Reductiones ad Absurdum of the "P versus NP Problem" and P =NP/AP/CH

Deng Ke[1]

Abstract

There exist latent meta-axioms concerning propositions in mathematics that are often overlooked. A discussion of these axioms simultaneously brings to light the secondarity of mathematics with respect to reality. Applying one of these axioms to the "P/NP problem" can reduce the problem to absurdity; however, precisely due to this secondarity of mathematics, such a dissolution can only ever be a "dissolution" rather than a "solution." To approach the "P/NP problem", we again adopt a reductio ad absurdum method: understanding the limitations of polynomial-time algorithms and identifying such limitations within NP-complete problems, thereby obtaining $\mathrm{P} \neq \mathrm{NP}$. Extending this reductio from the specific contradiction between polynomial-time algorithms and NP-complete problems to the more general contradiction between guarantee of winning and symmetric mutually-exclusive games, we arrive, in the computational and purely logical domains respectively, at the conclusions $\mathrm{P} \neq \mathrm{AP}$ and $\mathrm{P} \neq \mathrm{CH}$.

Keywords: proposition, condition, secondarity, polynomial-time algorithm, symmetric mutually-exclusive game

1 Reductio ad Absurdum of the "P vs. NP Problem": Axioms of Proposition and Their Application to the "P vs. NP Problem"

1.1 Conditions of Proposition

1.1.1 Existential Condition/Formal Existential Condition of Proposition

In mathematics, a proposition P contains two core elements: condition A (hypothesis/premise) and conclusion B. It requires a part that is known to hold, and a result derived from that part. The standard logical form of a proposition is

$$\text{If } A, \text{ then } B. \quad (A \to B.)$$

However, the "condition A" appearing here is an existential condition, a condition for its formal existence, which is the second condition A_2, a subordinate condition. Such a condition merely allows a complete proposition to exist and be describable in one description; it is a part of a complete proposition, is internal, and does not depend on anything external. The emergence of the proposition itself requires a guarantee—a more prior, external guaranteeing condition, namely the proof condition/first condition A_1.

1.1.2 Proof Condition/Mathematical Formal Existential Condition of Proposition

If something is merely stated/described, it is just a sentence, not a mathematical proposition. That is, proposition must not only exist formally but also exist specifically in mathematical form.

Proposition requires a finite sequence of symbols—a proof—that proceeds from accepted axioms, via valid rules of inference, to arrive at/realize itself. The existence of such a sequence from the axioms to P is a proof. Without a proof, P does not exist (even if everyone believes it

[1] Independent Scholar, DengKee@proton.me.

exists). Proof directly participates in giving the meaning of "existence"; it is a necessary condition for "existence":

$$\text{If } A_1, \text{ then } (\text{If } A_2, \text{ then } B). \quad [A_1 \to (A_2 \to B).]$$

The formal existential condition A_2 allows something to be described; the proof condition A_1 allows this thing to be described mathematically, to become mathematical content. Proof is a necessary condition for "existence" in mathematics. Without a proof, something has only a non-mathematical linguistic definition, not a mathematical definition.

"Being proved" is a judgment that a proposition exists mathematically, and it is also a proposition that distinguishes whether a proposition "is a proposition." Because such a proposition describes and judges other propositions, it belongs to the category of first-level propositions P_1/meta-propositions. The propositions we use in daily life are second-level propositions P_2. Generally, we do not need to make such a fine distinction. Strictly speaking, the complete description of any everyday proposition is encoded as a nested proposition: the P_2 we want to describe is always described by directly describing P_1, and P_2 then appears as part of it. The complete description of any true everyday proposition needs to consider the first proposition P_1 and the proof condition A_1; it is just that usually P_1 and A_1 are omitted as common knowledge.

1.1.3 Truth Condition/Actual Existential Condition of True Proposition

1.1.3.1 Truth Condition A proposition that exists formally and mathematically is not necessarily true—that is, it may not exist in reality, because it could be false.[2] A false proposition can be described (non-mathematical formal existence) and can be proved (mathematical formal existence), meaning it can be proved false/does not hold—"holding" is a property of true propositions. Thus, a true proposition requires more than a false one: it needs to hold.

If a true proposition only exists internally, it is merely a formal existence; if the form does not undergo a proof that yields truth, the proposition could be false, and a false proposition does not hold. Therefore, for a proposition to truly exist, i.e., one that yields true results B_{T} (in other words, if B has a subscript T, then a truth condition is required), it needs to exist in two dimensions simultaneously: in form (including non-mathematical and mathematical, but we generally refer to mathematical form) and in reality. If it does not exist in form, it cannot be described; if it does not exist in reality, it might describe something that does not exist. Existing in reality is called "holding." It is a guaranteeing existence that is even more external than the proof condition. Therefore, a true proposition requires, in addition to the formal existential conditions A_2 and A_1, a condition for its existence in reality, A_0. Without A_0, there is no A_1 or A_2:

$$\text{If } A_0, \text{ then } [\text{If } A_1, \text{ then } (\text{If } A_2, \text{ then } B_{\mathrm{T}})]. \quad \{A_0 \to [A_1 \to (A_2 \to B_{\mathrm{T}})].\}$$

A complete true proposition must be stated in this way. Here, the zero-th proposition P_0 containing A_0 takes the meta-position formerly occupied by the first proposition.

Proof merely provides a mathematical condition for a conditional proposition; verification provides a real-world condition for a proposition with a mathematical condition. So this just says that a truth condition is needed, but what exactly is truth condition?

---

[2] A P's holding mathematically is merely being true in the mathematical sense, i.e., mathematical truth, and mathematical truth is not absolute truth. Absolute truth is reality truth, tied to reality, it is not mathematical. "True" in this paper is used in a new sense and does not subsume the "true" of traditional mathematics; the traditional one is downgraded to "being proved"/"having A_1."

1.1.3.2 The Form of Truth Condition—Verification Condition "Truth condition" is just a name for this condition, named after its consequence ("it bestows truth,") but it does not express the specific mechanism of that condition. So what is the specific mechanism of truth condition?

Proof condition is only a necessary condition for "holding," not a sufficient one. "Holding" also requires a criterion for judgment. "Proof" is still an internal operation; to know whether it is ultimately correct, we need to introduce external verification. If we do not connect the form with real-world models and concrete instances, a purely formal system can become detached from intuition and even produce paradoxical conclusions, such as a seemingly perfect proposition that entails $1+1=3$. On this basis, objects requiring exhaustive verification, such as universal propositions involving infinite objects (e.g., the Goldbach conjecture), are in fact forever unable to "hold" and can only exist mathematically.

