CRA-Form

The Clauses Renaming & Ordering Algorithm (CRA^+) is a procedure which takes an arbitrary 2CNF Clause Set S in a space-i as input and applies CRA repetitively generating a new mapping and a new space each time. After each step the intermediate Clause Set is sorted as required by Definition 1b) before iterating back. This is done until renaming Literal indices in two consecutive steps yields *tMapping*, i.e, the Stable Set becomes equivalent with the Set LIT(S), while the output Clause Set S' becomes l.o.
The following recursive pseudo-formal description of this procedure is used in the below proofs:

CRA^+:
Inputs: An arbitrary 2CNF Clause Set **S**
Output: l.o. Clause Set **S'**
Steps:
1- set CurrentMapping = null, CurrentSet=S
2- while (CurrentMapping != tMapping)
   i. currentSet=CRA(CurrentSet)
   ii. sort CurrentSet as instructed in Definition 1 b)
   iii. set CurrentMapping=Mapping passed by CRA
3- S'=CurrentSet
4- return S', S' is called the **CRA-Form of S**.

Example: Following this procedure for the above Set S = {{0,5}{0,2}{1,3}{1,4}{2,3}} applying CRA to get S' = {{0,1}{0,2}{3,4}{3,5}{2,4}} and a sorting step giving the above S"={{0,1}{0,2}{2,4}{3,4}{3,5}}.
A new CRA-iteration will yield the following Connection Matrix:

	C_0	C_1	C_2	C_3	C_4
0	**True**	**True**	False	False	False
1	**True**	False	False	False	False
2	False	**True**	**True**	False	False
4	False	False	**True**	**True**	False
3	False	False	False	**True**	**True**
5	False	False	False	False	**True**

It is then transformed to:

	C_0	C_1	C_2	C_3	C_4
0	**True**	**True**	False	False	False
1	**True**	False	False	False	False
2	False	**True**	**True**	False	False
3	False	False	**True**	**True**	False
4	False	False	False	**True**	**True**
5	False	False	False	False	**True**

Mapping:
{{0,0}{1,1}{2,2}{4,3}{3,4}{5,5}},
Stable Set: {0,1,2,5} yields S'"={{0,1}{0,2}{2,3}{3,4){4,5}} when applied on S". S'" is l.o. already and needs no further sorting. Note that in the last matrix all literals are forming an ordered sequence which means that any further renaming would result in *tMapping*. This is the termination condition.

Definition 10: Sequentially-Ordered, Multi-Space SR-DAG, Multiple Space Block, Multi-spaced Symmetric Block, Target Space, Multiple Space Common-node

An MST whose Clause Sets are all l.o. is called: Sequentially-Ordered, Multi-Space Resolution Tree/SR-DAG ($MSRT_{s.o.}$), **if** $\forall n_{space\text{-}i}$**:Node∈SR-DAG:-** $(2CNF_n)_{space\text{-}i}$ **is l.o.** A block B_x whose Clause Set or derivations thereof (all or part of them) belong to more than one VS is called a Multiple Space Block, **MSB** (Notation also: $B_x^{S1,S2,..}$,S_1,S_2,.. Variable Spaces). Similar to Single Space Blocks:

An MSB may be symmetric or dissymmetric.

Formally: **MSB = {**
$(B_{x1})_{space\text{-}i}$:2CNF Clause Set |
∃space-j, $(B_{x2})_{space\text{-}j}$:2CNF Clause Set,
M: Mapping, where:
$((B_{x1})_{space\text{-}i} \Leftrightarrow_M (B_{x2})_{space\text{-}j}$) Or $((B'_{x1})_{space\text{-}i} \Leftrightarrow_M (B'_{x2})_{space\text{-}j})$), B'_{x1}, B'_{x2} are Derivations of B_{x1}, B_{x2}, in respective Spaces}

Definition 10.1: An MSB B_x is called Multi-spaced Symmetric Block (MSSB)
- MSSB = {
$(B_{x1})_{space\text{-}i}$:2CNF Clause Set |
∃space-j, $(B_{x2})_{space\text{-}j}$:2CNF Clause Set,
M: Mapping, where
$((B_{x1})_{space\text{-}i} \Leftrightarrow_M (B_{x2})_{space\text{-}j}$
Or
$(B'_{x1})_{space\text{-}i} \Leftrightarrow_M (B'_{x2})_{space\text{-}j}$)
B'_{x1}, B'_{x2} are Derivations of B_{x1}, B_{x2}, in respective Spaces and ∃$A_{space\text{-}i}$, $A_{space\text{-}j}$: Assignment such that:
instSimple($A_{space\text{-}i}$:{X_1=TRUE},
$(B_{x1})_{space\text{-}i}$) $\Leftrightarrow_M$
instSimple($A_{space\text{-}j}$:{X_2=FALSE},
$(B_{x2})_{space\text{-}j}$)
}

Definition 10.2: A node in a space S_T (called: Target Space, TS) which is target of two or more Variable Spaces is called Multiple Space Common Node, MSCN (Notation: $[q]_{ST}^{S1,S2,..}$,S_1,S_2,..,S_T Variable Spaces to which the node belongs). Formally: A node is called MSCN **if ∃n_1,n_2 ∈ $MSRT_{s.o}$ not necessarily of the same space: [q] adjacent to both n_1 and n_2,** i.e., in step k of the resolution it becomes common child/adjacent to two or more nodes, possibly of different spaces $[x]^{S1}$, $[y]^{S2}$, $[z]^{S3}$, … in (Figure 11)[28] generated in steps <k. This happens when there exist mappings M_1,M_2,M_3…, such that: x=M_1(x'),y=M_2(y'),z=M_3(z'),…, where x, y, z are literals in S_T, and x', y', z' are literals replaced by TRUE or FALSE in their respective Clause Sets and respective Spaces.
The common-node $[q]_{ST}^{S1,S2,..}$ contains the first appearance of its name Literal (NL) q in all branches of the $MSRT_{s.o}$ containing $[x']^{S1}$, $[y']^{S2}$, $[z']^{S3}$, … etc. and there exist literals q', q'', q''', etc. in Spaces S_1,S_2,S_3,… such that: q=M_1(q')=M_2(q'')=M_3(q''')=… etc.

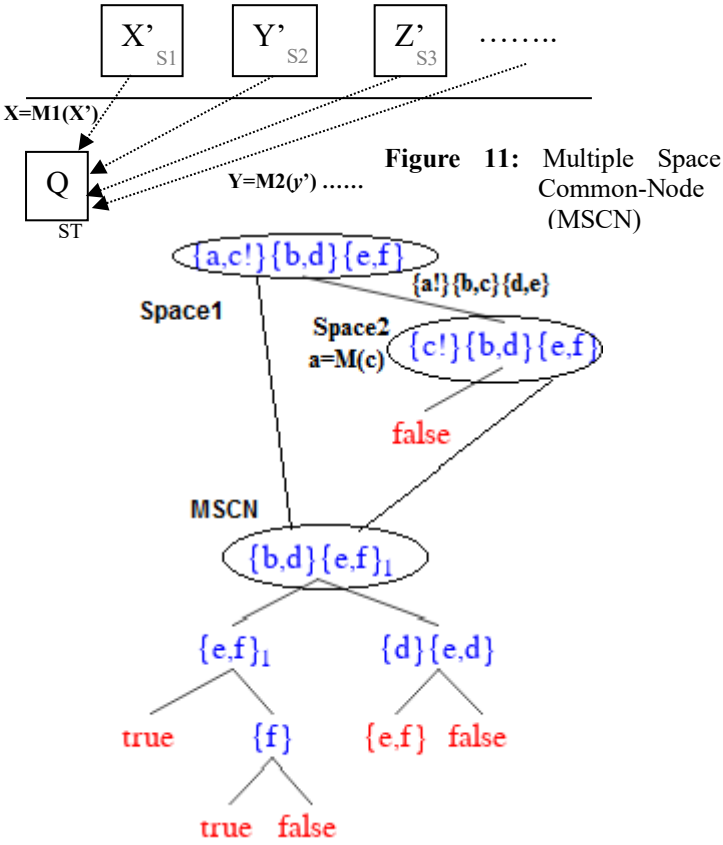


Figure 11: Multiple Space Common-Node (MSCN)

Figure 12: Illustration of Definition 10 where S_T=$Space_1$,M_0 is the trivial Mapping, $[b]_{ST}^{Space1}$={{b,d}{e,f}} is a MSCN,$[c]^{Space1}$={{¬c}{b,d}{e,f}}, $[a]^{Space2}$={{¬a}{b,c}{d,e}} for M={(c>a), (b>b),(d>c){e>d}{f>e}}. Then it is clear that $[c]^{Space1}$=$[a]^{Space2}$=$[q]^{Space1,Space2}$,where q=M_0(c)=M(c). Also: $[b]_{ST}^{Space2}$ is obviously child to both, $[a]_{ST}$=BS and $[a]^{Space2}$ with edge-literals a=M_0(a) and a=M(c) respectively.

[28] The notation $[x]^{S1}$ is read: Node [x] in Variable Space S_1.

Definition 11: Double-Sided MSCN with respect to Literal *z*, Single-Sided MSCN with respect to Literal *z*, trivial MSCN

An MSCN $[q]_{\text{space-i}}$ is called DS-MSCN$_z$ (Double-Sided MSCN with respect to Literal *z*) **if $\exists n_1,n_2 \in$ MSRT$_{s.o}$ of 2CNF Clause Set S, $\exists x_{\text{space-j}}$, $y_{\text{space-k}}$:Literal, $\exists M_1,M_2$: Mapping, such that: $[q]_{\text{space-i}}$ is adjacent to both n_1 and n_2 and $z_{\text{space-i}} = M_1(x_{\text{space-j}})$, $z_{\text{space-i}} = M_2(y_{\text{space-k}})$, where $y_{\text{space-k}}$ has the opposite sign of $x_{\text{space-j}}$,** i.e., there exist at least two edge- or branch-literals *x*, *y* from Spaces space-j, space-k respectively and a Literal *z* from the target space-i such that both literals are translated to *z* within their respective spaces and have opposite signs. Literals *x* and *y* are also called distinguished (c.f. Definition 4, (Distinguished Literal)).
if $\exists n_1,n_2 \in$ MSRT$_{s.o}$ of 2CNF Clause Set S, $\exists x_{\text{space-j}}$, $y_{\text{space-k}}$, $\exists M_1,M_2$:Mapping, such that: $[q]_{\text{space-i}}$ is adjacent to both n_1 and n_2 and $z_{\text{space-i}} = M_1(x_{\text{space-j}})$, $z_{\text{space-i}} = M_2(y_{\text{space-k}})$, where $y_{\text{space-k}}$ has the same sign as $x_{\text{space-j}}$, i.e., a MSCN is formed through only *+ve* or only *-ve* instantiations of edge- or branch-literals *z* or its images in respective spaces, *z* is not distinguished, then the MSCN is called SS-MSCN$_z$ (Single-Sided MSCN with respect to *z*). $[b]_{ST}^{Space2}$ in the example above of (Figure 12) is thus a SS-MSCN$_a$.

An MSCN $[q]$ is called trivial MSCN, (tMSCN), **if $\exists n \in$ MSRT$_{s.o}$ whose Clause Set is a MSSB, Child($[q]$,*n*)=TRUE,** i.e., $[q]$ is formed through a newly resolved clause in step *k*, who belonged to a MSSB to which one or more of its parents belonged in steps <*k*.

Concepts defined here are used mainly in (Lemma 8), (Lemma 9-a), (Lemma 9-b) and (Lemma 9-c)

Definition 12: Aligned Trees, Alignment Clause

A MSRT$_{s.o}$ of a 2CNF Clause Set S is said to be aligned **if** $\exists C \in S$, C' derivation of C such that:$\forall n \in$MSRT$_{s.o.}$, S' is 2CNF$_n$, $\forall C_x \in$ S' the following is true:

a) SortOrder(C', S')>SortOrder(C_x,S')
b) S' is l.o.

In other words: Either C or one of its derivations C' are the last clauses in any Clause Set of the MSRT$_{s.o}$. C is called Alignment-Clause.

Definition 13: Aligned Nodes, Alignment Clause Set of S, Alignment MSRT$_{s.o}$s

A node *n* of size M is said to be *aligned* **if**:

a) For M<=2: *n* possesses a Clause Set with an aligned MSRT$_{s.o}$
b) For M>2:
   (i) All nodes or sub-trees of size M possesses Clause Sets which are l.o.
   (ii) All nodes or sub-trees of size <M are aligned

The Set of all unique clauses and their derivations used for the alignment of all nodes of a MSRT$_{s.o}$ of an arbitrary 2CNF Clause Set S is called Alignment Clause Set of S (ACS). It is formally given by: **ACS=$\cup$ perm$_{Ci \in S}$** for all Ci$\in$S. Obviously, ACS cannot have more than RCC$_{2\text{-SAT}}$*M elements/clauses containing all possible permutations of literals in linear- or non-linear sequence. An MSRT$_{s.o}$ whose nodes are all aligned is called Alignment MSRT$_{s.o}$

Definition 14: Resolution procedures: 2SAT-GSPRA$^+$, Align

2SAT-GSPRA$^+$:
Inputs: Arbitrary 2CNF Clause Set **S** of size M
Output: $MSRT_{s.o}$
Steps: -
1- convert arbitrary clauses in S to a.a. ones (only sorting literals inside each clause).
2- choose a clause $C_0 \in S$
3- convert S to a l.o. Set using CRA$^+$ (the version with DB-Sorting, c.f. Section III, Lemma 8)
4- convert C_0 to a SR-DAG using Convert(C_0)
5- set IRT (**Intermediate Resolution Tree**) = SR-DAG produced in 4
6- ∀ $C_i \in S$ (one by one)
   a. IRT=Align(IRT , C_i)
7- return IRT

Align (SR-DAG, C):
Inputs: An $MSRT_{s.o}$ with base-node *n* and S the Clause Set of *n*, an a.a. 2CNF clause C
Outputs: $MSRT_{s.o}$
Data Structure: List of Tuples: <Clause Set, Node index> (called: **LCS**) initially empty
Steps: -
- If ($MSRT_{s.o}$ =FALSE-DAG)
  Return FALSE-DAG
else
  If ($MSRT_{s.o}$ =TRUE-DAG)
  {
  - Result = Convert(C)
  - Store S=C in LCS in its CRA-Form, index is the base node of Result
  Return Result
  }
  else
  { <bracket-1>
  a- Update S in node *n* with C: **S=S ∪ C**
  b- If (S is in LCS)
    Return SubTree(foundNodeIndex)
  c- If (S is l.o.)
  {<bracket-2>
  - X=least Literal in S
  - leftC=instSimple$_C$({X=TRUE},C)
  - rightC= instSimple$_C$({X=FALSE},C)
  - if (leftC=empty)
    (i..e. C evaluated to TRUE via InstSimple)
    leftResult=LeftDAG(*n*)
    else
    If (leftC=Nil)
    (i..e. C evaluated to FALSE via InstSimple)
    leftResult=FALSE-DAG
    else
    {<bracket-3>
    leftResult=
    Align(LeftDAG(*n*), leftC)
    }<bracket-3>
- if (rightC=empty)
    rightResult=RightDAG(*n*)
    else
    If (rightC=Nil)
    rightResult=FALSE-DAG
    else
    {<bracket-3>
    rightResult=
    Align(RightDAG(*n*), rightC)
    }<bracket-3>
- Result= $MSRT_{s.o}$ formed from node *n*, left- and rightResult
- Store S in LCS in its CRA-Form giving it as index the node *n*

- Return Result

}<bracket-2>
else (of step c-)
If (S is not l.o.)
  {<bracket-2>
  1- Choose a clause $C_0 \in S$, S'=CRA$^+$(S), the version with DB Sorting
  2- If S' has already been stored in LCS, erase its entry
  3- C may have changed its place due to sorting in CRA$^+$. $MSRT_{s.o}$ for all clauses except the last one must be created again: Let S''=S'\A, A is the last clause in S'
  4- NewDAG=2SAT-GSPRA$^+$(S''), Construct all nodes whose Clause Sets start with S'' again, assigning to them NewDAG and updating LCS with adequate information.
  5- Result=**Align(NewDAG,A)**
  6- Store S' in LCS in its CRA-Form giving it as index the base node of Result
  7- Return Result
  }<bracket-2>
}<bracket-1>

Definition 15: 2SAT Fast Generic Pattern Resolution Algorithm

2SAT-FGPRA:
Inputs: Arbitrary 2CNF Clause Set **S** of size M
Output: $MSRT_{s.o}$
Data Structure: List of Tuples: <Clause Set, Node index> (called: **LCS**) initially empty
Steps: -

1- convert arbitrary clauses in S to a.a. ones (only sorting literals inside each clause).
2- choose a clause $C_0 \in S$
3- convert S to a l.o. Set using **CRA^+** (the version with DB-Sorting, c.f. Section III, Lemma 8)
4- **Create base node *n***, Set S to be the Clause Set of *n*,
5- Process *n* as follows:
- if (size of *n* > 1) && (clauses are neither evaluated all to TRUE nor containing a clause evaluated to FALSE))

{<bracket-1>

(Form left- and right Clause Sets for *n* instantiating the least Literal to TRUE and FALSE respectively. Make sure the resulting Clause Sets are l.o.)

a. X=Least Literal of S
b. leftClauseSet= **InstSimple**({X=TRUE},S)
c. rightClauseSet= InstSimple({X=FALSE},S)
d. leftClauseSet= CRA^+(leftClauseSet)
e. rightClauseSet= CRA^+(rightClauseSet)

f. **Search for leftClauseSet in LCS**

if (leftClauseSet found)

leftResult= **SubTree**(foundIndex)

else

{

1-leftResult=2SAT-FGPRA(leftClauseSet)

2- Store leftClauseSet in LCS in its CRA-Form giving it as index the base node of leftResult

}

g. Search for rightClauseSet in LCS

if (rightClauseSet found)

rightResult= SubTree(foundIndex)

else

if (S has only one clause C)

{

1-rightResult=2SAT-FGPRA(rightClauseSet)

2-Store rightClauseSet in LCS in its CRA-Form giving it as index the base node of rightResult

}

- Result= $MSRT_{s.o}$ formed from node *n*, left- and rightResult
- Store S in LCS in its CRA-Form giving it as index the node *n*

- Return Result
} <bracket-1>

else

if (S has only one clause C)

{

- Result = **Convert**(C)

- Store S in LCS in its CRA-Form giving it as index the node *n*

Return Result

}

Else {

If (clauses are evaluated all to TRUE)

Return TRUE-DAG

Else (clauses contain a clause evaluated to FALSE)

Return FALSE-DAG

}

III-2 Converting arbitrary 2CNF Sets to l.o.u and l.o. ones

Can we always convert arbitrary Sets to s.o. or lo.o. ones? To answer this question we need to investigate how to convert a.a. Clause Sets[29] to l.o.u. and l.o. ones.

Lemma 1: CRA is guaranteed to convert an a.a. Clause Set S into a l.o.u. Clause Set. It takes O(N*M) steps[30] to do so for M = number of clauses, N = number of variables. Moreover:

a. CRA always produces monotone mappings (mM).
b. $(x \mid y)$ iff $(x<y)$ for literals $x,y \in$ Lit(S) in any l.o. Clause Set S.
c. In sequential, clause by clause resolution:
Let $x,y \in$ Lit(S), S is l.o., $S=2CNF_{BN}$, $x \in C_1, y \in C_2, C_1 \neq C_2$, $FIRST_C(x)=1$, $FIRST_C(y)=2$, SortOrder(C_1,S)< SortOrder(C_2,S)
and
$\exists n$:Node,space-i:VS where: $S'=2CNF_n$, Child(n,BN)=TRUE such that:
$x_{space\text{-}i}, y_{space\text{-}i} \in$ Lit(S'), S' is l.o., $x_{space\text{-}i} \in C_1', y_{space\text{-}i} \in C_2'$, $C_1' \neq C_2'$, $FIRST_C(x_{space\text{-}i})=1$, $FIRST_C(y_{space\text{-}i})=2$, SortOrder($C_1$',S')< SortOrder($C_2$',S')
and
$C_1', C_2' \in$ S' images or derivation of images of $C_1, C_2 \in$ S **then**:
$(x_{space\text{-}i} \mid y_{space\text{-}i})$ *iff* $(x \mid y)$[31]

Proof: c.f. the three conditions of (Definition 1) for a Clause Set to be l.o.u.:

a) $\forall a_i, b_{ij} \in C_{i+j}: a_i < b_{ij}$
c) $\forall x \in$ LIT(S), $\forall C \in$ S: if x not $\in$ LEFT(x,C) then $\forall y \in$ LEFT(x,C): $x>y$
d) Clauses appear only once in S

It is clear that a) and d) are fulfilled by any output of CRA as they constitute the mere definition of a.a. Sets. For Condition c): Suppose some Literal L in a clause C_i={... L ...} $\in$ S' (S' = output Set) breached Condition c): This means that L is new in the clause sequence starting with C_0 until C_i, but there exists L' to its left where L<L'. This cannot be the case, since any such L' would have to appear in a row before L in the connection matrix (step 2-b, Definition 7) and thus get a smaller index in the renaming step 3-. For the complexity assertion: The number of cells to be created in a Connection Matrix is always N*M.
To show the mM property a-: $\forall x,y$ literals in a Clause Set: CRA's way of giving them new names is - as seen - to assign each one a row in the connection matrix in the order of their appearance and then rename the rows by counting from 0-n, finishing up with a strict order (c.f. Definition 7, steps 2-a, 3 and 4 as well as the example). Therefore: If $(x \mid y)$, then, unless clauses are re-ordered, after one application of CRA: $M(x)<M(y)$.

For b-: $(x \mid y)$ iff $(x<y)$ in any l.o. Clause Set. To see this, the only direction we still need to show is: $(x<y)>(x \mid y)$. Suppose in a l.o. Set: $(x<y)$. Either $(x \mid y)$ or $(y \mid x)$. In case $(y \mid x)$, this means that the first occurrence of y comes before the first occurrence of x and both appear in different clauses. But then, x should have been $> y$ as per condition c in Definition 1 which prescribes that in a l.o. Clause Set a new Literal must be strictly greater than all literals occurring to its left.
This means $(x \mid y)$.

---

[29] Converting an arbitrary Clause Set to an almost arbitrary one (a.a.) being a trivial exercise needing only sorting literals inside each clause in ascending order and taking care that clauses have unique occurrences.

