A Polynomial Time Algorithm for 3SAT

ROBERT QUIGLEY

It is shown that any two clauses in an instance of 3SAT sharing the same terminal which is positive in one clause and negated in the other can imply a new clause composed of the remaining terms from both clauses. Clauses can also imply other clauses as long as all the terms in the implying clauses exist in the implied clause. It is shown an instance of 3SAT is unsatisfiable if and only if it can derive contradicting 1-terminal clauses in exponential time. It is further shown that these contradicting clauses can be implied with the aforementioned techniques without processing clauses of length 4 or greater, reducing the computation to polynomial time. Therefore there is a polynomial time algorithm that will produce contradicting 1-terminal clauses if and only if the instance of 3SAT is unsatisfiable. Since such an algorithm exists and 3SAT is NP-Complete, we can conclude P = NP.

CCS Concepts: • **Theory of computation** → **Complexity classes**.

Additional Key Words and Phrases: NP-Complete, SAT, 3SAT, Satisfiability, P vs NP, Polynomial Time, Non-deterministic Polynomial Time, Complexity Theory, Computational Complexity, Complexity Classes, Cook-Levin Theorem, Karp's 21 NP-Complete Problems

ACM Reference Format:
Robert Quigley. 2024. A Polynomial Time Algorithm for 3SAT. 1, 1 (March 2024), 22 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

This section introduces the 3SAT problem, the implications of solving it in polynomial time, and the structure of the paper.

As seen in [1], The boolean satisfiability problem is given as a set of terminals, $x_1, x_2, ..., x_n$, each of which can be assigned a value of True or False, combined by logical AND operators, logical OR operators, and negations. The problem is to determine whether or not there exists an assignment for each terminal that allows the instance to evaluate to True. As seen in [2], the satisfiability problem with exactly 3 literals per clause is NP-Complete. This is the same as the boolean satisfiability problem, but it is presented such that a clause contains exactly three terminals combined with logical OR operators and possibly negations, and each clause is combined with logical AND operators. The idea of NP-completeness shows that if one NP-complete problem can be solved in polynomial time, then all problems in the class NP can be solved in polynomial time. In other words, P = NP.

The paper is structured as follows:

(1) Introduction

Author's address: Robert Quigley, robertquiggleson@gmail.com.

(2) Standard Definitions
- Terms relating to the 3SAT problem

(3) Algorithm-Specific Definitions
- Terms relating to specific aspects of 3SAT regularly referenced in this paper

(4) Reformatting and Processing
- Defines how clauses and instances of the 3SAT problem will appear within the paper

(5) Lemmas
- A list of lemmas and their proofs pertaining to the algorithm

(6) Algorithm
- A step-by-step description of the algorithm to solve 3SAT in polynomial time

(7) Time Complexity Analysis
- An analysis of the time complexity of the algorithm

(8) Proof of Correctness
- A proof showing the algorithm works for every instance of 3SAT

(9) Conclusion

(10) References

2 STANDARD DEFINITIONS

Definition 2.1. Terminals: symbols used in the 3SAT problem that can be assigned a value of either 0 or 1, True or False, or any other binary assignment. They usually take the form x_i where i is a natural number.

Definition 2.2. Terms: terminals in either the positive or negated form that appear in a clause. If a term is positive, then the value of the terminal will be the same as the value of the term. If a term is negated, then the value of the terminal will be the opposite of the value of the term.

Definition 2.3. Clauses: a set of terms combined by logical OR operators. Clauses usually take the form $(x_i \vee \neg x_j \vee x_k)$ where $x_i \neq x_j \neq x_k$.

Definition 2.4. (3SAT) Instance: a set of any number of clauses combined by logical AND operators. Terminals may not repeat within a clause [1] , but they are free to repeat between clauses. Instances usually take the form:
$(x_i \vee \neg x_j \vee x_k) \wedge (x_l \vee x_m \vee x_n)$.

Definition 2.5. Assignment: A list of values in which each value represents either True or False such that each item in the list corresponds to a terminal and all terminals are assigned a value.

Definition 2.6. Satisfying Assignment: An assignment, A, is said to satisfy the instance if applying A will make the instance evaluate to True.

Definition 2.7. Satisfiable: an instance is satisfiable iff there exists a satisfying assignment.

Definition 2.8. Unsatisfiable: an instance is unsatisfiable iff there does not exist a satisfying assignment.

[1]See Lemma 5.3

3 ALGORITHM-SPECIFIC DEFINITIONS

Definition 3.1. Blocking an Assignment: An assignment, A, is said to be blocked by a clause, C if, given an instance containing C, there is no way that A allows C to evaluate to True, and thus there is no way A allows the instance to evaluate to True.

Definition 3.2. Implication: A clause, C, is said to imply another clause, D, if all assignments blocked by D are also blocked by C.

Definition 3.3. Given Clauses: clauses which were given in the original instance.

Definition 3.4. Derived or Implied Clauses: clauses which are implied by the clauses in the original instance.

Definition 3.5. k-terminal (k-t) clause: a clause is described as a k-terminal or a k-t clause if there are k terminals in the clause.

Definition 3.6. Reduction: A special type of implication in which the implied clause is shorter than the implying clause(s).

Definition 3.7. Expansion: A special type of implication in which the implied clause is longer than the implying clause(s) and all terms in the implying clause exist in the implied clause.

Definition 3.8. Contradicting 1-terminal Clauses: A set of two clauses are considered to be contradicting 1-terminal clauses if (1) they are both of length 1, (2) they contain the same terminal, and (3) the terminal is positive in one clause and negated in the other.

4 REFORMATTING AND PROCESSING

Since there are a lot of constant characteristics about an instance of 3SAT, we can remove most of them to allow ourselves to focus only on what changes from instance to instance. A list of unchanging characteristics follows:

- the symbol x
- logical AND operators
- logical OR operators

The only difference between instances, therefore, is the subscript of the terminal.

Additionally, the following items will be changed to improve compatibility with the Python programming language wherein an instance is expressed as a list of lists and each inner list represents a clause:

- parentheses will become square brackets
- negation symbols will become minus signs
- an instance may be surrounded with square brackets to show it is a list of lists

For example, the instance:
$(\neg x_a \vee x_b \vee x_c) \wedge (x_a \vee x_d \vee x_e)$
will be written as:
$[[-a, b, c], [a, d, e]]$

Instances of 3SAT will further be processed by removing any clauses that do not block any assignments. Since this algorithm relies on implications of new clauses, if we ever come across a clause that blocks no assignments, then by definition it will not be able to imply any additional clauses that block at least one assignment.