As stated above, we need existence in both formal and real dimensions, and existence in reality requires "verification"—it guarantees that our judgments about "proof" are reliable, and that the axioms and rules we use are not spinning arbitrarily. The proof condition A_1 merely ensures that an ordinary judgment becomes a proposition (enters mathematics); the verification condition A_0 alone can guarantee that a proposition holds. Holding requires external instances to verify it—that is the mechanism of truth condition.

Verification condition is not a logical condition like proof condition that participates in general mathematical operations; rather, it seeks instances to confirm, acting as a guaranteeing/corrective condition that provides a guarantee for mathematical operations and definitions.

1.1.3.3 "Purely Formal Truth" and the Secondarity of Mathematics Isn't it too strict to say "if no real-world instance can be found, then it does not hold"? If we reserve a theoretical place for "purely formal truth" and "purely formal holding," many objects requiring exhaustive verification could hold. But if we reconsider the purpose of "mathematics," we find that hooking onto reality is merely a basic requirement.

Do we discover mathematics or invent mathematics? The crux of this question is not "discover" or "invent," but the object "mathematics." No one thinks that mathematical symbols pre-exist humans, so those who say "invent mathematics" actually mean "mathematical language/mathematical tools," which are indeed invented. Both sides actually hold the same view, but the referent of "mathematics" is not unified. Let us first take this object to be "mathematical language," and in that case, we are inventing mathematics. If we think we are "inventing," then we also assume there exists a "mathematical reality"—and the purpose of inventing mathematical language is to describe this reality.

But that which is thought to require description by mathematical language, the "mathematical reality," is actually already reality—just a pattern or information within reality. The "mathematical reality" that can be described is also a part of reality; there is no need to demarcate it specially. The concept "mathematics," when uttered, is already distinct from reality; it is a descriptive tool. That is, it has always been "discover reality, invent mathematics."

In this view, reality is primary, and mathematics is secondary. Since the very emergence of mathematics is to describe reality, demanding a connection with reality is the most basic requirement. "Truth" and "holding" need no definition separate from reality; from the most fundamental purpose, they are already descriptions of things that exist in reality, and the language describing things that exist in reality should of course correspond to reality.

Purely formal content has never needed "truth" or "holding"—it does not need to correspond to reality, yet it can still have value in mathematics. For example, the Goldbach conjecture has genuine value in number theory, yet everyone also knows it has almost no direct, practical real-world use. That is to say, "purely formal truth is never needed and its absence does not affect its value" has always been a consensus; it only seems counter-intuitive when explicitly articulated. Strictly speaking, purely formal mathematics does not require "verification," because it has no need to correspond to reality. Verification condition is not a universal yoke imposed on all mathematical content, but a hard requirement for those propositions that are claimed to be "true" or "holding," i.e., those that are relevant to reality.

As with the Goldbach conjecture, there indeed exists in mathematics abstract content that is not easily reproduced in reality. However, this should not be seen as a separate domain. It is not that there are two major categories in mathematics: abstract mathematics and non-abstract mathematics. Rather, "abstract mathematics is an extension of the task/goal of 'describing reality'". (Simply put, if it were not an extension but independent, why would it not be nonsense or the creation of meaningless purely aesthetic mathematics? Why is it instead all useful mathematics? "Usefulness" is rooted in reality—even logic aids life.) It may merely be overly rich—and that richness reflects our enthusiasm for "describing reality."

Thus, not only is "purely formal truth" and "purely formal holding" unnecessary, but they would also oppose the birth of mathematics. Moreover, discovering content that is not easily reproduced in reality is itself a better understanding of reality—there exist aspects of reality that make abstract thinking necessary and valuable.

1.2 Application of Axioms of Proposition to the "P vs. NP Problem"

1.2.1 Applying Verification Condition to the "P vs. NP Problem"

1.2.1.1 First Contradiction of "Non-Verificational True Algorithm" Verification condition, as truth condition for a proposition, has the following bearing on the "$\mathrm{P/NP}$ problem"[3]: when verification is a necessary element for a true proposition, an "algorithm" that bypasses verification and directly computes the answer can only produce non-true results $B \in (B_{\mathrm{T}})^{\mathrm{c}}$ (the superscript c denotes the complement) /B_{F}, not true (i.e., established) results B_{T}. Consequently, a result directly computed without verification does not exist/hold, because such an "algorithm" lacks the function of producing true results, and therefore the "algorithm" itself does not exist/hold. That is:

- Verificational true algorithms can exist;
- Non-verificational non-true algorithms can exist;
- Non-verificational true algorithms cannot exist.

An "polynomial-time algorithm" (p-t algorithm) is precisely a "non-verificational true algorithm."

Since truth (holding) is directly conferred by verification condition, when a "p-t algorithm" rejects verification, it simultaneously rejects the truth of its computed results. Moreover, because its results are not guaranteed to be true, its own condition as an "algorithm" also does not hold. Its results can only possess formal existence (satisfying A_2) and mathematical formal existence (satisfying A_1), but not real-world existence (A_0). That is, the algorithm and its results can be described and mathematically described, but they describe non-existent things. Unless we allow the results to be non-true—i.e., treat it merely as a formal algorithm, not a true one, emphasizing

[3] Stephen A. Cook. *The complexity of theorem-proving procedures*.

only formal existence and not involving applied domains like cryptography, simply computing a formal result that need not hold—otherwise, once it touches real-world application domains, the algorithm does not hold mathematically.

Thus, if it is a "pure p-t algorithm" solution, it will ultimately be unable to confirm its own existence. A p-t algorithm must be paired with methods outside itself (such as separate verification) to be effective, yet the "$\mathrm{P/NP}$ problem" only asks "whether a p-t algorithm exists"—this is a question rooted in the mindset of a "pure p-t algorithm," which does not emphasize the need for verification in order for something to hold (be established).

People ask the "$\mathrm{P/NP}$ problem" because previously verification condition for true propositions was unclear. When verification condition is strictly required as a necessary condition for a true proposition, one generally no longer questions whether a "pure p-t algorithm" exists; instead, one clearly knows that it cannot exist, or can only exist as a formal algorithm that computes formal results, or alternatively, the question is explicitly framed as "whether a p-t algorithm paired with other methods exists."