[30] Steps are invocations of primitive operations as normally perceived in complexity analysis.

[31] Intuitively: If two literals x, y belonging to different, subsequent clauses of S, a l.o. Base Set, have images in another l.o. Set S' of some Space-i, and the order of clauses in S' preserves the relative precedence of images of Literal x on images of Literal y, this always means that $x_{space\text{-}i}$ proceeds $y_{space\text{-}i}$ in S'. The other direction is also true.

For c-: First direction: Suppose $x_{space\text{-}i}$ appears in C_1' for the first time in step k. X must also appear for the first time within C_1 in S in step k, because C_1' is the image or a derivation of an image of C_1 and $FIRST_C(x)=1$. $y_{space\text{-}i}$ appears then in S' in a step $j>k$, because $C_1' \neq C_2'$,SortOrder(C_1',S')<SortOrder (C_2',S'), S' is l.o. and resolution is sequential. Suppose now ($y \mid x$) in S. This means that C_2 must have appeared in a step $<k$ contradicting the fact that C_2', its image, appeared in $j>k$. Therefore it must be that: ($x \mid y$). The other direction is similar: When ($x \mid y$), then ($y_{space\text{-}i} \mid x_{space\text{-}i}$) cannot be the case unless either the order of S is not preserved in S' or $y_{space\text{-}i}$ appears for the first time in a clause other than C_2'. Both conditions contradict the assumptions.
(Q.E.D.)

Lemma 2: For a 2CNF Clause Set S it is true that:
a- S is l.o. **iff** $CRA^+(S)$ reaches a Stable-Set of literals equivalent to LIT(S)
b- S is satisfiable **iff** $CRA^+(S)$, the CRA-Form, is satisfiable
c- S and $CRA^+(S)$ are logically equivalent

Proof: a- Suppose S is l.o. This means that it is fulfilling all Conditions a)-d) of Definition 1. Any attempt to use CRA^+, i.e., rename the literals and then sort them, must generate a Stable-Set = LIT(S) after only one CRA- and sorting iteration, since otherwise (i.e., if a Literal gets a new Name/Index after such an iteration) this would mean a breach of one or all of those conditions. Other direction: Suppose S reached such a Stable-Set through application of CRA^+, i.e., CRA^+ terminated. If S is not l.o., then it must be at least l.o.u. (because of Lemma 1). The only reason for S not to be l.o. would thus be that clauses are not sorted correctly. This is not possible because CRA^+ can only become a Stable-Set equivalent to LIT(S) if two consecutive renaming iterations assign literals with the same names/indices, the first of which is followed per definition by a sorting operation.
b- The proof is by induction on M, the number of clauses in S.

Base-Case: M=1: For S={{a,b}} CRA^+ terminates after one iteration yielding the Clause Set S'={{a',b'}} with a',b' new Indices/Names for a,b, a'=M(a), b'=M(b), M the mapping produced by CRA^+. Let A be an Assignment satisfying S, A={{$a=v_1$}{$b=v_2$}}, $v_1,v_2 \in$ {TRUE, FALSE}. If we set A'={{$a'=v_1$}{$b'=v_2$}}, then S' is satisfied by A', since nothing has changed except variable names. The other direction is similar.

Induction Hypothesis: S is satisfiable iff $CRA^+(S)$ is satisfiable for $Size_S$=M

Induction step: If $Size_S$=M+1: Suppose $\boldsymbol{A}$ is the Assignment which satisfies S[32]. We distinguish two cases:

Case 1- S'=CRA^+(S), CRA^+ does not alter the order of clauses in S. Assume S={C_0,..,C_M}, S'={C_0',..,C_M'}, where C_M={a,b}, C_M'={M(a),M(b)}. $\boldsymbol{A}$ must also satisfy S''=S\$C_M$ which is of size M and per induction hypothesis there exists $\boldsymbol{A'}$ satisfying S'''=S'\$C_M$'. The following cases can then occur:
a- Literals $a,b \in C_M$ are new, i.e., $a,b \notin$ Lit(S''). M(a) and M(b) are also $\notin$ Lit(S''') per monotone mapping property of M. Extend $\boldsymbol{A'}$ to include {M(a)=v_1,M(b)=v_2}, where $v_1,v_2 \in$ {TRUE, FALSE} are values given to a, b in assignment $\boldsymbol{A}$. This extended $\boldsymbol{A'}$

[32] The other direction: [CRA+(S) is satisfiable =>S is satisfiable] can be shown using similar arguments and is not included here to avoid unnecessary length.

satisfies C_M' and thus also S'[33], otherwise ***A*** couldn't be satisfying C_M (remembering that names of variables are different in C_M and C_M', but signs are the same).

b- Either Literal a or b or both are ∈ Lit(S''). It must be then the case that v_1,v_2∈{TRUE, FALSE} used in assignment ***A*** for any such a or b to satisfy S'' do not falsify C_M, otherwise ***A*** wouldn't be satisfying S. Per induction hypothesis: A' satisfies S''' using, per definition, for any of M(*a*) or M(*b*) the same values v_1 and/or v_2. They can only falsify C_M' if they falsify C_M which is not the case.

Case 2- S'=CRA^+(S) alters the order of clauses in S. Let S'={C_0',..,C_M'}. Re-arrange S such that clauses are ordered like in S'. Call the new Clause Set S'', i.e., S''={C_0,..,C_M}. S is, per definition, satisfiable iff S'' is satisfiable. Apply the same arguments used in Case 1 on S' and S''.

c- S has a CRA-Form S'=CRA^+(S) and thus S ⇔$_M$ S', (Definition 8.3), i.e., ∃M:Mapping such that: Apply(M,S)=S', S' is the exact syntactic image of S. This means: Any Truth Assignment A satisfying S can be converted to a Truth Assignment A' satisfying S' by simply substituting variables *x* with M(*x*). The other direction is also possible.
(Q.E.D.)

Lemma 3: CRA^+ takes a number of steps which is in O(M^2(logM+N)). More precisely M CRA-iterations and M sorting operations[34] (M = number of clauses in S, an a.a Set).

Proof: (by induction on M)

Base-Case: M=1: For S={*a,b*} CRA^+ takes one CRA and one sorting operation to generate *tMapping* per definition (Definition 8.4).

Illustration Case: M=2[35]
Let S={{*a,b*},{*d,e*}}={C_0,C_1}

Case 1: No literals in common between C_0 and C_1: In that case *a*<*b*<*d*<*e*.
S is l.o. No CRA- or sorting iterations needed.

Case 2: Only head-Literal in common: S={{*a,b*}{*a,e*}} for example: Same as Case 1, S is also l.o. No CRA or sorting needed.

Case 3: Only tail-Literal in common (Case I): S={{*a,b*}{*b,e*}} for example: S' is converted after one CRA-iteration to S={{*a,b*}{*a,c*}}, because of Definition 7, 2a, Renaming Precedence Condition (RPC). Thus, no sorting needed.

Case 4: Only tail-Literal in common (Case II): S={{*a,b*}{*c,b*}} for example: S' is converted after one CRA-iteration to S={{*a,b*}{*a,c*}}, because of Definition 7, 2a), Renaming Precedence Condition (RPC), no sorting needed.

Resuming Base-Cases M=1,2: Although we may not need CRA or sorting, CRA^+ takes at most one iteration (i.e., one CRA- and one sorting operation) to generate *tMapping* and to terminate.

Induction Hypothesis: For M clauses: M CRA-iterations (M^2*N) as well as M sorting operations (M^2logM) are needed in the worst case to make S l.o.

Induction step: For any additional clause C_{M+1} = {*x*,*y*} we have the following cases (c.f. Definition 9, pseudo formal procedure):

[33] Since the truth value of S''' is not affected by the new variables

[34] Assuming that a sorting operation takes O(M log M) primitive operations.

[35] Monotone +*ve* 2-SAT case is used here and in the next Lemma (w.l.o.g.), since CRA^+'s behavior does not depend neither on Literal signs nor on clause breadth.

1. ***x,y* are new literals not appearing before in any Clause C_i:** This case is straightforward in that no sorting is needed, i.e., only CRA (renaming) in the worst case.
2. **One or more literals of *x,y* appeared in a previous clause:** For Example: Suppose S={{0,1} {0,2} {0,4} {0,6} {2,8} {9,10} {11,12}} which is l.o. adding the clause {4,6}, the following steps are required:

a) S={{0,1}{0,2}{0,4}{0,6} {2,8}{9,10}{11,12}{4,6}} input

b) S={{0,1}{0,2}{0,4}{0,6} {2,8}{4,6}{9,10}{11,12}} sort

c) S={{0,1}{0,2}{0,3}{0,4} {2,5}{3,4}{6,7}{8,9}} CRA, S in step c) is already l.o.

For a Clause Set of size M: S={{*a,b*} {*b*,…} {*d*,…}…} where, as per induction hypothesis, it is assumed that it is l.o. and we add a clause containing one or more literals which appeared before, we note that S is l.o.u. A sorting step is what is required to align the new clause to its right place. If this step is done, then another CRA-step guarantees l.o.u (per Lemma 1). This means that we need an additional CRA (renaming) as well as a sorting step for this case.

Resuming the induction step: One additional CRA- and one additional sorting step is needed in the worst case for M+1
(Q.E.D.)

This section concludes with a Lemma showing **that any** a.a**. Set can be converted to** a **l.o. Set**, i.e., application of CRA^+ on any a.a. Set always terminates yielding the right result.

Lemma 4: CRA^+ terminates always converting any arbitrary 2CNF Clause Set S of size M to a Stable-Clause Set.

Proof: (by induction on M)

Base-Case M=1: For S={{a,*b*}} as seen in the Base-Case of (Lemma 3) CRA^+ terminates after one iteration yielding the Clause Set S'={{*a'*,*b'*}} where *a'*,*b'* are new indices/names for *a,b*. S' is stable.

Illustration Case M=2: Let S={{*a,b*}{*x,y*}}. As seen in all Base-Cases for M=2 of (Lemma 3): One iteration of CRA and one sorting operation converts S to a l.o. Set. This means any further iteration of CRA^+ yields a Stable-Set (per definition of CRA^+) letting the algorithm terminate.

Induction Hypothesis: Application of CRA^+ for a number of iterations *k* on a 2CNF Clause Set S of size M converts S to a Stable-Clause Set (i.e., CRA^+ produces M stable clauses after *k* iterations).

Induction Step: Per induction hypothesis for S having M+1 clauses, there are M stable clauses in iteration *k*. Let C={*x,y*} be the clause which is not stable. After step *k* the position of C cannot be before any other stable clause C'={*i,j*}, e.g., as in {{*a,b*}…{*x,y*} {*i,j*}…}, because this would mean that CRA-operations will have to change indices *i,j* to new ones for C' contradicting its stability assumption, i.e., C has to be the last clause in S.
In that case, even if literals in C would not fulfill the l.o. condition for whatever reason other than sorting (because C is already in its place), further CRA-steps in iterations >=*k* guarantee to convert C into a stable clause (per definition of CRA^+)[36] causing CRA^+ to terminate with a Stable-Clause Set of size M+1.

(Q.E.D.)

[36] CRA renders S ∪ C l.o.u., i.e., any new literal *v* of C is > LEFT(*v*, C) after such an iteration.

III-3 Way of work of 2SAT-GSPRA$^+$

The main difference between 2SAT-GSPRA and 2SAT-GSPRA$^+$ is that the latter uses CRA$^+$ to convert Clause Sets to l.o. ones. It is necessary to understand what 2SAT-GSPRA$^+$ really does when it imposes the l.o. condition on clauses. Central in this respect are the following points:

i- Counting the number of new nodes created in each step is essential. As resolution is sequential, a new clause resolved in such a step has to traverse all nodes of the previous IRT if necessary (c.f. Figure 10 for an illustration). It is therefore clear that, unless nodes are left untouched or are copied (i.e., Splits occur), the contribution of the new clause is either to augment sizes of already existing nodes or to add new size-1 ones.

ii- It is also imperative to understand how 2SAT-GSPRA$^+$ recognizes equivalent Clause Sets so that it is not obliged to repeat similar calculations. The equivalence notion adopted in [Abdelwahab 2016-2] is structural (algorithmic), i.e., two Clause Sets are equivalent only when their generated resolution trees are[37]. As we are always trying to minimize nodes in generated trees, this notion is sufficient for our purpose. 2SAT-GSPRA$^+$ implements it (compare with Definition 14, Align function, Point b,) by requiring a Clause Set to be stored in the LCS list only when CRA$^+$ is applied to it. This has the advantage of normalizing all stored Clause Sets so that their sub-trees can be retrieved easily when encountered again during resolution, remembering that all resolution steps may require using CRA$^+$. [Abdelwahab 2016-2] calls this: (CRA-form).

iii- Sorting condition b) in (Definition 1) prescribes distinguishing *+ve* and *–ve* literals of the same variable while ordering a Clause Set without giving any preference to the best way of doing that, leaving it to implementations of CRA$^+$. Some implementations may have the effect of building SBs and tCNs as seen in (Definition 4) and (Figure 7) which may split. It is shown here that this situation can always be avoided without disturbing the essential (RPC) condition of CRA by appropriately choosing which sign to prioritize while applying the (DB Sorting) Condition[38].

The following lemmas allow us to get a more precise picture of the above ideas.

Lemma 5 (Expansion of MSRT$_{s.o}$s):
a- $\forall n_1,n_2$ nodes $\in$ MSRT$_{s.o}$: if n_1,n_2 are not directly connected in steps $<=k$ then they cannot be directly connected in steps $>k$, if the sort order of their Clause Sets is not altered, except in the trivial case when the new clause belongs to a block, parents of n_1,n_2 were instantiating in steps $<=k$ and n_1, n_2 become equivalent (tCN, tMSCN).

b- $\forall$M>1: A node [q] of size M is CN/MSCN iff $\exists$CN/MSCN [q'] of size M-1 augmented in size by a clause C such that: [q]=[q']

c- Let up_1,up_j be upper bounds of nodes generated during the whole process of

[37] S_1={{a, ¬b}{b,d}{¬d,e}}, S_2={{¬d,e}{b,d}{a, ¬b}} are for example considered to be different from the structural point of view although they are logically the same.

[38] In this present work RPC is restrained to HLs only while in [Abdelwahab 2016-2] it is applied to all literals. There, a stronger property than the one seen in Lemma 8 is shown, namely: That appropriately sorting blocks to avoid tCNs (there called the l.o.s condition) produces the same amount of unique nodes as not doing any extra sorting. This was necessary there to imply that tCNs and their Splits don't harm the near-to-minimal node counts of GSPRA$^+$ trees.

resolution in size-levels 1 and j, respectively, where $1<j<=M$. If Splits are not accounted for in any size-level j, then: $up_j <= up_1$

Proof:

a- When 2SAT-GSPRA$^+$ is applied on a BS, it uses (LLR_{BS}) in the Align Algorithm. This rule is applicable within a space as well as between spaces in the following way: If nodes n_1 and n_2 belong to different spaces and were not directly connected in step k, then, unless the sort order of their Clause Sets is not altered, they cannot be directly connected in steps $>k$, because newly resolved clauses don't affect old results of an application of the least-Literal-rule, i.e., least literals in old nodes remain the same for l.o. Clause Sets (c.f. Definition 4 and Figure 7 for an illustration). tCN and tMSCN exception cases are explicitly dealt with in (Lemma 8) below.

b- $\forall M>1$: If $\exists$CN/MSCN [q'] of size M-1 constructed in steps$<k$ and augmented in size by a clause C in step k such that: [q]=[q'], then per (Definition 4), [q] is a CN/MSCN and its size is M. Other direction: We need only to investigate the case when a node [q] of size M was *not* a CN/MSCN in steps $<k$ and became CN/MSCN in step $>=k$. As per a- this cannot happen unless the sort order of one of at least two nodes involved is altered. Let [q'] be the node of size M whose sort order is changed in step k and whose SR-DAG is completed in steps$>k$ such that $2CNF_{[q]}=2CNF_{[q']}$. This can only happen in 2SAT-GSPRA$^+$ if [q'], when passed to the Align-Algorithm is not found to be l.o. and CRA$^+$ is used (Definition 14, bracket-2, steps: 1-7). In this case: The last clause A of the re-arranged Clause Set is separated (step 3) and the SR-DAG of node [q'] is formed again, first with $2CNF_{[q']=}$ $2CNF_{[q]}\backslash A$ (step 4), before a recursive call of Align is attempted (step 5). In those first steps: $Size_{[q']}=M-1$. Since: $2CNF_{[q']}=2CNF_{[q]}\backslash A$ and as per (step 4) all nodes whose Clause Sets begin with $2CNF_{[q]}\backslash A$, i.e., [q] as well, are reconstructed: [q'] must have been a CN/MSCN of size M-1, before its size is augmented by A. When Align is called then in (step 5) with the last clause A, $Size_{[q']}=Size_{[q]}=M$ which was to be shown.

c- If Splits are *not* accounted for at any size-level $j>1$, then: Per (Definition 14) of 2SAT-GSPRA$^+$: A node can have in any step only one copy which either remains at such a level-j or is propagated up one level to become part of level-j+1, *but not both*. Recall that this is not like the case of a Split, where one copy of the node remains as it is and another copy (or more) is resolved with a new clause/Clause Set moving up the hierarchy (recall Definition 6, Splits). Hence, we can show the property using induction on j: $1<j<=M$ as follows:

Base-Case: For j=2: Since up_1 is the upper bound of nodes generated in size-level 1 during the whole process of resolution, the worst case is that all up_1 are added to level 2. Since Splits are not counted at level 2, they must be also the *only* nodes added at that level. Therefore: $up_2 <= up_1$

Induction Hypothesis: $up_j <= up_1$ for size-level j, $j>1$

Induction Step:: Because any node formed at level j+1 at any step of the resolution can either come from the lower j-level, or formed via Split and we don't count Splits: up_{j+1} cannot be $> up_j$, which means $up_{j+1} <= up_j$ and thus per induction hypothesis $up_{j+1} <= up_1$
(Q.E.D.)

Lemma 6: (Aligned $MSRT_{s.o}$ Base Cases) All size 1,2 nodes of any $MSRT_{s.o}$ of a 2CNF Clause Set S produced by 2SAT-GSPRA$^+$ are aligned.

Proof: For size 1 nodes it is clear that the $MSRT_{s.o}$ representing any single clause is aligned per (Definition 12) with the single clause itself being the Alignment-Clause. For size 2 nodes of the form S={{*a,b*}{*x,y*}} let's recall that 2SAT-GSPRA$^+$ converts any such Clause Set to a l.o. Clause Set using CRA$^+$ (step 3, Definition 14). This leads to the following cases:

Case 1 (Figure 13): No literals are common between the two clauses. {*x,y*} is then the Alignment-Clause

Case 2 (Figure 14): There is one Literal in common independent of the specific place of this Literal. Because of RPC of CRA (c.f. Definition 7, 2-a), all Clause Sets will be converted via CRA$^+$ to the form {*a,b*}{*a,y*} which has {*a,y*} as Alignment-Clause.
(Q.E.D.)

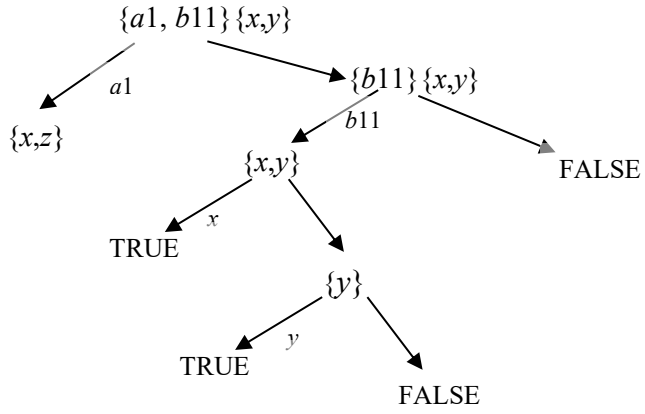


Figure 13

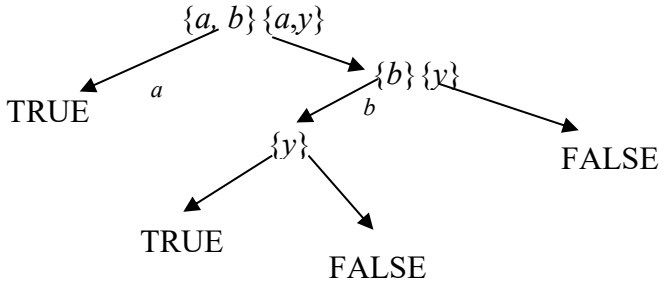


Figure 14

Lemma 7: (Alignment $MSRT_{s.o}$s) 2SAT-GSPRA$^+$ produces $MSRT_{s.o}$s with aligned nodes[39] and if Splits are not counted, then for the whole process of resolution: The total number of generated size-1-level nodes cannot exceed $RCC_{2\text{-}SAT}*M^2$

Proof:

1. **Alignment $MSRT_{s.o}$s (Induction on M)**
   Base Case: M=3-sized $MSRT_{s.o}$s are aligned because their M=2-sized nodes or sub-trees produced by 2SAT-GSPRA$^+$ are all aligned (Lemma 6) and (as per Definition 13) their M=3-sized nodes or sub-trees are l.o. The fact that all size M=3 nodes or sub-trees are aligned makes in the same way all size M=4 nodes aligned and so forth. **Inductively**: All M-sized nodes are aligned because all their M-1-sized nodes or sub-trees are aligned and their M-sized nodes of sub-trees are l.o. This implies that any final $MSRT_{s.o}$ is an Alignment $MSRT_{s.o}$.
2. Size-1 level nodes created in any step k<=M can only come from ACS and ACS cannot have more than $RCC_{2\text{-}SAT}*M$ per (Definition 13) i.e., the total number of generated size-1 nodes for all steps cannot exceed $RCC_{2\text{-}SAT}*M^2$

(Q.E.D.)

Lemma 8: ∀SB, DB, tCN such that SB⊆DB and tCN formed in SB: tCN can always be avoided by appropriately choosing the DB Sorting Condition. Similarly: tMSCNs can be avoided as well.