As such, we will ignore any clauses given or derived that are described in Lemma 5.3

5 LEMMAS

Lemma 5.1. *A clause can block an assignment.*

Proof. Recall an assignment is blocked if it does not allow the clause to evaluate to True.

Since terms in a clause are combined by logical OR operators, a clause cannot evaluate to True if all terms in the clause evaluate to False.

A term evaluates to False if it's either (1) negated and the terminal's value is True or (2) positive and the terminal's value is False.

Given a clause, we know there are some number of unique terminals.

Want to find an assignment where all the terms are assigned a value of False.

Since an assignment exists for all possible values for each terminal, then there exists an assignment such that all the terms in the clause evaluate to False.

Since all the terms evaluate to False and are combined by logical OR operators, the clause will evaluate to False.

Since the instance is composed of clauses combined by logical AND operators and one clause evaluates to False, then the entire instance evaluates to False.

Since the instance evaluates to False, the assignment cannot satisfy the instance. □

Lemma 5.2. *For a given instance with n terminals, there are 2^n possible assignments.*

Proof. An assignment for this instance consists of n values, each with two possible values, True or False.

Therefore, there are 2^n possible assignments. □

Lemma 5.3. *If a clause contains the same terminal in its negated and positive form, it will not block any assignments.*

Proof. Consider a clause containing the same terminal in both the positive and negative form.

We know that terminal must either be True or False. Consider both cases:

That terminal's value is True: The positive form of the terminal will be True and the clause will evaluate to True.

That terminal's value is False: The negated form of the terminal will be True and the clause will evaluate to True.

Since the clause evaluates to True in every case, there will never be an assignment with which it is impossible to make this clause evaluate to False.

In other words, this clause blocks no assignments. □

Lemma 5.4. *Each clause of length k blocks 2^{n-k} assignments.*

Proof. Consider the generic k-terminal clause, C, in an instance with n terminals.

As seen in Lemma 5.1, this blocks all assignments where all of the terms evaluate to False.

Since the values for k terminals are set, there are $n - k$ terminals left whose values could be True or False.

Since an assignment exists for every possible way to assign values to each terminal, we know an assignment exists for every possible way to assign a value for these $n - k$ terminals.

There are two possible ways to assign values to each of these $n - k$ terminals so there are 2^{n-k} unique assignments blocked by C. □

Lemma 5.5. *For any clause, C, if we select a terminal, t, that's not in C then half of the assignments blocked by C will assign True to t and the other half will assign False to t.*

Proof. We have a clause, C, of fixed yet arbitrary length, k:

$[a, b, c, ...]$

Now we select a terminal, t, that's not in C.

Want to show half of the assignments blocked by C assign True to t and the other half assign False to t.

We know there will be no overlap between these assignments because a single assignment cannot assign both the values True and False to the same terminal.

Now we just have to show that exactly half of the assignments are blocked by assigning either True or False to t.

By Lemma 5.4, C blocks 2^{n-k} assignments.

If we fix the value of t, then there are only $n - k - 1$ terminals whose values could be 0 or 1.

Since there are two choices per terminal and there are $n - k - 1$ terminals, then there are 2^{n-k-1} assignments blocked by C where the value of t is fixed.

Divide to get the ratio of the number of assignments blocked by adding t to the number of assignments blocked by C without t:

$2^{n-k-1}/2^{n-k}$

$= 2^{n-k-1-(n-k)}$

$= 2^{-1}$

$= 1/2$

This shows that half of the assignments blocked by C assign a fixed value to t.

Since there are two possible values for t and each block mutually exclusive halves of the assignments blocked by C, the lemma holds. □

Lemma 5.6. *Given a clause, C, and another clause, D, such that all of the terms in C also exist in D, then all of the assignments blocked by D are also blocked by C.*

Proof. Given a clause, C, of a fixed yet arbitrary length:

$C := [a, b, c, ...]$

And another clause, D, containing all the terms in C with possibly additional terms:

$D := [a, b, c, ..., d, e, f, ...]$

Want to show all the assignments blocked by D are also blocked by C.

We know that C blocks all assignments that cause all the terms to evaluate to False.

In other words, C blocks all assignments where

$a = b = c = ... = False$

Similarly, D blocks all assignments where

$a = b = c = ... = d = e = f = ... = False$

Clearly all assignments consistent with the terminal assignments from D are also consistent with the terminal assignments from C.

Therefore, every assignment blocked by D is also blocked by C. □

Lemma 5.7 (Reduction). *Given the following conditions:*

- *A is a clause of length k*
- *B is a clause of length k*
- *A and B share $k - 1$ identical terms*
- *A and B share one terminal that is negated in one clause and positive in the other*
- *C is a clause of length $k - 1$ composed of the shared terms from A and B*

Then A and B imply C.

Proof. Given clauses consistent with the description:

$A := [a, b, c, ...i]$

$B := [a, b, c, ..., -i]$

where $a, b, c, ...$ is shared between the clauses and i is a terminal not in $a, b, c, ...$

Want to show this implies C.

Recall by Lemma 5.3 that the same terminal cannot appear both negated and positive within the same clause and still block an assignment, so i cannot exist in $a, b, c...$

Consider the clause

$C := [a, b, c, ...]$

We know by Lemma 5.5 that if we select a terminal that's not in C, say t, then half of the assignments blocked by C assign True to t and the other half of the assignments blocked by C assign False to t.

Let this terminal t that's not in C be the terminal i that's in A and B.

We know that A blocks all assignments blocked by C where i is assigned the value of False.

We know that B blocks all assignments blocked by C where i is assigned the value of True.

Since A and B both block mutually exclusive halves of the assignments blocked by C, then all of the assignments blocked by C are blocked by A and/or B and we can say that A and B imply C. □

Lemma 5.8 (Expansion). *Given a clause, C, and a terminal, t, that's not in C, then two new clauses can be implied consisting of all the terms of C appended to either the positive form of t or the negated form of t.*

Proof. Given a clause, C, and a terminal, t, that's not in C:

$C := [a, b, c, ...]$

We can compose two new clauses:

$D := [a, b, c, ..., t]$

$E := [a, b, c, ..., -t]$

Since all of the terms in C exist in D and E, then by Lemma 5.6 all of the assignments blocked by D or E are blocked by C and we can say D and E are implied by C. □

Lemma 5.9 (General Lemma 5.7). *If two clauses share the same terminal, t, such that t is positive in one clause and negated in the other, then these clauses imply a new clause which is composed of all the terms in both clauses except terms containing t.*

Proof. Consider two clauses,
$C := [a, b, c, ..., t]$
$D := [d, e, f, ..., -t]$
Where within a clause, the same terminal does not repeat, but between clauses the same terminal may repeat.