1.2.1.2 Second Contradiction of "Non-Verificational True Algorithm" Even if we go a step further and allow the "p-t algorithm" to exist—i.e., to exist and bypass verification to compute true results, and even allow it to be self-referential—any true result, even the verification of itself, can be computed by itself in polynomial time, possessing the function to compute anything that would otherwise be waiting to be verified, including computing its own verification result (without needing to "find" an external instance to confirm the existence of the "p-t algorithm," directly "computing" an external instance), thus rejecting verification to the very end:

We find that even then, if it still does not emphasize the supporting methods outside the p-t algorithm itself, it still falls into a contradiction: The parts of this algorithm that would require verification are directly computed. If the directly computed result is true, it itself is still awaiting verification. Thus, the algorithm falls into a loop—

(1) Compute result $(B_{\mathrm{T}})_1$;

(2) $(B_{\mathrm{T}})_1$ is also awaiting verification. Compute the verification of it, which is $(B_{\mathrm{T}})_2$;

(3) Use the same "bypass verification" special skill to compute $(B_{\mathrm{T}})_2$;

(4) $(B_{\mathrm{T}})_2$ also requires verification. Compute the verification of it, which is $(B_{\mathrm{T}})_3$;

(5) $(B_{\mathrm{T}})_3$ also requires verification. And so on.

The algorithm can indeed guarantee that each individual computation finishes in polynomial time, but such computations must continue forever—because it rejects verification. And once the computation becomes eternal, the required total time is positive infinity, which certainly exceeds polynomial time. A "polynomial time" algorithm exceeds polynomial time—a contradiction arises.

This is not a design flaw in the algorithm—we have already made concessions to allow it to "hold," so it runs into a dead end while otherwise operating normally. This is not a contradiction of the algorithm itself, but a contradiction arising from encountering a true proposition. It is a contradiction between the hard requirement of a true proposition for verification and the algorithm's rejection of verification—an infinite regress inevitably triggered when verification is rejected in the domain of true propositions that have a hard requirement for verification. That is, it is not a problem of design, but a problem between purpose and method: the purpose of a "p-t algorithm" is to compute real-world true results (e.g., for cryptography), yet its method rejects verification condition that defines true results.

1.2.2 Applying the Secondarity of Mathematics to the "P vs. NP Problem"

An "p-t algorithm" could exist in fact without holding mathematically—reality never depends on mathematics. According to the secondarity of mathematics, since people use mathematics to understand reality, it is entirely possible that there exists a reality we cannot yet properly comprehend mathematically: For example, in reality, if a p-t algorithm exists, it would almost certainly come with a verification method—an effective p-t algorithm only needs to run once, because the verification required by $(B_{\mathrm{T}})_1$ (the first contradiction) is carried out by the verification method paired with the algorithm. Therefore, criticizing the "pure p-t algorithm" through the "$\mathrm{P/NP}$ problem"'s failure to discuss verification methods is merely a theoretical exercise; in practice, such a "pure" solution would almost never occur.

The situation where "something exists in reality but does not exist in mathematics" is reasonable. Mathematical language itself is not identical to reality; it is a tool that can be developed and improved. Therefore, "truth" and "holding" as we define them in mathematics are independent of reality—a non-verificational true algorithm is merely not allowed within mathematics, but it is entirely possible that someone is already using a "p-t algorithm" to break cryptographic codes. That person can use it, because "not holding" does not affect use; use does not require mathematical description. "Use" is a matter of reality; "verification" is only a tool that tries its best to approach reality—attempting to approach reality, but not identical to reality.

The answer to the "$\mathrm{P/NP}$ problem" is: because mathematics has secondarity—the "$\mathrm{P/NP}$ problem" is a pseudo-problem in mathematics, but a real problem in reality. Even though a "p-t algorithm" does not exist in current mathematics, it could still exist in reality. The application of the conditions of a proposition to the "$\mathrm{P/NP}$ problem" is not about solving the "$\mathrm{P/NP}$ problem," but about reflecting on the axiomatic foundation of the "$\mathrm{P/NP}$ problem" and thereby dissolving it. So if we want to get as close as possible to proving that $\mathrm{P} = \mathrm{NP}/\mathrm{P} \neq \mathrm{NP}$, we need to first assume that the "$\mathrm{P/NP}$ problem" exists and $\mathrm{P} = \mathrm{NP}$ or $\mathrm{P} \neq \mathrm{NP}$, then restart from another perspective.

Appendix: Issues with the Definition of the "P vs. NP Problem"

Should brute-force search be tied to exponential time? We instinctively view brute-force enumeration as super-polynomial time, but this is not an absolute logical necessity—pure logical deduction: humanity inventing a fast enumeration technique, a method to "traverse" an exponentially large solution space in polynomial time. Under classical computational models (turing machines, ordinary computers), it has been proven mathematically impossible to "quickly traverse exponentially many things"—information-theoretic lower bound: outputting or accessing exponentially many distinct states requires at least exponential steps. However, this has never precluded changing the computational model.

Brute-force search is traditionally viewed as super-polynomial time, and p-t algorithms discussed in the "$\mathrm{P/NP}$ problem" are usually implicitly assumed to be non-brute-force enumeration algorithms—but in fact, if $\mathrm{P} = \mathrm{NP}$, it could also be due to the existence of a fast brute-force search method.

And if the "$\mathrm{P/NP}$ problem" allows a p-t exhaustive search method for NP problems, then this reductio ad absurdum of p-t algorithms could become invalid, because exhaustive search is precisely brute-force verification—which includes the verification condition A_0.

2 Reductio ad Absurdum of P = NP: P-T Algorithm Contradiction of NP-Complete Symmetric Simultaneous Mutually-Exclusive Game Problems

2.1 Idea

Assume

$$\neg(\exists A_0) \Rightarrow \exists[(\mathrm{P}=\mathrm{NP})\vee(\mathrm{P}\neq\mathrm{NP})?]\wedge(\mathrm{P}\neq\mathrm{NP}),$$

proof method: Reductio ad absurdum of P = NP—Construct an NP-complete (NPC) problem such that if a p-t algorithm existed, a contradiction would arise, thereby proving that no NP problem that reduces to this NPC problem has a p-t algorithm. That is,

Step 1: Understand the limitations of p-t algorithms, i.e., what situations they cannot allow;

Step 2: Understand whether the class of NPC problems contains an NPC problem that encompasses the limitation from Step 1. That is, a problem that has a limitation preventing the existence of a p-t algorithm, yet belongs to the NPC class.