Proof: According to (Definition 1), a block is called DB if *-ve* and/or *+ve* instantiations of block Literal a result in Sets S_1, S_2 respectively and either $S_1 \subseteq S_2$ or $S_2 \subseteq S_1$. Figures 15 below shows an example for such a dissymmetric block B_a={{*a, b*}{¬*a, b*}{¬*a, c*}{*a, c*}{*a, d*}} (SB={{*a, b*}{¬*a, b*}{¬*a, c*}{*a, c*}}) sorted in two ways: One prioritizing clauses with *–ve* occurrences of a (Figure 15a) and the other prioritizing those with *+ve* occurrences (Figure 15b). Only the first, relevant parts of the resolution trees are shown. An SB as well as a tCN is formed in the first case and bound to split in any further step, while the second case avoids such formation by utilizing the dissymmetry in clause {*a,e*} to prioritize *+ve* occurrences of a. Clauses with *–ve* occurrences of the block Literal just fill then the TRUE leaf node in any further step. As (DB Sorting Condition) does not affect any special condition used in CRA$^+$ (especially the RPC condition in Definition 7 which only relates to HLs set here to block Literal *a*), a constellation like (Figure 15b) can always be reached w.l.o.g. by letting clauses with the most common block-literal-sign (in Figure 15b: *+ve*) appear before the others in the sort order.

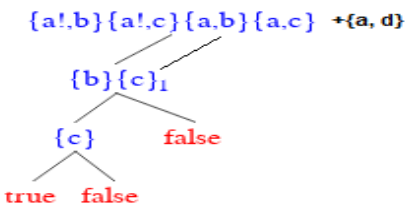


Figure 15-a

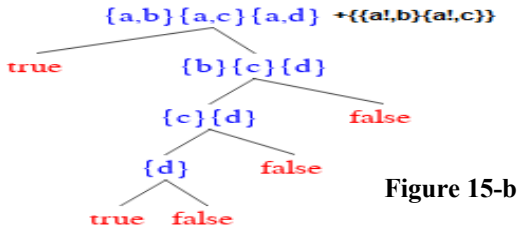


Figure 15-b

[39] As per (Definition 12) and (Definition 13): There is a subtle difference between aligned $MSRT_{s.o}$s and Alignment $MSRT_{s.o}$s. While the former represent trees with only one clause or its derivation entailing all Clause Sets, the latter represent trees in which all nodes were aligned, not necessarily with the same clause.

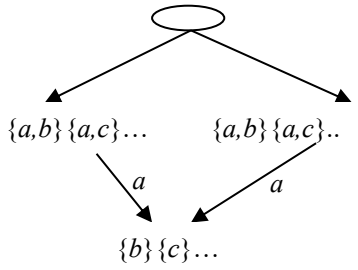


Figure 16

What if B_a is a MSB, i.e., some of its Clause Sets belong to more than one Space (Definition 10)? It suffices to observe that tCNs cannot be formed in blocks scattered between different, mutually exclusive branches of the tree. I.e., the constellation for B_a in (Figure 16) is ***not*** possible: The reason being that branch Literal a is the head of rank-2 clauses occurring also in the base Set BS. Thus, to scatter them between different, mutually exclusive branches an additional variable would be needed, contradicting the fact that BS is a 2CNF Clause Set. Therefore: B_a must occur in one and only one node which might be shared by many branches coming from different spaces. But then: Even if B_a or any of its Clause Sets were parts of more than one space, the same arguments used above would apply, if one of those spaces is chosen for the node in which B_a is occurring. In other words: When 2SAT-GSPRA$^+$ reaches this node it can apply DB-Sorting in CRA$^+$ as instructed in (Definition 14) and the proof of this Lemma without any additional effort. (Q.E.D.)

III-4 CN-Splits in MSRT$_{s.o}$s

The most important contributions of this work are the observations related to Splits of resolution trees on which l.o. conditions are imposed. As mentioned before: Only CN-Splits need to be thoroughly investigated. The other type of Splits, N-Splits (c.f. Definition 6), cannot occur during resolution work of 2SAT-GSPRA$^+$, since no node n containing Clause Set S and formed in step k, can be duplicated in steps $>k$, while S is resolved with a clause whose least-Literal is new and has an index strictly smaller than all or any indices of head-literals in S. Such a case would be a breach of the l.o. condition imposed by 2SAT-GSPRA$^+$ on all Clause Sets of all nodes (this is formally shown below in Lemma 9-b). As for CN- as well as MSCN-Splits, the following two cases in (Figure 17 and Figure 18) show practical situations occurring during resolution of l.o. Clause Sets, motivating the more abstract investigations of the Lemma 9. In (Figure 17), SS-MSCN$_3$ is formed through instantiation of Clause Sets {{¬2,3}} and {{2}{¬2,3}} by substituting TRUE for Literal 2. It is clear that MSCN [3][40] can be augmented in size by adding additional clauses of the form $\{\neg 2, x\}$ to the BS. A clause {2, x}, on the other hand, does not have any effect on [3], since it disappears from [3] the moment it is added to {{¬2,3}} and {{2}{¬2,3}}, i.e., continuing the current instantiation block B_2 in BS either augments the size of MSCN [3] or doesn't have any effect on it. If we attempt to split this node using clauses of the form $\{1, y\}$ or $\{\neg 1, y\}$ there is yet another restriction: The fact that BS is l.o. cannot allow any new blocks B_x starting after B_2 to contain: $x<2$. Therefore: [3] cannot be split in any further step.

[40] The notation [3] stands for [q], q=3.

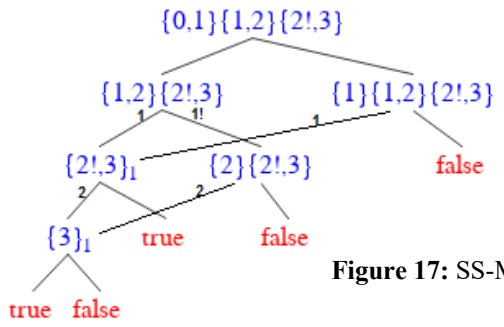


Figure 17: SS-MSCN$_3$

In (Figure 18), DS-MSCN$_2$ Splits the moment block B_1 is continued in any way in the BS, i.e., when clauses of both forms $\{1,x\}$ and $\{\neg 1,x\}$ are resolved.

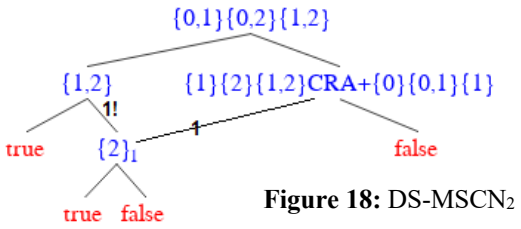


Figure 18: DS-MSCN$_2$

The only way to augment the size of [2] is by starting a new block B_x which has to fulfill $x>1$, because of the l.o. condition imposed on the BS. However: When such a block starts, [2] cannot be split in any further step, since no more B_1 or B_0 clauses are permitted. Are those the only possible cases of CN/MSCN Splits? Or are there other situations in which Splits can occur after MSCNs are augmented to big sizes? This is answered by the next central Lemma which investigates all possible situations encountered when Splits are attempted.

Lemma 9: MSRT$_{s.o}$s formed by 2SAT-GSPRA$^+$ during resolution of a 2CNF Clause Set have the following properties:

a. CNs and MSCNs containing clauses belonging to the BS or their images cannot split.
b. N-Splits cannot exist, but Rank-1, size-1 CN/MSCN Splits can.
c. Rank-1, size-1 CNs and MSCNs which are not tCNs or tMSCNs and which are augmented to sizes>1 in step k, cannot split in steps >k.

Proof:

We recall the generic form of a MSCN $[q]_{ST}^{sp1,sp2,sp3,..}$ (Figure 19) which is a generalization of a CN and shall be used here w.l.o.g. and which - as opposed to tMSCNs - was *not* formed in a symmetric block. Its edge- or branch-literals can be either distinguished or not (c.f. Definition 10, Definition 11, Figure 12):

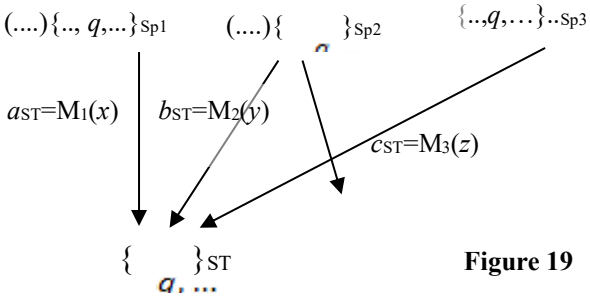


Figure 19

a- If the size of $[q]_{ST}^{sp1,sp2,sp3,..}$ gets augmented by a rank 2 clause $C_1'=\{a', b'\}_{ST}$ in step k, then, obviously, there exists a clause $C_1=\{a, b\}\in$BS and a mapping M such that: $a'=M_{ST}(\boldsymbol{a})$, $b'=M_{ST}(\boldsymbol{b})$, i.e., C_1' is an image of C_1. In this step k: All literals of C_1 and all their images were ***new*** in all branches and spaces leading to the MSCN per Definition 10, i.e., $\forall i, l_{space\text{-}i}, S'$, where $l_{space\text{-}i}$ is a branch- or edge-Literal of $[q]_{ST}^{sp1,sp2,sp3,..}$, S' Clause Set of a parent node containing $l_{space\text{-}i}$:

$l_{space\text{-}i} \mid a_{space\text{-}i}$[41]

Since all Clause Sets and all nodes are l.o., this means also that $l_{space\text{-}i} < a_{space\text{-}i}$ according to (Lemma 1-b).

[41] Reads: The first appearance of an image of a in space-i occurs after the first appearance of an image of **l** in any Clause Set S' of the same space.

Per (Lemma 1-a) we have:

$$M(l_{space\text{-}i})<M(a_{space\text{-}i}) \text{ -1}$$

To be able to split $[q]_{ST}^{sp1,sp2,sp3,..}$ in any step>k, a subsequent clause $C_2=\{\boldsymbol{x}, \boldsymbol{y}\}\in$BS must traverse some or all branches and spaces leading to the MSCN and form two different Derivations (Definition 6, CN-Split). For at least one of those Derivations: Some parent node ***p***, Clause Set S of ***p***, space-i and edge- or branch-Literal $l_{space\text{-}i}$ must satisfy:

$x_{space\text{-}i}=l_{space\text{-}i}$ or $y_{space\text{-}i}=l_{space\text{-}i}$ where $C_2'=\{x, y\}_{space\text{-}i}$. Substituting in above formula -1 we have: $M(x_{space\text{-}i})<M(a_{space\text{-}i})$ [42]

On the other hand: As per the l.o. condition imposed on BS: $a<=x$ and we have two cases[43]:

If $a=x$ then $x_{space\text{-}i}=a_{space\text{-}i}$ and C_2' is added to S and augments the size of the MSCN instead of splitting it.

If $a<x$, then as per (Lemma 1-b): *$(a \mid x)$*, because BS is l.o., such a BS may have ***only*** one of the following two generic forms which realize the requirement that the first appearance of Literal x comes after that of Literal a[44]:

$$\{..\{..,a\}..\{r, x\}..\{s, \neg x\}\ldots\{a, b\}\ldots\{x,y\}\ldots\} \quad -2$$

Or

$$\{\ldots\{a, b\}\ldots\{x, y\}\ldots\} \quad -3$$

We will show in what follows that both forms lead to inconsistency with respect to the given case assumption. To see this: BS in form-3 satisfies, per Lemma 1-c for any space-i: $(a_{space\text{-}i}|x_{space\text{-}i})$ and thus also: $M(a_{space\text{-}i})<M(x_{space\text{-}i})$, per monotone property of mappings. Contradiction. Note that $\{x, y\}_{space\text{-}i}$ can only come before $\{a, b\}_{space\text{-}i}$ when it is 'pulled' by a clause $\{.., x\}_{space\text{-}i}$ appearing before $\{a, b\}_{space\text{-}i}$ In that case: Clauses are re-arranged through renaming to guarantee l.o. as shall be seen. However: Because x appears in BS for the first time in $\{x, y\}$ and not as a TL in any clause $\{.., x\}$ prior to $\{a, b\}$ such a situation *cannot happen* and the relative position of x or any of its images to an image of Literal a in an arbitrary space remains the same for this case.

By contrast: If BS is of form-2, this means that in some step>k it may be that: Either B_a comes before B_x or vice versa. Constellations like:

$S=\{..\{x\}..\{a,b\}..\{x,y\}..\}_{space\text{-}i}$ **or**

$S=\{..\{\neg x\}..\{a,b\}..\{x,y\}..\}_{space\text{-}i}$

---

42 The same arguments hold if $y_{space\text{-}i}$ is used instead of $x_{space\text{-}i}$ or if C_2 is a unit clause. Those cases are omitted here to avoid unnecessary length.

43 In [Abdelwahab 2016-2] the '<' relationship alone is used to show a similar contradiction for the 3CNF case. The reasoning shown there, **which is equally valid here and may be considered a shorter version of the proof of Lemma 9-a of this work**, goes, informally, as follows: *"If a MSCN is augmented in size by an image of a clause C from the BS, then all literals of C or their images must be '>' any branch- or edge-literals of the MSCN. Since BS is l.o.: Any clause D from the BS, coming after C, can only possess literals which are '>=' the HL of C, i.e., also '>' branch- or edge-literals of the MSCN. To split a MSCN, however, there needs to be at least one branch- or edge-Literal of the MSCN '=' to a Literal in D. Contradiction"* In this work the precedence relation '|' is used to allow a thorough investigation of permutation possibilities of BS, leading all to the same contradiction as well. For 3CNF a lot more BS cases are involved, explaining why '|' could not be used there.

44 Because of the l.o. condition, any l.o. Clause Set cannot have a form in which blocks B_a or B_x are interrupted like in: $\{..\{a, z\}..\{..,x\}\{a,b\}\{x,y\}\}$ or $\{..\{..,a\}..\{x,..\}\{a,b\}\{x,y\}\}$ for example.

which are both not l.o. To make S l.o. in such a step, clauses are re-arranged, literals renamed and the sub-tree reconstructed by 2SAT-GSPRA$^+$ such that:
(i)-$\{\{x\}..\{x, y\}\}_{\text{space-i}}$ **or**
(ii)-$\{\{\neg x\}..\{x, y\}\}_{\text{space-i}}$ comes either *before* or *after* $\{a, b\}_{\text{space-i}}$.
If $\{x, y\}\in B_x$ comes after B_a in a space-i, the situation is similar to Form-3 discussed above and leads to a contradiction, when a split is attempted. The MSCN is augmented, since $\{a<x\}_{\text{space-i}}$.
On the other hand: If $\{x, y\}\in B_x$ comes before B_a in any one or more Spaces, it must be the case that only one Derivation of $\{x, y\}$ is generated, otherwise the MSCN would split, *before* it is augmented contradicting the case assumption.
In Summary: Because of the l.o. condition which prescribes that instantiation blocks cannot be interrupted (c.f. Footnote 44), Clause $C_2=\{x, y\}\in BS$ in Form-2 or Form-3 as well as all its possible derivations can only either augment the size of the MSCN or leave it untouched, but not split it.
Same Arguments apply for unit (rank-1) clauses $C_1=\{a\}\in BS$ which have images in $[q]_{ST}^{sp1,sp2,sp3,..}$ [45].

b- (Figures 20) shows a Split of a rank-1, size-1 MSCN occurring in the $MSRT_{s.o}s$ for $S=\{\{0,1\}\{0,2\}\{1,2\}\{1,3\}\}$.

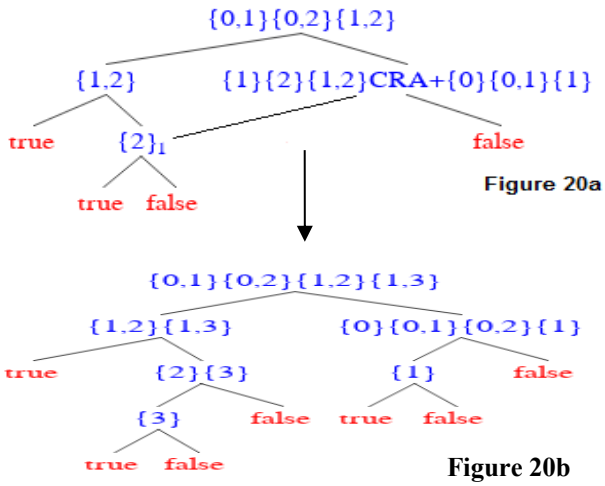


Figure 20a

Figure 20b

To show that N-Splits cannot exist: Suppose they do exist, this means - per definition of a Split (Definition 6) - that: ∃S':2CNF Clause Set such that: For some n_1,n_2:Node∈SR-DAG, S_1 is $2CNF_{n1}$, S_2 is $2CNF_{n2}$, $n_1\neq n_2$: $S'\subseteq S_1$, $S'\subseteq S_2$ and $\nexists n$: $Child(n,n_1)=Child(n,n_2)=TRUE$ (i.e., there are no common sub-trees between n_1, n_2, but there is a common sub-Set of clauses). This means also: Neither n_1 nor n_2 nor any children of them were CNs/MSCNs before (S' was not the Clause Set of a CN/MSCN).
Let F={..}+C+D+S, be the l.o. 2CNF Clause Set whose instantiation results in a sub-tree like in (Figure 20c) in which S is a Clause Set, S' a Set containing Derivations of clauses in S[46],

[45] The intuition behind this central observation of Lemma 9-a is the following: When a clause C∈BS has an image C' in a formed MSCN, then, per definition, all its literals and/or images of literals must have been new with respect to branches and edges leading to the MSCN as well as literals in Clause Sets of parent nodes. In that case: Any attempt to split the node using another, subsequent clause D∈BS will be in vain, because of the l.o. condition imposed on BS by 2SAT-GSPRA$^+$ which prescribes **either** that literals and/or images of literals in D be as new as those of C **or** that they be 'pulled' by TLs occurring in clauses before C, *contributing thus to the formation not the splitting of any MSCN augmented in size by C.*

[46] Clauses C, D whose images are not common between the two involved nodes may appear in F either before or after or bracing Sub-Set S, the origin of S'. F={..}+S+C+D, assumes S is resolved to create S' *before* C, D. It relates, therefore, to CN- not N-Splits and is dealt with,

S_1=D'+S' (left node n_1) and S_2=C'+S' (right node n_2), where C,D∈F and C={*x*, *y*}, D={*a, b*}. C' and D' are Derivations of C, D and C',D' as well as any other Derivations of C,D ∉S', C'≠D'[47].

Then: If L is the least Literal used to instantiate F:

Case-1, L≠*x*: C'=C should have been ∈S_1 as well as ∈S_2 which means C'∈S', since in the left node C'≠D'. Contradiction.

Case-2, L=*x*: We distinguish three cases**:**

i- *x*≠*a*, *x*∉D: D'=D should have been ∈S_1 as well as ∈S_2 which means D'∈S', since in the right node C'≠D'. Contradiction.

ii- *x*=*a*, *x*∈D: C'={*y*}, D'={}, D''={*b*} are left- and right-Derivations of D. Then: S_2={*y*}+{*b*}+S', which contradicts S_2={*y*}+S', because D''={*b*}∉S'.

iii- *x* = *a*, ¬*x* ∈D: C'={*y*}, D'={*b*}, D''={} are left- and right-Derivations of D. Then: Because F is l.o., *x*=*a* was least Literal in F: Both *y*, *b* must be <= any literals L∈S'. This means that clauses {*y*} and {*b*} are going both to appear and get instantiated *before* any clauses in S' in both branches of the tree ({*b*} in the left branch, {*y*} in the right branch)[48]. This instantiation creates a common sub-tree between nodes n_1, n_2 whose Clause Set is S' contradicting the definition of a Split.

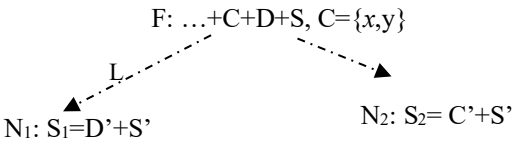


Figure 20c

c- Suppose $[q]_{ST}^{sp1,sp2,sp3}$ is a MSCN which is augmented in size in step *k* by a clause C'. We have just shown that if C' or images of it are ∈BS, then no Splits can occur in any steps >*k*. What about the case where C' is a unit clause, say {*z*}, but ∉BS and there are no clauses D' in the Clause Set of the MSCN such that D' is image of a D∈BS? Augmenting the

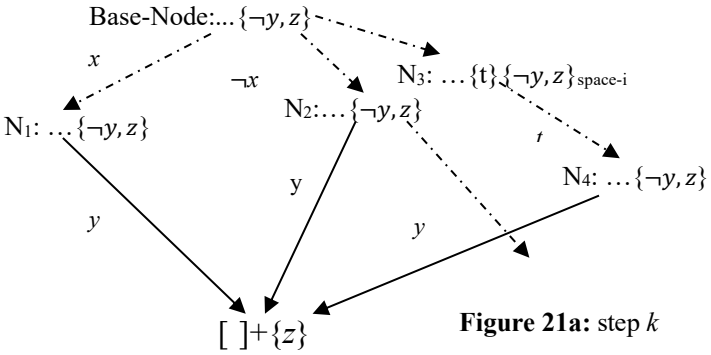


Figure 21a: step *k*

size of $[q]_{ST}^{sp1,sp2,sp3}$ with such a C' in step *k* means that there is a Literal L∈ C, where C={¬L, *z*}∈BS such that all instantiations of C through branches leading to [*q*] agree on its truth value, otherwise a Split would occur in this step. L is a (Non-Distinguished Literal),

indirectly, in the other two parts of this Lemma showing that such a Split can only occur if $rank_{S'}$<2 (Lemma 9-a) and $Size_{S'}$=1 (Lemma 9-c). Form: F={..}+C+S+D is basically a combination between F={..}+C+D+S, the investigated one, and F={..}+S+C+D, i.e., does not provide substantially different insights and is therefore skipped here to avoid unnecessary length.

[47] For showing the result it is actually sufficient to consider the difference between S_1 and S_2 constituting of only one clause (i.e., putting in the argument above either D'={} or C'={}), if we bear in mind that 2SAT-GSPRA$^+$ is a sequential Algorithm and Splits are therefore always formed in a single step in which only *one* clause is processed.