Want to show we can imply a clause consistent with the lemma description:
$E := [a, b, c, ..., d, e, f, ...]$
Let's define some additional clauses:
$E' := [a, b, c, ..., d, e, f, ..., t]$
$E'' := [a, b, c, ..., d, e, f, ..., -t]$
By Lemma 5.6, we know that C implies E' because all the terms in C exist in E.
By Lemma 5.6, we know that D implies E''.
By Lemma 5.7, since E' and E'' share all the same terms except for t, which is positive in one clause and negated in the other, we can create a new clause composed of all the shared terms in E' and E''.
Such a clause is already defined as E.
Now there are a couple extra cases to consider:

- There is some overlap between $a, b, c, ...$ and $d, e, f, ...$
- There is the same terminal that's positive in $a, b, c, ...$ and negated in $d, e, f, ...$

First, if the same term exists in $a, b, c, ...$ and $d, e, f, ...$, then we can just remove one of the duplicates since one term being true implies an identical term being True.
Secondly, if the same terminal exists, but is of the opposite form in $a, b, c, ...$ and $d, e, f, ...$ then by Lemma 5.3, this clause will always be True and thus blocks no assignments. In this case, the lemma is vacuously true, but we disregard the clause as it is of no value. □

Lemma 5.10. *Given two clauses of lengths k and m that imply another clause by Lemma 5.9, the length of the implied clause will fall in the range $max(k, m) - 1$ to $(k + m - 2)$ where the $max(k, m)$ represents the parameter with the greatest value.*

Proof. The smallest clause that can be implied by clauses of length k and m using Lemma 5.9 occur when all but one of the terms in one clause exist in the other.
As such, the unique terms will come from the clause that's longer.
Removing t, you are left with 1 less than the maximum of k and m.
The largest clause can be implied if there are no terms shared between the two clauses. In this case you subtract 1 from the length of each clause to account for t and since no duplicates will be removed, the resulting clause's length is 2 less than the sum of the lengths of the clauses. □

Lemma 5.11. *Given the following:*

- *A clause, A, of length less than k*

- *A clause, B, of length less than k*
- *A clause, C, of length less than k*
- *A clause, D, of length k or $k-1$*
- *A clause, E, of length k*
- *A and B imply E by Lemma 5.9*
- *C and E imply D by Lemma 5.9*

Then D can be derived by processing clauses with a maximum length of XXX.

Proof. Define the clauses in the following manner:
$A := [\alpha]$
$B := [\beta]$
$C := [\delta]$
Then the following are derived by Lemma 5.9:
$E = [\alpha \cup \beta - \{i, -i\}]$ (By A and B)
$D = [\alpha \cup \beta \cup \delta - \{i, -i\} - \{j, -j\}]$ (By C and E or by F and B)
$F = [\alpha \cup \delta - \{j, -j\}]$ (By A and C)
Where β, δ, and ϕ are generic sets of terms in the instance and $i, j \in \alpha$, $-i \in \beta$, $-j \in \delta$
Consider the following figure:

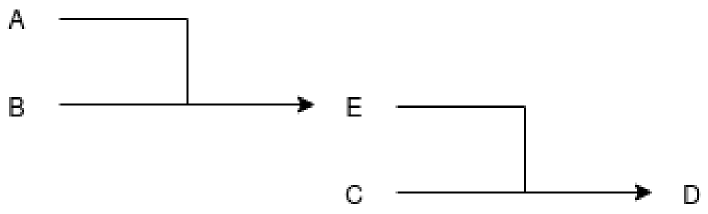

Fig. 1. A graph illustrating the derivations described by the lemma

Note that since C and E imply D, then C and E must share the same terminal such that it is positive in one clause and negated in the other.

Since A and B imply E by Lemma 5.9 then they must share an opposite form term. Let's say $i \in \alpha$ and $-i \in \beta$.

Since E and C imply D by Lemma 5.9 then they must share an opposite form term. Let's say $j \in E$ and $-j \in \delta$.

Notice all the terms in E come from A and B (not counting i or $-i$) so j must be in either A, B, or both.

Since A and B are treated the same, let's consider the cases where j is in either A or B as one case.

So we have two cases:

(1) j is in either α or β
(2) $j \in \alpha and j \in \beta$

(1) j is in either α or β

Since α and β are treated in the same manner, let's pick one set to contain j and fix it for the rest of the example. Let's say $j \in \alpha$.

Now we have the clauses:

$E = [\alpha \cup \beta - \{i, -i\}]$

$D = [\alpha \cup \beta \cup \delta - \{i, -i\} - \{j, -j\}]$

$F = [\alpha \cup \delta - \{j, -j\}]$

How does the length of F compare to D?

Let β' be the set of terms in β that do not exist in α or δ.

$|F| = |D| - \beta' + 2$

$|F|$ is highest when β' is lowest.

The lowest value for β' is 1 because it must contain at least $-i$ and $-i$ cannot exist in α or δ because then A would block no assignments or F would block no assignments and thus they would not be able to imply any other clauses.

So the highest value of $|F|$ is

$|F| = |D| + 1$

Which means

$|F| = k + 1$ when $|D| = k$ and

$|F| = k$ when $|D| = k - 1$

(2) $j \in \alpha and j \in \beta$

If j is in both α and β then the lowest value for $|\beta'|$ is 2 since it must contain $-i$ as well.

Now the highest value for $|F|$ is:

$|F| = |D|$

Considering both cases for D, the length of F is:

$|F| = k$ when $|D| = k$ and

$|F| = k - 1$ when $|D| = k - 1$

So given input clauses of length less than k, we can derive D by processing clauses with a maximum length of $k + 1$

□

LEMMA 5.12. *Given the following:*

- *A is a clause of length less than k*
- *B is a clause*
- *C is a clause of length less than k*
- *D is a clause of length k or $k - 1$*
- *A expands to imply B by Lemma 5.8*
- *B and C imply D by Lemma 5.9*

Then D can be derived by processing clauses with a max length of $k - 1$.

