Polynomial Exact-3-SAT-Solving Algorithm

Matthias Michael Mueller
louis@louis-coder.com

Sat, 2017-11-04
Version DM-1.0

Abstract

This article describes a relatively simple algorithm which is capable of solving any instance of a 3-SAT CNF in maximal $O(n^{18})$, whereby n is the literal index range within the formula to solve. Under the supposition the algorithm is correct, the P-NP-Problem would be solved with the result that the complexity classes NP and P are equal.

1 Introduction

Problems denoted as "NP-complete" are those algorithms which need an amount of computing time or space which grows exponentially with the problem size n. If it should be accomplished to solve in general at least one of those problems in polynomial time and space, all NP-complete problems out of the complexity class NP would from then on be solvable much more efficiently. Finding the answer to the open question if such a faster computation is possible at all is called the P versus NP problem.[10] The present article describes and analyzes an algorithm which is supposed by its author to solve the NP-complete problem "Exact-3-SAT" in polynomial time and space. In case all crucial statements made in this document are correct, the P versus NP problem would be solved with the conclusion that NP-complete problems *can* be solved in polynomial time and space. In this case, NP-complete problems would all lie in the complexity class P and thus P = NP was proven.

2 Definitions

The following definitions will be used to describe and analyze the polynomial algorithm. Their purpose will become accessible in the course of the document.

2.1 3-SAT CNF

Given is a formula in conjunctive normal form:

$$CNF = \bigwedge_{i=1}^{\gamma} (\epsilon_{i1} x_{i1} \vee \epsilon_{i2} x_{i2} \vee \epsilon_{i3} x_{i3})$$

This 3-SAT CNF, synonymously called SAT CNF or just CNF, consists of γ many and-ed clauses. Within each clause, $x_{i1}, x_{i2}, x_{i3} \mid i_1, i_2, i_3 \in \{1, ..., n\}$ are or-ed Boolean variables called literals. There are always exactly three literals in each clause. Each literal has assigned a second variable epsilon: $\epsilon_{i1}, \epsilon_{i2}, \epsilon_{i3} \in \{0, 1\}$. If this ϵ_{ia} has the value 1, then the value of x_{ia} is negated when evaluating the CNF. If ϵ_{ia} has the value 0, no negation is done. The literal indices are chosen out of a set of natural numbers $\{1, ..., n\}$. The literal indices are pair-wise distinct: $i_2 \neq i_1, i_3 \neq i_1, i_3 \neq i_2$. n will be used as description of the problem size in the upcoming analysis of the solver's complexity.

The task of the presented polynomial solver is to find out if there is an assignment of either true or false to each literal so that a given 3-SAT CNF as a whole evaluates to true. Then we say this assignment, synonymously called solution, satisfies the CNF. If the CNF evaluates to true, we say the 3-SAT CNF is solvable. If it is not possible to satisfy the CNF, we say the 3-SAT CNF is unsatisfiable. The solvability must be determined by the solver in polynomial time and space. The solver only says *if* there is a solution or not, it does not output an assignment. This is still sufficient to solve the P versus NP problem.[10]

2.2 Possible Clauses

The set PC of Possible Clauses:

$$PC = \{(\epsilon_{i1}x_{i1} \vee \epsilon_{i2}x_{i2} \vee \epsilon_{i3}x_{i3})\}$$

with $\epsilon_{i1}, \epsilon_{i2}, \epsilon_{i3} \in \{0, 1\}, i_1, i_2, i_3 \in \{1, ..., n\}, i_2 \neq i_1, i_3 \neq i_1, i_3 \neq i_2$.

$|PC| = (2^3 \times \binom{n}{3})$, because there are 2^3 epsilon combinations and $\binom{n}{3}$ possibilities to choose 3 distinct literal indices out of $\{1, ..., n\}$. If a clause $C \in PC$ appears in the 3-SAT CNF to solve we say C is an initially false clause and write $C = 0$. If a clause $C \in PC$ does not appear in the 3-SAT CNF to solve we say C is an initially true clause and write $C = 1$. A clause $C = (\epsilon_{c1}x_{c1} \vee \epsilon_{c2}x_{c2} \vee \epsilon_{c3}x_{c3}) \in PC$ appears in the 3-SAT CNF if there's a clause $S = (\epsilon_{s1}x_{s1} \vee \epsilon_{s2}x_{s2} \vee \epsilon_{s3}x_{s3})$ in the 3-SAT CNF so that $\forall x \in \{1, 2, 3\} : (c_x = s_x \wedge \epsilon_{cx} = \epsilon_{sx})$.

2.3 Underlying Solutions

The set of underlying solutions is the set of all 2^n many producible $0, 1$ progressions:

$$US = \{U \in \{0, 1\}^n\}$$

US_x is the x-th underlying solution in the set. U_x is the x-th element in the single underlying solution U.

2.4 In Conflict

Two clauses $J, K \in PC$ are said to be "in conflict" if they have at least one literal index in common and the ϵ's concerned are not equal:

$$(J \not\equiv K) \Leftrightarrow \exists(i \in \{1, 2, 3\} \mid ((\epsilon_{j_i} = 0 \wedge \epsilon_{ki} = 1) \vee (\epsilon_{j_i} = 1 \wedge \epsilon_{ki} = 0)) \wedge (j_i = k_i))$$

Similar is defined for a clause $J \in PC$ and an underlying solution $U \in US$:

$$(J \not\equiv U) \Leftrightarrow \exists(i \in \{1,2,3\} \mid ((\epsilon_{j_i} = 0 \wedge U_{j_i} = 1) \vee (\epsilon_{j_i} = 1 \wedge U_{j_i} = 0)))$$

We define "not in conflict" as $(J \equiv K) \Leftrightarrow \neg(J \not\equiv K)$ resp. $(J \equiv U) \Leftrightarrow \neg(J \not\equiv U)$.

Furthermore it is defined for a clause $C \in PC$ and an underlying solution $U \in US$: $C \in U \Rightarrow C \equiv U$.

2.5 Clause Table

The Clause Table, abbreviated CT, is a formula in disjunctive normal form:

$$CT = \bigvee_{l=1}^{2^n} (\bigwedge_{c=1}^{\binom{n}{3}} (C_c \mid C_c \equiv US_l))$$

with line l and column c. Each CT line contains and-ed all possible clauses not in conflict with the l-th underlying solution out of the set US.

For better understanding, here the beginning of the clause table for $n = 4$. The underlying solution corresponding to each CT line is given as a remark:

$((\mathtt{0x_1} \vee \mathtt{0x_2} \vee \mathtt{0x_3}) \wedge (\mathtt{0x_1} \vee \mathtt{0x_2} \vee \mathtt{0x_4}) \wedge (\mathtt{0x_1} \vee \mathtt{0x_3} \vee \mathtt{0x_4}) \wedge (\mathtt{0x_2} \vee \mathtt{0x_3} \vee \mathtt{0x_4})) \vee [\mathtt{US_1} = \{0,0,0,0\}]$
$((\mathtt{0x_1} \vee \mathtt{0x_2} \vee \mathtt{0x_3}) \wedge (\mathtt{0x_1} \vee \mathtt{0x_2} \vee \mathtt{1x_4}) \wedge (\mathtt{0x_1} \vee \mathtt{0x_3} \vee \mathtt{1x_4}) \wedge (\mathtt{0x_2} \vee \mathtt{0x_3} \vee \mathtt{1x_4})) \vee [\mathtt{US_2} = \{0,0,0,1\}]$
$((\mathtt{0x_1} \vee \mathtt{0x_2} \vee \mathtt{1x_3}) \wedge (\mathtt{0x_1} \vee \mathtt{0x_2} \vee \mathtt{0x_4}) \wedge (\mathtt{0x_1} \vee \mathtt{1x_3} \vee \mathtt{0x_4}) \wedge (\mathtt{0x_2} \vee \mathtt{1x_3} \vee \mathtt{0x_4})) \vee [\mathtt{US_3} = \{0,0,1,0\}]$
$((\mathtt{0x_1} \vee \mathtt{0x_2} \vee \mathtt{1x_3}) \wedge (\mathtt{0x_1} \vee \mathtt{0x_2} \vee \mathtt{1x_4}) \wedge (\mathtt{0x_1} \vee \mathtt{1x_3} \vee \mathtt{1x_4}) \wedge (\mathtt{0x_2} \vee \mathtt{1x_3} \vee \mathtt{1x_4})) \vee [\mathtt{US_4} = \{0,0,1,1\}]$
...

