On A "Quantum" Proof Of The Equivalence of P And NP

Adewale Oluwasanmi

waleoluwasanmi@yahoo.com

Abstract

We present an algorithm, along with a correctness proof, for solving the 3 Satisfiability problem that is inspired by quantum mechanical principles and that runs in polynomial time with respect to the size of the input problem. Even though we term both our algorithm and its associated proof as quantum (for reasons which we will demonstrate), it is intended to be run on standard classical computing (Turing complete) architecture. In the article, we posit that the 3 Satisfiability problem has an intrinsic complex quantum form that can be programmed in order to build a model of the solution space for satisfiable instances or show that such a model cannot be constructed. This yields surprising results on the ability for classical systems to abstractly simulate general quantum systems.

1. Introduction

The 3 Satisfiability problem is a popular problem in the field of computer science notable for being one of the quintessential problems in the NP complete space. The full classification of the NP space as well as the question of whether P = NP had the earliest developments in the combined works of Cook [2], Levin [3] and Karp [4]. In this article, we continue that analysis by providing a definite answer to this question in the form of a "quantum" algorithm that solves the 3 Satisfiability problem in polynomial time. Our quantum dynamical approach is unique and has the closest analogue in existing literature to the holographic algorithms investigated by Valiant [1] and others. Similar to the algebraic methods (but with a more directly functional approach) used in [1], in the course of our proof, we construct "holographic" reductions to quantum valued 2 dimensional "planar" objects which form the real (directly match-able) critical points in all assignment paths. As also conjectured/suggested by Valiant [1], these techniques provide a complete, if algorithmically convoluted/complex means, to resolve the P/NP issue.

In this work, we do not treat the Satisfiability problem in its regular logical form as a conjunctive normal formula over disjunctive unit clauses. We instead cast the problem into a novel quantum computational form or "type". This treatment converts the initial propositional formula into a quantum programmable type (strengthening the propositions as types conception) whose particular type is algebro-combinatorial (an idea to be fleshed out in detail). This assumes that the reader is familiar, if needed, with the normal classically logical formulations of the problem in CNF form as well as how the maximal operational space of that formulation classifies the problem as NP complete in the literature.

The approach we present is intended to address the major source of uncoordinated exponential blow up or “drift towards infinity” in the classical approach, namely that attributable to the worst-case element of brute force search. To address this, we propose an alternative, “renormalizable” model where the original clauses are converted to a discrete solution form and the totality of potential and actual solution spaces are modelled as sets of “evolutions” of complex combinatorial species along with an associated algebra of mutation/generator functions which extend the species by operating over their interference patterns.

The algebraic species are quantum “compressions” or representations of partial or complete solution spectra (if any solutions exist) and the generator functions construct complex generations of this space. In the case of unsatisfiable instances and instances with just one solution, the initial generation, termed first generation, is deemed complete. For instances with more than one solution, first generations serve as a complex differentiable base case over which other generations can be computed. All successive species constructed after the initial generation are collectively termed second generation species. Actual sets of satisfying assignments to unit propositions can be regarded as output/base species (that is they are of the same type, species, as the clausal space in our formulation and are complex homeomorphic with other species – if we regard all species as quantum deformable functions, with unit assignments as invertible constant functions as will be demonstrated).

We have used the word complex in many places in a similar sense to that in other mathematical literature, that is, a complex entity (function) is capable of splitting into two or more other complex or real entities, by configuration, based on the (bounded, imaginal) action of the user. Here we associate the sense “real” to objects identifiable with lines or linear actions which for example would be the linear assignment of truth values to individual variables in our provided SAT formula.

The article is divided into 2 further sections as follows:

2. The Algorithm.
3. Conclusion.

The overall style pursued is functional in spirit as opposed to formal since our approach is directly algorithmic and relies less directly on formal numerical methods established in more rigorous mathematical literature (for example, [1]). Hence, our main instruments of proof will be functional descriptions of structures (all algebraic) and the functions that operate on these structures along with arguments to prove the correctness (soundness and completeness) of the entire procedure. A main requirement will be for the reader to be able to logically follow along and understand these (algorithmic/functional) structures, the functions/operations that change them and the (further) structures implied from applying the operations.