[48] This remains the case even if CRA$^+$ is used, since the (RPC-condition) has no effect on unit clauses. The reader may have noticed that the argument used in Lemma 9-b is independent of renaming and variable spaces and relates only to the l.o. condition and the application of the Least Literal Rule on a Clause Set.

because distinguished ones like ***x*** in (Figure 21a) lead to different instantiations for different branches, i.e., Splits, per definition. It is not a Literal like ***t*** which, although non distinguished, appears only in some, not all branches of the tree. Such a Literal ***t*** would also create dissymmetry and hence Splits when C is instantiated. L is called a CNAL (Definition 4). The Argument below amounts to showing that, in case such CNAL L is used to augment the size of $[q]_{ST}^{sp1,sp2,sp3}$ in any step, no Splits can occur in furtherance, unless Clause Sets of the form:

{..{..<Literal *i*>..} {..no < Literal *i* >..}
{..< Literal *i* >}…} **– wrong-form**

are allowed for <Literal *i*>, used for splitting the MSCN, a situation which, in the studied cases, leads to inconsistency between imposed l.o. conditions on all sets (including BS) on the one hand and the to-be induced Split[49] on the other. Intuitively, the Argument goes as follows: If CNAL L augments the size of the MSCN through a clause, say C={¬L, *z*}∈BS, then L cannot be used to split the same node in any further step, because any clause E containing L and coming after C in the BS can **either** agree with C in the sign of L and shall be thus augmenting the MSCN, not splitting it, **or** disagree and in that case it leaves the MSCN untouched. If on the other hand a <Literal *i* >, different from CNAL L, is used to split the MSCN, i.e., E={*i, j*}, its first appearance in the BS *must come before* the instantiation Block of CNAL L, because otherwise <Literal *i*> would be greater than all branch- and edge-literals of the MSCN, including CNAL L, per the l.o. condition of BS, and thus not able to split the node. A block headed by <Literal *i*> cannot be interrupted as in the above **wrong-form**, which leaves then only one constellation of the BS to be thoroughly investigated in which <Literal *i*> is a TL of some clause before C such as:

{..{*a, i*}..{¬L, *z*}…{*i, j*}…}

Although such a constellation is l.o., where *a*<*i*<L<*z*, instantiation of least literals by 2SAT-GSPRA$^+$ necessarily results in the following non l.o. form:

{..{*i*}..{¬L, *z*}…{*i, j*}…}

In any space, the conversion of this form to l.o. (similar to what we have seen in Lemma 9-a) 'pulls' the clause E to a position in which it can only produce *one single Derivation through all spaces* and contribute to the formation of the MSCN rather than to splitting it.

Formally, we distinguish the two ***only***[50] cases:

[49] Note that Clause Sets similar to **wrong-form** are not always breaching l.o. conditions. For example: S={{0,1}{0,2}{1,3}} (putting <literal *i*> = 1).

[50] The shown two cases are the ***only*** ones, because <literal *i*> which causes the contradiction may here be anyone of the non CNAL literals ***x*** (first case)/***y*** (second case) or ***t***. With respect to what needs to be shown: Those literal types are similar. They: 1- must disappear when the MSCN is augmented in step *k* and 2- can theoretically cause Splits in steps>*k*. The argument shown uses <literal *i*>=***t*** to illustrate the idea w.l.o.g. ***t*** can either appear before or after the CNAL.

Case 1 (step *k*)- CNAL L=y appears in BS after <Literal i> i.e., $(x \mid y)$, $(t \mid y)$ and thus also $x<y$, $t<y$, since BS is l.o. (Figure 21a).

To split $[q]_{ST}^{sp1,sp2,sp3}$ using t or any of its images in a step>k and space-i, two possibilities may occur with respect to Clause Set $S=\{..\{t\}\{\neg y, z\}\}_{\text{space-i}}$ of node N_3:

a) S becomes = $\{..\{t\}\{\neg y, z\}\{t,z'\}\}_{\text{space-i}}$, then **either** $\{t\}$ was already $\in$BS and thus an image of S is $\subseteq$ BS indicating a breach of the l.o. condition, because t must then be both $<y$ and $>=y$ (per l.o.) **or** there exists a clause D$\in$BS such that: D=$\{a, t\}$. But then the BS contains a subset of clauses or images of the form $\{..\{a,t\}..\{\neg y, z\}..\{t,z'\}..\}$ where $t>=y$ also leads to the same inconsistency.

b) S becomes = $\{..\{t\}\{\neg y, z\}\{z', t\}\}_{\text{space-i}}$. As $t>z'>=y$, this means that we have a contradiction for all possible cases of the BS like in a)

Case 2 (step *k*)- CNAL L=x appears in BS before <Literal i> i.e., $x \mid y, x \mid t$ and $x<y, x<t$, (Figure 21b), we have $\forall \underline{i}{:}t_{\text{space-i}}<z_{\text{space-i}}$, as well since $\{z\}_{\text{space-i}}$ must augment the size of the MSCN.

As before: To split $[q]_{ST}^{sp1,sp2,sp3}$ using t or any of its images in a step>k and space-i, two cases may occur with respect to Clause Set $S=\{..\{t\}\{z\}..\}_{\text{space-i}}$ of node N_3:

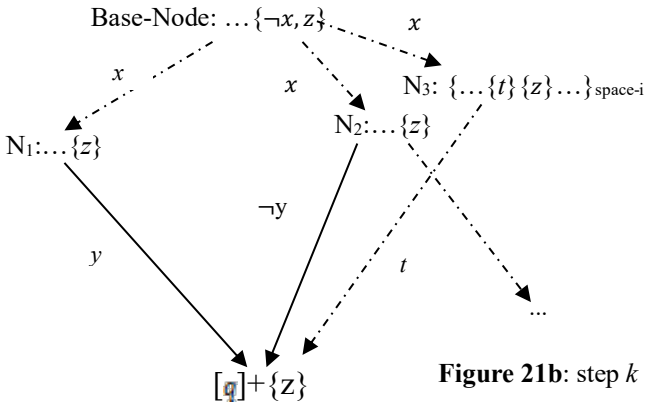


Figure 21b: step k

a) S becomes = $\{..\{t\}\{z\}..\{t,z'\}\}_{\text{space-i}}$, then **either** $\{t\}$ was $\in$BS and in that case $\{..\{t\}..\{\neg x,z\}..\{t,z'\}..\}\subseteq$ BS is a breach of the l.o. condition[51] **or** $\{..\{a,t\}..\{\neg x,z\}..\{t,z'\}\}\subseteq$BS and we have to consider two possibilities:

$\forall \underline{i}{:}(a \mid x)_{\text{space-i}}$: In that case BS has ***only*** one of the two forms[52]:

$\{..\{a,t\}..\{\neg x, z\}..\{t,z'\}..\}$- form1

Or

$\{\{r, a\}..\{s,x\}..\{a,t\}..\{\neg x,z\}\{t,z'\}\}$- form2

Form1 leads to a contradiction with the case assumption, since x appears for the first time in $\{\neg x, z\}$ and is thus per l.o. condition > all literals to its left including t according to (Definition 1-c).

Form2 needs to be transformed by 2SAT-GSPRA$^+$ to $S=\{..\{t\}\{z\}\}_{\text{space-i}}$ in steps<= k according to case assumption. In this form r,s must be $<a, x, t$ as per l.o. condition of BS. This transformation, which creates intermediate spaces, can only be

[51]The arguments used in [Abdelwahab 2016-2] for the corresponding 3CNF case amount to showing that Clause Sets similar in form to: $\{..\{t\}..\{\neg x,z\}..\{t,z'\}..\}$will ***always*** occur in parent sets of $[q]$, if such a node is supposed to be augmented first in size by a CNAL z, then split using t, consistently breaching the l.o. condition and requiring re-arrangement of clauses by CRA$^+$. To completely avoid the impression that this re-arrangement may lead to the same node-count as the one obtained when Splits are allowed, *the arguments used here reflect on the original BS, rather than any arbitrary parent Set*, *showing that all possible l.o. BS forms (used by* 2SAT-GSPRA$^+$*) for the constellations shown in Figure 21 cannot allow – without contradiction - first augmenting the size, then splitting such a [q].*

[52] C.f. Footnote 44 in point a- of this Lemma.

done - using the least Literal rule - as follows:

Suppose *r*<s, then form2 yields two sub-sets:

$\{\{s, x\}..\{a,t\}..\{\neg x,z\}\{t,z'\}\}$ - subset1

$\{\{a\}..\{s,x\}..\{a,t\}..\{\neg x,z\}\{t,z'\}\}$ - subset2

Because subset2 is not l.o., CRA$^+$ converts it giving a form where $\{a\}$ and $\{a, t\}$ are joined in one block B_a after which clause $\{t,z'\}$ appears, i.e.:

$\{\{s,x\}....\{\neg x,z\}..\{a\}\{a,t\}..\{t,z'\}\}_{\text{space-j}}$ - subset2'

Thus, in such a space-j: $z_{\text{space-j}}<t_{\text{space-j}}$ contradicting the case assumption as well as $(a \mid x)_{\text{space-i}}$.

Subset1 yields when resolved two additional Clause Sets:

$\{..\{a,t\}\{\neg x,z\}\{t,z'\}\}$ - subset3

$\{\{x\}..\{a,t\}..\{\neg x,z\}\{t,z'\}\}$ - subset4

Subset3 is similar to form1 and leads to a contradiction with $\forall i{:}t_{\text{space-i}}<z_{\text{space-i}}$. Subset4 needs to be converted to l.o.:

$\{..\{a,t\}..\{t,z'\}..\{x\}..\{\neg x,z\}..\}_{\text{space-l}}$ - subset4'

Where $\{x\}$ and $\{\neg x,z\}$ are summed up in one block B_x which has the effect in such a space-l that $(t_{\text{space-l}} \mid x_{\text{space-l}})$ contradicting the assumption that $t_{\text{space-l}}$ splits the MSCN after it is augmented using the CNAL $x_{\text{space-l}}$.

Suppose *r*=s, then form2 becomes:

$\{\{r, a\}..\{r,x\}..\{a,t\}..\{\neg x,z\}\{t,z'\}\}$ - form2'

Instantiating this formula in steps<=k produces two sub-formulas:

$\{..\{a,t\}\{\neg x,z\}\{t,z'\}\}$ - subset5

$\{..\{a\}..\{x\}..\{a,t\}..\{\neg x,z\}\{t,z'\}\}$ - subset6

Subset5 is again similar to form1 above. Subset6 has to be converted to l.o. yielding blocks B_a, B_x which are uninterrupted and come behind each other, since $(a < x)_{\text{space-i}}$ as per case assumption.

$\{..\{a\}\{a,t\}..\{t,z'\}\ldots\{x\}\{\neg x,z\}\}_{\text{space-m}}$ - subset6'

In such space-m: $(t_{\text{space-m}}|x_{\text{space-m}})$ contradicting the assumption that $t_{\text{space-m}}$ splits the MSCN after it is augmented using the CNAL $x_{\text{space-m}}$.

∀i:(x | a)*<u>*space-i*</u>:** In that case BS has ***only one of the two forms:

$\{..\{\neg x, z\}..\{a,t\}\ldots\{t,z'\}..\}$ - form3

Or

$\{\{r, x\}..\{s,a\}..\{a,t\}..\{\neg x,z\}\{t,z'\}\}$- form4

Form3 makes $z<t$ and forbids thus, because of (Lemma 1-c), in any formed space-i, that: $t_{\text{space-i}} <z_{\text{space-i}}$, unless the precedence of $\{\neg x, z\}$ on $\{a,t\}$ is changed which would be a breach of the l.o. condition, since (x | a) for all spaces.

For Form4 there are two cases:

Suppose *r*<s Then because $x<s<a<t$ the following two subsets will result of the application of the least Literal rule and conversion to a l.o. set:

$\{ ..\{\neg x,z\}..\{s,a\}..\{a,t\}..\{t,z'\}\}_{\text{space-n}}$ - subset7

$\{..\{x\}\{\neg x,z\}..\{s,a\}..\{a,t\}..\{t,z'\}\}_{\text{space-o}}$ - subset8

Both forms don't fulfill case requirement: $t_{\text{space-i}} <z_{\text{space-i}}$

Suppose *r*=s: Then form4 becomes

$\{\{r, x\}..\{r, a\}..\{a,t\}..\{\neg x,z\}\{t,z'\}\}$- form4'

Where $r<x<a<t$ and the following two sub-forms result from the application of the least Literal rule and/or the l.o. condition:

$\{..\{\neg x, z\}..\{a,t\}\ldots\{t,z'\}..\}_{\text{space-p}}$ - subset9

$\{..\{x\}\{\neg x,z\}..\{a\}\{a,t\}..\{t,z'\}\}_{\text{space-q}}$ - subset10

Both forms don't fulfill case requirement: $t_{\text{space-i}} <z_{\text{space-i}}$

b) S becomes = $\{..\{t\}\{z\}\{z',t\}\}$ and since $t>z'>=z$, the same contradictions seen in a) can be shown for all possible BS constellations.[53]

Resuming all cases of Lemma 9-c: BS constellations supporting the intention of first augmenting the size of $[q]_{\text{ST}}^{sp1,sp2,sp3}$ using a CNAL L in step k, then splitting it using <Literal i> all lead to inconsistencies, if <Literal i>≠ L. Since L

[53] The reader may wish to verify this for him/herself in a way similar to the one done for a). One will find out, that changing the position of t in $\{t, z'\}$ to become $\{z',t\}$ does not affect any argument used here. The case is not extended to avoid unnecessary length.

itself cannot be used to split $[q]_{ST}^{sp1,sp2,sp3}$ in steps >*k*, as seen above, this means that such a MSCN cannot be split.

Here is yet another shorter version of the proof of Lemma 9-c using only the '>' relation for interested readers:

In step *k*: L is CNAL in a clause $\{\neg L, z\}$ augmenting the size of [*q*] with {*z*}. As per the definition of a MSCN : Literal *z* is > all branch and edge literals of [*q*], i.e., *z*>L,*a,b,c,d*, where *a,b,c,d*,.. are all edge- and/or branch-literals.

In any step >*k* a clause C={*x*,..} cannot use L to split [*q*], since any +*ve* occurrence of L in C will keep [*q*] as it is. A –*ve* occurrence of L in C will only augment the size of [*q*].

If (*x*>*z*) and (L≠x) then also *x*>*a,b,c,d*. However: To split [*q*]: X needs to be equal to either one of them. Contradiction.

If (*x*<*z*) and (L≠x) then per l.o. condition of BS also *x*>L and *x* cannot appear in a Clause C' before $\{\neg L, z\}$ as a HL, i.e., it must be that C'={..,*x*}, L<*x*<*z*. Since L, *z* are literals of the same clause, they must be kept together in l.o. Clause Sets of parent nodes of [*q*] and their images in any space will *always* appear either *before* or *after* Literal *x* or its images causing contradictions to the case assumption in all cases.

To see this : If *x* or any of its images split the MSCN there has to be an edge marked '*x*' of a parent node *n* in certain space-i such that

$2SAT_n=\{..\{x\}\ldots\{\neg L, z\}\ldots.\{x,\ldots\}\}_{\text{space-i}}$ which is not l.o.

Making $2SAT_n$ l.o. draws $Bx=\{\{x\}\{x,..\}..\}_{\text{space-i}}$ together *prior to* $\{\neg L, z\}_{\text{space-i}}$ which is supposed to augment the size of the MSCN before $\{x,..\}_{\text{space-i}}$ splits it. Contradiction. Even if B_x is drawn after $\{\neg L, z\}_{\text{space-i}}$ like in: $2SAT_n=\{..\{\neg L, z\}..\{x\}\{x,\ldots\}\}_{\text{space-i}}$, this makes $(x>z)_{\text{space-i}}$ and thus also $x_{\text{space-i}}$>all edge- or branch-literals of the MSCN in this space, i.e., not able to cause a split, and augmenting the size of [*q*] only. Contradiction. (Q.E.D.)

It is imperative to summarize the important findings of Lemma 9 before proceeding to the next section:

a) (Lemma 9-a) shows that BigSps, i.e., Splits of rank 2 CN- or MSCN nodes ***cannot*** occur during 2SAT-GSPRA$^+$ resolution. This *anchor result* of the work presented here puts a linear upper bound[54] on the number of nodes which may be created via duplication (Split) of any existing CN/MSCN in any single step and basically means that *sub-problems which need to be solved in different manners again and again by 2SAT-GSPRA$^+$ are always strictly easier to solve than the original problem*.
b) (Lemma 9-b) shows cases where size-1 Splits occur. It also shows another *anchor result*, namely: No N-Splits can occur, because of the l.o. condition.
c) (Lemma 9-c) shows that the linear upper bound of point a) is an exaggeration and *only* a *constant number of nodes* are generated whenever a CN/MSCN splits in any step, because Splits cannot occur for CN/MSCN sizes>1.

Demonstrating then that the maximum number of such CNs/MSCNs/sub-problems must also be small suffices for establishing the main node count result. This is done in the next section.

[54] Rank 1 nodes of any size (i.e., nodes containing only unit clauses) have a linear number of nodes or sub-trees (in M)

III-5 Complexity of 2SAT-FGPRA

We proceed by showing that the number of unique nodes generated by 2SAT-GSPRA$^+$ is bounded above by a polynomial in M, the number of clauses. As 2SAT-GSPRA$^+$ uses in each iteration a data structure in which newly created Clause Sets are stored in their CRA-Form (LCS, c.f. Definition 14) there is a guarantee that no more nodes/Clause Sets are generated than the ones given by the maximum unique node count. (Lemma 2-c) makes sure that CRA-Forms in CNs and/or MSCNs represent Clause Sets which are logically equivalent although they may belong to different spaces.

Lemma 10: In any step $i>=0$ of 2SAT-GSPRA$^+$ resolving an arbitrary BS of size M=i+1 with Clause C_i: Newly added clauses used to align any nodes/sub-trees of Clause Sets S' of size <M produced in steps <i can ***only*** come from ACS. The total number of unique-nodes produced by 2SAT-GSPRA$^+$ for S in the final $MSRT_{s.o}$, including those generated by Splits, is, therefore, bounded above by:

$$2+ c*RCC_{2\text{-}SAT}{}^2 *M^4 + RCC_{2\text{-}SAT} *M^3,$$
$$c<=2, \text{ i.e., } \mathbf{O(M^4)}$$

Moreover: This bound remains polynomial, i.e., **O(M^6)**, even if Splits are allowed which are not BigSps.

Proof: (by induction on M)

Base-Case: M=1: For size 1 nodes the $MSRT_{s.o}$ representing a single clause which is aligned per definition, the single clause itself being the (Alignment-Clause). For M=1 we have, therefore:

$$i=0: 2 <2+ 2*(4)^2 *(1)^4$$

Illustration Case: M=2: The alignment of clause C_1 to C_0 in step i=1 of the resolution adds in the worst case 2 to the nodes of the $MSRT_{s.o}$ of clause C_0 which are also 2 at most (c.f. Lemma 6 and with Figures 13 and 14). Thus, for step M=2 we have:

$$i=1: 2+2 <2+ 2*(4)^2 *(2)^4$$

The practically used ACS-portion is comprised of clause C_1 and/or its derivations.

Induction Hypothesis (size M):
An IRT with a base-node of size M (step i+1) in the form of (Figure 22) (here k=2) is produced by adding in each step only elements of the ACS to the size 1 nodes levels (while aligning clauses to the intermediate IRTs of previous steps) and the total number of unique-nodes, including those resulting from Splits, do not exceed:

$$2+c*RCC_{2\text{-}SAT}{}^2 *M^4 + RCC_{2\text{-}SAT}*M^3, c<=2$$

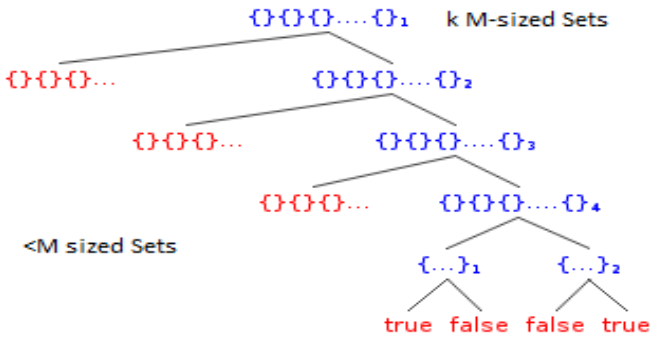


Figure 22: IRT with base-node size M

Induction Step (size M+1):
When IRT is resolved in step *i+2* via 2SAT-GSPRA$^+$ with a clause C:

1. *k* M-sized nodes shall become *k* M+1-sized nodes and l.o. as well (per definition of 2SAT-GSPRA$^+$ and the fact that the BS is l.o.). The breadth *k* of the first clause C_0 in S is not altered and thus also the number of nodes in the (Top-part). No other M+1-sized nodes can be formed.

2. Recall that as per (Lemma 7): The total number of generated size-1-level nodes cannot exceed $RCC_{2\text{-}SAT}*M^2$, if Splits are not counted. In essence we show the same again here, but in the context of only one resolution step: For all <M-sized nodes (when they are resolved with C forming nodes of Sizes <= M): The induction hypothesis applies, i.e., step *i*+1 produced for each one of them at most

$$|ACS|=RCC_{2\text{-}SAT}*M$$

new nodes of size 1 in their respective sub-trees (not counting Splits). Suppose now that in step *i*+2 C is aligned to such a node *n* (Figure 23) needing for the alignment of sub-trees of *n* (not necessarily in the same space) some other clauses C', C'' from ACS. If two or more sub-$MSRT_{s.o}$s of node *n* and/or any other node are aligned with the same clause C, C' or C'', then on size-1 level of the final, overall $MSRT_{s.o}$ a CN/MSCN possessing one unique CRA-form (c.f. Definition 14 in which CRA$^+$ is ***always*** applied before storing any Clause Set) will be built ***only one time*** within a space or between different spaces representing each one of C, C' or C''. In addition: All such non-trivial CNs/MSCNs[55] can only represent members of ACS per definition of ACS (Definition 13). Thus, the total number of newly formed, unique, size 1 nodes for all trees and sub-trees in this step (which may or may not become non-trivial CNs/MSCNs) cannot exceed |ACS| in the worst case[56], i.e.: $RCC_{2\text{-}SAT}$ *M.

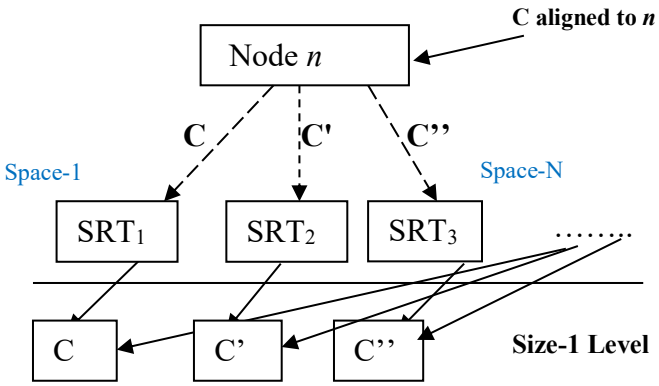


Figure 23

[55] Trivial CNs/MSCNs are not accounted for, because they can be avoided altogether w.l.o.g. as per (Lemma 8).