Completing these two steps would allow us to rely on the representativeness of NPC problems for NP problems to conclude that no NP problem has a p-t algorithm.

2.2 The Limitation of P-T Algorithms—Symmetric Mutually-Exclusive Games

We now proceed to Step 1: Discuss the limitations of p-t algorithms. For a p-t algorithm to be effective, it must not invalidate itself, i.e., it must not lead to a contradiction—this is not only a prerequisite for a p-t algorithm, nor just for any algorithm, but a prerequisite for the validity of anything. Therefore, a p-t algorithm has at least one limitation: it cannot be both effective and ineffective simultaneously.

So, what situation would cause a p-t algorithm to be both effective and ineffective? How could a p-t algorithm being effective cause itself to become ineffective? Being effective in a symmetric, mutually exclusive, zero-sum game would cause itself to become ineffective—simply put, if one uses a p-t algorithm in a symmetric, mutually exclusive game,

- due to mutual exclusivity, the only possible outcome is that I win, and the opponent loses — for example, in Rock-Paper-Scissors, if both players make the same move, the round is inconclusive and cannot be a final outcome. The definition of a "zero-sum game" itself allows draws, so we need to restrict the scope to zero-sum games that do not allow draws — mutually-exclusive zero-sum games. Since mutual exclusivity is a subset of zero-sum, we will simply refer to "mutually exclusive games" below. A p-t algorithm inherently requires computing a solution, and in a mutually exclusive game, the solution means a winning strategy. (If it were merely zero-sum, the solution would be just a non-losing strategy.) If I do not win/the opponent does not lose, then the basic function of the p-t algorithm would not be fulfilled, implying the p-t algorithm does not exist. Therefore, if a p-t algorithm exists and is used in a mutually exclusive game, it must win, and outcomes where the opponent wins, a draw, a double win, or a double loss are impossible;
- due to symmetry, given that both faces face the same problem, a p-t algorithm usable by myself would also be usable by the opponent—In a mutually exclusive game, a p-t algorithm is, by definition, required to compute a winning strategy, then if a double win occurred, a contradiction would arise—the p-t algorithm provides a solution that negates its own solution (one

player's p-t algorithm would yield a solution that negates the solution of the other player's p-t algorithm), yet both solutions would be claimed to be valid—would mean the p-t algorithm is ineffective—In a mutually exclusive game, what a p-t algorithm must compute is precisely a strategy that causes the opponent to lose. And if it is not a double win, then there are three possibilities—one side wins and the other loses, a draw, or a double loss—any non-double-win outcome proves that the p-t algorithm is ineffective, because theoretically, if a p-t algorithm existed, it ought to facilitate a double win, even if that leads to a contradiction.

P-t algorithms

- Can be used on zero-sum game problems—enabling oneself to win or not lose;
- Can be used on mutually exclusive game problems—enabling oneself to win;
- Can be used on symmetric game problems—resulting in a double win;
- Can be used on symmetric zero-sum game problems—resulting in a draw;
- Can even be used on symmetric mutually-exclusive game problems but just for "analysis" —acting as a third party that computes the outcome of the game in polynomial time, which could be a win and a loss (one side wins, the other loses...). At this point, the purpose of the algorithm is to provide results rather than winning strategies.

However, they cannot be used in "participating in" symmetric mutually-exclusive game problems (i.e., not analyzing the game, but playing as a participant in such a game)—In a game where double wins are not allowed, yet the definition and logic absolutely demand an impossible double win, a contradiction arises.

This does not require that an opponent actually exists; rather, the contradiction can form even if the opponent does not appear. Knowing that this game is symmetric, one can, even when acting as a single party, claim a sure win, which equivalently also claims a sure win for the opponent, declaring that the opponent would also win even before they have appeared. The p-t algorithm does not produce a contradiction only after the opponent appears; rather, once it appears in this type of game (even merely as one party), the contradiction already arises, and it is incompatible with this class of games.

Thus, for a p-t algorithm appearing in a symmetric mutually-exclusive game, all possible outcomes — double win, double loss, one win and one loss, draw — are self-contradictory. Therefore, a p-t algorithm cannot be used for symmetric mutually-exclusive games.

2.3 Constructing an NPC Symmetric Simultaneous Mutually-Exclusive Game Problem

2.3.1 Non-NP Symmetric Mutually-Exclusive Game Problems

We now proceed to Step 2: Find or construct a symmetric mutually-exclusive game problem that is in NP and is NPC, because p-t algorithms cannot be used for symmetric mutually-exclusive games, and if there exists an NPC symmetric mutually-exclusive game within the NP problems, NP problems would have no p-t algorithm. However, are typical symmetric mutually-exclusive games NPC or even NP? Let's first look at some common symmetric mutually-exclusive games:

Symmetric mutually-exclusive games have always existed—for example, chess,

- game theory usually allows a "first-move advantage," deeming a game symmetric as long as the overall game structure remains unchanged when swapping the roles of the two players (only the turn to move changes);
- board games are of course mutually exclusive, generally speaking.

However, such games are usually not NPC, and not even in NP—For board games with prohibited infinite repetition, the maximum number of moves is still exponential in the input size (exponential in the number of board squares; i.e., EXPTIME-complete), while for board games allowing infinite repetition without forced progress—the game can loop back to a previous state infinitely, leading to EXPTIME-complete games that are infinitely amplified — becoming EXPSPACE-complete. Therefore, symmetric mutually-exclusive games can be EXPTIME-complete or EXPSPACE-complete, and thus may not belong to NP. Proving that a harder problem has a p-t algorithm can prove that an easier problem also has a p-t algorithm;[4] conversely, proving that a harder problem has no p-t algorithm does not prove that an easier problem also has none. Hence, we need to convert the problem into an NPC problem first—by restricting the number of moves.

However, if we restrict the number of moves to a size solvable in polynomial time—such as coin tossing (single move), Rock-Paper-Scissors (single move or a few moves)—then the problem is no longer NP, but rather P or even trivial, because even without a p-t algorithm, one could find a solution via brute-force enumeration in polynomial time. Since our goal is to prove that NP problems have no p-t algorithms, proving that a P problem has none would be meaningless for our purpose.