PROOF. Consider the following figure:

Let the clauses be defined as follows:

$A := [\alpha]$

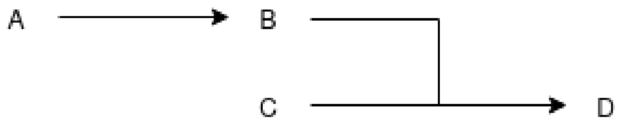

Fig. 2. An illustration of the derivations described in the lemma

$B := [\alpha \cup \beta]$
$C := [\delta]$
$D := [\alpha \cup \beta \cup \delta - \{i, -i\}]$
Where $i \in \alpha \cup \beta$ and $-i \in \delta$
Two cases exist:

(1) $i \notin \alpha$
(2) $i \in \alpha$

(1) $i \notin \alpha$
In this case, i must be in β.
Notice all of the terms in A exist in D so A can be expanded to D by Lemma 5.8.
And D can be derived by processing clauses with a maximum length of $|D| - 1$, which is shorter than k.
(2) $i \in \alpha$
In this case, a new clause can be derived using A and C with Lemma 5.9:
$E := [\alpha \cup \delta - \{i, -i\}]$
Notice all of the terms in E exist in D so E can be expanded to derive D by Lemma 5.8.
The length of E is greatest when $|\beta|$ is minimized.
This occurs when all terms in β exist in α or δ.
If all the terms in β exist in α or δ then E will be exactly the same as D so we derived D by processing clauses with a maximum length of less than k.
Notice $|E|$ is $|\beta'|$ less than $|D|$ where $|\beta'|$ is the set of terms in β that do not exist in α or δ.
When β' is 0, E is exactly D and when β' is greater than 0, then $|E|$ is less than k so we can always derive D by processing clauses with a maximum length of k-1.

□

Lemma 5.13. *Given an instance of 3SAT, you can expand all of the given clauses to the point where you are considering clauses of length n.*

Proof. Given an instance of 3SAT, we know all clauses are of length 3.
If we want to consider a generic n-terminal clause, B, that's implied by a given clause, A, then by Lemma 5.6 we know it's implied if all the terms in A exist in B. □

Lemma 5.14. *If you expand given 3-t clauses as described in Lemma 5.13, you will derive 2^n unique clauses of length n iff the instance is unsatisfiable.*

Proof. Want to show an unsatisfiable instance $\implies$ 2^n unique n-terminal clauses can be derived from the given 3-t clauses:

By lemma 5.4, a clause of length n blocks 1 assignment.

Recall an instance is unsatisfiable iff all 2^n assignments are blocked.

If a 3-terminal clause blocks an assignment, then it also implies the corresponding n-terminal assignment because there is one possible n-terminal clause for any given assignment.

Notice that for each of these n-terminal clauses, they must contain three terms from at least one given clause. If they didn't, then the assignment blocked by that n-terminal clause would not be blocked and the instance would be satisfiable.

Since each n-terminal clause blocks one assignment, blocking all assignments requires 2^n n-terminal clauses.

Want to show 2^n unique n-terminal clauses are derived by the given 3-t clauses $\implies$ then the instance is unsatisfiable.

By lemma 5.4, a clause of length n blocks 1 assignment.

Therefore if there are 2^n unique n-terminal clauses, then all 2^n assignments will be blocked.

Note that there will be no overlap because each n-terminal clause sets the value for each terminal and overlap would imply the same terminal having two values by the same assignment which is impossible. ☐

LEMMA 5.15. *The n-terminal clauses described in Lemma 5.14 can be reduced to derive any pair of contradicting 1-terminal clauses.*

PROOF. Given 2^n n-terminal clauses, want to show you can imply any pair of contradicting 1-terminal clauses by lemma 5.7.

Algorithm:

Pick a terminal that will not exist in the final 1-terminal clauses.

Notice half of the existing n-terminal clauses have that terminal assigned the value of False and the other half have that terminal assigned the value of True.

Pick one clause that blocks an assignment where the terminal is True.

Then there exists an assignment for each possible value for the remaining n-1 terminals.

Therefore, there must exist another clause that shares all of the same terms, but where that one terminal is assigned the value of False.

Using these two clauses, we can create a new clause by lemma 5.7.

Now all of the clauses of length n - 1 do not contain that terminal.

Repeat this process while never selecting the same terminal twice until you are left with two contradicting 1-terminal clauses.

Each clause is guaranteed to have a matching clause since every possible combination of terminal assignments exists and when you use Lemma 5.7 to create new clauses, the rest of all of the clauses remain the same except the selected terminal is removed. ☐

LEMMA 5.16. *Contradicting 1-terminal clauses can be expanded to imply every possible clause. TODO: "contra 1-t clauses imply unsatisfiability"*

PROOF. Let the following clauses be a pair of contradicting 1-t clauses:

$[a]$

$[-a]$

Any possible clause could either contain a, $-a$, or neither.

By lemma 5.8, we can expand to any clause that contains a or $-a$.

Now want to show these clauses imply another clause that does not contain a or $-a$.

Let the following be a generic k-terminal clause that does not contain a or $-a$:

$[b, c, d, ...]$

By Lemma 5.8, we know the 1-terminal clauses imply the following clauses:

$[a, b]$

$[-a, c, d, ...]$

By Lemma 5.9, these clauses imply:

$[b, c, d, ...]$

Therefore a set of contradicting 1-terminal clauses can imply any clause containing $a, -a$, or neither, which encompasses every possible clause. □

Lemma 5.17. *Given the following:*

- *A, B, C, and D, are clauses shorter than k*
- *E and F are clauses of length k*
- *G is a clause of length k or k - 1*
- *A and B imply E by Lemma 5.9*
- *C and D imply F by Lemma 5.9*
- *E and F imply G by Lemma 5.9*

Then G can be implied by processing clauses with a maximum length of XXX.

Proof. Consider the following figure:

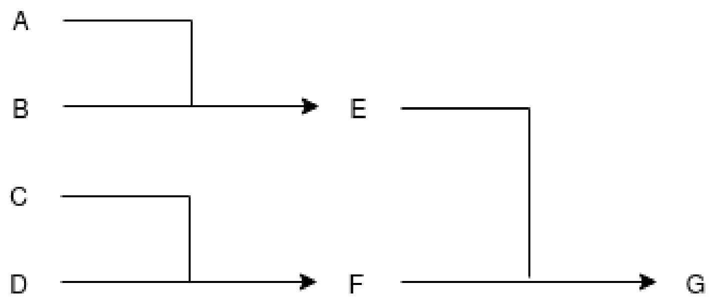

Fig. 3. An illustration of the derivations described in the lemma

Let the clauses be defined as follows:

$A := [\alpha]$

$B := [\beta]$

$C := [\delta]$

$D := [\phi]$

$E := [\alpha \cup \beta - \{i, -i\}]$

$F := [\delta \cup \phi - \{j, -j\}]$

$G := [\alpha \cup \beta \cup \delta \cup \phi - \{i, -i\} - \{j, -j\} - \{m, -m\}]$

Where the following terms exist in the following sets

- $m \in \alpha \cup \beta - \{i, -i\}$
- $-m \in \delta \cup \phi - \{j, -j\}$
- $i \in \alpha$
- $-i \in \beta$
- $j \in \delta$
- $-j \in \phi$

Couple cases for m and $-m$, I do not think this matters:

(1) m is in either α or β
(2) m is in both α and β
(3) $-m$ is in either δ or ϕ
(4) $-m$ is in both δ and ϕ

Let's say $m \in \alpha$ and $-m \in \delta$, then $-m \notin \beta$ and $m \notin \phi$.