[56] This is a theoretical exaggeration, since CRA-Forms of clauses like {*a,b*} and {*x,y*} are always the same in reality so that only $RCC_{2\text{-}SAT}$ size-1 nodes are practically added to the overall $MSRT_{s.o}$ in this step. Keeping the factor M lets us assume that 2SAT-GSPRA$^+$ handles permutations of different clauses of the base set differently, storing them in separate places when they appear. This is of course not how 2SAT-GSPRA$^+$ works, but gives us a good way to exaggerate our assumptions about its way of work so that we can get a more reliable upper bound. The exaggeration would be then: To leave the M-factor, while counting any possible Splits of all those redundant nodes as well.

3. As per 2., the total number of generated non-trivial CNs/MSCNs at level 1 cannot exceed $RCC_{2\text{-}SAT} * M^2$ in all steps without counting Splits. To count Splits at level 1: Recall that one copy of a node remains as it is and another copy is resolved with a new clause moving up the hierarchy (c.f., e.g., Figure 20). Assuming for the worst case that each one of those nodes is split by the newly resolved clause C in step *i+2 and* remains in the same level as it is as well: There are $RCC_{2\text{-}SAT}$ ways to do so for any CN/MSCN per definition[57]. Those Splits can only form Clause Sets of size 1 and produce only a constant amount $c(=<2)$ of new nodes each time[58]. If we assume (also as an exaggeration) that step $i+2$ adds all ACS-elements of point 2 as new nodes as well[59], this makes the maximum number of newly added size 1 nodes in this step:

$$c*RCC_{2\text{-}SAT}^2 * M^2 + RCC_{2\text{-}SAT} * M$$

This means that

$$c*RCC_{2\text{-}SAT}^2 * M^3 + RCC_{2\text{-}SAT} * M^2$$

is an upper bound of nodes added to size-level 1 during the whole process of resolution. What about added nodes of sizes >1? (Lemma 9-c) assures us that there are no Splits of nodes at j-size levels, for $j>1$. This means, we can apply the (expansion Lemma 5-c) which asserts that in the worst case and for the whole resolution process: The upper bound of the number of new nodes at all those j-size levels cannot exceed

$$c*RCC_{2\text{-}SAT}^2 * M^3 + RCC_{2\text{-}SAT} * M^2$$

confirming thus the given $O(M^4)$ bound for all levels.

Resuming again: The $O(M^2)$ nodes generated in size level 1, which include (as a worst case) also all possibilities of Splits of CN/MSCNs at this level, may in a further exaggeration all be propagated up the hierarchy of sizes to form at each step and for each size-j-level of nodes $O(M^2)$ additional, new ones. If they are not propagated, they remain in their respective levels and are not accounted for further up in the hierarchy[60].

4. What happens if we relax (Lemma 9-c), i.e., allow Splits at size-levels j, $j>1$, which are *not* BigSps? Any such Split would cause only O(M) new nodes to be generated each time it occurs (as the nodes involved can only be of rank 1). According to (Lemma 5-b) any CN/MSCN [q] in a size-level j and step k must be a CN/MSCN [q'] of size-level j-1

[57] Recall that $RCC_{2\text{-}SAT}$ is the cardinality of the Set of all clauses which are permutations of Literal arrangements of a 2CNF clause C.

[58] We are assuming hence that each newly resolved clause in each step $i+2$ comes with a least-literal equivalent to previously instantiated block literals of parent-nodes of every non-trivial CN/MSCN created before in every space and Splits this non-trivial CN/MSCN in all possible ways without breaching any l.o. condition. A clear exaggeration.

[59] Even if new nodes coming from ACS in this step are counted twice this way: It only helps the exaggeration intended here.

[60] Remember that, because there are no Splits at such levels, a node in any size-level-j, $j>1$, can either be propagated up in the hierarchy or left as it is, but not both, the argument here can also be expressed as follows: The $O(M^2)$ new nodes formed at size-level 1 in each step may in the worst case always stop at a certain level $j>1$ and not be propagated further up in the hierarchy. In that case level-j will contain at the end of the resolution process at most $O(M^3)$ unique nodes. Assuming that all other levels are similar to level-j (an exaggeration which can *never* happen), we get the $O(M^4)$ bound.

created in steps<k and augmented by the new resolved clause in step k. This means that the number of CNs/MSCNs which can split in any size-level j cannot exceed the maximum number of CNs/MSCNs at size-level j-1 and ultimately at size-level j=1, i.e., in the worst case $O(M^2)$ as seen in point 2. *Relaxing Lemma 9-c, we can, therefore, assume as a worst case that* a *new resolved clause at any step k splits all CNs/MSCNs residing in all levels j>=1 in* $RCC_{2\text{-}SAT}$ *possible ways creating at each level the maximum possible amount of* $RCC_{2\text{-}SAT}$*$O(M)$ *new nodes for each CN/MSCN and that those nodes may all be propagated up the hierarchy as well.* Thus, the upper bound of unique nodes created through Splits at any level j>=1 and in any step k is: $O(M^3)$, i.e., $O(M^4)$ for all steps. Using a simple inductive argument on size-levels 1<=j<=M, we can show that the overall upper bound of unique nodes is $O(M^6)$, whether nodes are generated through Splits or through propagation.

Base Case: Level j=1 contains at the end of the resolution at most: $O(M^3)$<=1*$O(M^4)$ unique nodes as just seen.

Induction Hypothesis: Size-level j contains at the end of the resolution: j*$O(M^4)$ unique nodes, j<=M

Induction Step: For size-level j+1: As per 2SAT-GSPRA$^+$ (Definition 14) a node can only become of size j in any step k when either it was of size j-1 in steps <k and it got augmented in size **or** when it was generated via Split. Unique nodes created through Splits cannot exceed $O(M^4)$ for all levels as just seen. Per induction hypothesis: The number of unique, size-level j nodes never surpasses j*$O(M^4)$, which makes the total number of unique nodes in size-level j+1 after resolution terminates: (j+1)*$O(M^4)$. As j<=M, we have in each such level j at the end: $O(M^5)$, making the overall upper bound for the whole $MSRT_{s.o}$: $O(M^6)$.

(Q.E.D.)

Finally: The following Lemma shows that 2SAT-FGPRA (Definition 15) can simulate 2SAT-GSPRA$^+$ correctly, i.e., producing exactly the same $MSRT_{s.o}$ when taking the same Clause Set sorting choices. It also gives an asymptotic upper bound of the number of operations needed by 2SAT-FGPRA[61].

Lemma 11: The following is true:

a- For any arbitrary 2CNF Clause Set S: ∃G:$MSRT_{s.o}$ such that: 2SAT-FGPRA(S)=2SAT-GSPRA$^+$(S)=G.

b- For 2SAT-FGPRA to produce G shown to exist in point a-: For the main Assistance Operations[62] used by 2SAT-FGPRA on 2CNF Clause Sets S of size M: Node creation and returning results (function SubTree), $MSRT_{s.o}$ creation for a single clause (function Convert), CRA$^+$, Forming new Clause Sets using least-Literal-rule (instantiation), Storing (nodes), Searching Clause Sets in LCS:

[61] Definitions: (14) and (15) of both Algorithms deliberately leave the issue of choosing C_0, the head clause of 2CNF Clause Set S to the respective implementations of the Algorithms, thus opening up the possibilities for choices which may lead to different node counts. (Lemma 11) shows that whatever those choices for 2SAT-GSPRA$^+$ are, 2SAT-FGPRA can simulate them correctly. Since only l.o. Clause Sets are used in any sub-problems generated by instantiation operations, 2SAT-FGPRA is producing a $MSRT_{s.o}$ equivalent to one produced by 2SAT-GSPRA$^+$, which always has a polynomial number of unique nodes as just seen in (Lemma 10).

[62] By *Assistance Operations* we mean modules and/or sub-functions used in the pseudo-code of 2SAT-FGPRA.

The total, worst case number of Primitive Operations[63] performed by any single one of them during a run of 2SAT-FGPRA is: **O(M^9)**. Moreover: Relaxing Lemma 9-c yields an upper bound of **O(M^{13})**.

Proof:

a- (induction on M, the size of S)**.** Assume that both Algorithms use the same ordering choices in CRA^+. Both Algorithms use CRA^+ in their preparation phases (points 2 and 3 in Definitions 14 and 15) on the same S, i.e., they order clauses in S in the same way. Remember also that they *always* convert Clause Sets to l.o., particularly in Top-Parts of resolution trees, using the same CRA^+ as well.

Base-Case: M=1: Because there is only one $C_0 \in S$, they convert it into the same $MSRT_{s.o}$ G. In that case obviously:
2SAT-FGPRA (S)=2SAT-$GSPRA^+$(S)=G.

Induction Hypothesis:
For all 2CNF Clause Sets S of size M: $\exists MSRT_{s.o}$ G such that:
2SAT-FGPRA (S)=2SAT-$GSPRA^+$(S)=G.

Induction Step: If S is l.o. of size M+1, then let S'=S\A, where A is the last clause in S. Per induction hypothesis: $\exists MSRT_{s.o}$ G such that:

2SAT-FGPRA(S')=2SAT-$GSPRA^+$(S')=G

and we distinguish two cases:

1- When A is aligned to G by 2SAT-$GSPRA^+$ to form G' of S there is no breach of any l.o. condition in any parts of G and A is appended to all Clause Sets of G in Top- as well as Bottom-parts. In that case: Top-parts of G' are clearly equivalent for both Algorithms because Literal choices of $C_0 \in S$ are not affected by the addition of clause A in either case and A is appended to Clause Sets which are exactly the same for both Algorithms. We use the induction hypothesis for Bottom-parts stating that there are always graphs G_1, G_2,…G_n which are equivalent for both Algorithms and can be substituted for Bottom-parts of G' to conclude that:
2SAT-FGPRA (S)=2SAT-$GSPRA^+$(S)=G'

2- When A is aligned to G by 2SAT-$GSPRA^+$ to form G' of S and there is a breach of the l.o. condition in some Clause Set S'' in the Top-part of G': Because this breach relates only to A, while all other clauses $C_i \in$ S'' are as per induction hypothesis the same for both Algorithms (and they use same choices for CRA^+ as well), both Algorithms fix the breach generating the same exact Clause Sets in Top-parts of G' and produce thus the same, related Bottom-parts. If on the other hand A causes the breach in Bottom-parts whose Clause Sets are all of size M, the induction hypothesis applies and there are graphs G_1, G_2,…G_n which are equivalent for both Algorithms and can be substituted for such Bottom-parts of G', thus:

2SAT-FGPRA(S)=2SAT-$GSPRA^+$(S)=G'.

b- Because of (Lemma 10), we know that the total number of unique-nodes in G cannot exceed $2+ c*RCC_{2\text{-}SAT}^2 *M^4 + RCC_{2\text{-}SAT} *M^3$, $c<=2$ (taking the result obtained *without* relaxing Lemma 9-c). Since G is produced by 2SAT-FGPRA as per point a- as well: The following are then upper bounds of the total number of invocations of Primitive Operations for all Assistance Operations listed above for that Algorithm (c.f. Definition 15):

[63] Primitive Operations take a constant amount of time in the RAM computing model.

1. $2+ c*RCC_{2\text{-}SAT}^2 *M^4 + RCC_{2\text{-}SAT} *M^3$ times CRA^+ (each node needs renaming of its Clause Set so that it can be stored in LCS in its CRA-Form). Through (Lemma 3) it is known that CRA^+ takes $O(M^2(\log M+N))$. Since N cannot exceed $c*M$, i.e., is in $O(M)$[64], this makes the total worst case number of Primitive Operations for this category: **$O(M^7)$.**
2. $2*(2+ c*RCC_{2\text{-}SAT}^2 *M^4 + RCC_{2\text{-}SAT} *M^3)$ times instantiation (two new Clause Sets are formed for each node in the worst case). Instantiating a Clause Set by substituting values TRUE or FALSE for a certain Literal in all M clauses is an operation in O(M). This makes the total number of Primitive Operations for instantiation: **$O(M^5)$**.
3. $2+ c*RCC_{2\text{-}SAT}^2 *M^4 + RCC_{2\text{-}SAT} *M^3$ times node creation assuming that it is in $O(c)$, i.e., **$O(M^4)$**. Same amount is needed for all SubTree function invocations, since getting from LCS a stored sub-tree using its index may be assumed to take $O(c)$ operations.
4. $2+ c*RCC_{2\text{-}SAT}^2 *M^4 + RCC_{2\text{-}SAT} *M^3$ times Storing/Appending in/to LCS assuming that it is in $O(c)$, i.e., **$O(M^4)$**.
5. $MSRT_{s.o}$ creation for a single clause: $O(c)$, since independent of M any clause can have at most 2 literals where 2 nodes are created for each one of them.
6. $2+ c*RCC_{2\text{-}SAT}^2 *M^4 + RCC_{2\text{-}SAT} *M^3$ times Searching Tuples in LCS. This search operation can be accomplished in the least efficient way[65] by sequentially comparing the sought Clause Set with all Clause Sets stored in the LCS, a single comparison of two Clause Sets being in O(M). In the worst case there are $2+ c*RCC_{2\text{-}SAT}^2 *M^4 + RCC_{2\text{-}SAT} *M^3$ Clause Sets in LCS, i.e., $O(M^8)$ comparisons are needed. This makes the total number of Primitive Operations for Searching **$O(M^9)$**.

If we relax Lemma 9-c we obviously get $O(M^{13})$ as the number of unique-nodes in G would be in $O(M^6)$ as per (Lemma 10) and the search operation in point 6 above is, as seen, the bottle-neck of 2SAT-FGPRA, requiring in the worst case: $O((\text{unique-nodes})^2*M)$ operations.
(Q.E.D.)

III-6 Counting Solutions

In this section we show that there exists an efficient Algorithm which counts solutions in the final $MSRT_{s.o}$ produced by 2SAT-FGPRA. We give an example of its application. Correctness and efficiency are shown in Lemmas (13) and (14) respectively.

Count2SATSolutions:
Inputs: The $MSRT_{s.o}$ generated by 2SAT-FGPRA for a 2CNF Clause Set S
Outputs: Solution Count (Integer)
Steps: -
1- NamedMSRT = Name nodes and edges starting from 0 and determine their levels. (Algorithm: **DetermineLevels** below)
2- Set Solution Count for node n0 = 0, and for edges on level 1 to be =1
3- For all levels *i* in NamedMSRT
a. For all edges e_{ij}, *j* is the index of an edge at level *i*:

[64] To c.f. this: Let $M=f(N)$. *f* can be exponential, i.e., N=O(log M), polynomial, i.e., $N=O(M^{1/k})$ for a given *k* or linear, i.e., $N=c*M$, $c<=2$, which is the largest count N can reach, representing the case where all clauses have distinct variables.

[65] The least efficient way is chosen to avoid any assumptions regarding sort- and search orders of Clause Sets in LCS.

i. Set Solution Count of e_{ij}=Solution Count of parent node

b. For all nodes n_{ik}, k is the index of a node at level i:

i. If n_{ik} is a TRUE leaf:
Solution Count of $\mathbf{n_{ik}=(\sum e_x * 2^{i-Le}) * 2^{N-i}}$, where x represents the index of any edge going into n_{ik}, $\mathbf{e_x}$ is the solution count of such an edge, Le is edge level of x [66], N number of variables in S

ii. Else
Solution Count of $\mathbf{n_{ik}=\sum e_x * 2^{i-Le}}$

4- Return SolutionCount==$\sum$ Tnd, Tnd is a TRUE leaf node

Algorithm – A3

Determining levels of nodes in the $MSRT_{s.o}$ in the first step of Count2SATSolutions requires calculating the longest path from the source node to each other node, since a node may have several paths and its level relates only to the longest one as per (Definition 0.3), a problem which is in general NP complete [Schrijver 2003][67]. However: The single-source longest path problem for an un-weighted DAG (like the $MSRT_{s.o.}$) has an efficient and even linear solution (O(|V|+|E|), V vertices and E edges) which uses topological ordering. In [Dasgupta 2006] (Ch. 4.7, p. 130), a single-source shortest-path algorithm for DAGs is described. It only needs to perform a sequence of updates that includes every shortest path as a subsequence. The key source of efficiency is that in any path of a DAG, the vertices appear in increasing linearized order. Therefore, it is enough to linearize (that is, topologically sort) the DAG by depth-first search, and then visit the vertices in sorted order, updating the edges out of each. The scheme doesn't require edges to be positive. In particular, one can find longest paths in a DAG by the same Algorithm: Just negating all edge lengths. The following slightly modified Algorithm first creates an ordering for the $MSRT_{s.o}$ and then calculates the longest distance from the source to each node (which is then set to the level of that node). Correctness and efficiency of the original algorithm is discussed in the above reference.

DetermineLevels:
Inputs: The $MSRT_{s.o}$ generated by 2SAT-FGPRA for a 2CNF Clause Set S
Outputs: Nodes named and their levels calculated
Steps: -
1- Scan the $MSRT_{s.o}$ recursively, rename edges and nodes and form the topological, lineralized order in a depth first traversal manner[68].
2- For all $u \in V$:

$dist(u) = \infty$

$dist(s) = 0$, s is source node

3- for each $u \in V$, in the linearized order:

$dist(u)$= **Dist(u, $MSRT_{s.o.}$)**

L_u=| $dist(u)$|

Dist:

Inputs: $u \in V$, DAG = (V,E)
Outputs: Integer representing distance from u to source of DAG
Steps: -
for all nodes $v_1, v_2, .. v_n \in V$ such that $(u, v_i) \in E$:

Dist (u,DAG) =

min{

[**Dist** (v_1,DAG) + l(u, v_1)], ….

[66] Recall as per (Definition 0.3): $L_e = L_{Sr}+1$ if Sr is the Source of e.

[67] The longest path problem for a general graph is not as easy as the shortest path problem because it doesn't have optimal substructure property, i.e., that sub-paths between two nodes have themselves to be optimal (enabling the greedy strategy).

[68] A topological sort order of nodes is basically an inequality, which may be formed in the following way: For any two nodes n_1, n_2 children of node n: create the inequality $n<n_1<n_2$ and add it to the final inequality formed recursively through depth first traversal. When the inequality is extended: Node n_1 and its children comes before n_2 and its children according to precedence in already constructed inequalities.

[**Dist** (v_n,DAG) + l(u, v_n)]

}

l(u,v_i,) is the length of the edge from u to v_i (which is always '-1').

Algorithm – A4

Applying this algorithm to the $MSRT_{s.o}$ produced for Clause Set: S={{0,1}{0,2}{3,4}} for example yields after the first step (Figure 24).

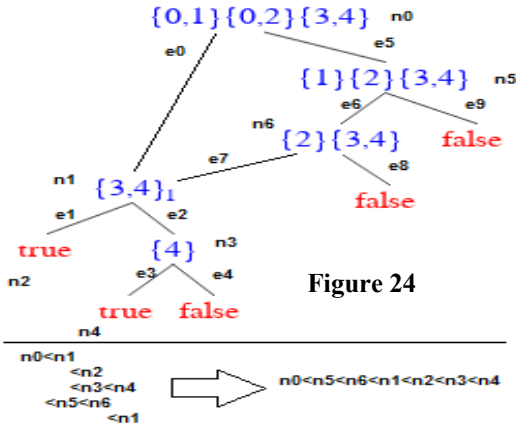


Figure 24

In the third step the following example sequence of operations is performed to get the longest distance from n_0 to n_2 whose absolute value corresponds to the level of n_2:

a) The only way to go from n_0 to n_2 is through the only direct predecessor node n_1. Thus dist(n_2) = Dist(n_1,DAG) -1
b) Dist(n_1,DAG)=min{Dist(n_0,DAG) -1,Dist(n_6,DAG)-1}= min{-1,Dist(n_6,DAG)-1}
c) Dist(n_6,DAG)= Dist(n_5) -1
d) Dist(n_5,DAG)= Dist(n_0) -1=-1
e) Dist(n_6,DAG)=-2
f) Dist(n_1,DAG)= min{-1,-3}=-3
g) dist(n_2) = -4
h) L_{n2}=4

(Figure 25) shows all nodes and their levels.

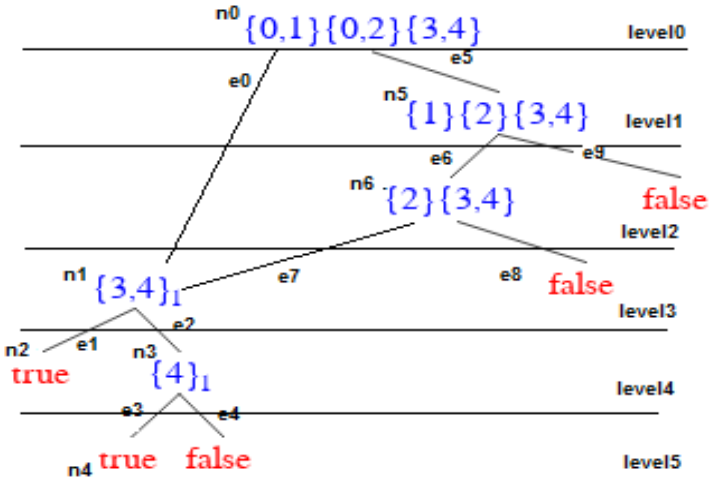


Figure 25

1 of **Count2SATSolutions** as just seen, executing steps 2, 3 through all levels gives the following sequen ce of operations completing thus the example:

a) Level-0: n_0=0
b) Level-1: e_0=1,e_5=1, n_5=e_5*$2^{i\text{-}Le5}$=1*$2^{1\text{-}1}$=1
c) Level-2: e_6=n_5=1, e_9=n_5=1, n_6= e_6*$2^{i\text{-}Le6}$=1*$2^{2\text{-}2}$=1
d) Level-3: e_7=n_6=1, e_8=n_6=1, n_1=e_0*$2^{i\text{-}Le0}$+e_7*$2^{i\text{-}Le7}$= 1*$2^{3\text{-}1}$+1*$2^{3\text{-}3}$=5
e) Level-4: e_1=n_1=5, e_2=n_1=5, n_2=(e_1*$2^{i\text{-}Le1}$)*$2^{N\text{-}i}$= (5*$2^{4\text{-}4}$)*$2^{5\text{-}4}$=10, n_3=e_2*$2^{4\text{-}4}$=5,
f) Level-5: e_3=n_3=5, e_4=n_3=5, n_4=(e_3*$2^{i\text{-}Le3}$)*$2^{N\text{-}i}$= (5*$2^{5\text{-}5}$) * $2^{5\text{-}5}$=5
g) Solution Count=n_4+n_2=**15**[69]

In the next Lemma we show that both 2SAT-GSPRA$^+$ and 2SAT-FGPRA are complete 2SAT-Solver Algorithms. As per (Lemma 11-a) 2SAT-FGPRA simulates 2SAT-GSPRA$^+$ correctly producing the same $MSRT_{s.o}$s. It is thus sufficient to prove this property for 2SAT-GSPRA$^+$. Doing this will enable us to focus in the correctness proof of **Count2SATSolutions** on $MSRT_{s.o}$s rather than on truth tables

[69] The reader may wish to verify this number by constructing the truth table and counting the assignments satisfying S

Lemma 12 (completeness, truth table equivalence): 2SAT-GSPRA$^+$ and 2SAT-FGPRA are complete, truth table equivalent Algorithms, i.e.: Let **S** be a 2CNF Clause Set, **A** any Assignment of truth values of literals in **S**, then: Applying **A** on the $MSRT_{s.o}$ produced by any of the two Algorithms leads to a TRUE leaf **iff A** satisfies **S**.

