P versus NP

Frank Vega

January 10, 2020

Open Problem

P versus NP is considered as one of the most important open problems in computer science. This consists in knowing the answer of the following question: Is P equal to NP? A precise statement of the P versus NP problem was introduced independently by Stephen Cook and Leonid Levin. Since that date, all efforts to find a proof for this problem have failed. We show some results that prove this outstanding problem with the unexpected solution of $P = NP$.

L versus NL

A logarithmic space Turing machine has a read-only input tape, a write-only output tape, and read/write work tapes. The work tapes may contain at most $O(\log n)$ symbols. In computational complexity theory, L is the complexity class containing those decision problems that can be decided by a deterministic logarithmic space Turing machine. NL is the complexity class containing the decision problems that can be decided by a nondeterministic logarithmic space Turing machine. Whether $L = NL$ is another fundamental question that it is as important as it is unresolved.

The problem *XOR 2SAT*

We denote the *XOR* function as $\oplus$. The *XOR 2SAT* problem will be equivalent to *XOR SAT*, but the clauses in the formula have exactly two distinct literals. *XOR 2SAT* is in L.

A certificate-based definition for *NL*

We can give a certificate-based definition for *NL*. The certificate-based definition of *NL* assumes that a logarithmic space Turing machine has another separated read-only tape. On each step of the machine, the machine's head on that tape can either stay in place or move to the right. In particular, it cannot reread any bit to the left of where the head currently is. For that reason this kind of special tape is called "read-once".

Definition of an NL problem

Definition 1

A language L_1 is in NL if there exists a deterministic logarithmic space Turing machine M with an additional special read-once input tape polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \{0,1\}^*$:

$$x \in L_1 \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M(x,u) = \text{“yes”}$$

where by $M(x,u)$ we denote the computation of M where x is placed on its input tape and the certificate u is placed on its special read-once tape, and M uses at most $O(\log |x|)$ space on its read/write tapes for every input x where $|\ldots|$ is the bit-length function. M is called a logarithmic space verifier.

Hypothesis 2

Given a nonempty language $L_1 \in L$, there is a language L_2 in NP–complete with a deterministic Turing machine M, where:

$$L_2 = \{w : M(w, u) = y,\ \exists \text{ string } u \text{ such that } y \in L_1\}$$

when M runs in logarithmic space in the length of w, u is placed on the special read-once tape of M, and u is polynomially bounded by w. In this way, there is an NP–complete language defined by a logarithmic space verifier M such that when the input is an element of the language with its certificate, then M outputs a string which belongs to a single language in L.

Two-way and one-way Turing machines

From the early days of automata and complexity theory, two different models of Turing machines are considered, the offline and online machines. Each model has a read-only input tape and some work tapes. The offline machines may read their input two-way while the online machines are not allowed to move the input head to the left. In the terminology of the (generalized) Turing machine models are called two-way and one-way Turing machines, respectively.

One-way logarithmic space Turing machine

Hartmanis and Mahaney have investigated the classes $1L$ and $1NL$ of languages recognizable by deterministic one-way logarithmic space Turing machine and nondeterministic one-way logarithmic space Turing machine, respectively. They have shown that $1L \neq 1NL$ (by looking at a uniform variant of the string non-equality problem from communication complexity theory) and have defined a natural complete problem for $1NL$ under deterministic one-way logarithmic space reductions. Furthermore, they have proven that $1NL \subseteq L$ if and only if $L = NL$.

When the Hypothesis 2 is true

Theorem 3

If the Hypothesis 2 is true, therefore when $L \neq NL$, then $P = NP$.

Proof

We can simulate the computation $M(w, u) = y$ in the Hypothesis 2 by a nondeterministic logarithmic space Turing machine N such that $N(w) = y$, since we can read the certificate string u within the read-once tape by a work tape in a nondeterministic logarithmic space generation of symbols contained in u. Certainly, we can simulate the reading of one symbol from the string u into the read-once tape just nondeterministically generating the same symbol in the work tapes using a logarithmic space.

Proof

If we suppose that $L \subset 1NL$, then we can accept the elements of the language $L_1 \in L$ by a nondeterministic one-way logarithmic space Turing machine M'. In this way, there is a nondeterministic logarithmic space Turing machine $M''(w) = M'(N(w))$ which will accept when $w \in L_2$. Consequently, M'' is a nondeterministic logarithmic space Turing machine which decides the language L_2.