2. The Algorithm

2.1 Quantum Programs

We construe our algorithm as manipulating quantum programs, hence the term, quantum algorithm. These quantum programs, which we also term quantum injunctions, can be taken to be the programmatic equivalents of standard quantum computational elements, say qubits or qudits, that is,

they are morphable and their morphisms can only take on standard quantum values. Ignore the fact that these programs themselves are the problems to be solved, that is, they extend classical quantum computation with the notion of self-solving quantum systems. This makes the programs a form of abstract quantum matter which, with some ingenuity, can be used to study other (physically concrete) quantum systems. Note that because we are dealing with a virtual form, these morphisms will always take on their desired quantum values, that is, we have no probabilistic element in the orientation or "spin" of the programs, providing an abstract and absolute form of topologically secure quantum computation. One can say, that by capturing and bounding the entire possibility space, we "encircle" the traditional probability layer of quantum mechanics.

2.2 Unit Variables and Literals

Unit literals represent traditional propositions that can be assigned true or false. Each literal presented can be identified by a subscript (natural number) and sign (+ for unnegated literals and – for negated literals). For brevity sake, we only show signs for negated (-) literals and assume that unsigned numbers are positively signed. Every literal and its opposite (inverse) are antipodes of the same variable (simply identified by the lower letter "x" attached to their common subscript) which is their algebraic "superposition". The signed representations (literals) are the only elements directly available to functional manipulation but we will continue to use the term "variable" to refer to the abstract sense of assignment so we can talk of assigning a variable, say x_3, to one of its assignable literals, 3 or -3 , that is, we make that particular literal true and the opposite literal (same variable, opposite sign), false. The selected notation for literals is similar in spirit to the DIMACS CNF format used by many classical SAT solvers.

2.3 Basic Quantum Program Structures

Each initial disjunctive clause is represented by a Quantum Injunctive Clause, which we will call an X structure. This is a comma separated list of literal values, where the integer part represents the variable subscript and the sign is the OPPOSITE of that carried by the literal in normal CNF format, nested within opening and closing braces. For example:

Clausal form $(\neg x_1 \vee x_4 \vee \neg x_7)$ is transformed to X structural form {1, -4, 7} and

Clausal form $(x_3 \vee \neg x_5 \vee x_6)$ is transformed to X structural form {-3, 5, -6}.

We will call all X structures that directly represent the clauses in the original formula, the BASIS X structures. We will shortly see how this basis is extended to compute the complete satisfiability model for the provided formula instance.

The entire CNF formula itself is represented by a different structure termed a Y structure which is a comma separated list of all (valid) X structures. We will shortly explain the notion of a valid X structure. This gives Y structures a richer, computationally effective (functional versus formulaic) overall structure. For example:

The formula $(\neg x_1 \vee x_4 \vee \neg x_7) \wedge (x_3 \vee \neg x_5 \vee x_6)$ is transformed to Y structural form {1, -4, 7}, {-3, 5, -6}.

The last class of structures, Z structures, are the actual satisfying solutions to some problem instance S, that is, a provided CNF formula. A Z structure is represented as a comma separated open list (no opening or closing braces) of assignments (literals that have been made true). For example:

> The Y structure: {1, -4, 7}, {-3, 5, -6} is satisfied by the Z structure: -1, -4, 7, 3, 5, -6.

Note how in our example, only the first literal in each Y structure has its sign reversed in the Z structure and is the only assignment that satisfies the respective underlying disjunctive clause (because, as we have explained, X structures reverse the sign).

Signs are reversed in X structures for the following reason: Computationally, X structures can then be interpreted as declared conditional (functional) programs for which the following axiom must be satisfied:

> **Axiom 1**: "For some X structure, X1, composed of n distinct elements and for any program P1 with an attached, partially completed Z structure, Z1, that satisfies (must satisfy) the given instance S containing X1, if n-1 of the literals in X1 have already been added to Z1, that is, these n -1 variables in Z1 have already been assigned to literals which do not satisfy the classical disjunctive clause represented by X1, the program P1 must immediately assign the nth, unassigned literal, call it k, in X1 to its opposite Thus, we extend Z1 to a new Z structure, Z2, that satisfies X1. We call Z2 a satisfying completion of X1 over k".