Claim: if and only if there is a CT line with clauses which are all absent from the 3-SAT CNF, then this 3-SAT CNF is solvable:

$$\exists(l \mid \forall c = \{1,...,\binom{n}{3}\} : (C_c \equiv US_l \wedge C_c = 1)) \Leftrightarrow \texttt{SAT CNF is solvable}.$$

Proof: We define for $S, S' \in US$: $S' = neg(S) := (\forall x \in \{1,...,n\} : S'_x = \neg S_x$ [1]). Furthermore we define that a clause $C = (\epsilon_{c1}x_{c1} \vee \epsilon_{c2}x_{c2} \vee \epsilon_{c3}x_{c3})$ is satisfied by an underlying solution $U = \{0,1\}^n$ if $\exists x \in \{1,2,3\} : \epsilon_{cx} = U_{cx}$. Notice that here merely one x is enough.

In case the SAT CNF is solvable: We know there is at least one solution $S \in US$ which satisfies the SAT CNF. The SAT CNF does not contain any clause from the CT line l with $US_l = neg(S)$ because such a clause would not be satisfied by S and thus S would not satisfy *all* clauses of the SAT CNF. So at least all clauses from the CT line l do not appear in the SAT CNF.

In case the SAT CNF is unsatisfiable: For each $S \in US$ there must be at least one clause C in the SAT CNF for which applies: $\forall x \in \{1,2,3\} : \epsilon_{cx} = \neg(S_{cx})$. Otherwise S would be a solution, as mentioned in the previous paragraph, and the SAT CNF would be solvable, what is not the case by the basic assumption of this proof. This means: $\forall(neg(S) \in US)\exists(C \equiv neg(S) \wedge C = 0)$. Because for absolutely every underlying solution $US_x \mid x \in \{1,...,2^n\}$ there's at least one clause $C = 0$ not in conflict with US_x, it is for sure we can conclude: $\forall(S \in US)\exists(C \equiv S \wedge C = 0)$. This implies: $\forall(US_l \mid l \in \{1,...,2^n\})\exists(C \equiv US_l \wedge C = 0)$. This does <u>not</u> fulfill $\exists(l \mid \forall c = \{1,...,\binom{n}{3}\} : (C_c \equiv US_l \wedge C_c = 1))$. This is the result we need, because it was claimed the latter condition is only fulfilled if there is a CT line with only initially true clauses. □

[1] $\neg 0 = 1$ and $\neg 1 = 0$

2.6 Being Contained

We regard the clauses $I = (\epsilon_{i1}x_{i1} \vee \epsilon_{i2}x_{i2} \vee \epsilon_{i3}x_{i3})$, $H = (\epsilon_{h1}x_{h1} \vee \epsilon_{h2}x_{h2} \vee \epsilon_{h3}x_{h3})$, $J = (\epsilon_{j1}x_{j1} \vee \epsilon_{j2}x_{j2} \vee \epsilon_{j3}x_{j3})$ and $K = (\epsilon_{k1}x_{k1} \vee \epsilon_{k2}x_{k2} \vee \epsilon_{k3}x_{k3})$.

We define the set D of tuples:

$$D = \{\{(j_a, \epsilon_{j_a})\} \cup \{(k_b, \epsilon_{kb})\} \mid a, b \in \{1, 2, 3\}\}$$

We say I is contained within (J, K) if:

$$(I \equiv J \wedge I \equiv K \wedge J \equiv K) \wedge$$
$$(\forall x \in \{1, 2, 3\} : (i_x, \epsilon_{ix}) \in D)$$

This means that each literal of I appears with same literal index and ϵ value in J or K or both. Also the three clauses are not in conflict pair-wise.

We say I and H are contained within (J, K) if:

$$(I \equiv J \wedge I \equiv K \wedge H \equiv J \wedge H \equiv K \wedge J \equiv K) \wedge$$
$$(\forall (a, b, c, d, e, f \in \{1, 2, 3\}, a \neq b, a \neq c, b \neq c, d \neq e, d \neq f, e \neq f, p \in \{1, ..., n\}) : ($$
$$((i_a, \epsilon_{ia}) \in D) \wedge ((i_b, \epsilon_{ib}) \in D) \wedge ((i_c, \epsilon_{ic}) \notin D) \wedge (i_c = p) \wedge (\epsilon_{ic} = 0) \wedge$$
$$((h_d, \epsilon_{hd}) \in D) \wedge ((h_e, \epsilon_{he}) \in D) \wedge ((h_f, \epsilon_{hf}) \notin D) \wedge (h_f = p) \wedge (\epsilon_{hf} = 1)))$$

This means that two literals of each I and H appear each with the same index and ϵ value in J or K or both. One literal index of I does not appear in J or K but is equal to some value p. The corresponding ϵ_{ic} is always 0. Similarly, one literal index of H does not appear in J or K but is equal to the value p, the same p as in the I case. The corresponding ϵ_{hf} is always 1.

In contrast, the statement that a clause table (CT) line contains one or more clauses just means the clauses appear in the clause table line. This has nothing to do with the property of being contained as defined here in 2.6.

2.7 Enabled/Disabled Clause Tuple

We will make use of the terms "enabled clause tuple(s)" and "disabled clause tuple(s)". An enabled tuple of two possible clauses J and K is noted as $(J, K) = 1$. A disabled tuple of two possible clauses J and K is noted as $(J, K) = 0$.

We say a tuple (J, K) is or gets disabled when the solver sets $(J, K) := 0$. It is not important if the tuple was enabled before, the crucial point is that it does (from then on) apply $(J, K) = 0$.

2.8 Possible Clause Locations in CT

Claim: if $I \in PC$ is contained within $(J, K) \mid J, K \in PC$, then I appears in any CT line l containing (J, K).

Proof: The requirement of three clauses $A, B, C \in PC$ to appear in the same CT line with $U \in US$ is that it must apply $A \equiv U \wedge B \equiv U \wedge C \equiv U$. This can be derived from definition 2.5. This means that then it must also apply $A \equiv B \wedge A \equiv C \wedge B \equiv C$, because all three clauses' ϵ's are uniformly determined by U. If I is contained within (J, K), it does apply $A \equiv B \wedge A \equiv C \wedge B \equiv C$ with $A = I, B = J, C = K$.

Thereby we see the requirements of I being contained within (J, K) does also fulfill the requirement of $I, J, K \equiv U$. $\square$

Claim: if $I \in PC$ and $H \in PC$ is contained within $(J, K) \mid J, K \in PC$, then either I or H, but not both, appears in a CT line l containing (J, K).

Proof: The situation is similar as in the previous proof, with the difference that I and H are in conflict at a literal index p, see definition 2.6. Because U_p can either be 0 or 1, not both I and H can be within the same CT line. $\square$

3 The Polynomial Exact-3-SAT Solving Algorithm

The polynomial solver decides in polynomial time and space if any given 3-SAT CNF is solvable or not. The polynomial solving algorithm consists of an initialization phase, followed by the iterated application of two rules.