And a new clause can be derived from A and C using Lemma 5.9:

$H := [\alpha \cup \delta - \{m, -m\}]$

And a new clause can be derived from H and B using Lemma 5.9:

$I := [\alpha \cup \delta \cup \beta - \{m, -m\} - \{i, -i\}]$

Notice G can be derived by using H and D with Lemma 5.9.

What can be said about $|H|$ and $|I|$ wrt to $|G|$?

Consider $|H|$, notice it is 4 greater than $|G| - |\beta'| - |\phi'|$

Where β' is the set of terms in β that do not exist in α or ϕ and ϕ' is the set of terms in ϕ that do not exist in α or ϕ

$A := [1, 2, 3]$ $B := [-1, 4, 5]$ $C := [6, -2, 8]$ $D := [-6, 9, 10]$ $E := [2, 3, 4, 5]$ $F := [-2, 8, 9, 10]$ $G := [3, 4, 5, 8, 9, 10]$ $H := [1, 3, 6, 8]$ $I := [3, 4, 5, 6, 8]$ $G := [3, 4, 5, 8, 9, 10]$

mimimize β' and ϕ'

$A := [1, 2, 3]$ $B := [-1, 9, 10]$ $C := [6, -2]$ $D := [-6, 9, 10]$ $E := [2, 3, 9, 10]$ $F := [-2, 9, 10]$ $G := [3, 9, 10]$ $H := [1, 3, 6]$ $I := [3, 6, 9, 10]$ $G := [3, 9, 10]$

Hmmm, $|I| > |G|$

□

Lemma 5.18. *Given the following:*

- *a clause, A, of length less than k*
- *a clause, B, of length less than k*
- *a clause, C, of length less than k*
- *a clause, D, of length k*
- *a clause E, of length k*
- *a clause F, of length $k - 1$ or k*
- *A and B imply D by Lemma 5.9*

- *C expands to E by Lemma 5.8*
- *D and E imply F by Lemma 5.9*

Then F can be implied by only processing clauses with a maximum length of $k - 1$.

Proof. Consider the following figure:

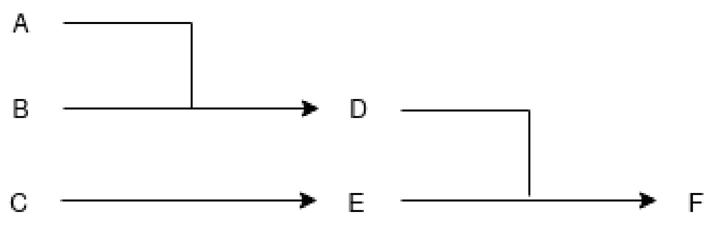

Fig. 4. An illustration of the derivations described by the lemma

Let the following clauses be defined:

$A := [a, b, \beta, i]$

$B := [c, d, \delta, -i]$

$C_1 := [-a, f, \phi]$

$C_2 := [e, f, \phi]$

Then the following clauses are implied:

$D = [a, b, \beta, c, d, \delta]$

$E_1 = [-a, f, \phi, g, h, \gamma]$

$E_2 = [e, f, \phi, -a, h, \gamma]$

$F_1 = [b, \beta, c, d, \delta, f, \phi, g, h, \gamma]$

$F_2 = [b, \beta, c, d, \delta, e, f, \phi, h, \gamma]$

Where β, δ, ϕ and γ are generic sets of terms such that A, B, and C block at least one clause by Lemma 5.3.

Notice E and D must share a term of the opposite form and there are two possible cases for what this term is:

- it exists in C (use C_1, E_1, and F_2)
- it does not exist in C (use C_2, E_2, and F_2)

Either way the opposite term must exist in A or B since it must exist in D and D is composed of terms from A or B.

Since A and B are both generic clauses treated in the same way, it does not matter which clause the term exists in as long as it is fixed.

Let's pick A to contain the opposite form term from C.

Consider (1) the opposite form term exists in C (use C_1, E_1, and F_1).

Want to show F_1 can be derived by processing clauses with a maximum length of k - 1.

Let the following clauses be defined:

$G = [b, \beta, i, f, \phi]$ (By clause A and C_1 with Lemma 5.9)

$H = [b, \beta, f, \phi, c, d, \delta]$ (By clause G and B with Lemma 5.9)

Since all of the terms in H exist in F_1, Lemma 5.8 can be used to derive F_1 from H.

Want to show (1a) G is shorter than k and (1b) H is shorter than k

(1a) G is shorter than k

Consider two cases (1ai) F_1 is of length k and (1aii) F_1 is of length $k-1$.

(1ai) F_1 is of length k

In the following equations, let the presence of a term represent a count of one, and the presence of a set of terms represent the number of terms in that set. If multiple sets of terms are shown in parentheses, let this represent the number of terms found in the intersection of both sets.

Recall F_1 is of length k so k can be defined as follows:

$k = b+c+d+f+g+h+\beta+\delta+\phi+\gamma-(\beta\delta)-(\beta\phi)-(\beta\gamma)-(\delta\phi)-(\delta\gamma)-(\phi\gamma)+(\beta\delta\phi)+(\beta\delta\gamma)+(\beta\phi\gamma)+(\delta\phi\gamma)-(\beta\delta\phi\gamma)$

Length of $G = b+i+f+\beta+\phi-(\beta\phi)$

Want to show length of G is less than k:

$b+i+f+\beta+\phi-(\beta\phi) < b+c+d+f+g+h+\beta+\delta+\phi+\gamma-(\beta\delta)-(\beta\phi)-(\beta\gamma)-(\delta\phi)-(\delta\gamma)-(\phi\gamma)+(\beta\delta\phi)+(\beta\delta\gamma)+(\beta\phi\gamma)+(\delta\phi\gamma)-(\beta\delta\phi\gamma)$

$\rightarrow i < c+d+g+h+\delta+\gamma-(\beta\delta)-(\beta\gamma)-(\delta\phi)-(\delta\gamma)-(\phi\gamma)+(\beta\delta\phi)+(\beta\delta\gamma)+(\beta\phi\gamma)+(\delta\phi\gamma)-(\beta\delta\phi\gamma)$

Which is true as long as at least two terms exist on the R.H.S.

Want to show at least two terms exist on the R.H.S.

Suppose not, then at most one term exists on the R.H.S.

Recall the clauses

$A := [a, b, \beta, i]$

$D := [a, b, \beta, c, d, \delta]$

We can redefine some clauses since at most one term may exists on the R.H.S.