Proof

The reason is because we can simulate the output string of $N(w)$ within a read-once tape and thus, we can compute in a nondeterministic logarithmic space the logarithmic space composition using the same techniques of the logarithmic space composition reduction, but without any reset of the computation. Certainly, we do not need to reset the computation of $N(w)$ for the reading at once of a symbol in the output string of $N(w)$ by the nondeterministic one-way logarithmic space Turing machine M'.

Proof

Therefore, L_2 is in NL and thus, $L_2 \in P$ due to $NL \subseteq P$. If any single *NP–complete* problem can be solved in polynomial time, then $P = NP$. Since $L_2 \in P$ and $L_2 \in$ *NP–complete*, then we obtain the complexity class P is equal to NP under the assumption that $L \subset 1NL$.

Proof

Hartmanis and Mahaney have also shown with their result that if $1NL \subseteq L$ or even $1NL \subset L$, then $L = NL$, because they proved there is a complete problem for both $1NL$ and NL at the same time. In this way, if $L \neq NL$, then $L \subset 1NL$ by contraposition. Since we already obtained that $P = NP$ under the assumption that $L \subset 1NL$, therefore if $L \neq NL$, then $P = NP$. □

Definition of a *coNL* problem

Definition 4

A language L_1 is in *coNL* if there exists a deterministic logarithmic space Turing machine M with an additional special read-once input tape polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \{0,1\}^*$:

$$x \in L_1 \Leftrightarrow \forall \textit{ appropriated } u \in \{0,1\}^{p(|x|)} \textit{ then } M(x,u) = \text{“}\textit{yes}\text{”}$$

where by $M(x,u)$ we denote the computation of M where x is placed on its input tape and the disqualification u is placed on its special read-once tape, and M uses at most $O(\log|x|)$ space on its read/write tapes for every input x where $|\ldots|$ is the bit-length function. M is called a logarithmic space disqualifier.

Example

For example, there is a well-known *coNL* problem that states: Given a directed graph $G = (V, E)$ and two nodes $s, t \in V$, is there no possible path from s to t? In that problem, an appropriated disqualification u is a sequence of nodes contained in V when s is the first node and t is the last one such that this sequence of nodes is not a path: There is at least a consecutive pair of nodes in the sequence where they are not connected by an edge.

Hypothesis 5

Given a nonempty language $L_1 \in 1NL$, there is a language L_2 in coNP–complete with a deterministic Turing machine M, where:

$$L_2 = \{w : M(w, u) = y, \ \forall \text{ appropriated string } u \text{ such that } y \in L_1\}$$

when M runs in logarithmic space in the length of w, u is placed on the special read-once tape of M, and u is polynomially bounded by w. In this way, there is a coNP–complete language defined by a logarithmic space disqualifier M such that when the input is an element of the language with any of its appropriated disqualification, then M always outputs a string which belongs to a single language in $1NL$.

When the Hypothesis 5 is true

Theorem 6

If the Hypothesis 5 is true, therefore when $L = NL$, then $P = NP$.

Proof

We can accept the elements of the language $L_1 \in 1NL$ by a nondeterministic one-way logarithmic space Turing machine M'. In this way, there is a nondeterministic logarithmic space Turing machine $M''(w,u) = M'(M(w,u))$ which will accept when $w \in L_2$ for all the appropriated disqualification u, where u is placed on the special read-once tape of M''.

Proof

The reason is because we can simulate the output string of $M(w, u)$ within a read-once tape and thus, we can compute in a nondeterministic logarithmic space the logarithmic space composition using the same techniques of the logarithmic space composition reduction, but without any reset of the computation. Certainly, we do not need to reset the computation of $M(w, u)$ for the reading at once of a symbol in the output string of $M(w, u)$ by the nondeterministic one-way logarithmic space Turing machine M'.

Proof

Consequently, M'' can be converted into a logarithmic space disqualifier for the language L_2 just assuming that $L = NL$, because of the nondeterministic logarithmic space Turing machine M'' could be simulated by a deterministic logarithmic space Turing machine. Therefore, L_2 is in $coNL$ and thus, $L_2 \in P$ due to $coNL \subseteq P$. If any single *coNP–complete* problem can be solved in polynomial time, then $P = NP$. Since $L_2 \in P$ and $L_2 \in$ *coNP–complete*, then we obtain the complexity class P is equal to NP under the assumption that $L = NL$. □

NAE 3SAT

Definition 7

INSTANCE: A Boolean formula ϕ in $3CNF$.
QUESTION: Is there a truth assignment for ϕ such that each clause has at least one true literal and at least one false literal?
REMARKS: $NAE\ 3SAT \in NP\text{–}complete$.