For example:

> The Y structure: {1, -4, 7}, {-3, 5, -6} has the partially completed Z structure: -1, -4, 7, 5 at some time t and at the successive time t + 1 we extend the solution to -1, -4, 7, 5, -6 by adding -6.
>
> Note that the second X structure is unsatisfied at the new time t + 1 and that we must immediately at what we call time t + 2, add 3 (the opposite of -3) to the Z structure, obtaining the satisfying solution: -1, -4, 7, 3, 5, -6.
>
> At time t + 2, we can regard the previous n–1 unsatisfying literal assignments as the INPUT portion for the inclusion of the satisfying literal assignment (the OUTPUT) of the anonymous function locally defined (in space and time) on the X structure. This qualifies us to call each structure X of size n, an n-dimensional program encoding n functions of input size n-1 and output size 1.

Axiom 1 treats X structures as algebraic constraint types from which we can construct a new type via pattern matching. Since Y and Z structures vary along with X structures, they can be regarded as algebraic types (varieties) themselves. Speaking algebraically then, we think it worthwhile to make the following section closing statements by speaking of the "cycle" structure of the 3 algebraic types:

> Every completed satisfying program P1 outputs a permuted/chained variation of a permutable set of assignments C (that is, the elements of C should be assignable in any order). We say each X structure carries a partial cycle on C chains.
>
> Every X structure of size n that carries a cycle on some combination C, over some output literal j, also carries a (exactly n-1) cycles on other C structures for which the negation (contra-variation)

of j is an input assignment. Thus, every X structure is a co-cycle over its n variants, that is, every X1 structure is a partial co-cycle over its n associated C structures. We will soon see how X structures can be regarded as projective varieties.

On the other hand, Y structures must necessarily carry a total co-cycle over every possible combination (where solutions exist), at every instance. We will soon see how Y structures can be regarded as affine (parallel connection over their manifold representation) varieties.

Z structures are just particular chain realizations of some covered combination C. We will soon see how Z structures can be regarded as smooth (continuously varying real/rational entities) varieties.

2.4 Injunctive Space Completion

Here, we introduce an additional axiom and its corollary that both define a partial binary "integration" operation over X structures.

Axiom 2: "Take any 2 valid (again, we will explain invalidation rules shortly) X structures, X1 of size m and X2 of size n. If X1 and X2 share a common variable x_i, such that x_i takes on opposite signs (literals) in both structures AND X1 and X2 share no other variables, DIFFERING IN SIGN, we must generate and add to the global Y structure, a new X structure X3, composed of the m-1 elements in X1 and n-1 elements in X2, differing from x_i".

This can be read more procedurally as: "If Y is to contain only valid X structures, then it must admit only X structures that do not lead to assignment contradictions when applied in parallel since by Axiom 1, if we do not construct X3, x_i will be assigned two contradictory values if it is the last value assigned in both X1 and X2 by some arbitrary program P". Thus, Axiom 2 defines a binary parallelization/synchronization operator on our X structural programs.

Corollary 2.1: Self Contradiction/Reduction: "Take 2 structures X1 and X2 both of size n having the same set of variables and in which n-1 of the variables are the same and agree in sign (same literals) and also having an nth common variable that differs in sign (opposite literals). We must generate a new structure X3, composed of the agreeing n-1 literals. X1 and X2 are now invalid as they have been replaced/reduced by X3, a valid substructure of each.".

2.5 First-Generation Algorithm

i. Take a given 3 SAT problem instance S, convert all of its clauses into X structures and add them to a list called "incoming". These initial set forms the basis.

ii. Create a new list called "processed" to add structures that have been picked up from "incoming" and processed as described in step iii.

iii. Process elements of "incoming" in a loop, and for each iteration:
Remove the first "incoming" structure X1 and compare it to each applicable element in "processed" using Axiom 2.
If we generate a new X structure during a comparison with 5 OR FEWER elements, add it to "incoming", else discard it (NOTE THIS STEP).
Add X1 to "processed.

Continue until “incoming” is empty.

iv. If at any point, we generate two X structures X1 and X2 each of size 1 and each containing the same variable but where each variable is the literal opposite of the other, X1 and X2 are invalid and S is unsatisfiable, otherwise, if we exhaust “incoming” and no such occurrences are recorded, S is satisfiable.

2.6 Proof of Correctness

It should be quite obvious from the description above that we are dealing with a polynomial time algorithm (all considered X structures are bounded by a finite size of 5) with no immediately evident linear reason why the procedure is correct. The best way to prove the correctness of this algorithm will be to examine its edge cases.