$D := [a, b, \beta, x]$

Where x represents one term from c, d, or δ.

Note that if x was more than one term, the two terms could be extracted and treated as c and d, but we know at most one of these terms exist.

Note the length of D is the same as the length of A.

This is a contradiction because the length of A is given as less than k while the length of D is given as k.

Therefore at least two terms exist on the R.H.S. and the inequality is true.

Therefore G is shorter than k when using F_1 and the length of F_1 is k.

(1aii) F_1 is of length $k-1$

Recall F_1 is of length $k-1$ so k can be defined as follows:

$k = b+c+d+f+g+h+\beta+\delta+\phi+\gamma-(\beta\delta)-(\beta\phi)-(\beta\gamma)-(\delta\phi)-(\delta\gamma)-(\phi\gamma)+(\beta\delta\phi)+(\beta\delta\gamma)+(\beta\phi\gamma)+(\delta\phi\gamma)-(\beta\delta\phi\gamma)+1$

Length of $G = b+i+f+\beta+\phi-(\beta\phi)$

Want to show length of G is less than k:

$b+i+f+\beta+\phi-(\beta\phi) < b+c+d+f+g+h+\beta+\delta+\phi+\gamma-(\beta\delta)-(\beta\phi)-(\beta\gamma)-(\delta\phi)-(\delta\gamma)-(\phi\gamma)+(\beta\delta\phi)+(\beta\delta\gamma)+(\beta\phi\gamma)+(\delta\phi\gamma)-(\beta\delta\phi\gamma)+1$

Similarly as before, the inequality will become:

$\rightarrow i < c+d+g+h+\delta+\gamma-(\beta\delta)-(\beta\gamma)-(\delta\phi)-(\delta\gamma)-(\phi\gamma)+(\beta\delta\phi)+(\beta\delta\gamma)+(\beta\phi\gamma)+(\delta\phi\gamma)-(\beta\delta\phi\gamma)+1$

Which is true as long as at least one term exists on the R.H.S.

We already showed at least two terms exist on the R.H.S. and the proof does not rely on the length of F_1.

Therefore at least one term exists on the R.H.S. and the inequality is true.

Therefore the length of G is less than k when using F_1 and the length of F_1 is $k-1$.

(1b) Want to show H is shorter than k.

We have two cases to consider: (1bi) F_1 is of length k and (1bii) F_1 is of length $k-1$

Consider (1bi) F_1 is of length k

Recall F_1 is of length k, so we can define k as follows:

$k = b+c+d+f+g+h+\beta+\delta+\phi+\gamma-(\beta\delta)-(\beta\phi)-(\beta\gamma)-(\delta\phi)-(\delta\gamma)-(\phi\gamma)+(\beta\delta\phi)+(\beta\delta\gamma)+(\beta\phi\gamma)+(\delta\phi\gamma)-(\beta\delta\phi\gamma)$

Length of $H = b+f+c+d+\beta+\delta+\phi-(\beta\delta)-(\beta\phi)-(\delta\phi)+(\beta\delta\phi)$

Want to show the length of H is less than k.

$b+f+c+d+\beta+\delta+\phi-(\beta\delta)-(\beta\phi)-(\delta\phi)+(\beta\delta\phi) < b+c+d+f+g+h+\beta+\delta+\phi+\gamma-(\beta\delta)-(\beta\phi)-(\beta\gamma)-(\delta\phi)-(\delta\gamma)-(\phi\gamma)+(\beta\delta\phi)+(\beta\delta\gamma)+(\beta\phi\gamma)+(\delta\phi\gamma)-(\beta\delta\phi\gamma)$

$\rightarrow b+f+c+d < b+c+d+f+g+h+\gamma-(\beta\gamma)-(\delta\gamma)-(\phi\gamma)+(\beta\delta\gamma)+(\beta\phi\gamma)+(\delta\phi\gamma)-(\beta\delta\phi\gamma)$

$\rightarrow 0 < g+h+\gamma-(\beta\gamma)-(\delta\gamma)-(\phi\gamma)+(\beta\delta\gamma)+(\beta\phi\gamma)+(\delta\phi\gamma)-(\beta\delta\phi\gamma)$

Which is true if at least one term exists on the R.H.S.

Want to show at least one term exists on the R.H.S.

Suppose not, then no terms exist on the R.H.S.

Recall the clauses

$C_1 := [-a, f, \phi]$

$E_1 := [-a, f, \phi, g, h, \gamma]$

Since no terms exist on the R.H.S., we can redefine some clauses:

$E_1 := [-a, f, \phi]$

Note that no other terms may exist in E_1 because any other terms could be used as g or h, but we know these don't exist.

Notice E_1 is exactly C_1.

This is a contradiction because the length of C is given as less than k while the length of E is given as k.

Therefore at least one term exists on the R.H.S. and the inequality is true.

Therefore H is shorter than k when using F_1 and the length of F_1 is k.

Consider (1bii) F_1 is of length $k-1$

Since F_1 is of length $k-1$, we can define k as follows:

$k = b+c+d+f+g+h+\beta+\delta+\phi+\gamma-(\beta\delta)-(\beta\phi)-(\beta\gamma)-(\delta\phi)-(\delta\gamma)-(\phi\gamma)+(\beta\delta\phi)+(\beta\delta\gamma)+(\beta\phi\gamma)+(\delta\phi\gamma)-(\beta\delta\phi\gamma)+1$

Length of $H = b+f+c+d+\beta+\delta+\phi-(\beta\delta)-(\beta\phi)-(\delta\phi)+(\beta\delta\phi)$

Want to show the length of H is less than k.

$b+f+c+d+\beta+\delta+\phi-(\beta\delta)-(\beta\phi)-(\delta\phi)+(\beta\delta\phi) < b+c+d+f+g+h+\beta+\delta+\phi+\gamma-(\beta\delta)-(\beta\phi)-(\beta\gamma)-(\delta\phi)-(\delta\gamma)-(\phi\gamma)+(\beta\delta\phi)+(\beta\delta\gamma)+(\beta\phi\gamma)+(\delta\phi\gamma)-(\beta\delta\phi\gamma)+1$

Similarly to before, this will derive

$\rightarrow 0 < g+h+\gamma-(\beta\gamma)-(\delta\gamma)-(\phi\gamma)+(\beta\delta\gamma)+(\beta\phi\gamma)+(\delta\phi\gamma)-(\beta\delta\phi\gamma)+1$

Which is always true.

Therefore the length of H is less than k when using F_1 and the length of F_1 is $k-1$.

Consider (2) the opposite form term does not exist in C, use C_2, E_2, and F_2.