Thus, too many moves lead to EXPTIME or above, too few moves lead to P or even trivial—problems solvable and verifiable in polynomial time are insufficient, and problems requiring more than polynomial time to solve and verify are also insufficient. What we need is a symmetric mutually-exclusive simultaneous game problem that cannot be solved in polynomial time but allows verification—that would be an NP symmetric mutually-exclusive game problem, and only after being an NP problem could it potentially be NPC.

Supplement Even if the origin of the problem is "Nature" — an unconscious, passive optimization process—it can still be reinterpreted as an active, adversarial problem. This is to translate a natural problem—for instance, the question "Does there exist a possible state?"—into the question "Do I possess a winning strategy?" The core of any NP problem is "∃ solution." This "∃ solution" structure is inherently simulatable by a game-theoretic structure of the form "∃ my strategy, ∀ your strategy."

According to the Cook-Levin theorem, every NP problem—regardless of its original form, whether it is "finding a path," "filling a grid," or "breaking a cipher"—can be "translated" into an adversarial game. This translation process is general, mechanical, and independent of whether the original problem contains an "opponent."

2.3.2 NP Symmetric Simultaneous Mutually-Exclusive Game Problems

Let's consider why the games mentioned earlier do not belong to NP. The reason coin tossing and Rock-Paper-Scissors are P problems is evidently because the number of moves is too small, so we mainly need to determine why board games are EXPTIME problems:

Board games belong to EXPTIME or harder complexity classes because of "alternating moves"—Player A makes a move, then Player B makes a move. This forces almost every step to adjust strategy based on the opponent's latest move, and the cumulative heavy computation required for each step results in very complex real-time simulation.

---

[4] There is an inclusion relationship between harder and easier problems. However, algorithms are not necessarily generalizable, as there is no "arbitrary-to-complete" reduction between different complexity classes, and it is mostly "easier-to-harder" reductions.

From this, we can observe that if we change from an "alternating-move game" to a "simultaneous game," such as coin tossing or Rock-Paper-Scissors where the outcome is determined simultaneously, we cut off the quantifier chain and remove the temporal depth, thereby eliminating the dependency chain that causes real-time simulation.

Of course, if it is truly coin tossing or Rock-Paper-Scissors, that falls directly to P or even trivial problems. Therefore, we need a problem that is simultaneous, but harder than coin tossing or Rock-Paper-Scissors[5]—a symmetric simultaneous mutually-exclusive game problem.

2.3.3 NPC Symmetric Simultaneous Mutually-Exclusive Game Problems

2.3.3.1 Idea There are two directions for constructing symmetric simultaneous mutually-exclusive game problems:

- turning alternating-move games into simultaneous ones;
- making simple simultaneous games harder.

The former involves modifying games like board games, while the latter involves modifying games like coin tossing/Rock-Paper-Scissors. "Turning alternating-move games into simultaneous ones," although logically feasible, is very cumbersome to implement in practice and can easily, by mistake, construct a P problem — making it too simple, falling into P : after changing to simultaneous moves, the choices of both players become independent, and the problem might degenerate into "whether there exists a pair of actions satisfying a condition," which is often a P problem. And if the modification is "too successful," it might retain too much interaction and exceed the NP category. Therefore, we choose the second direction, which is theoretically much simpler and more standard—making simple simultaneous games harder. But either way, the result is a symmetric simultaneous mutually-exclusive game problem.

Making simple simultaneous games harder — we aim to make simple simultaneous games harder. Starting from the simple simultaneous games mentioned earlier—but not coin tossing: coin tossing is a zero-sum game, although it does not allow double wins, it does allow draws (both calling the same side). Although in daily play, we naturally disregard draws as non-outcomes and continue (i.e., draws are not permitted in practice), we still consider a more standard, mutually exclusive game that by definition does not allow draws and where victory is defined by making the opponent lose — Rock-Paper-Scissors (RPS. Rock defeats Scissors by crushing it, Scissors defeats Paper by cutting it, Paper defeats Rock by covering it—victory strictly depends on the opponent's move/whether the opponent loses, whereas coin tossing only requires one to guess correctly, without checking the opponent's correctness):

We transform RPS game into a combinatorial RPS game — using SAT problem thinking: increasing from "moves" to "move combinations," the action space becomes larger. This transformation process makes the generated formula scale huge, turning RPS game not only into an NP problem but also into an NPC problem, and specifically the most standard type of NPC problem — the SAT problem. Now, we construct a SAT Duel problem by wrapping RPS game: Change the simple "Rock/Paper/Scissors" into "Rock-Paper-Scissors combinations." That is, in RPS game, all possible moves are only "Rock," "Scissors," and "Paper"—the action space is very small (only 3); now we replace this with all possible assignments of a SAT problem—all possible combinations (n).

---

[5] If it were just a reduction within NP problems, it would only be "more complex," not a leap in difficulty; but here we need to go from P to NP, so it must be "harder".

2.3.3.2 Construction Following the core idea of SATization, we combinatorize the RPS game — a combinatorial RPS game, where victory in a single round requires the entire move combination to defeat the opponent's combination. Originally, defeating just one move was needed; now, a whole set of moves must be defeated. For example, if the opponent simultaneously plays Rock-Paper-Scissors, then one would need the combination Paper-Scissors-Rock:

$$\begin{aligned} &P \to R \\ &S \to P \\ &R \to S, \end{aligned}$$

But, if we use such fixed combinations, e.g., if we only ever play three moves at once, the action space is constant and can still be solved by brute-force enumeration—falling into P. Therefore, we need non-fixed "arbitrary combinations," making the action space arbitrarily large (n , non-constant)—jumping out of P, while remaining a simultaneous game so as not to exceed NP:

$$\begin{cases} P \to R \\ S \to P \\ R \to S \end{cases} \to \begin{cases} \vdots \\ R \to S \\ P \to R \\ S \to P \\ R \to S \\ P \to R \\ \vdots . \end{cases}$$

To make the candidate actions combinatorial and the action space dependent on n, the core is to introduce exponential scaling in the problem's input. For example, in the original RPS, the input is 2 players and 3 actions, with only 3 outcomes leading to a win for a player. To create exponential scaling, one can input

- n players with 3 actions;
- 2 players with n actions;
- still 2 players and 3 actions, but with each player having n hands...