First, we must agree that if we complete our first generation such that by our criteria above, the problem instance S is satisfiable (to be proven), then all valid X structures describe necessary actions that must be taken at “boundaries” by any satisfying program P according to Axiom 1.

Secondly, during second generations, as P assigns variables to literals, any X structure can be rewritten/rescaled with a structure reflecting all completed assignments. For example:

Say we have some X structure {3, -5, -7}.

If we assign the variable x_5 to false, that is, our Z structure contains -5, we can perform the following rewrite:

{3, -5, -7} to {3, -7}.

This ensures that the X structure indicates the new exact boundary rule that satisfies the underlying disjunctive clause.

This is what we mean by X structures being quantum computational elements that can be reprogrammed by taking a quotient on a structure to obtain a valid (and necessary) quantum “adjoined” boundary rule.

Henceforth, we will continue to use particular X structures with clearly defined literals (1, -1, 2, and so on) to demonstrate all described operations in the proof. As the reader will see, this use is abstract enough to demonstrate all necessary points.

Next, consider any two X structures of size 2 such that X1 = {1, 3} and X2 = {1, -3}. Clearly, the variable x_1 cannot be assigned a positive value 1 at any point in the program. If we were forced to assign the literal 1 to true, $\mathbf{X}_3$ becomes unassignable. We will call this an assignment dilemma on $\mathbf{X}_3$.

The main line of the proof will show that upon successful completion of a first generation on some arbitrary 3 SAT instance, we never run into the assignment dilemma mentioned (in a very exact, abstract sense which we will demonstrate). This fact couples with the additional fact that because our X structures always denote only valid actions that must be taken at “boundaries”, only invalid solutions are ever excluded at each point in time. The result is that our proof is able to show that we

can after first generation, continuously assign values to all variables, where some assignments may be free and others bound (to the value necessitated by the boundary rule indicated by some valid X structure).

The approach taken can be used to show that our algorithm is sound and complete (at every point in time) with respect to satisfiable instances. The details are a little subtle and so we ask the reader to pay close attention to both individual steps and their combined effect in order to validate the reasoning.

Henceforth, we will convert the term first generation into a single formal word – first-generation (note the hyphen). We will also apply this compression to the associated term second generation which will be referred to as second-generation.

Before we proceed, we state the following assumption: Only X structures with no more than 3 elements need to be considered for reduction during our proof and for any second-generation truth assignment procedure. We formalize this as a hypothesis:

Hypothesis 1: "Any second-generation algorithm (single variable truth assignment) need only consider X structures of 3 element sizes and less".

In the course of analyzing our cases, we will state some axioms that are conditional on the validity of this hypothesis. We ask the reader to accept these axioms until we prove its validity via our last provided axiom. This is because there is an implicit cyclicity in the way this hypothesis will be proved to be true which partially depends on the completion of the conditional axioms themselves, as will be fully demonstrated.

Here are the edge cases we examine to prove our approach:

Case 1:

X structures {1, 3} and {1, -3} exist as part of the first-generation completion.

By Axiom 2, this cannot be the case at all since we will have combined both structures into the 1 element "constant" structure {1} which would yield a constant assignment of -1 to true.

Case 2:

X structures: {1, -3}, {1, 2}, {-2, 3} exist as part of first-generation completion.

Here assigning the variable x1 to 1 forces the following other assignments by virtue of Axiom 2 (X structures to the left of colon: assignment implications to right of colon):

{1, -3} :3

{1, 2}: -2

{-2, 3}: -3

But by Axiom 2, this cannot be the case as the following actions would have been taken in the first-generation stage:

{1, 3} and {-2, 3} would have yielded {1, -2} which would combine with {1, 2} to yield just {1} which would yield a constant assignment of -1 to true.

Case 3:

{1, 3} and {1, -3} exist as part of a rewrite of two independent X structures, X1 and X2 each of size 3.

This means we can't assign the literal 1 to true. We call this situation a "local" assignment blockage on assignment to the literal 1.

This can happen if say we originally had X1 = {1, 3, 4}, X2 = {1, -3, 5} and then assigned both 4 and 5 to true.

By axiom 2 however, during first-generation, we would have constructed an extra structure X3 = {1, 4, 5}, such that on adding 4 and 5 to the Z structure of a program, we will be forced to add -1 by virtue of Axiom 1.