Recall we have the clauses:

$C_2 := [e, f, \phi]$

$F_2 := [b, \beta, c, d, \delta, e, f, \phi, h, \gamma]$

Notice all of the terms in C_2 exist in F_2 so C_2 can expand to F_2 using Lemma 5.9.

Since the length of C is given as less than k, we can derive F_2 by processing clauses with a maximum length of $k-1$.

Since F_1 and F_2 are all possible cases of F and they can be derived without processing clauses longer than length $k-1$, F can be derived by processing only clauses with a maximum length of $k-1$. □

Lemma 5.19. *Given the following:*

- *a clause A*
- *a clause B*
- *a clause C*
- *a clause D*
- *a clause E*
- *A expands to C by lemma 5.8*
- *B expands to D by lemma 5.8*
- *C and D imply E by lemma 5.7*

Then E can be derived by processing clauses in the range max($|A|$, $|B|$) - 1 to $|E|$.

Proof. Consider the following figure:

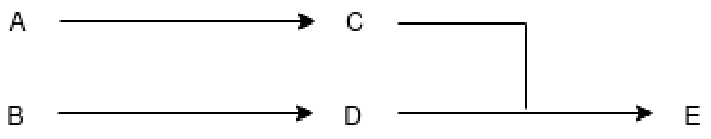
$$\begin{tikzcd} A \arrow[r] & C \arrow[dr] & \\ B \arrow[r] & D \arrow[r] & E \end{tikzcd}$$

Fig. 5. An illustration of the derivations described in this lemma

In the following clause definitions let β, δ, ϕ, and γ be fixed yet arbitrary set of terms in which the clauses block at least one assignment by Lemma 5.3.

In order for C and D to imply E by Lemma 5.9, they have to share a terminal which is positive in one clause and negated in the other.

Note that C is composed of terms in A and terms not in A.

Similarly D is composed of terms in B and terms not in B.

There are 3 cases for this opposite form term existing in regards to A and B:

(1) the opposite form term does not exist in A or B
(2) the opposite form term exists in either A or B but not both
(3) the opposite form term exists in A and B

Let the clauses be defined as follows:

$A := [\alpha]$

$B := [\beta]$

$C := [\alpha \cup \delta]$

$D := [\beta \cup \phi]$

$E := [\alpha \cup \delta \cup \beta\phi - \{i, -i\}]$

Consider case (1) the opposite form term does not exist in A or B

Then $i \notin \alpha$ and $-i \notin \beta$.

Notice then all of the terms in A exist in E so A can be expande to imply E by Lemma 5.8.

So E can be derived by processing clauses with a maximum length of $|E| - 1$ or if $A = E$ then E already exists when A exists.

Consider case (2) the opposite form term exists in either A or B, but not both

Then $i \in \alpha$ or $-i \in \beta$, but not both.

Let's say $i \in \alpha$, then $-i \notin \beta$.

Notice then all of the terms in B exist in E so B can be expanded to E by Lemma 5.8.

So E can be derived by processing clauses with a maximum length of $|E| - 1$ or if $B = E$ then E already exists when B exists.

Consider case (3) the opposite form term exists in both A and B

Then $i \in \alpha$ and $-i \in \beta$.

A new clause can be implied,

$F := [\alpha \cup \beta - \{i, -i\}]$

Which contains all of the terms in E so F can expand to E by Lemma 5.8.

If $F \neq E$, then F is shorter than E and E can be derived by processing clauses with a maximum length of $|E| - 1$

If $F = E$, then F is derived by processing A and B. □

6 ALGORITHM

```
(1) For each clause in the instance, C, of length 3 or less:
  (a) For each clause in the instance, D, of length 3 or less:
    (i) Get all clauses implied by C and D according to Lemma 5.9 and add them to the instance
    (ii) Check if this new clause is in the instance and update a flag accordingly
  (b) Expand C to get all possible clauses with a maximum length of 3 and add them to the instance
  (c) For each new clause from the last step
    (i) Check if this new clause is in the instance and update a flag accordingly
(2) For each clause in the instance, E, of length 1:
  (a) For each clause in the instance, F, of length 1:
    (i) if E and F contain the same terminal in which it is positive in one clause and negated in the other,
        the clauses are contradicting and the instance is unsatisfiable, end
(3) Repeat (1)-(2) until no new clauses are added
(4) If it reaches here, the instance is satisfiable, end
```

7 TIME COMPLEXITY ANALYSIS

In this section I will analyze the time complexity of the algorithm in section 4

(1) - $O(n^3)$ - At most $\binom{n}{3} * 8 + \binom{n}{2} * 4 + \binom{n}{1} * 2$ clauses to iterate which is on the order of $O(n^3)$

(1.a) - $O(n^3)$ - At most $\binom{n}{3} * 8 + \binom{n}{2} * 4 + \binom{n}{1} * 2$ clauses to iterate which is on the order of $O(n^3)$

(1.a.i) - $O(1)$ - For each terminal in C, check if it's opposite form is in D. Since each clause is of length 3 or less, the worst case we iterate over 3 terms in C and check each term in D. If it's a match, we iterate over each clause again and create a new clause as described in Lemma 5.9. This time complexity is intuitively $O(3^2 + 3^2)$ which is constant time.

(1.a.ii) - $O(n^3)$ - For each clause in the instance, iterate through the instance to check if it exists already. There are on the order of $O(n^3)$ clauses in the instance.

(1.b) - $O(n^2)$ - For a 2-terminal clause, there are $2 * n$ possible 3-terminal clauses that could be expanded to. For a 1-terminal clause, there are $4 * n^2$ possible 3-terminal clauses that could be expanded to. This is upper bounded by the latter case.

(1.c) - $O(n^2)$ - Upper bounded by at most $O(n^2)$ new clauses from the last step

(1.c.i) - $O(n^3)$ - At most $\binom{n}{3} * 8 + \binom{n}{2} * 4 + \binom{n}{1} * 2$ clauses to iterate which is on the order of $O(n^3)$

(2) - $O(n^3)$ - Iterating through an instance where there are on the order of $O(n^3)$ clauses

(2.a) - $O(n^3)$ - Iterating through an instance where there are on the order of $O(n^3)$ clauses

(2.b.i) - $O(1)$ - Constant time to check if two 1-terminal clauses contain the same terminal in the opposite form

(3) - $O(n^3)$ - Worst case, we add one new clause each time so we have to loop $\binom{n}{3} * 8 + \binom{n}{2} * 8 + \binom{n}{1} * 2$ times which is on the order of $O(n^3)$

(4) - $O(1)$ - Constant time to check and return satisfiable

The time complexity breaks down:

$(3) * ((1) * ((1.a) * ((1.a.i) + (1.a.ii)) + (1.b) + (1.c) * (1.c.i)) + (2) * (2.a) * (2.a.i)) + (4)$

It is seen the most computationally expensive steps are

$(3) * (1) * (1.a) * (1.a.oii)$

$= O(n^3) * O(n^3) * O(n^3) * O(n^3)$

$= O(n^{12})$

8 PROOF OF CORRECTNESS

Want to show an instance is unsatisfiable iff we can derive contradicting 1-terminal clauses by the algorithm.