INITIALIZATION We regard the entire set of tuples $\{(J, K)\} \mid J, K \in PC$. All those tuples are initially enabled which do not appear in the SAT CNF and whose clauses J and K are not in conflict:

$$(J = 1) \wedge (K = 1) \wedge (J \equiv K) \Rightarrow (J, K) := 1$$

$$(J = 0) \vee (K = 0) \vee (J \not\equiv K) \Rightarrow (J, K) := 0$$

RULE 1 $\exists(((I, J) = 0 \vee (I, K) = 0) \mid I$ is contained within $(J, K)) \Rightarrow (J, K) := 0$.

RULE 2 $\exists((((I, J) = 0 \vee (I, K) = 0) \wedge ((H, J) = 0 \vee (H, K) = 0)) \mid I$ and H are contained within $(J, K)) \Rightarrow (J, K) := 0$.

RULE 1 and RULE 2 are applied repeatedly until all $I, H, J, K \in PC$ combinations have been regarded once in RULE 1 and RULE 2 and no tuple (J, K) has been disabled any more. As last step the result of the polynomial solving process is determined:

$$\exists(J, K \in PC \mid (J, K) = 1) \Rightarrow \texttt{SAT CNF is solvable}.$$

$$\neg\exists(J, K \in PC \mid (J, K) = 1) \Rightarrow \texttt{SAT CNF is unsatisfiable}.$$

In words, if at least one enabled tuple rests, the SAT CNF is solvable. If all tuples were disabled, the SAT CNF is unsatisfiable.

4 Proof of Correctness

4.1 Why Solvable Detection is Reliable

Given: At least one CT line l containing only initially true clauses. This CT line l is to be called the active CT line.

$$\exists(l \mid \forall c = \{1, ..., \binom{n}{3}\} : (C_c \equiv US_l \wedge C_c = 1))$$

From 2.5 we know in this case the SAT CNF is solvable.

It will now be shown: None of the 3 solver rules will disable a tuple (J, K) with $J, K \in PC$ which appears in the active CT line:

INITIALIZATION There is no (J, K) with $(J = 0) \vee (K = 0) \vee (J \not\equiv K)$ for any J, K out of the active CT line l. So $\forall J, K \in US_l : ((J, K) = 1)$.

RULE 1 When I is contained within (J, K) then I appears in any CT line l containing (J, K). This has been shown in 2.8. As it is assumed in this proof that $C = 1$ if $C \in US_l$, it must apply: I contained within (J, K) with $J, K \in US_l \Rightarrow I \in US_l \Rightarrow I = 1$. This means $\forall I, J, K \in US_l : ((I, J) = 1 \wedge (I, K) = 1)$. Because $\neg\exists(((I, J) = 0 \vee (I, K) = 0) \mid I$ is contained within $(J, K))$, (J, K) will stay enabled.

RULE 2 When I and H are contained within (J, K) then either I or H, but not both, appears in a CT line l containing (J, K). This has been shown in 2.8. This means it applies: $I = 1 \vee H = 1$ and therewith $((I, J) = 1 \wedge (I, K) = 1) \vee ((H, J) = 1 \wedge (H, K) = 1)$. But this does not fulfill RULE 2 to disable (J, K), because RULE 2 demands $((I, J) = 0 \vee (I, K) = 0) \wedge ((H, J) = 0 \vee (H, K) = 0)$. So (J, K) will stay enabled.

Because $(\forall(J, K) \mid J \equiv US_l \wedge K \equiv US_l) : (J, K) = 1$, the solver determines "solvable".

4.2 Why Unsatisfiable Detection is Reliable

The solver needs to decide if:

Formula $F1 =$

$$(\forall(U \in \{0,1\}^n)\exists F = (\epsilon_{f_1} x_{f_1} \vee \epsilon_{f_2} x_{f_2} \vee \epsilon_{f_3} x_{f_3}) = 0) :$$
$$(f_1 \in \{1, ..., n\} \wedge f_2 \in \{1, ..., n\} \wedge f_3 \in \{1, ..., n\} \wedge$$
$$\epsilon_{f_1} = U_{f_1} \wedge \epsilon_{f_2} = U_{f_2} \wedge \epsilon_{f_3} = U_{f_3} \wedge$$
$$f_2 \neq f_1 \wedge f_3 \neq f_1 \wedge f_3 \neq f_2)$$

This means the solver needs to decide if there is for each underlying solution $US_x \mid x \in \{1, ..., 2^n\}$ at least one initially false clause F whose epsilon values $e_{f_1}, e_{f_2}, e_{f_3}$ are not in conflict with US_x. In this case the CNF is unsatisfiable, see 2.5. The literal indices f_1, f_2, f_3 can be chosen arbitrarily and do not decide in which CT line F is. The CT line index in which F appears is only determined by F's epsilon values.

The idea of this proof is to show that the polynomial solver will disable any tuple (J, K) with $J, K \in PC$ if there's at least one initially false clause in each CT line containing J and K.

Mathematically, the solver needs to decide if:

Formula $F2 =$

$$(\forall(U \in \{0,1\}^n \mid (U \equiv J \wedge U \equiv K))\exists F = (\epsilon_{f_1} x_{f_1} \vee \epsilon_{f_2} x_{f_2} \vee \epsilon_{f_3} x_{f_3}) = 0) :$$
$$(f_1 \in \{1, ..., n\} \wedge f_2 \in \{1, ..., n\} \wedge f_3 \in \{1, ..., n\} \wedge$$
$$\epsilon_{f_1} = U_{f_1} \wedge \epsilon_{f_2} = U_{f_2} \wedge \epsilon_{f_3} = U_{f_3} \wedge$$
$$f_2 \neq f_1 \wedge f_3 \neq f_1 \wedge f_3 \neq f_2)$$

This is almost the same decision as decided by formula $F1$, except only those underlying solutions (and therewith CT lines) containing J and K are taken into account.

The proof starts with a restricted literal index- and epsilon-set, which will be extended in every proof passage. This means the sets f_1, f_2, f_3 and $\epsilon_{f1}, \epsilon_{f2}, \epsilon_{f3}$ can be chosen from grow. At the end of the proof it will be clear the solver regards the complete literal index- and epsilon-set as stated in formula $F2$.

The solver needs to check if there's at least one initially false clause F in *each* CT line which contains J and K. This means the solver might need to look for multiple Fs if it should *not* apply for *all* initially false clauses F: $\forall x \in \{1,2,3\} : (f_x = j_x \vee f_x = k_x)$. In this case the solver needs, when extending the restricted literal index- and epsilon-set by one further index, to look for two epsilons, once 0 and once 1. This check for two epsilons is implemented by RULE 2, which requires two clauses I and H to be contained within (J, K). I has an epsilon = 0 at position p and H has an epsilon = 1 at position p. Please recall 2.6. It will be pointed out in the following proof that RULE 2 can in practice work recursively. This is the reason why the solver can check for more comprehensive epsilon combinations than just one 0 and one 1.

It follows the proof. Please keep in mind it will be shown each solver rule extends the set the literal indices and epsilons of the initially false clauses can be chosen from. At the end of the proof the required maximal extend of that set will be reached.