The question we then need to ask is, does the selection of 4, 5 and -1 lead to a dilemma over another variable, for example, say we have the following 2 structures:

{-1, 4, 5, 6} and {-1, 4, 5, -6}.

We know that this condition could not exist after a first-generation completion and so why this question? – This is purely in the interest of analysis (which will be fully appreciated in the next case). In the process of Z structure completion, we posit that we are only interested in guaranteeing continuity of "local" variable changes, that is, we are only interested in situations where we explicitly assign a single literal (a variable antipode) to true or are forced to assign it to true. Note that the only times we are "immediately" forced to assign any literal to true is at the level of literal assignments over 2-element X structures, say:

We have {1,2} and we select 1 (assign it to true), then we immediately have to select -2 (assign it to true).

In the case of our example where we have {-1, 4, 5, 6} and {-1, 4, 5, -6}, the selection 4 and 5 would immediately create the structure {-1, 6} and {-1, -6} (which we will call conditionally implied 2-element structures) which would then also render us unable to assign -1 to true.

In our case however, the presence of our two offending structures: {-1, 4, 5, 6} and {-1, 4, 5, -6}, as earlier implied, would have caused the construction of a new X structure {-1, 4, 5} during first generation. This would have combined with X3, that is, {1, 4, 5} to create {4, 5} which means that 4 and 5 could not be simultaneous assignments in any valid program and we would not have a blockage on either 1 or -1 (no local blockages on the 2 antipodes of any variable).

We then ask a follow up question: Assuming we only had {-1, 4, 5, 6} after first generation and not {-1, 4, 5, -6}. The assignment to true of 4 and 5 still leaves us with the 2-element structure {-1, 6}. Note that by hypothesis 1, we would not actually be performing reductions on 4-element structures but we immediately posit the following additional axiom:

Axiom 3: "Any 4-element and 5-element forced assignment rules will hold even if during actual variable assignment (second generation) we are only considering and reducing 3-element structures and lower".

The validity of this axiom should be self-evident if we consider that all injunctions can be seen as derived by necessity from the basis (original, non-completed) injunction set and that they will manifest logically (inductively) as we assign individual variables to literal truth values.

For the above question, let us agree with hypothesis 1 that only 3-element structures would actually be considered for reduction but agree with axiom 3 that the assignment to true of 4 and 5 still leaves us with the 2-element structure {-1, 6}. Now say we also have the following 3 element structure:

(-1, -6, 7}.

This structure implies that we could not assign 7 to true after assigning 4 and 5 to true but it also easy to see that the following structure would have been generated during first-generation:

{4, 5, 7}

Be convinced that 4 and 5 in this case need not necessarily be different literals, but by virtue of their difference, show the maximal extension of this case.

We now add a new axiom as a result of our analysis of this case. This is an axiom whose validity is based on hypothesis 1 being valid:

Axiom 4: "After rewriting any two 3-element X structures, if we derive from these rewrites, a local blockage on assignment to a literal x, our program can assign the negation of x to true, that is, add the negation of x, which has no local blockage, to the current Z structure, without producing a negative side effect on the functional validity of any other 3 element structure".

Here we used the term functional validity to mean that a forced assignment resulting from rewriting a 3-element structure will not contradict that resulting from rewriting any other 3-element structure simply because up to this point, we cannot construe of any resulting 2-element structures that would result in such a functional invalidation.

Case 4:

Variation on case 2 for rewrites.

We have following X structures as a result of rewriting three different 3 element structures: {1, -3}, {1, 2}, {-2, 3}.

As shown in case 2, this also generates a dilemma for assigning x_1 to the literal 1. Let us again, use a maximally extended case in which the 3 different structures use 3 different extra variables:

X1 = {1, -3, 4}, X2 = {1, 2, 5}, X3 = {-2, 3, 6}

On first-generation, applying axiom 2, we would have generated the following X structures:

X4 = {1, -2, 4, 6}, X5 = {1, 3, 5, 6} and X6 = {1, 4, 5, 6}.

It is again clear that assigning 4, 5 and 6 will explicitly force us to assign x_1 to -1.