MAXIMUM EXCLUSIVE-OR 2SAT

Definition 8

INSTANCE: A positive integer K and a Boolean formula ϕ that is an instance of *XOR 2SAT*.
QUESTION: Is there a truth assignment in ϕ such that at most K clauses are unsatisfied?
REMARKS: We denote this problem as $MAX \oplus 2SAT$.

Theorem 9

$MAX \oplus 2SAT \in NP\text{–complete.}$

Proof

It is trivial to see $MAX \oplus 2SAT \in NP$. Given a Boolean formula ϕ in $3CNF$ with n variables and m clauses, we create three new variables a_{c_i}, b_{c_i} and d_{c_i} for each clause $c_i = (x \vee y \vee z)$ in ϕ, where x, y and z are literals, in the following formula:

$$P_i = (a_{c_i} \oplus b_{c_i}) \wedge (b_{c_i} \oplus d_{c_i}) \wedge (a_{c_i} \oplus d_{c_i}) \wedge (x \oplus a_{c_i}) \wedge (y \oplus b_{c_i}) \wedge (z \oplus d_{c_i}).$$

Proof

We can see P_i has at most one unsatisfied clause for some truth assignment if and only if at least one member of $\{x, y, z\}$ is true and at least one member of $\{x, y, z\}$ is false for the same truth assignment. Hence, we can create the Boolean formula ψ as the conjunction of the P_i formulas for every clause c_i in ϕ, such that $\psi = P_1 \wedge \ldots \wedge P_m$.

Proof

Finally, we obtain that:

$$\phi \in \textit{NAE 3SAT if and only if } (\psi, m) \in \textit{MAX} \oplus \textit{2SAT}.$$

Consequently, we prove $\textit{NAE 3SAT} \leq_p \textit{MAX} \oplus \textit{2SAT}$ where we already know the language $\textit{NAE 3SAT} \in \textit{NP–complete}$. To sum up, we show $\textit{MAX} \oplus \textit{2SAT} \in \textit{NP–hard}$ and $\textit{MAX} \oplus \textit{2SAT} \in \textit{NP}$ and thus, $\textit{MAX} \oplus \textit{2SAT} \in \textit{NP–complete}$. □

Hypothesis 2 proof

Theorem 10

There is a deterministic Turing machine M, where:

$$MAX \oplus 2SAT = \{w : M(w, u) = y, \exists \text{ string } u \text{ such that } y \in XOR\ 2SAT\}$$

when M runs in logarithmic space in the length of w, u is placed on the special read-once tape of M, and u is polynomially bounded by w.

Proof

Given a valid instance (ψ, K) for $MAX \oplus 2SAT$ when ψ has m clauses, we can create a certificate array A which contains K different natural numbers in ascending order which represents the indexes of the clauses in ψ that we are going to remove from the instance. We read at once the elements of the array A and we reject whether this is not an appropriated certificate: That is when the numbers are not sorted in ascending order, or the array A does not contain exactly K elements, or the array A contains a number that is not between 1 and m.

Proof

While we read the elements of the array A, we remove the clauses from the instance (ψ, K) for $MAX \oplus 2SAT$ just creating another instance ϕ for $XOR\ 2SAT$ where the Boolean formula ϕ does not contain the K different indexed clauses ψ represented by the numbers in A. Therefore, we obtain the array A would be valid according to the Theorem 10 when:

$(\psi, K) \in MAX \oplus 2SAT \Leftrightarrow (\exists\ appropriated\ array\ A\ such\ that\ \phi \in XOR\ 2SAT)$

Proof

Furthermore, we can make this verification in logarithmic space such that the array A is placed on the special read-once tape, because we read at once the elements in the array A and we assume the clauses in the input ψ are indexed from left to right. Hence, we only need to iterate from the elements of the array A to verify whether the array is an appropriated certificate and also remove the K different clauses from the Boolean formula ψ when we write the final clauses to the output. □

3UNSAT

Definition 11

INSTANCE: A Boolean formula ϕ in $3CNF$.
QUESTION: Is ϕ unsatisfiable?
REMARKS: $3UNSAT \in coNP\text{–}complete$.