INITIALIZATION Trivially, the polynomial solver's initialization rule disables any (J, K) if J or K are initially false clauses. See 3. This is expressed mathematically as:

$(\mathtt{J},\mathtt{K}) := \mathtt{0\ if}$
$(\exists \mathtt{J} = \mathtt{0\ with\ } (\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3\}), (\epsilon_{\mathtt{j}_1}, \epsilon_{\mathtt{j}_2}, \epsilon_{\mathtt{j}_3} \in \{\epsilon_{\mathtt{j}_1}, \epsilon_{\mathtt{j}_2}, \epsilon_{\mathtt{j}_3}\})) \vee$
$(\exists \mathtt{K} = \mathtt{0\ with\ } (\mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3 \in \{\mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3\}), (\epsilon_{\mathtt{k}_1}, \epsilon_{\mathtt{k}_2}, \epsilon_{\mathtt{k}_3} \in \{\epsilon_{\mathtt{k}_1}, \epsilon_{\mathtt{k}_2}, \epsilon_{\mathtt{k}_3}\}))$

RULE 1 We suppose there is no $J = 0$ and no $K = 0$. Then some $I = 0$ being contained within (J, K) can disable (J, K). Being contained means in this case each of I's literal index- and ϵ-tuples is equal to one from J or/and K. See definition 2.6. So after applying the INITIALIZATION rule and RULE 1 it applies:

$(\mathtt{J},\mathtt{K}) := \mathtt{0\ if\ } \exists \mathtt{I} = \mathtt{0\ with\ } \mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3\}$

$$\forall x \in \{1,2,3\} : \epsilon_{ix} = \begin{cases} \epsilon_{jx} & \text{if } i_x = j_x \\ \epsilon_{kx} & \text{if } i_x = k_x \end{cases}$$

Each of the literals i_1, i_2, i_3 can be chosen out of the stated set to disable (J, K). i_1, i_2, i_3 must be distinct pair-wise, because i_1, i_2, i_3 are the literals out of a 3-SAT clause. The need for distinction applies to all literal triples regarded in this proof. To avoid excessive and therefore confusing notation work, this is not always explicitly mentioned.

RULE 2 $\underline{\lambda = 0}$

Solver RULE 2 is left. RULE 2 disables (J, K) if there are *two* initially false clauses I and H.

$(\mathtt{J},\mathtt{K}) := \mathtt{0\ if}$
$(\exists \mathtt{I} = \mathtt{0\ with\ } \mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0\}) \wedge$
$(\exists \mathtt{H} = \mathtt{0\ with\ } \mathtt{h}_1, \mathtt{h}_2, \mathtt{h}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0\})$

$$\forall x \in \{1,2,3\} : \epsilon_{ix} = \begin{cases} \epsilon_{jx} & \text{if } i_x = j_x \\ \epsilon_{kx} & \text{if } i_x = k_x \\ 0 & \text{if } i_x = p_0 \end{cases}, \quad \epsilon_{hx} = \begin{cases} \epsilon_{jx} & \text{if } h_x = j_x \\ \epsilon_{kx} & \text{if } h_x = k_x \\ 1 & \text{if } h_x = p_0 \end{cases}$$

This is because

$\mathtt{I\ with\ } \mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0\} \mathtt{\ and}$
$\mathtt{H\ with\ } \mathtt{h}_1, \mathtt{h}_2, \mathtt{h}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0\} \mathtt{\ are\ contained\ within}$
$\mathtt{J\ with\ } \mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3\} \mathtt{\ and}$
$\mathtt{K\ with\ } \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3 \in \{\mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3\}.$

Because $I = 0$ and $H = 0$ are contained in (J, K) as defined in 2.6, RULE 2 will set $(J, K) := 0$.

$\underline{\lambda = 1}$

We suppose there's no $J = 0$, no $K = 0$ and no $I = 0$ or/and no $H = 0$. Even in this case $(J, K) := 0$ is possible, even though we exhausted all possibilities of the three rules of the polynomial algorithm. The reason is that one or more of the tuples (I, J), (I, K), (H, J), (H, K) might get disabled in the same way as (J, K). So we must take into consideration a *recursive* usage of RULE 2. The recursion is not purposely implemented but happens in practice because clause tuples depend from each other in what concerns their enabled state.

We know that $(J, K) := 0 \; if \; ((I, J) = 0 \vee (I, K) = 0) \wedge ((H, J) = 0 \vee (H, K) = 0)$. The four tuples of this formula can get disabled in the same way as (J, K):

Observation:

$(\mathtt{I}, \mathtt{J}) := \mathtt{0}$ **if** $((\mathtt{I1}, \mathtt{I}) = \mathtt{0} \vee (\mathtt{I1}, \mathtt{J}) = \mathtt{0}) \wedge ((\mathtt{H1}, \mathtt{I}) = \mathtt{0} \vee (\mathtt{H1}, \mathtt{J}) = \mathtt{0})$
$(\mathtt{I}, \mathtt{K}) := \mathtt{0}$ **if** $((\mathtt{I1}, \mathtt{I}) = \mathtt{0} \vee (\mathtt{I1}, \mathtt{K}) = \mathtt{0}) \wedge ((\mathtt{H1}, \mathtt{I}) = \mathtt{0} \vee (\mathtt{H1}, \mathtt{K}) = \mathtt{0})$
$(\mathtt{H}, \mathtt{J}) := \mathtt{0}$ **if** $((\mathtt{I2}, \mathtt{H}) = \mathtt{0} \vee (\mathtt{I2}, \mathtt{J}) = \mathtt{0}) \wedge ((\mathtt{H2}, \mathtt{H}) = \mathtt{0} \vee (\mathtt{H2}, \mathtt{J}) = \mathtt{0})$
$(\mathtt{H}, \mathtt{K}) := \mathtt{0}$ **if** $((\mathtt{I2}, \mathtt{H}) = \mathtt{0} \vee (\mathtt{I2}, \mathtt{K}) = \mathtt{0}) \wedge ((\mathtt{H2}, \mathtt{H}) = \mathtt{0} \vee (\mathtt{H2}, \mathtt{K}) = \mathtt{0})$

We define the recursion depth as λ. This λ grows by one each time one clause tuple out of $((tuple) \vee (tuple)) \wedge ((tuple) \vee (tuple))$ is turned into the tuple to be disabled in the next deeper recursion layer.

We regard the first line of the observation and see:

Either $(\mathtt{I}, \mathtt{J}) := \mathtt{0}$ **or** $(\mathtt{I}, \mathtt{K}) := \mathtt{0}$ **if**
$(\exists \mathtt{I1} = \mathtt{0}$ **with** $\mathtt{i1}_1, \mathtt{i1}_2, \mathtt{i1}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, \mathtt{p}_1\}) \wedge$
$(\exists \mathtt{H1} = \mathtt{0}$ **with** $\mathtt{h1}_1, \mathtt{h1}_2, \mathtt{h1}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, \mathtt{p}_1\})$

$$\forall x \in \{1, 2, 3\} : \epsilon_{i1x} = \begin{cases} \epsilon_{j_x} & \text{if } i1_x = j_x \\ \epsilon_{k_x} & \text{if } i1_x = k_x \\ \epsilon_{p_0} & \text{if } i1_x = p_0 \\ 0 & \text{if } i1_x = p_1 \end{cases}, \quad \epsilon_{h1x} = \begin{cases} \epsilon_{j_x} & \text{if } h1_x = j_x \\ \epsilon_{k_x} & \text{if } h1_x = k_x \\ \epsilon_{p_0} & \text{if } h1_x = p_0 \\ 1 & \text{if } h1_x = p_1 \end{cases}$$

This is because

$\mathtt{I1}$ **with** $\mathtt{i1}_1, \mathtt{i1}_2, \mathtt{i1}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, \mathtt{p}_1\}$ **and**
$\mathtt{H1}$ **with** $\mathtt{h1}_1, \mathtt{h1}_2, \mathtt{h1}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, \mathtt{p}_1\}$ **are contained within**
$\mathtt{I}$ **with** $\mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0\}$ **and**
$\mathtt{J}$ **with** $\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3\}$

or

$\mathtt{I1}$ **with** $\mathtt{i1}_1, \mathtt{i1}_2, \mathtt{i1}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, \mathtt{p}_1\}$ **and**
$\mathtt{H1}$ **with** $\mathtt{h1}_1, \mathtt{h1}_2, \mathtt{h1}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, \mathtt{p}_1\}$ **are contained within**
$\mathtt{I}$ **with** $\mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3 \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0\}$ **and**
$\mathtt{K}$ **with** $\mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3 \in \{\mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3\}$

Reason for being contained:

- First, it is important to recognize $i1_1, i1_2, i1_3$ and $h1_1, h1_2, h1_3$ will *not* all be out of $j_1, j_2, j_3, k_1, k_2, k_3, p_0$. If this was the case, RULE 2 would already have set $(J, K) := 0$, as explained in the $\lambda = 0$ passage. So we know at least one literal index of $i1_1, i1_2, i1_3$ and one literal index of $h1_1, h1_2, h1_3$ is not in $j_1, j_2, j_3, k_1, k_2, k_3, p_0$. The indices are either p_1 or some higher-indexed p (p_2, p_3 etc.) in a deeper recursion layer. So there are only two literal indices of $I1$ and two literal indices of $H1$ which must be equal to literal indices out of I and (J or K).

