P versus *NP*

The proof on
`https://www.preprints.org/manuscript/201908.0037/v1` [1]

Frank Vega

November 5, 2019

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 $P = NP$.

Another definition of NP

We can define an NP problem as

$$L2 = \{w : M(w, c) = y \ \textit{where } y \in L1\}(\textit{y is the output in the halting state})$$

when $L1$ is a language in P, M is a deterministic Turing machine that runs in polynomial time in the length of w and c (the certificate) is polynomially bounded by w.

Equivalent definition

This is easy to prove since the polynomial time composition reduction can be done in polynomial time. Therefore, the previous definition complies with the existence of a polynomial time verifier M such that the NP problem $L2$ is defined as

$$L2 = \{w : M'(w, c) = \text{``}yes\text{''}\}(\textit{``yes'' is the acceptance state})$$

where the computation of $M'(w, c)$ is equal to $N(M(w, c))$ when N is the Turing machine which accepts $L1$ in polynomial time.

The logarithmic space verifier

On the other hand, the verifier for a language in *NL* (nondeterministic logarithmic space class) has as premise that the certificate c is placed in a special tape that is read only and read at once (that means we cannot read to the left).

Our result

Theorem 1

In our work was proved that we can define a NP–complete problem as

$$L2 = \{w : M(w, c) = y \text{ where } y \in L1\}(y \text{ is the output in the halting state})$$

when L1 is a language in L (deterministic logarithmic space class), M is a deterministic Turing machine that runs in logarithmic space in the length of w and c (the certificate) is polynomially bounded by w such that c is placed in the special tape in M that is read only and read at once.

Converting the verifier

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

The classes $1L$ and $1NL$

Definition 2

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. The one-way machines are not allowed to move the input head to the left.

Supposition

If we suppose that $L \subset 1NL$, then we can accept the elements of the language $L1 \in L$ 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 halt in the acceptance state when $w \in L2$. Consequently, M'' is a nondeterministic logarithmic space Turing machine which decides the language $L2$.

The arguments

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'.

Consequences

Therefore, $L2 \in NL$ and thus, $L2 \in P$ due to $NL \subseteq P$. If any single *NP–complete* problem can be solved in polynomial time, then $P = NP$. Since $L2 \in P$ and $L2 \in$ *NP–complete*, then we obtain the complexity class P is equal to NP under the assumption that $L \subset 1NL$. Hartmanis and Mahaney have also shown with their result that if $1NL$ is a subset of L, then $L = NL$, because they proved there is a complete problem for both $1NL$ and NL at the same time. If 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 $P = NP$.

The classes $2L$ and $2NL$

Definition 3

The class 2L contains those languages that are deterministic logarithmic space reduced to a language in 1L. The class 2NL consists in those languages that are nondeterministic logarithmic space reduced to a language in 1NL.

Inclusion

We obtain that $2L \subseteq L$ and $2NL \subseteq NL$ by the definition of L and NL under the *L–reduction* and *NL–reduction*, respectively. Certainly, since the output string will be in $1L$ or $1NL$, then we do not need to reset the computation of the *L–reduction* or *NL–reduction* in order to be decided by a deterministic or nondeterministic logarithmic space Turing machine under logarithmic space composition, respectively.

Proof of $L = 2L$: First part

On the one hand, every language in $L' \in L$ which is decided by a deterministic logarithmic space Turing machine M could be *L–reduced* to a language in $1L$. The reason is simple, because the Turing machine M could output the sequence of symbols that is read continuously in the input tape during the whole computation in case of acceptance. Indeed, for every read symbol in the input tape in M, then this is written to the output tape in a sequential way.

Proof of $L = 2L$: Second part

In this way, the output string could be accepted by a deterministic one-way logarithmic space Turing machine, that would be the same Turing machine M where this one will always read on the input tape from left-to-right. Certainly, when M tries to move the head of the input tape to the left into the output string, then this will move contiguously to the right. Consequently, M in case of acceptance will output the strings that consist in a language in $1L$. Hence, we obtain that $L \subseteq 2L$. Since we already know that $2L \subseteq L$, then $L = 2L$.

Proof of $NL = 2NL$: First part

On the other hand, every language in $L'' \in NL$ which is decided by a nondeterministic logarithmic space Turing machine N could be *NL–reduced* to a language in $1NL$. The reason is simple, because the Turing machine N could output the sequence of symbols that is read continuously in the input tape during the whole computation in case of acceptance. Indeed, for every read symbol in the input tape in N, then this is written to the output tape in a sequential way.

Proof of $NL = 2NL$: Second part

In this way, the output string could be accepted by a nondeterministic one-way logarithmic space Turing machine, that would be the same Turing machine N where this one will always read on the input tape from left-to-right. Certainly, when N tries to move the head of the input tape to the left into the output string, then this will move contiguously to the right. Consequently, N in case of acceptance will output the strings that consist in a language in $1NL$. Hence, we obtain that $NL \subseteq 2NL$. Since we already know that $2NL \subseteq NL$, then $NL = 2NL$.

Conclusion: $P = NP$

In this way, if $L = NL$ then $2L = 2NL$. However, we know that $2L \neq 2NL$ because of $1L \neq 1NL$ since Hartmanis and Mahaney have shown that $1L \neq 1NL$ by looking at a uniform variant of the string non-equality problem from communication complexity theory. Therefore, we prove the complexity class L is not equal to NL. Since we show that $L \neq NL$, then we prove the complexity class P is equal to NP.

Code

In this Scala Project [2], we use the Assertion on the properties of the instances of each problem and the Unit Test for checking the correctness of the Theorem 1. We need to install JDK 8 in order to test the Scala Project. In addition, we need to install SBT to run the unit test (we could run the unit test with the **sbt test** command).

References

Frank Vega.
Logarithmic Space Verifiers on NP-complete.
In *Preprints*, 2019, 2019080037.
doi:10.20944/preprints201908.0037.v1.

Frank Vega.
VerifyReduction, August 2019.
In a GitHub repository at
https://github.com/frankvegadelgado/VerifyReduction.