Again, as in case 3, the follow up question we need to ask (for local variable assignment coverage) is, if only {1, 4, 5, 6} exist and not {-1, 4, 5, 6}, combinatorial-wise, how does the selection of 4,5,6 and -1 "act" on the space. Specifically, is it possible to have the following two 5 element structures:

{-1, 4, 5, 6, 7} and {-1, 4, 5, 6, -7}?

The existence of these 2 structures would mean that we could not assign -1 as expected in addition to 4, 5 and 6 since we would then not be able to select any value for X_7 and would therefore need to generate a new structure {-1, 4, 5, 6} which would combine with {1, 4, 5, 6} to yield {4, 5, 6}. A 3-element structure.

Now a variation of the follow up question we asked in case 3 becomes doubly important in this case. This question translates in this case as follows: Assuming we only had {-1, 4, 5, 6, 7} after first generation and not {-1, 4, 5, 6, -7}. The assignment to true of 4, 5 and 6 still leaves us with the 2-element structure {-1, 7} which means selecting -1 forces us to select -7. Now, say we also have the following 3 element structure:

(-1, -7, 8}.

This structure implies that we could not assign 8 to true after assigning 4, 5 and 6 to true and in this case it also easy to see that the following structure would have been generated during first-generation:

{4, 5, 6, 8}.

Here, it would seem we would have to rewrite 4-element structures to derive the fact that 8 could not be assigned to true after assigning 4, 5 and 6 to true.

So far, we have assumed that 4,5 and 6 are not in a dependent 3-element relationship so that the selection of say 4 and 5 causes us to select 6, that is, the following structure does not exist:

{4,5, -6}.

Now if such a structure did indeed exist in our Y structure, let us examine its effect on the structures originally specified at the start of the case analysis, that is:

X1 = {1, -3, 4}, X2 = {1, 2, 5}, X3 = {-2, 3, 6}

It is clear that X3 would combine with this new structure, {4,5, -6}, to give {-2, 3, 4, 5} which would combine with both X1 and X2 respectively to give {1, -2, 4, 5} and {1, 3, 4, 5}. These two 4-element structures would then each recombine with the original 3 element structures as follows:

{1, -2, 4, 5} combines with X2 = {1, 2, 5} to give {1, 4, 5} and

{1, 3, 4, 5} combines with X1 = {1, -3, 4}, to also give {1, 4, 5}.

Now we have an even stricter requirement on the trio of 4,5 and 6 to assign -1 to true as originally deduced.

Now if we assume that 4,5 and 6 do not exist in such a dependent relationship as construed, but we still had {-1, 4, 5, 6, 7} and {4, 5, 6, 8}, then we need only to explicitly consider the case where the

selection of -1 and -7 (since {-1, 7} becomes implied by the selection of 4,5 and 6} requires the selection of 8, that is, the X structure:

{-1, -7, -8}.

Now this 3-element structure combines with our provided 4 and 5 element structures as follows:

{-1, -7, -8} combines with {4, 5, 6, 8} to give {-1, 4, 5, 6, -7} which then combines with {-1, 4, 5, 6, 7} to give {-1, 4, 5, 6} which would then combine with deduced X6 structure, {1, 4, 5, 6} to give {4,5,6}.

Now we have all our valid logic in 3-element structures (which also includes the new introduction (-1, -7, 8})

Again, we cannot conceive of a scenario in this case where the result of rewriting a 3-element structure as part of second-generation would contradict the result of rewriting any other 3-element structure.

Again, as in case 3, We add a new axiom as a result of our analysis of this case:

Axiom 5: "After rewriting any three 3-element X structures, if we derive from these rewrites, a blockage on assignment to a literal x, our program can assign the negation of x to true, that is, add the negation of x, which has no local blockage, to the current Z structure, without producing a negative side effect on the functional validity of any other 3 element structure".

Case 5 (terminality case):

Note that we have proven conditionally up to this point that any assignment blockages we encounter upon a rewrite of up to three 3 element structures can be resolved by reassigning the blocked literal to its negation.

Please be convinced that 3 is the maximal number of 3-element structures we need to consider in a single step for analyzing local blockage on any literal that can result from rewriting any number of 3-element structures (based on our stated conditions resulting from hypothesis 1, that is, case 4 captures the maximal blockage pattern on any such literal).