If there are originally 3 actions (a fixed number) and we input n players, then the number of actions changes from 3 to 3^n:

$$3 \times 3 \times \cdots \times 3 = 3^n;$$

if we fix 2 players and input n actions, then the number of actions becomes n^2:

$$n \times n = n^2;$$

if we fix 2 players and 3 actions and input n hands, then the number of actions becomes 3^{2n}:

$$3^n \times 3^n = 3^{2n}.$$

It is worth noting that although the second method, because it involves n, can encode arbitrarily large information through accumulation and can also be used for NP-completeness constructions, it only makes the number of actions polynomial rather than exponential. This growth is not sufficiently "steep" and may lead to less natural reductions or proofs, and is not a standard model. Therefore, we generally use the other two methods.

When the number of players, or the number of actions, or the number of hands becomes n, the difficulty of confrontation in each round increases dramatically, because the action space

becomes exponential in size — NP , or even SAT. Definition SAT-RPS/SAT-Symmetric Simultaneous Mutually-Exclusive Game (SAT-SSMG):

Given a Boolean formula ϕ (containing n Boolean variables $x_1, \ldots, x_n$), two players P_1 and P_2 engage in a symmetric, simultaneous, mutually exclusive game.

Action Space: Each player's action set is a complete truth assignment $\alpha \in \{0,1\}^n$;

Symmetry: Exchanging the sets and roles of the two Ps leaves the structure unchanged;

Simultaneity: Both players choose an assignment at the same time;

Payoff Function (Win/Loss Determination):

- P_1 wins if and only if

$$[\phi(\alpha_1) = \text{True}] \wedge [\phi(\alpha_2) = \text{False}];$$

- P_2 wins if and only if

$$[\phi(\alpha_2) = \text{True}] \wedge [\phi(\alpha_1) = \text{False}].$$

Therefore, the game is mutually exclusive: there does not exist any pair of action sets (α_x, α_y), including those (α_1, α_2) that would make P_1 and P_2 win respectively, such that both players win simultaneously. Moreover, due to symmetry, this mutual exclusivity can be utilized by either side — if a p-t algorithm existed, the non-existent (α_x, α_y) exist — contradiction. Hence, there is no p-t algorithm for SAT-SSMG.

Furthermore, due to the mutual exclusivity, the game requires a "winning certificate" — an ordered pair (α, β). This is because a player must choose an assignment that makes the formula true, while simultaneously ensuring that the opponent has at least one choice that makes the formula false. (Otherwise, no matter what the opponent chooses, the formula would be true, resulting only in a draw or a win for the other player.) Thus, the verifier needs to determine whether there exists another action set β such that

$$[\phi(\alpha) = \text{True}] \wedge [\phi(\beta) = \text{False}]:$$

- Verify $\phi(\alpha) = \text{True}$.
- Verify $\phi(\beta) = \text{False}$.

Both steps are substitution computations taking $O(n)$ time. Therefore, verification is polynomial, and the problem belongs to NP.

In the above game, for a player to guarantee a win, they must choose an α that makes ϕ true. That is, a winning strategy exists in this game if and only if ϕ is satisfiable, which is equivalent to "finding an assignment that satisfies all clauses." Hence, it is a form of SAT, and the game inherits the NP-completeness of SAT: As is commonly feared, breaking cryptographic systems can be naturally encoded as SAT instances; breaking a cipher is equivalent to finding an action set that makes ϕ true in the SAT-RPS/SAT-SSMG game. Since SAT-SSMG admits no deterministic p-t algorithm, any cryptographic system whose security relies on its intractability remains secure in polynomial time.

At the same time, we observe that even a special model such as SAT-RPS, which may appear non-trivial to reduce from, can be reduced to by any NP problem — this is a manifestation of the Cook-Levin theorem (as in the Supplement to 2.3.1). The reduction only requires that "the existence of a solution" be preserved; it does not require that "the real-world circumstances of the solver" be simulated. (For instance, the lack of mutual exclusivity in decryption problems is likely

to be questioned, but it need not be simulated, even if one could construct mutual exclusivity in decryption. Mutual exclusivity is an artificially imposed rule used to reinforce the "uniqueness of the solution," but it does not alter the existence of the solution.) The equivalence between "the existence of a key" and "the existence of a winning strategy" is established through the strategy of "persistently choosing a true assignment," and this equivalence relies entirely on logical correspondence, not on any real-world adversarial component.

The reduction does not consist in "making the game simulate SAT"; rather, it consists in "encoding a SAT instance as an input to the game, such that the game's solution corresponds exactly to the solution of the SAT instance." That is, if the original NP problem (e.g., decryption) has a solution, then that solution itself constitutes a winning strategy in the game. The reduction from any NP problem to SAT-SSMG is not merely legitimate—it is trivial, for it proceeds entirely through SAT as an intermediary. Therefore, SAT-SSMG has no p-t algorithm, and consequently, all NP problems represented by SAT-SSMG have no p-t algorithm:

$$\mathrm{P} \neq \mathrm{NP}.$$

Notes (1) This is because I used RPS as an example. For other NP problems to be reduced to a SAT problem, merely including n in the input might only increase the problem scale, not the action space. For instance, for the problem "which number is larger," even if the candidates range from $-\infty$ to $+\infty$, the action space can remain very small—I could simply always choose $+\infty$, which would always be correct, as nothing is larger. RPS, however, does not ask "which action is better," but "which action defeats the opponent's action," strictly examining the opponent's move. This cyclic countering structure with no absolutely optimal move allows the action space to naturally grow exponentially with the input size. So, for other NP problems, specific reduction methods must be constructed.

(2) The winning condition of SAT-SSMG requires "the existence of an assignment β that makes the formula false," for a tautology (where every assignment makes the formula true), such a β does not exist. This appears to break the reduction's equivalence, since a standard SAT instance that is satisfiable would have no winning strategy in SAT-SSMG. However, this critique rests on a fundamental misconception: it confuses "reduction" with "problem equivalence." The "solution to a SAT-SSMG instance" is not the "solution to the original SAT instance." A reduction is a transformation—an encoding—not a subset relation. A valid polynomial time reduction does not require the solution set of the target problem (SAT-SSMG) to coincide with that of the source problem (SAT). It only requires the existence of a polynomial time transformation f that maps every instance of the source problem to an instance of the target problem, such that the property of "having a solution/having no solution" is preserved.