SUM ZERO

Definition 12

INSTANCE: A collection of integers C such that $0 \notin C$ and every integer in C has the same bit-length of the number that represents the cardinality of C multiplied by 3 (we do not take into account the symbol minus in counting the bit-length of the negative integers).
QUESTION: Are there two elements $a, b \in C$, such that $a + b = 0$?
REMARKS: We denote this problem as $0SUM$.

Contained in $1NL$

Theorem 13

$0SUM \in 1NL$.

Proof

Given a collection of integers C, we can read its elements from left to right, verify that every element is not equal to 0, check that every element in C has the same bit-length and count the amount of elements in C to finally multiply it by 3 and compare its bit-length with the single bit-length from the elements in C. In addition, we can nondeterministically pick two elements a and b from C and accept in case of $a + b = 0$ otherwise we reject. We can make all this computation in a nondeterministic one-way using logarithmic space.

Proof

Certainly, the calculation and store of the bit-length of the elements in C could be done in logarithmic space since this is a unique value. On the one hand, we can count and store the number of elements that we read from the input and multiply it by 3 to finally compare its bit-length with the stored unique bit-length from the elements of the collection, since the cardinality of C multiplied by 3 could be stored in a binary number of bit-length that is logarithmic in relation to the encoded length of C.

Proof

On the other hand, the two elements a and b that we pick from C have a logarithmic space in relation to the encoded length of C, because of every integer in C has the same bit-length of the number that represents the cardinality of C multiplied by 3. Indeed, we never need to read to the left on the input for the acceptance of the elements in $0SUM$ in a nondeterministic logarithmic space. □

Hypothesis 5 proof

Theorem 14

There is a deterministic Turing machine M, where:

$3UNSAT = \{w : M(w, u) = y, \forall$ *appropriated string* u *such that* $y \in 0SUM\}$

when M runs in logarithmic space in the length of w, u is placed on the special read-once tape of M, and u is polynomially bounded by w.

Proof

Given a Boolean formula ϕ in $3CNF$ with n variables and m clauses, we can create a disqualification array A which contains m positive integers between 1 and 3 which represents the literals of the clauses in ϕ which appear from left to right. We read at once the elements of the array A and we reject whether this is not an appropriated disqualification: That is when the array A does not contain exactly m elements, or the array A contains a number that is not between 1 and 3.

Proof

While we read the elements of the array A, we select from the clauses ϕ the literals such that these ones occupy the position that represents the number between 1 and 3, that is the first, second or third place within the clause from left to right. In this way, we output the selected literals that are represented by a positive or negative (in case of a negated variable) integer just creating another instance C for $0SUM$ where the collection C contains those integers which are the selected literals for each clause in ϕ.

Proof

Therefore, we obtain that all the appropriated array A would be valid according to the Theorem 14 when:

$$\phi \in 3UNSAT \Leftrightarrow (\forall \textit{ appropriated array } A \textit{ such that } C \in 0SUM)$$

since we assume the positive and negated literals of some variable in the input ϕ correspond to a positive integer and its negative value, respectively.

Proof

Furthermore, we can make this disqualification in logarithmic space such that the array A is placed on the special read-once tape, because we read at once the elements in the array A. Hence, we only need to iterate from the elements of the array A to verify whether the array is an appropriated disqualification and pick the m literals from the Boolean formula ϕ when we write the final integers that represent these literals to the output.

Proof

Note, that every possible literal in ϕ could have a representation by an integer between $-3 \times m$ and $3 \times m$ with the exception of 0, where m is the cardinality of the collection C. In this way, we guarantee the output collection C is an appropriated instance of $0SUM$ just filling with zeroes to the left the elements with bit-length lesser than $|3 \times m|$ where $|\ldots|$ is the bit-length function. $\square$

Final Result

Theorem 15

$P = NP$.

Proof

If $L \neq NL$, then $P = NP$ as a consequence of the Hypothesis 2. If $L = NL$, then $P = NP$ as a consequence of the Hypothesis 5. Since we have either $L \neq NL$ or $L = NL$ is true, then the complexity class P is equal to NP. $\square$

THE END

QUESTIONS?