Proof: We are going to show the result w.l.o.g. for 2SAT-GSPRA$^+$ only (by induction on M, the number of clauses in S)

Base: M=1[70] for the following $MSRT_{s.o}$:

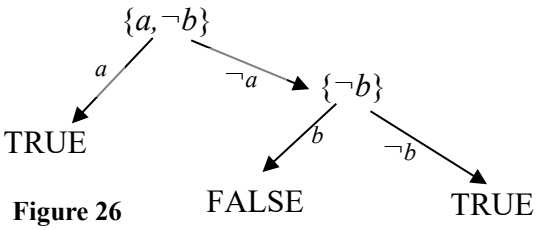


Figure 26

If we construct the Truth Table T2

a	*b*	S
0	0	1
0	1	0
1	0	1
1	1	1

Truth Table - **T2**

and use the following propagation rule to apply any Assignment A to any node in the $MSRT_{s.o:}$

"If the input value of the least Literal in A is TRUE go left, else go right. Apply this rule to all literals in A and nodes in the $MSRT_{s.o}$ until you reach* a *leaf".

Then, the obtained results are equivalent to the ones found in the truth table. Check the two marked cases: For assignment A="01" the base-node will take us right through edge $\neg a$, then left through edge *b* making the overall value FALSE as the one indicated in the truth table. For Assignment A="10" we are taken by edge a directly to the value TRUE which is the value of the truth table as well.

Induction Hypothesis: For all Assignments **A** of truth values to literals in **S**, $Size_S$=M: Applying A to the $MSRT_{s.o}$ using the above propagation rule returns TRUE **iff A** satisfies **S**.

Induction Step: Let S=S'+C, $Size_S$=M+1. Remembering that S must be l.o.: When C= {*x*,*y*} is added to S' the following cases can be distinguished:

1. *x,y are new with respect to S':* 2SAT-GSPRA$^+$ propagates C until leaves are reached (per l.o. condition the new variables are > all literals in branches of the previous tree). If leaves are +*ve* then the tree representing **C** will substitute them, otherwise FALSE is left. Each branch ending with TRUE stands per induction hypothesis for the fact that - without the newly added clause {*x*,*y*} - the Set **S'** had already a satisfiable assignment **A** and what is missing is to satisfy {*x*,*y*} only by extending **A** with a partial assignment giving *x*, *y* truth values so that **A** becomes **A'**. This is done through the extension produced by 2SAT-GSPRA$^+$ which is a tree **T** similar to the one in the base case. Because we need only to check the two new variables, it is easily seen (as in the base case) that for all TRUE leaves of **T**, reachable using the propagation rule: **A'** satisfies **S'** +{*x*,*y*} and vice versa, i.e., if a given **A'** satisfies **S'**+{*x*, *y*} through giving literals *x* or *y* the value TRUE, then a TRUE leaf in **T** must be reachable via the above procedure. When on the other hand a branch terminates with FALSE, reachable through any assignment **A,** it is guaranteed by induction

[70] The case used here (w.l.o.g.) is not the only permutation of +*ve*/-*ve* literals *a,b* combined in a clause. The reader is encouraged to check other permutations and verify the validity of the property for M=1 in a similar way to the one shown here.

hypothesis also that **S'** is not satisfiable by **A** even without taking the new clause into consideration. Thus, **A'** does not satisfy **S'**+{*x*,*y*} as well for any truth values given to *x* and/or *y*.

2. *x exists in S', while y is new:* When **C** is propagated through branches of the tree, those terminating with FALSE and reachable through assignment **A** - as seen in the previous case - are not dependent on the new clause and will keep their values and guarantee (per induction hypothesis) that **S'** is not satisfiable. Therefore for that case: Any new assignment **A'** adding a new variables to **A** is not satisfying **S'**+{*x*,*y*} as well. For all those branches which terminate with TRUE it either might be the case that this truth value is independent of the new variable *y* and thus kept as it is per induction hypothesis (i.e., **A** satisfies **S'**+{*x*,*y*}), **or** it is dependent on *y* and the branch (and per induction hypothesis its corresponding assignment **A**) is extended with a sub-tree containing two possibilities of partial assignments satisfying the single new clause {*y*}: (*y*=TRUE) and (*y*=FALSE).Then: If A'=A+(*y*=TRUE) satisfies **S'**+{*x*,*y*}, it leads to a TRUE leaf using the above procedure and if A'=A+(*y*=FALSE) doesn't it leads to a FALSE leaf (first direction) while if **S'**+{*x*,*y*} has to be satisfied and we are on a TRUE leaf, A'=A+(*y*=TRUE) can be used to do that (other direction).
   Resuming the case of C = {*x*,*y*}: **Either** no new nodes are added to the tree in all those branches where *x* and/or *y* already exist and where per induction hypothesis the tree is already equivalent to the right truth table **or** *x* and/or *y* are new in some branch. In that case they will be added to the *+ve* leaves accordingly and correspond to specifications of truth table values which were *don't cares* before[71].

(Q.E.D.)

The following Lemma shows then the correctness of **Count2SATSolutions**.

Lemma 13 (Correctness): Let **S** be a 2CNF Clause Set for which 2SAT-GSPRA$^+$ or 2SAT-FGPRA produce a $MSRT_{s.o}$, AllAssignments the set of all satisfibale Assignments of S, then: Count2SATSolutions(S)=|AllAssignments|.

Proof: (by induction on N, the number of levels of nodes in the $MSRT_{s.o}$)

Base: N=1: Let S={{*a*}}, then Count2SATSolutions produced in step 1 the following tree:

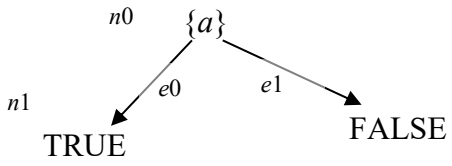


Figure 27

After which the following sequence of operation steps follow:

a) Level-0: n_0=0
b) Level-1: *e*0=*e*1=1, *n*1=*e*0=1
c) Result =1

Which represents the single assignment satisfying S, namely: {(*a*=TRUE)}

[71] For illustration: Consider the case where {1,2} is added to {0,1}{0,2}. The left branch of the tree for {0,1}{0,2} which is the leaf TRUE, corresponds to the fact that values of 1&2 are not relevant for the overall value of the formula {0,1}{0,2} when literal 0 is set to TRUE following this particular assignment branch, i.e., they are *Don't Cares*. When {1,2} is added, its tree replaces TRUE indicating for what values of 1 & 2 the same truth table gives truth values capturing satisfiability conditions of the newly added clause {1,2}.

Induction Hypothesis: For all levels N in the $MSRT_{s.o.}$: All node- and edge values calculated via Count2SATSolutions for that level represent the exact number of solutions possible through the respective node or edge.

Induction Step: For level N+1, when node- and edge-values of that level are calculated:

1- Edge values are equivalent to values of parent nodes which are all correct per induction hypothesis.

2- Node values are summations of edge values **either** from the same level and in that case (per -1) correct **or** from prior levels. Call an edge from prior levels ***e***. The value of ***e*** is also correct per induction hypothesis, but in need for a multiplication factor (step 3-b-ii): $\mathbf{2^{(N+1)-L_e}}$ representing the number of exponential possibilities of partial assignments lost by ***e*** through skipping variables. Trivially: Any skipped variable is accounted for by the multiplication factor of 2.

3- Values for nodes which are TRUE leaves are, with respect to whatever happened before them, correct (as per -1 and -2), but in need of another multiplication factor $\mathbf{2^{NumberOfVars-(N+1)}}$ representing the number of exponential possibilities of partial assignments lost through stopping at that level.

Therefore, the conclusion is that assuming Count2SATSolutions counts the solutions correctly for any level N, it does the same for level N+1.
(Q.E.D.)

This section concludes with an upper bound on the number of operations needed by Count2SATSolutions.

Lemma 14 (Efficiency): Let S be a 2CNF Clause Set for which 2SAT-GSPRA$^+$ or 2SAT-FGPRA produce a $MSRT_{s.o}$: The number of steps taken by Count2SATSolutions to count all exact solutions of S is in $\mathbf{O(M^9)}$, M being the number of clauses (size) of S. If we relax (Lemma 9-c) we get $\mathbf{O(M^{13})}$.

Proof: Remembering that the number of nodes/vertices of a $MSRT_{s.o}$ is $O(M^4)$ (as per Lemma 10) and edges cannot exceed thus $O(M^8)$ in this DAG, we have the following:

1- Step 1 in Count2SATSolutions, i.e., DetermineLevels Algorithm, takes an amount of steps linear in the number of nodes and edges, i.e., $O(M^8)$:

a- Scanning the $MSRT_{s.o}$ in the first step to rename nodes and edges and calculate the topological order is in $O(M^8)$

b- Applying the single-source shortest-path algorithm for DAGs is in $O(M^8)$ as well (c.f. [Dasgupta 2006], Ch. 4.7, p. 130)

2- In further steps Count2SATSolutions loops through all levels calculating edge- and node-values for each level. In the worst case, this would be $O(M^8*N)$, where N is the number of variables in S. Since N is in O(M) (c.f. Lemma 11 Footnote 64), we get an upper bound of $O(M^9)$. Relaxing (Lemma 9-c) gives us as per (Lemma 10) $O(M^6)$ for the unique node count, which makes counting in $O(M^{13})$ in that case.
(Q.E.D.)

Now we are ready for the main theorem of this paper.

III-7 Main Result

Theorem 1:

a- Let S be a *k*CNF Clause Set, $k>0$, *k*SAT-GSPRA$^+$ and *k*SAT-FGPRA Algorithms which are generalizations of 2SAT-GSPRA$^+$ and 2SAT-FGPRA allowing *k*CNF Clause Sets as input, but agreeing on all other resolution steps, in particular those related to:

i- Imposing l.o. conditions via CRA$^+$ and using least literals for instantiation

ii- Creating CNs/MSCNs at any size-level *j* only from CNs/MSCNs at size-level *j*-1

and for which we can show that:

1- No N-Splits can exist
2- No Splits of CN/MSCN nodes of rank *k* can exist
3- *k*SAT-FGPRA simulates *k*SAT-GSPRA$^+$ correctly

And let U^k denote an upper bound of the number of unique nodes generated in a $MSRT_{s.o}$ through anyone of *k*SAT-GSPRA$^+$ or *k*SAT-FGPRA while resolving S, ***then***:
$U^k<=U^{k-1}*O(M^5)$ where U^{k-1} is polynomial in M, M number of clauses of S. *k*SAT-FGPRA is in P, more particularly in: $O(M)*(U^k)^2$. This implies that P=NP.

b- Counting the exact number of Assignments which satisfy Q, a 2CNF Clause Set, (called the #2SAT problem) is in P: $O(M^9)$, or, if (Lemma 9-c) is relaxed: $O(M^{13})$. Because of this also: P=NP.

Proof:

a- Proof is by induction on *k*, the rank of the *k*CNF Clause Set S:

Base Cases: *k*=1: Obviously: If S is a 1CNF Clause Set (i.e., formed only of unit clauses) we get $MSRT_{s.o}$s with O(M) unique nodes which are formed via anyone of 1SAT-GSPRA$^+$ or 1SAT-FGPRA through instantiation of uniquely occurring literals one by one (after converting S to a l.o. 1CNF Clause Set). Complexity of 1SAT-FGPRA is $O(M^3)$ as searching already resolved 1CNF Clause Sets requires: $O((\text{unique-nodes})^2*M)$ operations (c.f. Lemma 11). 1SAT-FGPRA is in P. CNs do not exist. Splits don't exist as well, because sub-formulas can only appear in one node.

***k*=2**: (Lemma 10) in this work asserts that: If S is a 2CNF Clause Set, then even relaxing the property that Splits (produced by anyone of 2SAT-GSPRA$^+$ or 2SAT-FGPRA) in size-levels $j>1$ of a $MSRT_{s.o}$ cannot exist, shown to be true in (Lemma 9-c), yields a unique node count of only $O(M^6)$. Recall that this node count was obtained as follows: Because rank $k=2$ CN/MSCNs cannot split as per (Lemma 9-a) and no N-Splits can occur as per (Lemma 9-b) as well, and because CNs/MSCNs at any size-level *j* only come from CNs/MSCNs at the lower size-level *j*-1 (Lemma 5-b), only $O(M^2)$ rank $k=1$ CN/MSCNs may split in the worst case at any one step forming each O(M) new nodes (the node count of 1CNF $MSRT_{s.o}$s) at any size-level *j*. For all steps this makes them $O(M^4)$ nodes generated via Splits per size-level. A size-level $j<=M$ accumulates in the worst case also whatever may have been generated in the lower size-level *j*-1, which is given by $(j-1)*O(M^4)$ making the overall node count $j*O(M^4)=O(M^5)$ per size-level. For all size-levels we get then the $O(M^6)$ bound in (Lemma 10). Putting $U^1=O(M)$ in inequality $U^2<=U^1*O(M^5)$ yields the same result. For the complexity of 2SAT-FGPRA: $O(M^{13})= O(M)*(M^6)^2$ as shown in (Lemma 11). 2SAT-FGPRA is in P.

***k*=3**[72]: In [Abdelwahab 2016-2] it is shown that:

1- No Splits of CN/MSCN nodes of rank k=3 can exist (Lemma 9)

2- FGPRA simulates GSPRA$^+$ correctly (both of them conceived for k=3).

Although it is not explicitly shown there that N-Splits don't exist in MSRT$_{s.o}$s produced by anyone of FGPRA or GSPRA$^+$, the argument seen in (Lemma 9-b) in this work can be extended to demonstrate that it is indeed the case[73]. GSPRA$^+$ has also the feature of reconstructing sub-trees in case a Clause Set is found to be not l.o. This is the same condition which enabled us to deduce (Lemma 5-b) that: CNs/MSCNs at any level j can only come from CNs/MSCNs at the lower level j-1. Although Lemma 13 in [Abdelwahab 2016-2] shows, similar to (Lemma 10) in this work, an upper bound of $O(M^4)$ of unique nodes, because size-j>1 Splits are not possible, relaxing this condition enables us to use exactly the same arguments used for the above k=2 base case. When we do so: Putting $U^2=O(M^6)$ in $U^3<= U^2*O(M^5)$, gives us the unique node count of $O(M^{11})$, and a complexity of $O(M^{23})=O(M)*(M^{11})^2$ for FGPRA which, although larger than the $O(M^9)$ result of [Abdelwahab 2016-2] is still in P of course.

Induction Hypothesis[74]**:** For any kCNF Clause Set S, k>0, kSAT-GSPRA$^+$ and kSAT-FGPRA Algorithms satisfying conditions 1, 2 & 3 above: $U^k<=U^{k-1}*O(M^5)$, where U^{k-1} is a polynomial expression in M, kSAT-FGPRA is efficient, more particularly its time complexity is given by: $O(M)*(U^k)^2$.

Induction Step: Suppose for a (k+1)CNF formula F that we can show (k+1)SAT-GSPRA$^+$ has the following properties:

1- No N-Splits can exist
2- No Splits of CN/MSCN nodes of rank k+1 can exist
3- (k+1)SAT-FGPRA simulates (k+1)SAT-GSPRA$^+$ correctly.

We do this, for example, by extending the arguments used, per induction hypothesis, to show the same for kSAT-FGPRA and kSAT-GSPRA$^+$. Then our argument for k+1 may go as follows: Because rank k+1 CN/MSCNs cannot split and no N-Splits can occur as well, and because CNs/MSCNs at any level j only come from CNs/MSCNs at the lower level j-1, only $O(M^2)$ rank k CN/MSCNs may split in the worst case at any one step forming, per induction hypothesis, each at most U^k new nodes at any size-level j. For all steps this makes $U^k*O(M^3)$ nodes generated via Splits per level. A level $j<=$M accumulates in the worst case also whatever may be generated in the lower level j-1, which is as already seen above $(j\text{-}1)*U^k*O(M^3)$ making the overall count $j*U^k*O(M^3)= U^k*O(M^4)$ per level. For all levels we get then the inequality $U^{k+1}<=U^k*O(M^5)$. The complexity expression follows, as

[72] Base cases k=1, k=2 are enough for this inductive argument and make the results shown here independent of any investigations given in [Abdelwahab 2016-2]. It is, nevertheless, important to show the link to - and thus the continuity of - ideas presented there as well.

[73] Recall that this argument only uses the l.o. condition imposed on all Clause Sets to arrive at the result (c.f. Lemma 9-b).

[74] This induction hypothesis implies P=NP.

seen in all base cases, from the bottle-neck search condition requiring: $O(M)*(U^{k+1})^2$ operations. Since U^k is, per induction hypothesis, a polynomial expression and *k*SAT, for $k>2$, an NP-compete problem, it follows that:

P=NP.

Suppose now that we don't show for F that (*k*+1)SAT-GSPRA$^+$ and (*k*+1)SAT-FGPRA Algorithms satisfy conditions 1,2 & 3. Even then, remembering that *k*SAT is NP-complete for any $k>2$: There is a polynomial time reduction from (*k*+1)SAT to *k*SAT. We could for example convert F to a *k*CNF formula F' via an equisatisfiable transformation and use *k*SAT-FGPRA to solve it. The number of clauses of F' would be bounded above by (*k*+1)*M, M number of clauses of F, because such transformations generate always at most (*k*+1) clauses for any clause C∈F. As per induction hypothesis: *k*SAT-FGPRA's time complexity is given by:

$O(M)*(U^k)^2$, where U^{k-1} is a polynomial expression of degree, say, ***d*>0** in **M** and $U^k<=U^{k-1}*O(M^5)$. Substituting (*k*+1)*M for M in this inequality gives:

$U^k<=(k+1)^{d+1}*U^{k-1}*O(M^5)$ and does not disturb the polynomial behavior of *k*SAT-FGPRA as expected. Since $U^{k+1}=U^k$, this trivially means also that: $U^{k+1}<=U^k*O(M^5)$ which was to be shown. F' can thus be solved by a polynomial time Algorithm producing a polynomial number of unique nodes, i.e.,

P=NP.

No surprise since P=NP was already embedded in the strong induction hypothesis.

b- The same main result follows also directly from the following observations:

1- Using 2SAT-FGPRA to produce a $MSRT_{s.o}$ for Q is, as per (Lemma 11) in this work, in $O(M^9)$ or in $O(M^{13})$ if we relax (Lemma 9-c).

2- Counting the exact number of solutions using Count2SATSolutions is, for the same reason, also either in $O(M^9)$ or in $O(M^{13})$ as per (Lemma 14).

3- This means that any Algorithm solving #2SAT using 2SAT-FGPRA first to construct the $MSRT_{s.o}$ and then Count2SATSolutions needs in the worst case only $O(M^9)$ or $O(M^{13})$ primitive operations. #2SAT is known to be #P-complete (c.f. [Valiant 1979]), therefore:

P=NP

(Q.E.D.)

IV DISCUSSION OF RESULTS

This work shows that small FBDDs for base cases of *k*SAT: *k*=1, *k*=2 are achievable via SPR-like Algorithms which neither possess N- nor Big-Splits. Moreover: The nature of those Algorithms permits a uniform expression of result parameters of 2SAT-GSPRA$^+$/2SAT-FGPRA versions in terms of 1SAT-GSPRA$^+$/1SAT-FGPRA versions for both: The upper bound of the number of unique nodes in generated FBDDs and the worst case time complexity. This is sufficient to prove P=NP in the following two different ways:

a- FBDDs of polynomial sizes for arbitrary 2CNF formulas enable the definition of efficient model counting solutions resulting in solving #2SAT in a polynomial number of steps (Theorem 1-b).

b- Uniformly linking efficient 1SAT- and 2SAT-versions of SPR-Algorithms, while proving small, upper bounds on unique node counts, enables formulating the strongest possible induction hypothesis, namely: That *k*SAT-FGPRA is a polynomial time Algorithm producing polynomial number of unique nodes in a FBDD (which means: P=NP). This in its turn facilitates using *k*SAT-FGPRA to solve (*k*+1)CNF formulas via equisatisfiable translations in the induction step, completing thus a third way of showing that P=NP in (Theorem1-a)[75].

The core work of demonstrating that FBDDs for a 2CNF formula F can always be small strongly relates to the concept of a *Split*, which expresses the fact, that some sub-formulas of F may be repeatedly processed during resolution. Fatal cases of processing sub-formulas of the same difficulty as the original problem from scratch again and again (N- and Big-Splits) are shown to be avoided using imposed l.o. conditions. The rest of existing rank 1- and/or size 1-Splits facilitate a uniform formulation of the relation between *k*- and (*k*-1)SAT-SPR-Algorithms when some lemmas are relaxed.

Splits are not mere accidents which don't have a rational reason. They reflect consequences of tangible pattern-properties of variables found in nature and enforced on Clause Sets to serve, in addition to usual container-properties, in the definition of SPR-like procedures.

Finally: Discussing the consequences of our findings is beyond the scope of this work.

[75] Counting also the solution of 3SAT presented in [Abdelwahab 2016-2].