– The polynomial solver will detect $I1$ and $H1$ being contained within I and (J or K) if:
  * two literal indices of $I1$ are equal to two literal indices of I.
  * one literal index of $H1$ is equal to one literal index of I.
  * one literal index of $H1$ is equal to one literal of either J or K.

  This is an excerpt of the possible cases. The same situation with I and H swapped would be accepted as being contained as well. However, it suffices to take into consideration the described sub case.

$I1 = 0$ and $H1 = 0$ is required if there's no initially false I, J and K. Additionally, or solely, $I2 = 0$ and $H2 = 0$ is required if there's no initially false H, J and K. This case resembles the previous one:

$\texttt{Either } (\texttt{H}, \texttt{J}) := \texttt{0 or } (\texttt{H}, \texttt{K}) := \texttt{0 if}$
$(\exists \texttt{I2} = \texttt{0 with } \texttt{i2}_1, \texttt{i2}_2, \texttt{i2}_3 \in \{\texttt{j}_1, \texttt{j}_2, \texttt{j}_3, \texttt{k}_1, \texttt{k}_2, \texttt{k}_3, \texttt{p}_0, \texttt{p}_1\}) \wedge$
$(\exists \texttt{H2} = \texttt{0 with } \texttt{h2}_1, \texttt{h2}_2, \texttt{h2}_3 \in \{\texttt{j}_1, \texttt{j}_2, \texttt{j}_3, \texttt{k}_1, \texttt{k}_2, \texttt{k}_3, \texttt{p}_0, \texttt{p}_1\})$

$$\forall x \in \{1,2,3\} : \epsilon_{i2x} = \begin{cases} \epsilon_{jx} & \text{if } i2_x = j_x \\ \epsilon_{kx} & \text{if } i2_x = k_x \\ \epsilon_{p0} & \text{if } i2_x = p_0 \\ 0 & \text{if } i2_x = p_1 \end{cases}, \quad \epsilon_{h2x} = \begin{cases} \epsilon_{jx} & \text{if } h2_x = j_x \\ \epsilon_{kx} & \text{if } h2_x = k_x \\ \epsilon_{p0} & \text{if } h2_x = p_0 \\ 1 & \text{if } h2_x = p_1 \end{cases}$$

This is because

$\texttt{I2 with } \texttt{i2}_1, \texttt{i2}_2, \texttt{i2}_3 \in \{\texttt{j}_1, \texttt{j}_2, \texttt{j}_3, \texttt{k}_1, \texttt{k}_2, \texttt{k}_3, \texttt{p}_0, \texttt{p}_1\} \texttt{ and}$
$\texttt{H2 with } \texttt{h2}_1, \texttt{h2}_2, \texttt{h2}_3 \in \{\texttt{j}_1, \texttt{j}_2, \texttt{j}_3, \texttt{k}_1, \texttt{k}_2, \texttt{k}_3, \texttt{p}_0, \texttt{p}_1\} \texttt{ are contained within}$
$\texttt{H with } \texttt{h}_1, \texttt{h}_2, \texttt{h}_3 \in \{\texttt{j}_1, \texttt{j}_2, \texttt{j}_3, \texttt{k}_1, \texttt{k}_2, \texttt{k}_3, \texttt{p}_0\} \texttt{ and}$
$\texttt{J with } \texttt{j}_1, \texttt{j}_2, \texttt{j}_3 \in \{\texttt{j}_1, \texttt{j}_2, \texttt{j}_3\}$

or

$\texttt{I2 with } \texttt{i2}_1, \texttt{i2}_2, \texttt{i2}_3 \in \{\texttt{j}_1, \texttt{j}_2, \texttt{j}_3, \texttt{k}_1, \texttt{k}_2, \texttt{k}_3, \texttt{p}_0, \texttt{p}_1\} \texttt{ and}$
$\texttt{H2 with } \texttt{h2}_1, \texttt{h2}_2, \texttt{h2}_3 \in \{\texttt{j}_1, \texttt{j}_2, \texttt{j}_3, \texttt{k}_1, \texttt{k}_2, \texttt{k}_3, \texttt{p}_0, \texttt{p}_1\} \texttt{ are contained within}$
$\texttt{H with } \texttt{h}_1, \texttt{h}_2, \texttt{h}_3 \in \{\texttt{j}_1, \texttt{j}_2, \texttt{j}_3, \texttt{k}_1, \texttt{k}_2, \texttt{k}_3, \texttt{p}_0\} \texttt{ and}$
$\texttt{K with } \texttt{k}_1, \texttt{k}_2, \texttt{k}_3 \in \{\texttt{k}_1, \texttt{k}_2, \texttt{k}_3\}$

Proving this works analogous to the $I1, H1$ case.

Summarized, the polynomial solver will disable (J, K) if:

$(\texttt{J}, \texttt{K}) := \texttt{0 if } ($
$(\exists \texttt{I1} = \texttt{0 with } \texttt{i1}_1, \texttt{i1}_2, \texttt{i1}_3 \in \{\texttt{j}_1, \texttt{j}_2, \texttt{j}_3, \texttt{k}_1, \texttt{k}_2, \texttt{k}_3, \texttt{p}_0, \texttt{p}_1\}) \wedge$
$(\exists \texttt{H1} = \texttt{0 with } \texttt{h1}_1, \texttt{h1}_2, \texttt{h1}_3 \in \{\texttt{j}_1, \texttt{j}_2, \texttt{j}_3, \texttt{k}_1, \texttt{k}_2, \texttt{k}_3, \texttt{p}_0, \texttt{p}_1\})$
$) \wedge ($
$(\exists \texttt{I2} = \texttt{0 with } \texttt{i2}_1, \texttt{i2}_2, \texttt{i2}_3 \in \{\texttt{j}_1, \texttt{j}_2, \texttt{j}_3, \texttt{k}_1, \texttt{k}_2, \texttt{k}_3, \texttt{p}_0, \texttt{p}_1\}) \wedge$
$(\exists \texttt{H2} = \texttt{0 with } \texttt{h2}_1, \texttt{h2}_2, \texttt{h2}_3 \in \{\texttt{j}_1, \texttt{j}_2, \texttt{j}_3, \texttt{k}_1, \texttt{k}_2, \texttt{k}_3, \texttt{p}_0, \texttt{p}_1\})$
$)$

It is important to notice that each time RULE 2 is examined recursively in this proof, the count of required I (resp. $I1$, $I2$ and so on) and H (resp. $H1$, $H2$ and so on) doubles. Although there's this theoretical doubling and thus a supposed exponential growth of complexity, this is in practice not the case for the presented polynomial solver. The reason is that even in the most comprehensive RULE 2, not more than quadruples of possible clauses are regarded. As the count of possible clauses grows polynomially with the problem size n, it is impossible to get an exponential complexity. If the recursion of RULE 2 would be implemented using recursive procedure calls, quadruples of possible clauses would be regarded by RULE 2 multiple times in the recursive sub calls. If a tuple has already been disabled, it is surplus work to regard it multiple times in recursive sub calls. If a tuple can not be disabled because other tuples it depends from have not yet been disabled, it

is again surplus work to regard this tuple multiple times in recursive sub calls. So because the polynomial solver relinquishes recursive procedure calls and therewith the surplus work, a polynomial complexity is achieved.