By definition, SAT-SSMG does not require "the existence of a false assignment" to be an additional hard problem that must be solved. It is merely a logical fact observed in proving satisfiability—a fact that necessarily exists for any satisfiable non-tautological formula—and it serves as a logical tool used in proving equivalence. In the construction of the reduction, we do not even need to "find" this false assignment; we only need to know that it exists logically. What we prove is this: ϕ is satisfiable (a true assignment exists) if and only if the player has a winning strategy in this game. This proof is standard, direct, and relies perfectly on the Cook–Levin theorem. Therefore, the reduction is both legitimate and fully adequate.

If a treatment for the tautology case is indeed required: For a satisfiable formula, P_1's strategy is to consistently select a satisfying assignment. If the opponent chooses a falsifying assignment, P_1 wins; if the opponent also chooses a satisfying assignment, the result is a draw. However, draws can be replayed, and since the number of satisfying assignments is finite, either the opponent is forced into a losing move or the draws exhaust all possibilities. Both outcomes—a decisive loss or an infinite loop—demonstrate the non-existence of a p-t algorithm, as such an algorithm would be unable to terminate with a decisive result. Alternatively, the special case of tautologies can be eliminated via a polynomial time preprocessing step: By introducing a new variable y and constructing $\phi' = \phi \wedge y$, the formula is transformed into a satisfiable, non-tautological formula, thereby ensuring that the condition "there exists a falsifying assignment" is always met. This preprocessing avoids the boundary case of tautologies and guarantees that the precondition for the existence of a falsifying assignment holds uniformly.

(3) If we choose to introduce a new variable (without relying on the multi-round mechanism) to handle the tautology case: Low-move, single-move models, such as the original RPS, can also be used to construct NPC SSMG problems—jumping out of P because P problems have few moves, but some P problems can be quickly transformed into an NP problem by adding a constraint: although few moves allow fast enumeration, if a single move is very hard—e.g., asking "is there a way to win immediately in the first move," disallowing subsequent moves, thus rejecting enumeration and requiring a conventional algorithm—then it becomes an NP problem: verifying whether a first move wins is simple, but finding such a move is hard. A P class problem can be transformed into an NP problem under a specific constraint, like the "win-in-one-move" rule. Then, it possesses the properties of NP, symmetry, simultaneity, and mutual exclusivity, only needing to be SATized. However, as noted at the beginning of 2.3.3, this is non-standard, so it is merely a supplement: a low-move model does not necessarily have to be extended to many moves to become an NP problem, and if the core of becoming an NP problem is creating an exponential action space, such an action space can also occur within a single move.

Moreover, once we consider single-move exponential computation, we can better refine our SAT-RPS—only requires combinatorial victory, but what exactly is "combinatorial victory"? Does it mean the player whose combination defeats more of the opponent's actions wins, or that one's entire combination must defeat all of the opponent's actions to win? Although it is generally assumed by default that all actions/the entire action combination must win (the latter), we realize that without specifying the victory condition, we wouldn't know whether to count a round as a win or skip to the next round—if there is a draw. Wouldn't that make the game prone to being non-mutually-exclusive, just like coin tossing mentioned in 2.3.3.1? Now, we can define SAT-RPS as a single-round action combination confrontation, and it does not lose its NP-ness due to being a "single-step" game—the action space remains exponential. It is somewhat similar to the case of 2 players with n actions, where the space is n^2 — non-standard, but usable. Of course, this supplement is not necessary. "Allowing draws" never affects the construction of an NPC SSMG problem, we can simply treat "draws" as non-outcomes to be skipped. As long as it is SAT, we can always add this extra requirement according to the Cook-Levin theorem. It's just that supplementing the victory condition for a single-round confrontation makes the SAT-RPS more standardized, coherent, and natural.

3 Reductio ad Absurdum of P =AP/CH : Guarantee of Winning Contradiction of Symmetric Mutually-Exclusive Game Problems

As you can see, although the above focuses on NP problems, the contradiction between p-t algorithms and Symmetric Mutually-Exclusive Games (SMG, not limited to SSMG, including but not limited to simultaneous cases) actually implies not only $P \neq NP$, but also that problems of difficulty above NP are also not equal to P. As mentioned in 2.3.1, we first discovered that SMGs like chess, which are EXPTIME-complete or even EXPSPACE-complete, have no p-t algorithms, and only then realized the need to convert to NP problems. That is, if we allow the alternating-move form of games to generate dependency chains, we can directly observe the contradiction between p-t algorithms and SMGs in general, not just SSMGs. However, since the theme revolves around the P/NP problem, we only formally constructed the case for NP problems; now that we have addressed the NP case, we can come back and supplement: in any symmetric, mutually exclusive deterministic game (whether simultaneous, fixed-turn, or arbitrary-turn), there is no guarantee of winning (GoW), including by p-t algorithms (not just p-t algorithms, but any method claiming to guarantee a win is false). Because if both sides use the same method, its deterministic output under symmetric initial conditions inevitably leads to a contradiction (a draw or a double-win paradox).

Since GoW is incompatible with SMG, it follows that for PH, PSPACE, EXPTIME, EXPSPACE, and even Undecidable (U), as long as each has its own SMG that is complete, none of them can have GoW—

(1) PH contains P, theoretically, even if a PH-complete (PHC) SMG existed, it would not by itself imply $P \neq PH$, but since $P \neq NP$ (2.3.3.2), at least $\Sigma_1 P$ does not collapse to $\Sigma_0 P$, consequently, PH cannot collapse to P, and hence $P \neq PH$. But this conclusion is reached indirectly via $P \neq NP$, if one seeks an independent treatment: PH is an infinite union of layers, there is no single "PHC" problem—only complete problems for each individual layer, hence no PHC-SMG exists. To represent other PH problems, we cannot rely on "completeness." Instead, we construct a $\Sigma_i P$-SMG (SMG for each layer $\Sigma_i P$): both players assign values to the variables of a Boolean formula ϕ, and the winner is determined by the truth value of ϕ. By swapping the first move, the construction extends to $\Pi_i P$—The same rule applies uniformly for any i. The parameter i serves as a substitute for completeness, enabling representation of other PH problems. Therefore, the earlier statement "each has its own SMG and is complete, hence none has GoW" should be revised to: "each has its own SMG and collectively provides representativeness (PHC → PHR ∋ Σ_iP-SMG/PHR-SMG), hence none has GoW."