WTS Contradicting 1-terminal clauses can be derived $\implies$ the instance is unsatisfiable

Contradicting 1-terminal clauses take the form:

$[a]$

$[-a]$

Where a is a terminal in the problem.

In all possible assignments, a can have the value of True or False.

If a is True, the clause $[-a]$ will be False and the assignment does not satisfy the instance. If a is False, the clause $[a]$ will be False and the assignment does not satisfy the instance.

Since all possible assignments do not allow both clauses to be true, the instance is unsatisfiable.

Therefore contradicting 1-terminal clauses can be derived $\implies$ the instance is unsatisfiable

Want to show an unsatisfiable instance $\implies$ the algorithm will derive contradicting 1-terminal clauses

By Lemma 5.14, since the instance is unsatisfiable, the given 3-terminal clauses can be expanded to every possible n-terminal clause.

By Lemma 5.15 these n-terminal clauses can be reduced by Lemma 5.7 to derive contradicting 1-terminal clauses.

So we know if we can derive these n-terminal clauses, we can derive contradicting 1-terminal clauses.

The idea behind the proof is that we know the n-terminal clauses can be used to derive the 1-terminal via reduction but we'll show that we don't ever have to process a clause above length 3 to derive these contradicting 1-terminal clauses.

This relies on the fact that all clauses of length 4 or greater, say of length k, have to be derived by shorter clauses and we can process the shorter clauses to derive any clauses that the clauses of length k would derive.

We can do this without processing a clause of length k or greater.

The way in which we derive these 1-terminal clauses is we use Lemma 5.7 to reduce the n-terminal clauses to $(n-1)$-terminal clauses which are reduced to $(n-2)$-terminal clauses which are reduced to ... which are reduced to 2-terminal clauses and which finally get reduced to 1-terminal clauses.

Notice this passes through every possible k from length 2 to n.

Recall that deriving all of the n-terminal clauses was done by using Lemma 5.8.

These n-terminal clauses were then used to derive clauses of length $(n-1)$ by Lemma 5.7.

By Lemma 5.18, such a case allows us to derive the clauses of length $(n-1)$ without ever having to process a clause of length n.

Now we have all of the clauses of length $(n-1)$ that we would have derived if we processed clauses of length n.

Note how each of these clauses are either derived from given clauses by Lemma 5.8 or by Lemma 5.9 (all Lemma 5.7 derivations are a subset of all Lemma 5.9 derivations).

We now want to use these $(n-1)$-terminal clauses to derive clauses of length $(n-2)$, but we want to do it without processing clauses whose length is greater than $(n-2)$.

Since it takes two $(n-1)$-terminal clauses to derive a $(n-2)$-terminal clause by Lemma 5.7, the possible clauses could be of the form:

(1) both clauses were derived by Lemma 5.8 (expansion)
(2) both clauses were derived by Lemma 5.9 (reduction/implication)
(3) each clause was derived in a different manner

Note that this list is exhaustive because these are the only manner of implications used in the lemmas that allowed us to derive these clauses.

We can use the following lemmas to handle each case:

(1) Lemma 5.19
(2) Lemma 5.17
(3) Lemma 5.18

As such we can derive the clauses of length $(n-2)$ without processing a clause whose length is greater than $(n-2)$.

In a more general sense, for any implied clause of length k, say C, that is used to derive a clause of length k or $k-1$, say D, then we can use the fact that C is derived by shorter clauses and we can use these shorter clauses to directly derive D without ever processing a clause of length k or greater.

At this point, we have 4-terminal clauses and we want to derive 3-terminal clauses.

Notice that Lemmas 5.17, 5.18, and 5.19 require the input clauses to be derived so we cannot use those lemmas to derive all the clauses of length 3 we need.

We need two input clauses to derive the necessary 3-terminal clauses and we have three cases:

(1) Both inputs are of length 4
(2) One input is of length 4, the other is of length 3
(3) Both inputs are of length 3

We can disregard the last point because we want to derive a 3-terminal clause without processing a clause of length 4 or greater so the claim is vacuously true in this case.

In the case where both inputs are of length 4, we know they are both derived using smaller clauses and we can use Lemmas 5.17, 5.18, or 5.19.

In the case where one input is of length 4 and the other is of length 3, there are two cases for how the 4-terminal clause was derived:

(1) It was derived using Lemma 5.9 (reduction/implication)
(2) It was derived using Lemma 5.8 (expansion)

We can handle the cases in the following ways:

(1) Using Lemma 5.11
(2) Using Lemma 5.12

Now we have all the 3-terminal clauses that would have been derived while the n-terminal clauses were being reduced to contradicting 1-terminal clauses.

We can now reduce the 3-terminal clauses to 1-terminal clauses.

Since we have all the necessary 3-terminal clauses without having to process a clause of length 4 or greater, this shows we do not have to process any clauses of 4 or greater to derive contradicting 1-terminal clauses.

Even though the algorithm only explicitly uses Lemma 5.9 and Lemma 5.8, this will cover cases where reduction is needed because Lemma 5.9 is a more general case of Lemma 5.7. Notice, too, that Lemmas 5.11, 5.12, 5.17, 5.18, and 5.19 rely on Lemmas 5.8 and 5.9 so we do not have to explicitly capture the cases where the intermediate lemmas would apply.

Therefore, this coincides with the described algorithm.

9 CONCLUSION

In this paper, we present an algorithm to solve 3SAT in polynomial time.

This algorithm relies on Lemmas 5.8 and 5.9. The former of which says a clause can expand to imply another clause by appending any term that's not in the original clause. The latter of which says two clauses sharing one terminal which is positive in one clause and negated in the other can imply a new clause composed of the rest of the terms in either clause. Using these strategies, we can process an instance of

3SAT while only considering clauses of length 3 or less and are guaranteed to derive a pair of contradicting 1-terminal clauses if and only if the instance is unsatisfiable.

According to [2], 3SAT is NP-complete so if 3SAT can be solved in polynomial time then every problem in NP can be solved in polynomial time.

Since such an algorithm exists, all problems in NP can be solved in polynomial time.

Thus, P = NP.

REFERENCES

[1] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.

[2] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.