As already mentioned, the count of initially false clauses required to disable the basis (J, K) grows when λ grows. There is a kind of branching, as visible in the just presented formula, see "Summarized, the polynomial solver will disable (J, K) if:". All tuples appearing in the sub branches are to be disabled according to the same scheme. For this reason the proceeding in the upcoming proof passage "$\lambda \geq 2$" can be used on any of the sub branches in the same way and will therefore be shown once only.

$\underline{\lambda \geq 2}$

If (J, K) has still not been disabled, further recursive tuple disabling operations are required. This means further recursion layers with higher λ values need to be examined in this proof. All cases with $\lambda \geq 2$ will now be taken in account by an induction proof.

This following induction proof shows:

- If at some recursion depth λ, it applies that RULE 2 accepts:
  $\mathtt{I}_\lambda$ with $\mathtt{i}_{\lambda_1}, \mathtt{i}_{\lambda_2}, \mathtt{i}_{\lambda_3} \in \mathtt{S}_{\mathtt{I}\lambda}$ and
  $\mathtt{H}_\lambda$ with $\mathtt{h}_{\lambda_1}, \mathtt{h}_{\lambda_2}, \mathtt{h}_{\lambda_3} \in \mathtt{S}_{\mathtt{H}\lambda}$, then
- At $\lambda + 1$, it applies that RULE 2 accepts:
  $\mathtt{I}_{\lambda+1}$ with $\mathtt{i}_{\lambda+1_1}, \mathtt{i}_{\lambda+1_2}, \mathtt{i}_{\lambda+1_3} \in \{\mathtt{S}_{\mathtt{I}\lambda} \cup \{\mathtt{p}_{\lambda+1}\}\}$
  $\mathtt{H}_{\lambda+1}$ with $\mathtt{h}_{\lambda+1_1}, \mathtt{h}_{\lambda+1_2}, \mathtt{h}_{\lambda+1_3} \in \{\mathtt{S}_{\mathtt{H}\lambda} \cup \{\mathtt{p}_{\lambda+1}\}\}$.
- This means that for a large enough λ, it applies:
  $\mathtt{I}_{\lambda+\mathtt{x}}$ with $\mathtt{i}_{\lambda+\mathtt{x}_1}, \mathtt{i}_{\lambda+\mathtt{x}_2}, \mathtt{i}_{\lambda+\mathtt{x}_3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_\mathtt{m}\}$
  $\mathtt{H}_{\lambda+\mathtt{x}}$ with $\mathtt{h}_{\lambda+\mathtt{x}_1}, \mathtt{h}_{\lambda+\mathtt{x}_2}, \mathtt{h}_{\lambda+\mathtt{x}_3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_\mathtt{m}\}$.
  Here m represents the count of literal indices $\in \{1, ..., n\}$ which do neither appear in the basis J nor in the basis K of the tuple to be disabled. This means m describes how many distinct positions p, as defined in 2.6, exist for the current basis (J, K).
- The conclusion is that the literal indices
  $\mathtt{i}_{\lambda+\mathtt{x}_1}, \mathtt{i}_{\lambda+\mathtt{x}_2}, \mathtt{i}_{\lambda+\mathtt{x}_3}$ and
  $\mathtt{h}_{\lambda+\mathtt{x}_1}, \mathtt{h}_{\lambda+\mathtt{x}_2}, \mathtt{h}_{\lambda+\mathtt{x}_3}$
  can be chosen out of $\{1, ..., n\}$ for a large enough x. This means the literal indices of each initially false clause in each CT line can be chosen arbitrarily to disable the basis (J, K).

$\underline{\text{Preliminary consideration}}$

Claim:

$\mathtt{I}_\lambda$ with $\mathtt{i}_{\lambda_1}, \mathtt{i}_{\lambda_2}, \mathtt{i}_{\lambda_3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-1}, \mathtt{p}_\lambda\}$ and
$\mathtt{H}_\lambda$ with $\mathtt{h}_{\lambda_1}, \mathtt{h}_{\lambda_2}, \mathtt{h}_{\lambda_3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-1}, \mathtt{p}_\lambda\}$ are contained within
$\mathtt{J}_\lambda$ with $\mathtt{j}_{\lambda_1}, \mathtt{j}_{\lambda_2}, \mathtt{j}_{\lambda_3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-1}\}$ and
$\mathtt{K}_\lambda$ with $\mathtt{k}_{\lambda_1}, \mathtt{k}_{\lambda_2}, \mathtt{k}_{\lambda_3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-2}\}$

Proof: It must be heeded we cannot assign more than summarized three literals of I_λ and H_λ to each J_λ and K_λ. This is because all clauses are exact-3-SAT clauses and thus have exactly 3 literals. So it is important to regard the count of I_λ and H_λ literals being assigned. Additionally it must be recognized K_λ cannot contain any literal index equal to $p_{\lambda-1}$. So a literal of J_λ must be used therefore. K_λ's literal index range goes up to $p_{\lambda-2}$ only, as visible in the formula above.

Keeping this in mind it can be gathered that the polynomial solver will detect I_λ and H_λ as being contained within J_λ, K_λ if:

- If one I_λ literal and one H_λ literal is $p_{\lambda-1}$
  - Those I_λ and H_λ literals (because non-distinct, maximal 1) are equal to some literal in J_λ.
  - The resting I_λ literals (maximal 2) are equal to some literals in J_λ and
  - the resting H_λ literals (maximal 2) are equal to some literals in K_λ.
- If one I_λ literal and no H_λ literal is $p_{\lambda-1}$
  - This I_λ literal (maximal 1) is equal to to some literal in J_λ.
  - The resting I_λ literals (maximal 2) are equal to some literals in J_λ and
  - the resting H_λ literals (maximal 3) are equal to some literals in K_λ.
- If no I_λ literal and one H_λ literal is $p_{\lambda-1}$
  - This H_λ literal (maximal 1) is equal to some literal in J_λ.
  - The resting H_λ literals (maximal 2) are equal to some literals in J_λ and
  - the resting I_λ literals (maximal 3) are equal to some literals in K_λ.
- If no I_λ literal and no H_λ literal is $p_{\lambda-1}$
  - All I_λ literals (maximal 3) are equal to some literals in J_λ and
  - all H_λ literals (maximal 3) are equal to some literals in K_λ.

This is an excerpt of the situations in which the polynomial solver accepts I_λ and H_λ being contained within J_λ, K_λ. However, the just stated situations are in practice sufficient to make the polynomial solver work. This is forecasted by theory and I could also not find any errors by computer-aided verification using self-written computer programs.

It might be not all J_λ and K_λ literals have to be equal to some I_λ or H_λ literals. This happens if I_λ and/or H_λ have one literal index equal to p_λ. Then only 4 respectively 5 literals must be equal. The resting J_λ and K_λ literals can be chosen arbitrarily. Furthermore it is stated "(*maximal* x) [literals are equal ...]" because one of the x many literals might be equal to p_λ. In this case one of the x many literals need not be equal to any J_λ and also not to any K_λ literal. The count x is then $x - 1$. $\square$

INDUCTION PROOF.