(2) The standard QBF game[6] is a classic PSPACE-complete problem[7], but it is only mutually exclusive, not symmetric—given a QBF formula ϕ, the variable grouping is changed from quantifier-prefix order to a predetermined systematic partition (e.g., by parity), not tied to logical roles. Players take turns assigning values to variable groups. (Groups are unbiased, assigned in index order regardless of quantifier type.) The player roles are no longer fixed as "existential" or "universal"; instead, they become completely equivalent, distinguished only by name. The payoff matrix no longer presumes who "pursues truth" or "pursues falsity"; rather, the outcome depends on "whose assignment makes the formula true." The trade-off is that this

[6] Chandra, A. K., Kozen, D. C., and Stockmeyer, L. J. *Alternation*.
[7] Stockmeyer, L. J. *The complexity of decision problems in automata theory and logic*.

converts the game from mutually-exclusive to zero-sum (allowing draws). However, adding the rule "ignore draws and restart the game" does not invalidate the equivalence proof: the answer (true/false) of the original QBF problem remains in one-to-one correspondence with "whether there exists a winning strategy" in our constructed SMG.

(3) EXPTIME and EXPSPACE contain generalized board games that are complete and conform to the symmetric mutually-exclusive structure (2.3.1).

(4) Since a "polynomial time reduction" is itself an "algorithmic" procedure, and U problems admit no algorithmic solution whatsoever, it is impossible to "compute" a reduction from an U problem to another problem — doing so would effectively decide the U one — U problems are not reduced via standard polynomial time reductions, but via a more direct "encoding" method—encoding the problem as an equivalent logical formula and then embedding it into SMG rules, i.e., constructing a G_{Halt} such that:

- If the program halts, then there exists a GoW in G_{Halt}.
- If the program does not halt, then there does not exist a GoW in G_{Halt}.

This reduction is not "polynomial time" but "logically constructible." If an algorithm exists that can guarantee a win in G_{Halt}, then it must be able to correctly determine and output a GoW on all inputs (including encodings of U problems). But if that algorithm could determine the GoW for G_{Halt} in polynomial time, then it could solve the Halting Problem in polynomial time — the Halting Problem has been proven to have no algorithm at all (regardless of time). It is impossible for even "any algorithm" to exist, let alone a "p-t algorithm."

The contradiction between GoW and SMGs is actually a more fundamental, universal contradiction. At different complexity levels above P (AP, from NP to U. And we can see that the name "NP" is not very accurate, because when P is the simplest, non-P is of course above P—AP), it can be used to prove that there is no p-t algorithm in that level:

$$\mathrm{P} \neq \mathrm{AP}.$$

This conclusion does not depend on whether the problem belongs to P, NP, PH, PSPACE, EXPTIME, EXPSPACE, or U, it is a conclusion independent of complexity levels, and it is not a new conclusion but merely another interpretation of existing facts. That is, theoretically the order is: there exists a phenomenon, and the contents of the complexity hierarchy also exhibit this phenomenon. That is, in 2.2, we actually first discovered the contradiction between GoW and SMG, and then discovered the contradiction between p-t algorithm and SMG — p-t algorithm (in SMG) is one of the GoW.

Then, why AP rather than all difficulty levels? Theoretically, P-class also certainly have SMGs. As mentioned in 2.3.1, the original RPS exists and it could very well be "P-complete (PC)," then it should not be "P that are not reducible within PH can also be proven to have no GoW" and P ≠ Computability Hierarchy (CH, all difficulty levels. It cannot be CC — Complexity Classes, since Complexity Classes do not encompass U), ultimately revealing an even more severe contradiction P ≠ P? The reason is that P is reduced to PC via log-space reduction, while the input space of the original RPS is extremely small, and its output space is even smaller, always constant, this does not satisfy the requirement that the output length of a log-space reduction must be polynomial in the length of the input, thus it is impossible to encode complex P into trivial games like the original RPS. However, our path is no longer about proving that P have no p-t algorithms, it is about proving that SMG has no GoW. As in the case of PH/U, P do not need to be unified

through traditional reductions. We have shifted from examining "completeness" to examining "representativeness ($\mathrm{PC} \to \mathrm{PR}$)." The original RPS, which can be embedded into any ordinary P (i.e., any ordinary P can be complicated into it), possesses representativeness. Since it has no GoW, ordinary P have no p-t algorithms either. Therefore, from the computational direction and the logical direction, there is one conclusion each:

$$\begin{cases} \mathrm{P} \neq \mathrm{AP}, \\ \mathrm{P} \neq \mathrm{CH}. \end{cases}$$

From the computational direction, AP problems all lack p-t algorithms; from the logical direction, CH problems all lack GoWs, and the definition of P leads to a contradiction—it cannot equal itself. The very definitions of fundamental concepts in computational complexity theory need to be reconsidered:

(1) The "p-t algorithm" that P is believed to possess always operates under certain conditions—it is not a universal algorithm for all cases. (I.e., it does not necessarily yield a result; rather, it is a type of algorithm that strictly examines the environment.) That is not the same as the "p-t algorithm" we are discussing.

(2) The "p-t algorithm" we are discussing is, in fact, one that is required to always produce a result (and to be universally applicable to any instance, as long as it falls within the relevant complexity class), because it is an algorithm for a complete problem, and complete problems have representativeness. This distinction was not recognized when discussing NP or higher complexity classes—it only becomes apparent when discussing P, since P in theory has p-t algorithms, yet PR-SMG is indeed representative (i.e., it has no p-t algorithm).

This means our discussion of "p-t algorithm" has not been sufficiently accurate.

References

Chandra, A. K., Kozen, D. C., and Stockmeyer, L. J. 1981. "Alternation." *J. ACM* 28, 1 (Jan. 1981).

Stephen A. Cook. 1971. "The complexity of theorem-proving procedures." In *Proceedings of the third annual ACM symposium on Theory of computing*.

Stockmeyer, L. J. 1974. *The complexity of decision problems in automata theory and logic*. Ph.D. Dissertation. Massachusetts Institute of Technology, Cambridge, MA, USA.