V REFERENCES

1. [Abdelwahab 2016-1]: N. Abdelwahab, *On the dual Nature of logical Variables and Clause Sets*, J. Acad. (N.Y.) 2016, Vol. 6, 3:202-239.
2. [Abdelwahab 2016-2]: N. Abdelwahab, *Constructive Patterns of Logical Truth*, J. Acad. (N.Y.) 2016, Vol. 6, 2:99.
3. [Gal 1997]: Anna Gal, *A simple function read-once that requires exponential size branching programs*, Information Processing Letters 62 (1997), 13-16.
4. [Beame 2013]: Beame, P., Li, J., Roy, S. and Suciu, D. 2013. *Lower bounds for exact model counting and applications in probabilistic databases.* In Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11 - 15 2013.
5. [Bolling 1996]: Bollig, B.; Wegener, *I*., *Improving the Variable Ordering of OBDDs is NP-Complete*, IEEE Transactions on Computers, Vol. 45, 1996.
6. [Bryant 1986]: Randal Bryant, *Graph-Based Algorithms for Boolean Function Manipulation*, IEEE Transactions on Computers, C-35-8, 677-691, August, 1986
7. [Burch 1991]: Jerry R. Burch, *Using BDDs to verify multipliers*, DAC'91 Proceedings of the 28th ACM/IEEE Design Automation Conference, 408-412.
8. [Darwiche 2002]: Adnan Darwiche, Pierre Marquis, *A knowledge compilation map*, Journal of Artificial Intelligence Research 17 (2002) 229-264, AI Access Foundation and Morgan Kaufmann Publishers.
9. [Dasgupta 2006]: Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Vazirani. 2006. Algorithms (1 ed.). McGraw-Hill, Inc., New York, NY, USA.
10. [DeItaLuna 2012]: De Ita, Guillermo & Marcial-Romero, J. (2012). Computing #2SAT and #2UNSAT by binary patterns. 273-282. 10.1007/978-3-642-31149-9_28.
11. [Fuerer 2007]: Fürer M., Kasiviswanathan S.P., *Algorithms for Counting 2-Sat Solutions and Colorings with Applications.* In: Kao MY., Li XY. (eds.) Algorithmic Aspects in Information and Management. AAIM 2007. Lecture Notes in Computer Science, Vol. 4508. Springer, Berlin, Heidelberg.
12. [Handbook of Satisfiability 2009]: Armin Biere et al., *Handbook of Satisfiability*, Publisher IOS Press, Nieuwe Hemweg 6B, 1013 BG Amsterdam Netherlands.
13. [Rudell 1993]: R. Rudell, Dynamic Variable ordering for ordered binary decision diagrams, ICCAD-93. Digest of Technical Papers., 1993 IEEE/ACM International Conference on Computer-Aided Design, 1993.
14. [Sauerhoff 2003]: Sauerhoff and P. Woelfel, *Time-space tradeoff lower bounds for integer multiplication and graphs of arithmetic functions*. In Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), pp. 186–195.
15. [Sieling 1995]: D. Sieling and I. Wegener, *Graph driven BDDs - a new data structure for Boolean functions*, Theoretical Computer Science, 141, 1995,283-310.

16. [Schrijver 2003]: Combinatorial Optimization: *Polyhedra and Efficiency*, Volume 1, Algorithms and Combinatorics, 24, Springer, p. 114.
17. [Valiant 1979]: Valiant, L. G. *The complexity of enumeration and reliability problems.* SIAM Journal of Computing 8, 3 (1979), 410–421.
18. [Wegener 1988]: I. Wegener, *On the complexity of branching programs and decision trees for clique functions*, Journal of the ACM, 35, 1988,461-471.
19. [Wegener 2000]: Ingo Wegener, *Branching Programs and Binary Decision Diagrams*, Theory and Applications, SIAM 2000.
20. [Zak 1984]: S. Zak, *An exponential lower bound for one-time-only branching programs*, MFCS'84, LNCS 176, 1984, 562-566.

VI APPENDICES

VI-A Formal terms, their definitions and usage

Term/(Acronym,Link)	Definition	Formally	Used in	Comment
Variable, Literal, Clause, 2CNF Formula/Clause Set	0.1	Standard	Basic	---
Truth Assignment, Partial Assignment, Restricted Assignment	0.1	f:Var =>{0,1}. When f is partial it is called Partial Assignment, when it is restricted to only one variable it is called Restricted Assignment	Basic, (Lemma 2)	---
2SAT Decision Problem	0.2	Standard	Basic	---
Graphs, Vertices/Nodes, Edges, adjacent vertex, Source, Target , reachable, Child, Parent, Base Node (BN), Path, Branch, acyclic, Length of Path/Branch, Directed Acyclic Graph (DAG)	0.3	Standard	Basic	---
Source Path of node n (SP_n)	0.3	SP_n:List<Edges>	Counting Models	Used for determining node levels in the (**Count2SATSolutions**) procedure (Section III-6)
Level of node n (L_n) in a DAG	0.3	L_n=Max(length(SP_n^1)..length(SP_n^k)) where any SP_n^i is a Source Path of n.	Counting Models	Used in the (**Count2SATSolutions**) procedure (Section III-6)
Level of edge e (L_e) in a DAG	0.3	Le=L_{SR}+1, where SR is Source of e	Counting Models	Used in the (**Count2SATSolutions**) procedure (Section III-6)
Topological Ordering of a DAG (TO)	0.3	$\forall e$:Edge, e=(v_i , v_j), v_i,v_j∈V: i< j	Counting Models	Used in the (**Count2SATSolutions**) procedure (Section III-6)
-Sequential Resolution DAG (SR-DAG) - 2CNF Clause Set of a node ($2CNF_{node}$), - Base Clause Set (BS), - (TRUE-DAG) - (FALSE-DAG)	0.3	-SR-DAG: $\forall n$:Node∈d:DAG: ∃S, S is 2CNF Clause Set, S is the Clause Set of n ($2CNF_n$). BS is $2CNF_{BN}$. - TRUE-DAG: SR-DAG with one node only labeled TRUE. FALSE-DAG: similar.	Basic	---
-($rank_C$) -($rank_{Node}$) -($rank_{2CNF}$)	0.3	- $rank_C$: (clause) => N - $rank_S$=$rank_{Node}$= Max($rank_C(C_1)$..$rank_C(C_m)$), C_1-C_m∈S, S is $2CNF_{Node}$	Basic	- $rank_C$: Number of literals in clause C
-Size of a node n ($Size_n$), -Size of a 2CNF Clause Set S ($Size_S$)	0.3	Standard	Basic	- $Size_n$: Number of clauses in the $2CNF_n$ - $Size_S$: Number of clauses in a 2CNF Clause Set S
-(Top-Part) of a SR-DAG	0.3	$Top_{d:SR\text{-}DAG}$={n:Node∈d | ∃S, S is $2CNF_n$, $Size_S$=M or $Size_S$=M-1, $Size_{BN∈d}$=M}	Basic	---

-(LeftDAG) - (RightDAG) - (SubTree)	0.3	LeftDAG: (*n*:Node)=>SR-DAG rightDAG: (*n*:Node)=>SR-DAG SubTree: (*n*:Node)=>SR-DAG	Basic	- Functions returning SR-DAGs of left- and right Child nodes of a node *n* - SubTree: Is a Function which, given a node *n* of a SR-DAG, returns the portion of the SR-DAG starting with *n*.
Literals in a 2CNF Clause Set S (LIT)	0.4	LIT: (S) => Var	Basic, (Lemma 1) (Lemma 2)	Function returning all literals in S
Left literals of Literal *x* (LEFT)	0.4	LEFT:(*x*:Literal∈C,C:Clause∈S) => Var	Basic	- LEFT: Function returning literals occurring to the left of a Literal *x* in the string representation of S
(SortOrder)	0.4	SortOrder:(C:Clause∈S, S:2CNF Clause Set)=>int	Basic, (Lemma 1)	- Function mapping clause C∈ S and 2CNF Clause Set S to an integer number representing the position of C within S
-Head-Literal, Tail-Literal (HL,TL) -Connectivity of a Literal *x* in a 2CNF Clause Set S ($Connect_{x,S}$)	0.4	HL={L:Literal | C∈ S, S is 2CNF Clause Set, C={L, *t*}} TL={L:Literl | C∈ S, S is 2CNF Clause Set, C={*t*, L}} Connectivity:(*x*:Literal∈ S,S)=>int	Basic	-First Literal in any clause is called Head-, last one is called Tail-Literal - Connectivity: Is a Function mapping a Literal *x* in a Clause Set S to the number of clauses of S in which the Literal *x* appears. It is used in CRA
-Permutations of C∈S, S is 2CNF Clause Set ($perm_C$). -Resolution Complexity Coefficient (RCC) - Alignment 2CNF Clause Set of S (ACS).	0.4, 13	- $perm_C$={C∈ S | C={*a, b*} or C={*b, a*} or C={*a*} or C={*b*}, *a*, *b*:Literal∈C} -$RCC_{k\text{-}SAT}={}^kP_k+{}^kP_{k-1}+{}^kP_{k-2}+\ldots{}^kP_1$ i.e., for 2SAT $RCC_{2\text{-}SAT}={}^2P_2+{}^2P_1=4$ - ACS=$\cup perm_{Ci\in S}$ for all C*i*∈S	(Lemma 7) (Lemma 10)	- $perm_C$ is the Set of all clauses which use permutations of Literals in C∈ S - ACS is the Set of all unique clauses and their derivations used for the alignment of all nodes of a $MSRT_{s.o}$
-Instantiations of Literals, - (Derivation) of C∈ S and S is 2CNF Clause Set, -(Linear Derivation) of C∈ S, -(Empty Derivation) of C∈ S, -(Positive Derivation) of C∈ S, -(Negative Derivation) of C∈ S, -(Every Derivation) of C∈ S, -(InstSimple) - $InstSimple_C$, - Satisfiability of S	0.4	-Inst:(A:Assignment,S) => 2CNF Clause Set - InstSimple=Inst(A:RestrictedAssignment,S) => 2CNF Clause Set. -$InstSimple_C$: (A:RestrictedAssignment,C:Clause) => 2CNF Clause Set a- $InstSimple_C$:(A:Assignment,C:Clause) => Clause b- Derivation of a clause C is∈{C':Clause | C'∈ $perm_C$}. c- Linear Derivation of C is ∈ {C':Clause| C'={*a,b*} or C'={*b*} , *a, b*:Literal∈C, *a*<*b*} d- Empty Derivation of C is ∈ {C':Clause| C'={TRUE} or	Basic, (Lemma 2)	- Instantiations are functions using Total or Partial Truth Assignments to create new Clause Sets. They substitute literals in Clause Sets by Boolean truth values given in the Assignment. - The clause resulting from applying an instantiation on any C∈ S is called a derivation of C. - It is called linear Derivation if consecutive instantiations respect the linear order of literals in C.

		{FALSE} or {TRUE,FALSE} or {FALSE,TRUE} or {FALSE,FALSE} or {TRUE,TRUE}} e- Positive Derivation of C is ∈ {C':Clause| TRUE ∈C'} f-Negative Derivation of C is ∈ {C':Clause| C'={FALSE,FALSE} or C'={FALSE}} g- Every Derivation of C is ∈{C':Clause| C'∈ perm$_C$ or C'∈ Empty Derivation of C}		- If consecutive instantiations result in a clause containing only truth values and no literals, the derivation is called: Empty Derivation - A Derivation containing one TRUE value is called Positive Derivation. - A Derivation containing only FALSE values is called Negative Derivation. - Derivations can be directly evaluated to TRUE or FALSE. Evaluation is embedded in the Inst function. If this evaluation results in the TRUE, S is said to be *satisfiable* by A. - When Partial Assignments used by Inst are related to only one variable, Inst is called InstSimple. InstSimple can be restricted to only one clause and becomes InstSimple$_C$ - S is said to be satisfiable by A: If Inst(A,S) results in the overall value TRUE C.f.: (Lemma 2)
- (Convert) a clause to SR-DAG, -(FIRST) occurrence of a Literal in a 2CNF Clause Set S, -(SELECT) a Literal from a 2CNF Clause Set S	0.4	- Convert(C:Clause∈S)=>SR-DAG - FIRST/FIRST$_C$(L:Literal, S)=>int - SELECT(S)=>int	Basic, (Lemma 1)	- Convert is a function mapping a 2CNF Clause C={a_1,b_{11}} to a SR-DAG by substituting in two subsequent simple instantiation steps first a_1 with TRUE and FALSE creating Clause Sets and placing them in the respective nodes of the SR-DAG and then doing the same for $b11$ (Figure 2). - FIRST: is a function mapping a Literal and a Clause Set S to the integer position (starting from the left) of the Literal in the string representation of S. FIRST$_C$ is the version of this function which returns the index of the clause in which L appears for the first time, c.f.: (Lemma 1-c)

				- SELECT: Is a Function selecting a Literal from LIT(S). Although generic, it is only used in Algorithms of this work to select the least Literal according to LLR
-Linearly Ordered- (l.o.) , -Linearly Ordered, but unsorted (l.o.u.), -Almost Arbitrary (a.a.) Clause Sets	1	For a 2CNF formula S, S is called l.o. if the following Conditions hold: a) $\forall a_i,b_{ij} \in C_{i,j}$: $a_i < b_{ij}$ b) **$\forall i,j,x,y$: if $i<j$ then $L_2 \in C_{j,x} >= L_1 \in C_{i,y}$, where L_2 is HL of $C_{j,x}$ and L_1 HL of $C_{i,y}$, *SortOrder*($C_{j,x}$,S)> *SortOrder*($C_{i,y}$,S)** c) $\forall x \in$ LIT(S),$\forall C_{i,j} \in$S: if $x \notin$ LEFT(x,$C_{i,j}$) then $\forall y \in$ LEFT(x,$C_{i,j}$): $x>y$ d) S is a Set If S fulfills Conditions a), c), d), but not b) it is l.o.u. If S fulfills Conditions a), d) only it is a.a.	All Lemmas	a) Literal names/indices are sorted in ascending order within clauses. b) S is sorted by a_i & b_{ij} in ascending order taking into consideration negation signs. c) all new Names/Indices of literals occurring for the first time in a clause C of S are strictly larger than all the Literal Names/Indices occurring before them in S d) Clauses appear only once in S.
- Blocks (B_a), - Block-Literal, - Block-Sequence (B_{seq}), - Symmetric Block (SB), - Dissymmetric Block (DB), -(DB Sorting Condition)	1	- B_{ax}={{a_x,b_{x1}}{a_x,b_{x2}} .. {a_x,$b_{xi,}$}} is a 2CNF Clause Set. - a_x is Block-Literal - S={B_a…B_n} is Block-Sequence - A Block B_x is called SB if ∃A: Assignment such that: instSimple(A:{X=TRUE},B_x}= instSimple(A:{X=FALSE}, B_x} -It is called DB if ∃A:Assignment such that: instSimple(A:{X=TRUE},B_x}=S_1, instSimple(A:{X =FALSE}, B_x}=S_2 and either $S_1 \subseteq S_2$ or $S_2 \subseteq S_1$.	Basic, (Lemma 8) (Lemma 9)	- Blocks are referred to by the name of the leading Literal (in this case S is called a_x-Block). - Clauses having a_x as leading Literal are said to belong to the a_x-Block. - A Block B_x is called SB if *-ve* and/or *+ve* instantiations of Block Literal x result in the same Clause Set. - A Block B_x is called DB if *-ve* and/or *+ve* instantiations of Block Literal x result in Sets S_1, S_2 and one of them is included in the other. - DB Sorting Condition: If a DB B_x is sorted such that all clauses containing *–ve* instances of Literal x are placed before all those containing *+ve* instances or vice versa
-(2SAT-GSPRA Procedure), - (Align Procedure),	2	-2SAT-GSPRA Procedure (c.f. Section III.1)	(Lemma 5)	- A node in a SR-DAG is symbolized by [x] if the

-Name Literal (NL), -(Edge Literal) - (Branch Literal) - Least Literal Rule of a 2CNF Clause Set S (LLR_S), -Variable Ordering ($\prod_p$), -CanonicalOrdering ($\prod^c_p$)		-Align Procedure (c.f. Section III.1) - $NL=LLR_S=\{i$:Literal $\mid \exists BS$: 2CNF Clause Set, $\exists n$:Node$\in$SR-DAG_{BS}, S is $2CNF_n$, SELECT(S)=i and $\forall x \in LIT(S)$: $i<x\}$ -$\prod_p=<i,j,k,\ldots>$ where $i,j,k,\ldots$ integers such that $i<j<k<\ldots$.		lead clause in its Clause Set is headed by a least-Literal x. Moreover: x is called the NL of this Clause Set/node. - Edges going out of a SR-DAG node $[x]$ are marked with x and represent instantiations of the NL x of the Clause Set of that node (this fact is called LLR). -Literals on edges of branches leading indirectly to a node n are called branch-literals of n while literals on edges connected directly to n are called edge-literals of n. Every edge-Literal is a branch-Literal, but not vice versa. - A variable ordering of a problem p ($\prod_p$) expressed as a 2CNF Clause Set S and resolved by any resolution procedure PR is a list of integers $<i,j,k,\ldots>$ representing indices of Literal/variable names indicating priorities of instantiations of literals/variables of S used in PR. If $\prod_p$ represents the canonical, truth table ordering of variables the following notation is used: $\prod^c_p$.
-(Sequentially ordered SR-DAG) - Strongly ordered SR-DAG (s.o.) - Loosely ordered SR-DAG (lo.o.)	3	- Sequentially Ordered SR-DAG: $\forall S$, $n \in$SR-DAG, S is $2CNF_n$: $S=\{C_i,C_j,\ldots C_M\}$ for some $i<j<\ldots.<M'$, $M'<=M$. M number of clauses in S, C_x's are clauses or derivations of clauses enumerated from left to right in S - Strongly Ordered SR-DAG: $\forall S$, $n \in$SR-DAG, S is $2CNF_n$: S is linearly ordered (l.o.) - Loosely Ordered SR-DAG: $\forall S$, $n \in$SR-DAG, S is $2CNF_n$: S is either l.o. or l.o.u.	All Lemmas	- Strongly ordered Sets are always linearly ordered, the inverse is not always the case, i.e., some l.o. Sets may have Clause Sets in their SR-DAGs which are not l.o. - If a Set S has a BS which is l.o. while some other Clause Sets in its generated SR-DAG are l.o.u., then S as well as its SR-DAG is called loosely ordered
- Common Node (CN), - Head-CN (HCN), - Tail-CN (TCN),	4	-$[q]\in$SR-DAG is CN if $\exists n_1,n_2 \in$SR-DAG such that: $[q]$ adjacent to both n_1 and n_2	(Lemma 8) (Lemma 9)	A CN $[q]$ is supported in a step $l>k$ if its Clause Set S gets clauses appended to

- Trivial-CN (tCN), - (Supported CN) - Supporting Parent, -(Direct Parent), -(Direct Child), -Double-Sided CN from the perspective of x ($DSCN_x$), -Single-Sided CN from the perspective of x ($SSCN_x$), -(Distinguished Literal), -(Non-Distinguished Literal), -CN-Augmenting Literal (CNAL)		- A CN $[q]$ ∈SR-DAG is HCN if its Clause Set has a leading/head clause C∈S, NL q is HL of C - A CN $[q]$ ∈SR-DAG is TCN if its Clause Set has a leading/head clause C' which is a derivation of a clause C∈ S, NL q is TL of C - $[q]$ ∈SR-DAG is tCN if $\exists n \in$ SR-DAG, S is $2CNF_n$, S is SB, Child($[q]$,n)=TRUE -A CN $[q]$ ∈SR-DAG with S=$2CNF_{[q]}$, S=B_{seq} produced in steps <=k, is said to be supported in a step l>k if ∃C:Clause, C∈B_x such that: S=S ∪ C in step l>k while in steps <=k: ∃n ∈SR-DAG, Parent(n,$[q]$), S' is $2CNF_n$, S' is B_{seq} and $B_x \notin$ S' - CN $[q]$ ∈$SR\text{-}DAG_{BS}$ is called $DSCN_x$ if ∃n_1,n_2:Node∈$SR\text{-}DAG_{BS}$, x,y:Literal, S_1 $2CNF_{n1}$, S_2 $2CNF_{n2}$ such that: $LLR_{S1}=x$, $LLR_{S2}=y$, $x=\neg y$, **Parent(n_1,$[q]$)=TRUE,** **Parent(n_2,$[q]$)=TRUE.** - CN $[q]$ ∈$SR\text{-}DAG_{BS}$ is called $SSCN_x$ if ∃n_1,n_2:Node∈$SR\text{-}DAG_{BS}$, x,y:Literal, S_1 $2CNF_{n1}$, S2 $2CNF_{n2}$ such that: $LLR_{S1}=LLR_{S2}=x$, **Parent(n_1,$[q]$)=TRUE,** **Parent(n_2,$[q]$)=TRUE.** - CNAL={L:Literal∈C:Clause, $[q]$ is CN∈$SR\text{-}DAG_{BS}$ formed in steps<=k, L is non-distinguished for $[q]$ | $Size_{[q]}$ is augmented in steps>k through invocations: $InstSimple_C$ ({L=TRUE},C) or $InstSimple_C$ ({L=FALSE},C)}		its head in step l which don't belong to any Block instantiated in steps <=k by one or more of its parents. A parent-set of such a CN is called supporting. If a head-clause of a CN is also a clause of one of the Clause Sets of its parents, then this parent is called direct parent of the CN. The CN itself is called direct child of this parent A CN $[q]$ formed within a Block B_x through +ve as well as -ve edge- or branch-literals x is $DSCN_x$. Such a x is called in this case distinguished Literal for $[q]$. A CN $[q]$ formed within a Block B_x through only +ve or only -ve edge- or branch-literals x is $SSCN_x$. x is called in this case non-distinguished Literal for $[q]$. If for a CN $[q]$ there is no distinguished Literal x such that the CN is $DSCN_x$, then $[q]$ is SSCN. If a non-distinguished Literal x for a CN $[q]$ formed in steps <=k is used to augment the size of $[q]$ in step l>k, i.e., x is instantiated in a clause added to the clauses of $[q]$ in l, then x is CNAL for $[q]$.
- Dependency Graph (DG), - Leaves of Dependency Graphs, - Free Binary Decision Diagram (FBDD)	5	- DG is a DAG <V,E> where V is the Set of all NLs, E the Set of ordered pairs <v_1,v_2> , v_1,v_2∈ V	All Lemmas	- DGs can be deduced from SR-DAGs in a canonical way and used as practical alternatives for truth tables. They are equivalent to FBDDs. - DGs (FBDDs) have the following properties a- Each NL can appear only once in a branch. b- Branches can have different Literal/variable orderings $\prod_p$ depending on the sub-problem p they belong to c- A leaf of a DG is a node whose value is TRUE or FALSE.

