P versus NP

The proof on

https://www.preprints.org/manuscript/201908.0037/v1 [1]

Frank Vega

November 18, 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

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

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

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

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 an NP–complete problem as

$L2 = \{w : M(w, c) = y \text{ where } y \in L1\}$(*y 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 complexity class *LNL*

Definition 3

The class *LNL* contains those languages that are decided by a nondeterministic logarithmic space Turing machine N such that for every element $x = yz$ of these languages, there are a prefix and suffix substrings y and z where N moves strictly deterministically on y and strictly nondeterministically on z when strictly deterministically means there is no a possible nondeterministic step and strictly nondeterministically means there is at least one nondeterministic step on the computation.

The class *BNL*

Sauerhoff has investigated the class *BNL* of languages recognizable by nondeterministic logarithmic space Turing machine, that only use nondeterministic moves before reading their input ("nondeterminism at the beginning").

Proof of $LNL = NL$

We have that $NL \subseteq LNL$, because the prefix substring y of an instance $x = yz$ could be the empty string. Certainly, all the languages in the class NL could be decided by nondeterministic logarithmic space Turing machines such that they always do a single nondeterministic step after the original acceptance state choosing nondeterministically the same acceptance state within two equals choices from the original and modified deterministic or nondeterministic logarithmic space Turing machines which decide these languages (this is assuming the prefix substring y is the empty string in the elements $x = yz$). Moreover, we have that $LNL \subseteq NL$, because the languages in LNL are decided by nondeterministic logarithmic space Turing machines. Since $NL \subseteq LNL$ and $LNL \subseteq NL$, then the complexity class LNL is equal to NL.

Proof of $L \neq NL$

However, we can state that $BNL \neq LNL$, because there is no possible way over the Definition 3 of LNL for an element $x = yz$ in some languages in BNL when the prefix substring y is the empty string, such that we could move strictly nondeterministically on y (that would be a "nondeterminism at the beginning") and move strictly deterministically on the nonempty string z. Hence, we obtain that $BNL \neq NL$ by transitivity. Nevertheless, Sauerhoff has also shown that $L \subseteq BNL \subseteq NL$. Consequently, we prove the complexity class L is not equal to NL.

Conclusion: $P = NP$

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.