Please be also convinced that we need to only consider actual and "implied" local blockages (analyzable from 2-element structures) in our analysis, that is, we can always guarantee that no two opposing literals of the same variable are both locally blocked at any point in time and that there is a guaranteed local continuity of assignment over every variable point. This is a notion of continuity that has to be carefully and intuitively grasped in order to ascertain the full correctness of our procedure.

We now add a closing axiom:

Axiom 6: "Any program which can continuously rewrite all the BASIS injunctions that directly represent the original clauses (which all contain at most 3 elements by natural transform) such that none of them assigns a literal to true which is assigned false by another, is sufficient to prove that the original formula is satisfiable".

This axiom necessarily validates hypothesis 1 and by extension all axioms that were conditionally declared based on its supposed validity.

If we then agree that all our axioms are sound, we can also likewise conclude that since all structures only exclude invalid assignments at any point in time, then our algorithm is (always) complete, and indeed we need no more than structures of a maximum of 5 elements to be constructed during first-generation to fully determine the satisfiability of our original clause.

One last set of non-crucial points to observe:

1. Freely assignable variables not bound to a represented structure may occur at any point during the course of running the algorithm. We will call these structures radicals and
2. 1 element structures (constants) may occur during first-generation prior to second-generation selections and that in fact all variables over the instance signature may collapse into just one assignable literal each, generating only a single combination instance, in which case there are no possibilities for second-generation modifications. This is a closed loop.
3. If we accept that our algorithm is sound and complete, assignments need not perform an actual reduction on any 3-element structures but simply treat all such structures as "concurrent" functional programs that always agree on their combined outputs. In this case, we can simply match up the Z structure to each valid X structure of size 3 and less, adding each required output to the Z structure as we progress.

Combining all said proof points, we conclude that if we can't construct a first-generation model (dilemma over some arbitrary variable), our instance S is unsatisfiable, otherwise it is satisfiable and we can generate a sound and complete exact model of its solution space.

2.7 Runtime Analysis

This analysis is straightforward and does not take into consideration the many ways that the algorithm could be made more efficient in practice such as both parallel computation (possible with classical and actual quantum systems) and data partition strategies that reduce the number of structures that would be compared with each other.

Since X structures cannot have more (width) than 5 elements, we can have no more than n^5 distinct X structures overall. If we compare each X structure to every other X structure during first-generation we get n^5 by n^5 total comparisons giving us a total of n^{10} comparisons. All modifications are constant time and so our runtime cannot exceed that needed to make individual comparisons, giving us a maximum of n^{10} total number of operations.

For second generation, if we use only the recommended basis set, similarly, we get no more than n^3 possible X structures. Here, we do not compare structures but instead loop over their collection for a possible maximum of n times to derive a solution. This gives us a possibility for no more than n^4 possible operations.

Considering the totality of all possible operations, we can see that our algorithm has a maximal runtime complexity of $O(n^{10})$.

3. Conclusion

We have introduced a quantum algorithm that runs on classical computers, a first from what we know. We hope that this would prove useful to mathematicians and physicists who aim to study quantum systems and their related mathematical structures and properties. We feel that the greatest contribution we would be able to make in addition to being able to solve the class of problems currently known as NP in the computing literature efficiently (and we hope to have successfully proved is P is equal to NP) is in the scalable and iterative modelling of quantum systems.

We conjecture that problems with existing rational order – mathematical and physical, would be particularly tractable using our algorithm as they would already have some intrinsic quantum computational/algorithmic patterns embedded in their structure. In particular, this should help us better understand quantum systems and their efficient design and improvement. We hope that this spurs a new era in work on both quantum and classical algorithms and systems as we better understand their properties and possibilities on all possible computing platforms.

References

[1] Valiant, Leslie (17–19 October 2004). Holographic Algorithms (Extended Abstract). FOCS 2004. Rome, Italy: IEEE Computer Society. pp. 306–315. doi:10.1109/FOCS.2004.34. ISBN 0-7695-2228-9.

[2] Stephen A. Cook. The complexity of theorem-proving procedures. In Proceedings of the 3rd ACM Symposium Theory of Computing, pages 151–158, Shaker Heights, Ohio, 1971.

[3] Leonid A. Levin. Universal search problems. Problemy Peredachi Informatsii, 9(3):265–266, 1973.

[4] Richard M. Karp. Reducibility among combinatorial problems. Complexity of Computer Computations, (R. Miller, J. Thatcher eds.), pages 85–103, 1972.