- (Splits), - (N-Splits) - (CN-Splits) - (Split Node) - Big-Splits (BigSps)	6	- Split: A SR-DAG is said to possess a Split if ∃S':2CNF Clause Set such that: For some n_1,n_2:Node∈SR-DAG$_{BS}$, S_1 is 2CNF$_{n1}$, S_2 is 2CNF$_{n2}$, $n_1 \neq n_2$: S'⊆ S_1, S'⊆ S_2, ∄n: Child(n,n_1)=Child(n,n_2)=TRUE - Splits are called CN-Splits, if, in addition to the formal condition above: ∃q:Node, ∃C:Clause: S' is 2CNF$_{[q]}$, [q] is CN/MSCN in step k and C is resolved in steps >k such that: $C_1 \subseteq S_1$, $C_2 \subseteq S_2$, $C_1,C_2 \notin$S', $C_1,C_2 \in$Every Derivation of C, $C_1 \neq C_2$. - If a Split is not a CN-Split, it is called a N-Split. - BigSps: Are Splits of a CN [q] where rank$_{[q]}$=rank$_{BN}$	(Lemma 9)	Split: There exists a sub-Set of clauses common between two or more Clause Sets of different nodes which don't possess common sub-trees. Splits are formed when **either** node n containing Clause Set S constructed in step k is duplicated one or more times in steps >k together with all or parts of its nodes or sub-trees, the cause of this duplication being that S is resolved with a clause whose least-Literal was new in that step and had an index < all or any indices of head-literals in S (N-Split) **or** a CN [q] constructed in step k and/or any of its nodes or sub-trees are duplicated with variations one or more times in steps >k (CN-Split). If [q] is a CN of a SR-DAG which is split in step k, then the new node [q]'=[q]+C' formed in k, because C∈BS is resolved (C' is a Derivation of C) is called: Split-Node. BigSps occur when a CN is split which has the same rank as the rank of the base node. They are causes of exponential behavior of 2SAT-GSPRA.
- Clauses Renaming Algorithm (CRA), -(Connection Matrix), - Renaming Precedence Condition (RPC)	7	- CRA c.f. Definition 7	(Lemma 1) (Lemma 3) (Lemma 6) (Lemma 8)	Connection Matrix: Rows are Literal Names/Indices, Columns are clauses, Entries are TRUE/FALSE according to whether the Literal occurs in the given clause or not RPC: Arrange literals in ascending order within any $C_i \in S$ such that literals which were not renamed before and appear more often in other clauses become HLs before those which appear

				less often or which only appear in C_i.
- (Mapping), - (Image), - Variable Space/Space (VS), - 2CNF Clause Set in space-i ($S_{space-i}$), - Node in space-i ($Node_{space-i}$)	8.1	- Mapping: (N) => N - VS=Mapping*(N)	(Lemma 1) (Lemma 9)	- Mapping is a bijective function giving a Literal Name/Index in a 2CNF Clause Set S its new Name/Index after a renaming operation using CRA. - The new Name/Index is also called: Image of the Literal. New names of literals forming single clauses or Clause Sets are called Images of clauses or Clause Sets. – A VS is a subsequent application of mappings starting from the Base Clause Set of a 2CNF formula. - To express that a Clause Set is formed in a certain space-i the notation: $S=\{\{..\}\dots\{..\}\}_{space-i}$ or just $S_{space-i}$ is used. - To express that a node is formed in a certain space-i the notation: $Node_{space-i}$ is used.
- (Apply) - (InvApply)	8.2	- Apply: (M:Mapping, S:2CNF Clause Set) => 2CNF Clause Set	All Lemmas	- Apply is a function which replaces occurrences of literals in a 2CNF Clause Set S with their Names/Indices given by the mapping M. - InvApply is similarly defined, but applies to S: M^{-1} instead of M.
- Equivalence via Mapping ($S_1 \Leftrightarrow_M S_2$), - (Syntactic Image)	8.3	- $S_1 \Leftrightarrow_M S_2$: if $\exists M_1, M_2$:Mapping such that: $Apply(M_1,S_1)=Apply(M_2,S_2)=S'$. S' is called syntactic image of both S_1, S_2.	(Lemma 2)	- Equivalent via Mapping: Are 2CNF Clause Sets which reside in MSCNs, i.e., CNs which are formed between different Variable Spaces
- trivial Mapping (tMapping), - (Stable Set of literals), - (Stable Clause) - Stable Clause Set	8.4	- tMapping: $\exists$M:Mapping, S a 2CNF Clause Set, $\forall x \in LIT(S)$: $M(x)=x$ - Sub is a Stable Set of literals: If $\exists$M:Mapping produced in step k such that: $\forall x \in Sub$, $Sub \subseteq Lit(S)$: $M(x)=x$ in any step $>k$ - Stable Clause: $\forall x$:*Literal* $\in C_i$, $x \in Sub \subseteq Lit(S)$, Sub is a Stable Set of literals	(Lemma 2) (Lemma 3)	- tMapping: Each Literal index is given itself after a renaming operation using CRA. -Stable Set of literals: a subset of Literal indices is mapped to itself via CRA in step k and remains always mapped to itself for any step$>k$,

		- Stable Clause Set: $\forall C_i \in$ S:Clause Set, C_i is stable, then: S is a Stable Clause Set.		
- Mixed-Space Node (MSN), - Single-Space Nodes (SSN)	8.5	- MSN: S_1, S_2 are 2CNF Clause Sets of nodes $n_1, n_2 \in$ SR-DAG, respectively, and $S_1 \neq S_2$, but $n_1 = n_2 = n$.	(Lemma 9)	- MSNs possess two syntactically non-equivalent Clause Sets, because of the application of CRA^+ - SSNs are nodes in which CRA^+ was not applied
- Mixed-Space Tree (MST), - Single-Space Tree (SST) - Literal in space-i ($L_{space\text{-}i}$) - Literal x proceeds Literal y in a Clause Set S of space-i ($(x \mid y)_{space\text{-}i}$) - Mapping in space-i ($M_{space\text{-}i}$)	8.6	- $(x \mid y)_{space\text{-}i}$: If $\exists$space-i:VS such that: $S_{space\text{-}i}$ is a 2CNF Clause Set where: $FIRST(x, S_{space\text{-}i}) < FIRST(y, S_{space\text{-}i})$	(Lemma 1) (Lemma 9)	- MST: SR-DAG with MSN nodes - SST: SR-DAG with only SSNs. – $L_{space\text{-}i}$ refers to the name of Literal L given by a mapping M in space-i. - x proceeds y in space-i: Within space-i the first occurrence of Literal x in a Clause Set S comes before the first occurrence of Literal y. When space-i is known, its subscript is omitted. Since S is always apparent from the context a reference to it is omitted as well. - $M_{space\text{-}i}$: Refers to the mapping created by a CRA operation within space-i.
- Monotone Mapping in space-i ($mM_{space\text{-}i}$)	8.7	- A mapping is monotone when $\forall x,y \in LIT(S_{space\text{-}i})$: if $(x \mid y)_{space\text{-}i}$ then also $M_{space\text{-}i}(x) < M_{space\text{-}i}(y)$	(Lemma 1) (Lemma 9)	-This property is intrinsic in all GSPRA Algorithms
- Clauses Renaming and Ordering Algorithm (CRA^+), - (CRA-Form) - Sequentially-Ordered, Multi-Space Resolution Tree/SR-DAG ($MSRT_{s.o.}$), - Multiple Space Block (MSB)	9, 10	- CRA^+: Pseudo-Code Definition 9, $CRA^+(S)$ is called the CRA-Form of S. - $MSRT_{s.o.}$: Is a SR-DAG such that: $\forall n_{space\text{-}i}$:Node$\in$SR-DAG: $(2CNF_n)_{space\text{-}i}$ is l.o. - MSB = {$(B_{x1})_{space\text{-}i}$:2CNF Clause Set | $\exists$space-j, $(B_{x2})_{space\text{-}j}$:2CNF Clause Set, M: Mapping, where: $((B_{x1})_{space\text{-}i} \Leftrightarrow_M (B_{x2})_{space\text{-}j})$ Or $((B'_{x1})_{space\text{-}i} \Leftrightarrow_M (B'_{x2})_{space\text{-}j})$, B'_{x1}, B'_{x2} are Derivations of B_{x1}, B_{x2}, in respective Spaces }	(Lemma 2) (Lemma 3) (Lemma 8) (Lemma 9) (Lemma 10)	-$MSRT_{s.o}$ is a MST whose Clause Sets are all l.o. - MSB: A block B_x whose Clause Set or derivations thereof (all or part of them) belong to more than one VS (Notation also: $B_x^{S1,S2,..}$,$S_1,S_2,..$ Variable Spaces). - Similar to Single Space Blocks: An MSB may be symmetric or dissymmetric.
-Multi-spaced Symmetric Block (MSSB)	10.1	- MSSB = { $(B_{x1})_{space\text{-}i}$:2CNF Clause Set | $\exists$space-j, $(B_{x2})_{space\text{-}j}$:2CNF Clause Set, M: Mapping, where	(Lemma 8) (Lemma 9)	- MSSB is the structure in which a tMSCN can occur

		$((B_{x1})_{space\text{-}i} \Leftrightarrow_M (B_{x2})_{space\text{-}j}$ Or $(B'_{x1})_{space\text{-}i} \Leftrightarrow_M (B'_{x2})_{space\text{-}j})$ B'_{x1}, B'_{x2} are Derivations of B_{x1}, B_{x2}, in respective Spaces and $\exists A_{space\text{-}i}, A_{space\text{-}j}$: Assignment such that: $instSimple(A_{space\text{-}i}:\{X_1=TRUE\}, (B_{x1})_{space\text{-}i}) \Leftrightarrow_M instSimple(A_{space\text{-}j}:\{X_2=FALSE\}, (B_{x2})_{space\text{-}j})$ }		
- Multiple Space Common Node (MSCN) - Target Space (TS)	10.2	- MSCN:- if $\exists n_1,n_2 \in MSRT_{s.o}$ not necessarily of the same space: $[q]$ adjacent to both n_1 and n_2	(Lemma 8) (Lemma 9) (Lemma 10)	- Target Space: The VS of a node which is target of two or more MSNs in a $MSRT_{s.o.}$
- Double-Sided MSCN with respect to Literal z ($DS\text{-}MSCN_z$), - Single-Sided MSCN with respect to Literal z ($SS\text{-}MSCN_z$), - trivial MSCN (tMSCN)	11	- $[q]_{space\text{-}i}$ is $DS\text{-}MSCN_z$, if $\exists n_1,n_2 \in MSRT_{s.o}$ of 2CNF Clause Set S, $\exists x_{space\text{-}j}$, $y_{space\text{-}k}$:Literal, $\exists M_1,M_2$: Mapping, such that: $[q]_{space\text{-}i}$ is adjacent to both n_1 and n_2 and $z_{space\text{-}i}=M_1(x_{space\text{-}j})$, $z_{space\text{-}i}=M_2(y_{space\text{-}k})$, where $y_{space\text{-}k}$ has the opposite sign of $x_{space\text{-}j}$ - $[q]_{space\text{-}i}$ is $SS\text{-}MSCN_z$, if $\exists n_1,n_2 \in MSRT_{s.o}$ of 2CNF Clause Set S, $\exists x_{space\text{-}j}$, $y_{space\text{-}k}$, $\exists M_1,M_2$:Mapping, such that: $[q]_{space\text{-}i}$ is adjacent to both n_1 and n_2 and $z_{space\text{-}i}=M_1(x_{space\text{-}j})$, $z_{space\text{-}i}=M_2(y_{space\text{-}k})$, where $y_{space\text{-}k}$ has the same sign as $x_{space\text{-}j}$, - $[q]$ is tMSCN, if $\exists n \in MSRT_{s.o}$ whose Clause Set is a MSSB, $Child[q],n)$=TRUE	(Lemma 8) (Lemma 9)	- $DS\text{-}MSCN_z$: There exist at least two edge- or branch-literals x, y from Spaces space-j, space-k respectively and a Literal z from the target space-i such that both literals are translated to z within their respective spaces and have opposite signs. Literals x and y are also called distinguished (c.f. Definition 4, (Distinguished Literal)). - $SS\text{-}MSCN_z$: Similar definition, but x, y have same signs - tMSCN: $[q]$ is formed in step k and belongs to a MSSB to which one or more of its parents belonged in steps $<k$ - $DS\text{-}MSCN_z$ as well as $SS\text{-}MSCN_z$ are used to show that a MSCN cannot be first augmented to sizes >1 and then split except in the trivial case of a tMSCN (Lemma 9-c) - tMSCNs are called trivial, because they can result in Splits which happen only inside symmetric Blocks and thus can be avoided altogether when an appropriate sorting condition within CRA^+ is

				chosen (called: DB-Sorting, (Lemma 8)
(Aligned $MSRT_{s.o,}$), (Alignment Clause , (Aligned Node) (Alignment $MSRT_{s.o}$)	12,13	-Aligned $MSRT_{s.o}$:- $\exists C \in S$, C' derivation of C such that:$\forall n \in MSRT_{s.o.}$, S' is $2CNF_n$, $\forall C_x \in$ S' the following is true: a- SortOrder(C',S')>SortOrder(C_x, S') b- S' is l.o. - A node *n* of size M is said to be aligned if: - For M<=2: *n* possesses a Clause Set with an aligned $MSRT_{s.o}$ - For M>2: (iii) All nodes of sub-trees of size M are l.o. (iv) All nodes of sub-trees of size <M are aligned - An $MSRT_{s.o}$ whose nodes are all aligned is called Alignment $MSRT_{s.o}$	(Lemma 6) (Lemma 7) (Lemma 10)	- C is called Alignment Clause - The fact that a $MSRT_{s.o}$ produced by 2SAT-GSPRA$^+$ is always an Alignment $MSRT_{s.o}$ is used to show that the number of new nodes on size-level 1 in any inductive step cannot become more than the number of elements in ACS which are linearly many (Lemma 7)
Resolution procedures: (2SAT-GSPRA$^+$), (Align) (LCS)	14	c.f. Section III.1	(Lemma 6) (Lemma 7) (Lemma 8) (Lemma 9) (Lemma 10)	-Used to study the effect of resolving one single clause at a time and count the number of unique nodes produced in the final $MSRT_{s.o}$ - LCS: List of Tuples: <Clause Set, Node index> initially empty used to store already resolved Clause Sets and their generated sub-trees
2SAT Fast Generic Pattern Resolution Algorithm (2SAT-FGPRA)	15	c.f. Section III.1	(Lemma 11)	-This is the central, practical Algorithm proposed in this work (and a similar one is proposed in [Abdelwahab 2016-2] as well). It overcomes the main drawback of 2SAT-GSPRA$^+$ of having to re-construct sub-trees again and again in case their respective Clause Sets are not l.o. Instantiation is performed always in any node on the whole $2CNF_{node}$ rather than step wise one clause at a time.

				- 2SAT-FGPRA is shown to correctly simulate 2SAT-GSPRA^{+} (Lemma 11-a)

VI-B Selected Lemmas and their Dependencies on Formalized Concepts

Lemma 1: - CRA produces monotone Mappings - $(x \mid y)$ iff $(x<y)$ - $(x_{space\text{-}i} \mid y_{space\text{-}i})$ iff $(x \mid y)$ when involved Clause Sets are l.o. and order of clauses and images of clauses in respective spaces is preserved	l.o. Condition
	monotone Mapping
	CRA
	VSpace
	LIT
	$(x \mid y)$
	$FIRST_C$
	SortOrder
Lemma 2: For a 2CNF Clause Set S it is true that: - S is l.o. iff $CRA^+(S)$ reaches a stable Set equivalent to LIT(S) - S is satisfiable iff $CRA^+(S)$ is satisfiable - S is logically equivalent to $CRA^+(S)$	l.o. Condition
	Stable Set
	CRA^+
	Satisfiable
	Assignment
	LIT
Lemma 3: The complexity of CRA^+ is in $O(M^2(Log\ M+N)$	l.o. Condition
	l.o.u. Condition
	CRA^+
	CRA
	RPC
Lemma 4: CRA^+ terminates always converting an arbitrary 2CNF Clause Set to a stable one	l.o. Condition
	l.o.u. Condition
	CRA^+
	CRA
	Stable Set

Lemma 5: a- For all n_1,n_2 nodes ∈ $MSRT_{s.o}$: if n_1,n_2 are not directly connected in steps <=k then they cannot be directly connected in steps >k, if the sort order of their Clause Sets is not altered, except in the trivial case when the new Clause belongs to a block, parents of n_1,n_2 were instantiating in steps <=k and n_1, n_2 become equivalent (tCN, tMSCN). b- For all M>1: A node [q] of size M is CN/MSCN iff there exist CN/MSCN [q'] of size M-1 augmented in size by a clause C such that: [q]=[q'] c-Let up_1,up_j be upper bounds of nodes generated during the whole process of resolution in size-levels 1 and j, respectively, where 1<j<=M. If Splits are not accounted for in any size-level j, then: up_j<=up_1	2SAT-GSPRA LLR_{BS} CNs/MSCNs
Lemma 6: (Aligned Base Cases) All size 1,2 nodes of any $MSRT_{s.o}$ of a 2CNF Clause Set S produced by 2SAT-GSPRA$^+$ are aligned.	2SAT-GSPRA$^+$ RPC, Aligned $MSRT_{s.o}$
Lemma 7: (Alignment $MSRT_{s.o}$) 2SAT-GSPRA$^+$ produces aligned $MSRT_{s.o}$s and if Splits are not counted, then during the whole process of resolution: -The number of newly added size-1-level nodes cannot exceed $RCC_{2\text{-}SAT}$*M^2 - The number of newly added size-j-level nodes, j>1, cannot exceed $RCC_{2\text{-}SAT}$*M^2 as well, for any level j	2SAT-GSPRA$^+$ ACS Aligned $MSRT_{s.o}$
Lemma 8: ∀SB, DB, tCN such that SB⊆DB and tCN formed in SB:tCN can be avoided by appropriately choosing the DB Sorting Condition. Similarly: tMSCNs can be avoided as well.	SB, DB, tMSCN 2SAT-GSPRA$^+$
Lemma 9-a: CNs and MSCNs containing clauses belonging to the BS or their images cannot split. **Proof sketch: in step *k*:** there exists a Clause C_1={a, b}∈BS and a mapping M such that: a'=M_{ST}(a), b'=M_{ST}(***b***). In this step also: All literals of C_1 and all their images were ***new*** in all branches and spaces leading to the MSCN [q], i.e., for all i,$l_{space\text{-}i}$,S: where $l_{space\text{-}i}$ is a branch- or edge-literal of $[q]_{ST}^{sp1,sp2,sp3,..}$, S Clause Set of any parent node in space-i:: ($l_{space\text{-}i}$ | $a_{space\text{-}i}$) and per Lemma 1-a also: M($l_{space\text{-}i}$)<M($a_{space\text{-}i}$). **To split [*q*], in steps>*k*:**: there must exist a Clause C_2={***x***, **y**}∈BS and a parent node p of [q] such that: $x_{space\text{-}i}$ = $l_{space\text{-}i}$ for some literal $l_{space\text{-}i}$ in p, i.e., **M($x_{space\text{-}i}$)<M($a_{space\text{-}i}$)**. Then: Per l.o. of BS: **Either *x*=*a*** which means [q] is only augmented in size not split **or** a<***x*** and thus *(a | x)* per Lemma 1-b. BS is then in one of the forms: 1-{..{..,*a*}..{*r*, *x*}..{*s*, ¬*x*}…{*a, b*}…{*x*,*y*}…} or 2-{..{*a*,*b*}..{*x*,*y*},..}. Form 2 leads to ($a_{space\text{-}i}$ | $x_{space\text{-}i}$), hence: **M($a_{space\text{-}i}$) <M($x_{space\text{-}i}$).** Contradiction. Form 1 causes the MSCN to be augmented by {*x*,*y*}, not split. (A shorter version of this anchor proof of this work, using the '>' relation, can be found in Footnote 43)	BS, rank, size CN, MSCN Mapping , monotone Mapping Distinguished-, non-Distinguished Literal Lemma 1

Lemma 10: The total number of unique-nodes produced by 2SAT-GSPRA^{+} in the final MSRT$_{s.o}$, including those generated by Splits, is bounded above by:

$$2+ c*RCC_{2\text{-}SAT}^{2} *M^{4} + RCC_{2\text{-}SAT} *M^{3}, c<=2, \text{i.e., } \mathbf{O(M^{4})}$$

Moreover: This bound remains polynomial, i.e., **O(M^{6})**, if Splits are admitted which are not BigSps (i.e., Lemma 9-c relaxed).

- Alignment Clause
- CN, MSCN
- CRA-Form
- ACS
- Lemma 5
- Lemma 6
- Lemma 9

Lemma 11: The following is true:

a- For any arbitrary 2CNF Clause Set S: ∃MSRT$_{s.o}$ such that: 2SAT-FGPRA(S)=2SAT-GSPRA^{+}(S).

b- For 2SAT-FGPRA to produce the MSRT$_{s.o}$ shown to exist in point a-: For the main Assistance Operations used by 2SAT-FGPRA on 2CNF Clause Sets S of size M: The total, worst case number of Primitive Operations performed by any single one of them during a run of 2SAT-FGPRA is: **O(M^{9})**. If Splits are admitted which are not BigSps, i.e., Lemma 9-c is relaxed, then this bound is **O(M^{13})**.

- Top-parts
- l.o. Condition
- Lemma 10
- 2SAT-FGPRA

Lemma 12: 2SAT-GSPRA^{+} and 2SAT-FGPRA are complete, truth table equivalent Algorithms, i.e.: Let S be a 2CNF Clause Set, A any Assignment of truth values of literals in **S**, then: Applying A on the MSRT$_{s.o}$ produced by any of the two Algorithms leads to a TRUE leaf **iff** A satisfies S.

- Assignment
- Assignment Satisfies S

Copyright © 2018 JOURNAL ACADEMICA FOUNDATION • All rights reserved
ISSN 2161-3338 • www.journalacademica.org