Induction basis

We suppose it is given:

$\mathtt{J}_\lambda \text{ } \mathtt{with} \text{ } \mathtt{j}_{\lambda 1}, \mathtt{j}_{\lambda 2}, \mathtt{j}_{\lambda 3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-1}\}$
$\mathtt{K}_\lambda \text{ } \mathtt{with} \text{ } \mathtt{k}_{\lambda 1}, \mathtt{k}_{\lambda 2}, \mathtt{k}_{\lambda 3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-2}\}$

As proven in the earlier preliminary consideration we see that:

$\mathtt{I}_\lambda \text{ } \mathtt{with} \text{ } \mathtt{i}_{\lambda 1}, \mathtt{i}_{\lambda 2}, \mathtt{i}_{\lambda 3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-1}, \mathtt{p}_\lambda\} \text{ } \mathtt{and}$
$\mathtt{H}_\lambda \text{ } \mathtt{with} \text{ } \mathtt{h}_{\lambda 1}, \mathtt{h}_{\lambda 2}, \mathtt{h}_{\lambda 3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-1}, \mathtt{p}_\lambda\} \text{ } \mathtt{are} \text{ } \mathtt{contained} \text{ } \mathtt{within}$
$\mathtt{J}_\lambda \text{ } \mathtt{with} \text{ } \mathtt{j}_{\lambda 1}, \mathtt{j}_{\lambda 2}, \mathtt{j}_{\lambda 3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-1}\} \text{ } \mathtt{and}$
$\mathtt{K}_\lambda \text{ } \mathtt{with} \text{ } \mathtt{k}_{\lambda 1}, \mathtt{k}_{\lambda 2}, \mathtt{k}_{\lambda 3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-2}\}$

$$\forall x \in \{1,2,3\} : \epsilon_{i_\lambda x} = \begin{cases} \epsilon_{j_x} & \text{if } i_{\lambda x} = j_x \\ \epsilon_{kx} & \text{if } i_{\lambda x} = k_x \\ \epsilon_{p_0} & \text{if } i_{\lambda x} = p_0 \\ \epsilon_{p_{\lambda-2}} & \text{if } i_{\lambda x} = p_{\lambda-2} \\ \epsilon_{p_{\lambda-1}} & \text{if } i_{\lambda x} = p_{\lambda-1} \\ 0 & \text{if } i_{\lambda x} = p_\lambda \end{cases}, \quad \epsilon_{h_\lambda x} = \begin{cases} \epsilon_{j_x} & \text{if } h_{\lambda x} = j_x \\ \epsilon_{kx} & \text{if } h_{\lambda x} = k_x \\ \epsilon_{p_0} & \text{if } h_{\lambda x} = p_0 \\ \epsilon_{p_{\lambda-2}} & \text{if } h_{\lambda x} = p_{\lambda-2} \\ \epsilon_{p_{\lambda-1}} & \text{if } h_{\lambda x} = p_{\lambda-1} \\ 1 & \text{if } h_{\lambda x} = p_\lambda \end{cases}$$

The polynomial solver's RULE 2 allows:

$$(\mathtt{J},\mathtt{K}) := \mathtt{0}\ \mathtt{if}\ ((\mathtt{I},\mathtt{J}) = \mathtt{0} \vee (\mathtt{I},\mathtt{K}) = \mathtt{0}) \wedge ((\mathtt{H},\mathtt{J}) = \mathtt{0} \vee (\mathtt{H},\mathtt{K}) = \mathtt{0})$$

We insert the variables of this $\lambda \geq 2$ proof:

$$(\mathtt{J}_\lambda,\mathtt{K}_\lambda) := \mathtt{0}\ \mathtt{if}\ ((\mathtt{I}_\lambda,\mathtt{J}_\lambda) = \mathtt{0} \vee (\mathtt{I}_\lambda,\mathtt{K}_\lambda) = \mathtt{0}) \wedge ((\mathtt{H}_\lambda,\mathtt{J}_\lambda) = \mathtt{0} \vee (\mathtt{H}_\lambda,\mathtt{K}_\lambda) = \mathtt{0})$$

We can set (J_λ, K_λ) in this basis case equal to each of $(I1, I)$, $(H1, I)$, $(I2, H)$, $(H2, H)$ from the $\lambda = 1$ examination. Like this, we have a dependency of tuple enabled states from the maximal possible λ over $\lambda = 1$ down to $\lambda = 0$.

Induction step

Now we imagine it is required that $(I_\lambda, J_\lambda) := 0$ needs to occur before we can set $(J, K) := 0$. This means the solver has to apply another recursive usage of RULE 2:

$$\begin{aligned}&(\mathtt{J}_{\lambda+1} = \mathtt{I}_\lambda, \mathtt{K}_{\lambda+1} = \mathtt{J}_\lambda) = \\ &\quad((\mathtt{I}_{\lambda+1},\mathtt{J}_{\lambda+1}) = \mathtt{0} \vee (\mathtt{I}_{\lambda+1},\mathtt{K}_{\lambda+1}) = \mathtt{0}) \wedge \\ &\quad((\mathtt{H}_{\lambda+1},\mathtt{J}_{\lambda+1}) = \mathtt{0} \vee (\mathtt{H}_{\lambda+1},\mathtt{K}_{\lambda+1}) = \mathtt{0})\end{aligned}$$

For the next greater $\lambda + 1$, it applies:
The "new J", $\mathtt{J}_{\lambda+1}$, is the "old I", $\mathtt{I}_\lambda$.
The "new K", $\mathtt{K}_{\lambda+1}$, is the "old J", $\mathtt{J}_\lambda$.
This is evident from the RULE 2 formula above.

$$\begin{aligned}&\mathtt{J}_{\lambda+1} = \mathtt{I}_\lambda\ \mathtt{with}\ \mathtt{i}_{\lambda 1}, \mathtt{i}_{\lambda 2}, \mathtt{i}_{\lambda 3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-1}, \mathtt{p}_\lambda\}\ \mathtt{and} \\ &\qquad \mathtt{H}_\lambda\ \mathtt{with}\ \mathtt{h}_{\lambda 1}, \mathtt{h}_{\lambda 2}, \mathtt{h}_{\lambda 3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-1}, \mathtt{p}_\lambda\}\ \mathtt{are\ contained\ within} \\ &\mathtt{K}_{\lambda+1} = \mathtt{J}_\lambda\ \mathtt{with}\ \mathtt{j}_{\lambda 1}, \mathtt{j}_{\lambda 2}, \mathtt{j}_{\lambda 3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-1}\}\ \mathtt{and} \\ &\qquad \mathtt{K}_\lambda\ \mathtt{with}\ \mathtt{k}_{\lambda 1}, \mathtt{k}_{\lambda 2}, \mathtt{k}_{\lambda 3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-2}\}\end{aligned}$$

With the analogous examination for $(H_\lambda, J_\lambda) := 0$ it can be concluded:

$$\begin{aligned}&\mathtt{I}_{\lambda+1}\ \mathtt{with}\ \mathtt{i}_{\lambda+1 1}, \mathtt{i}_{\lambda+1 2}, \mathtt{i}_{\lambda+1 3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-1}, \mathtt{p}_\lambda, \mathtt{p}_{\lambda+1}\}\ \mathtt{and} \\ &\mathtt{H}_{\lambda+1}\ \mathtt{with}\ \mathtt{h}_{\lambda+1 1}, \mathtt{h}_{\lambda+1 2}, \mathtt{h}_{\lambda+1 3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-1}, \mathtt{p}_\lambda, \mathtt{p}_{\lambda+1}\}\ \mathtt{are\ contained\ within} \\ &\mathtt{J}_{\lambda+1}\ \mathtt{with}\ \mathtt{j}_{\lambda+1 1}, \mathtt{j}_{\lambda+1 2}, \mathtt{j}_{\lambda+1 3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-1}, \mathtt{p}_\lambda\}\ \mathtt{and} \\ &\mathtt{K}_{\lambda+1}\ \mathtt{with}\ \mathtt{k}_{\lambda+1 1}, \mathtt{k}_{\lambda+1 2}, \mathtt{k}_{\lambda+1 3} \in \{\mathtt{j}_1, \mathtt{j}_2, \mathtt{j}_3, \mathtt{k}_1, \mathtt{k}_2, \mathtt{k}_3, \mathtt{p}_0, ..., \mathtt{p}_{\lambda-1}\}\end{aligned}$$

$$\forall x \in \{1,2,3\} : \epsilon_{i_\lambda x} = \begin{cases} \epsilon_{j_x} & \text{if } i_{\lambda x} = j_x \\ \epsilon_{k_x} & \text{if } i_{\lambda x} = k_x \\ \epsilon_{p_0} & \text{if } i_{\lambda x} = p_0 \\ \epsilon_{p_{\lambda-1}} & \text{if } i_{\lambda x} = p_{\lambda-1} \\ \epsilon_{p_\lambda} & \text{if } i_{\lambda x} = p_\lambda \\ 0 & \text{if } i_{\lambda x} = p_{\lambda+1} \end{cases}, \quad \epsilon_{h_\lambda x} = \begin{cases} \epsilon_{j_x} & \text{if } h_{\lambda x} = j_x \\ \epsilon_{k_x} & \text{if } h_{\lambda x} = k_x \\ \epsilon_{p_0} & \text{if } h_{\lambda x} = p_0 \\ \epsilon_{p_{\lambda-1}} & \text{if } h_{\lambda x} = p_{\lambda-1} \\ \epsilon_{p_\lambda} & \text{if } h_{\lambda x} = p_\lambda \\ 1 & \text{if } h_{\lambda x} = p_{\lambda+1} \end{cases}$$

We see that through another recursive usage of RULE 2, the sets from which we can choose the initially false clauses' literal indices has grown. The sets have been extended by $p_{\lambda+1}$. This is guaranteed to happen with each recursive usage of RULE 2. This is for sure because RULE 2 generally accepts I and H clauses containing a formerly unused literal index p (synonymously called p_λ, $p_{\lambda+1}$ etc.). With a sufficiently large λ, the set from which we can choose the initially false clauses' literal indices is equal to $\{1, ..., n\}$. The result is that we can choose the literals for each initially false clause in each CT line arbitrarily. In any case it is guaranteed that $(J, K) := 0$ will occur. This means that if there's at least one initially false clause in each CT line, no matter which literal indices these initially false clauses have, any (J, K) with $J, K \in PC$ will get disabled. The polynomial solver will output 'unsatisfiable' because no $(J, K) = 1$ rests. $\square$

Further notes

In the $\lambda \geq 2$ case the proof suggests it is not required to involve (I_λ, K_λ) or (H_λ, K_λ). This has been tested by me using computer-aided verification. The result is that the K_λ tuples were really not required in all tests done. But this does not apply for the $\lambda = 1$ case, what can be derived from the proof. There it is shown that one I or H literal will be equal to one literal in *either* J *or* K. It is not known in advance if J or K will be required. This depends on the SAT CNF to solve. I verified also this statement (that (I, K) and (H, K) is mandatory) using a test program, with the observation the statement seems to be correct.

5 Complexity

The size of the set PC is of great importance because the polynomial solver's main work consists substantially of looping through the set of possible clauses. There are $|PS| = O(n^3)$ many possible clauses, because we can place the three indices of all possible clauses using three nested loops, each having an iteration range not larger than 1 to n. Furthermore there are $2^3 = 8$ possibilities for each clause to choose the three ϵ values out of $\{0, 1\}$. But because this is a constant complexity, it will not be observed in the O notation. Regarding all possible combinations of x many possible clauses one time has a complexity of $O((n^3)^x)$. This is the case because we had to implement x many nested loops, each having an iteration range of 1 to $|PS|$.

Next, we determine the complexity of all 3 solving steps. We regard the 3 steps independently because they are executed sequentially.

INITIALIZATION Regard $J, K \in PC \Rightarrow O((n^3) \times (n^3)) = O(n^6)$.

RULE 1 Regard $I, J, K \in PC \Rightarrow O((n^3) \times (n^3) \times (n^3)) = O(n^9)$.

RULE 2 Regard $I, H, J, K \in PC \Rightarrow O((n^3) \times (n^3) \times (n^3) \times (n^3)) = O(n^{12})$.

We apply RULE 1 and RULE 2 at maximum up to the point all $O((n^3) \times (n^3))$ many clause tuples have been disabled. This means we apply RULE 1 and RULE 2 maximal $O(n^6)$ times, whereby RULE 2 is the most comprehensive operation. So we get a total complexity of $O(n^{12} \times n^6) = O(n^{18})$. In this consideration it was assumed that checking for containment is done in constant time. This can be achieved by pre-computing if I, H is contained within (J, K). The pre-computing would require $O((n^3)^4) = O(n^{12})$ for examining all required clause combinations. Similarly, the solver can also pre-compute for each possible clause if it appears in the SAT CNF. The pre-computing would require $O((n^3) \times (n^3)) = O(n^6)$ to loop through all possible clauses to check for appearance, multiplied with the SAT CNF's highest possible clause count. The pre-computing does not increase the final overall complexity because it is independent from the work with highest complexity.

6 Further Reading

The present document explains the polynomial exact-3-SAT solving algorithm using mathematical notation. There's an older document version online which has more pages and uses more linguistic paraphrases. Furthermore there are C++ sample implementations of the algorithm available which run on Windows or Linux. All these items can be downloaded from the author's homepage www.louis-coder.com.

7 Acknowledgments

I thank Mr. Mihai Prunescu, Simion Stoilow Institute of Mathematics of the Romanian Academy, for helpful tips and a reference to the polynomial algorithm in one of his articles (see [8], resp. [9]).

References

[1] Michael R. Garey and David S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, W. H. Freeman & Co., 1979.

[2] Christos H. Papadimitriou, Computational complexity, Addison-Wesley, 1994.

[3] Uwe Schöning, Theoretische Informatik - kurz gefasst, Bibl. Institut Wissenschaftsverlag, 1992, ISBN 3-411-15641-4.

[4] Ingo Wegener, Theoretische Informatik - eine algorithmenorientierte Einführung (3. Auflage), B. G. Teubner Verlag / GWV Fachverlage GmbH, Wiesbaden 2005, ISBN 3-8351-0033-5.

[5] Volker Heun, Grundlegende Algorithmen (2. Auflage), Friedr. Vieweg & Sohn Verlag / GWV Fachverlage GmbH, Wiesbaden 2003, ISBN 3-528-13140-3.

[6] Daniel Grieser, Mathematisches Problemlösen und Beweisen, Springer Fachmedien Wiesbaden 2013, ISBN 978-3-8348-2459-2.

[7] Bronstein, Semendjajew, Musiol, Mühlig, Taschenbuch der Mathematik, Verlag Harri Deutsch, Thun und Frankfurt am Main 2000, ISBN 3-8171-2015-X.

[8] Prunescu, Mihai, About a surprizing computer program of Matthias Müller, https://imar.academia.edu/MihaiPrunescu (link checked 2017-November-03).

[9] Prunescu, Mihai, About a Surprising Computer Program of Matthias Müller, Convexity and Discrete Geometry Including Graph Theory: Mulhouse, France, September 2014, Springer International Publishing, ISBN 978-3-319-28186-5_9, http://dx.doi.org/10.1007/978-3-319-28186-5_9 (link checked 2017-November-03).

[10] Schöning, Torán, The Satisfiability Problem, Lehmanns Media Berlin 2013, ISBN 978-3-86541-527-1.