# Logarithmic Space Verifiers on NP-complete

**Frank Vega**
Joysonic, Uzun Mirkova 5, Belgrade, 11000, Serbia
vega.frank@gmail.com

**Abstract**

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. NP is the complexity class of languages defined by polynomial time verifiers 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 P. Another major complexity classes are L and NL. The certificate-based definition of NL is based on logarithmic space Turing machine with an additional special read-once input tape: This is called a logarithmic space verifier. NL is the complexity class of languages defined by logarithmic space verifiers M such that when the input is an element of the language with its certificate, then M outputs 1. To attack the P versus NP problem, the NP-completeness is a useful concept. We demonstrate 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. In this way, we obtain that L = NL and P = NP cannot be both false at the same time.

**2012 ACM Subject Classification** Theory of computation → Complexity classes; Theory of computation → Problems, reductions and completeness

**Keywords and phrases** complexity classes, completeness, verifier, reduction, polynomial time, logarithmic space

## 1 Introduction

In previous years there has been great interest in the verification or checking of computations [11]. Interactive proofs introduced by Goldwasser, Micali and Rackoff and Babi can be viewed as a model of the verification process [11]. Dwork and Stockmeyer and Condon have studied interactive proofs where the verifier is a space bounded computation instead of the original model where the verifer is a time bounded computation [11]. In addition, Blum and Kannan has studied another model where the goal is to check a computation based solely on the final answer [11]. More about probabilistic logarithmic space verifiers have been shown on a technique of Lipton [11]. In this work, we show some results about the logarithmic space verifiers applied to the class $NP$ which impact into one of the most important open problems in computer science, that is $P$ versus $NP$.

## 2 Motivation

The $P$ versus $NP$ problem is a major unsolved problem in computer science [4]. This is considered by many to be the most important open problem in the field [4]. It is one of the seven Millennium Prize Problems selected by the Clay Mathematics Institute to carry a US$1,000,000 prize for the first correct solution [4]. It was essentially mentioned in 1955 from a letter written by John Nash to the United States National Security Agency [1]. However, the precise statement of the $P = NP$ problem was introduced in 1971 by Stephen Cook in a seminal paper [4]. In 2012, a poll of 151 researchers showed that 126 (83%) believed the answer to be no, 12 (9%) believed the answer is yes, 5 (3%) believed the question may be independent of the currently accepted axioms and therefore impossible to prove or disprove, 8 (5%) said either do not know or do not care or don't want the answer to be yes nor the problem to be resolved [7]. It is fully expected that $P \neq NP$ [14]. Indeed, if $P = NP$ then there are stunning practical consequences [14]. For that reason, $P = NP$ is considered as a very unlikely event [14]. Certainly, $P$ versus $NP$ is one of the greatest open problems in science and a correct solution for this incognita will have a great impact not only in computer science, but for many other fields as well [1]. Whether $P = NP$ or not is still a controversial and unsolved problem [1]. We show some results that are a breakthrough in the future path of solving this outstanding problem.

## 3 Preliminaries

In 1936, Turing developed his theoretical computational model [16]. The deterministic and nondeterministic Turing machines have become in two of the most important definitions related to this theoretical model for computation [16]. A deterministic Turing machine has only one next action for each step defined in its program or transition function [16]. A nondeterministic Turing machine could contain more than one action defined for each step of its program, where this one is no longer a function, but a relation [16].

Let $\Sigma$ be a finite alphabet with at least two elements, and let $\Sigma^*$ be the set of finite strings over $\Sigma$ [3]. A Turing machine $M$ has an associated input alphabet $\Sigma$ [3]. For each string $w$ in $\Sigma^*$ there is a computation associated with $M$ on input $w$ [3]. We say that $M$ accepts $w$ if this computation terminates in the accepting state, that is $M(w) = 1$ (when $M$ outputs 1 on the input $w$) [3]. Note that $M$ fails to accept $w$ either if this computation ends in the rejecting state, that is $M(w) = 0$, or if the computation fails to terminate, or the computation ends in the halting state with some output, that is $M(w) = y$ (when $M$ outputs the string $y$ on the input $w$) [3].

Another relevant advance in the last century has been the definition of a complexity class. A language over an alphabet is any set of strings made up of symbols from that alphabet [5]. A complexity class is a set of problems, which are represented as a language, grouped by measures such as the running time, memory, etc [5]. The language accepted by a Turing machine $M$, denoted $L(M)$, has an associated alphabet $\Sigma$ and is defined by:

$$L(M) = \{w \in \Sigma^* : M(w) = 1\}.$$

We denote by $t_M(w)$ the number of steps in the computation of $M$ on input $w$ [3]. For $n \in \mathbb{N}$ we denote by $T_M(n)$ the worst case run time of $M$; that is:

$$T_M(n) = max\{t_M(w) : w \in \Sigma^n\}$$

where $\Sigma^n$ is the set of all strings over $\Sigma$ of length $n$ [3]. We say that $M$ runs in polynomial time if there is a constant $k$ such that for all $n$, $T_M(n) \leq n^k + k$ [3]. In other words, this means the language $L(M)$ can be decided by the Turing machine $M$ in polynomial time. Therefore, $P$ is the complexity class of languages that can be decided by deterministic Turing machines in polynomial time [5]. A verifier for a language $L_1$ is a deterministic Turing machine $M$, where:

$$L_1 = \{w : M(w, c) = 1 \textit{ for some string } c\}.$$

We measure the time of a verifier only in terms of the length of $w$, so a polynomial time verifier runs in polynomial time in the length of $w$ [3]. A verifier uses additional information, represented by the symbol $c$, to verify that a string $w$ is a member of $L_1$. This information is called certificate. $NP$ is the complexity class of languages defined by polynomial time verifiers [14].

▶ **Lemma 1.** *Given a language $L_1 \in P$, a language $L_2$ is in $NP$ if there is a deterministic Turing machine $M$, where:*

$$L_2 = \{w : M(w, c) = y \text{ for some string } c \text{ such that } y \in L_1\}$$

*and $M$ runs in polynomial time in the length of $w$. In this way, $NP$ is the complexity class of languages defined by polynomial time verifiers $M$ 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 $P$.*

**Proof.** If $L_1$ can be decided by the Turing machine $M'$ in polynomial time, then the deterministic Turing machine $M''(w, c) = M'(M(w, c))$ will output 1 when $w \in L_2$. Consequently, $M''$ is a polynomial time verifier of $L_2$ and thus, $L_2$ is in $NP$. ◀

## 4 Hypothesis

A function $f : \Sigma^* \to \Sigma^*$ is a polynomial time computable function if some deterministic Turing machine $M$, on every input $w$, halts in polynomial time with just $f(w)$ on its tape [16]. Let $\{0, 1\}^*$ be the infinite set of binary strings, we say that a language $L_1 \subseteq \{0, 1\}^*$ is polynomial time reducible to a language $L_2 \subseteq \{0, 1\}^*$, written $L_1 \leq_p L_2$, if there is a polynomial time computable function $f : \{0, 1\}^* \to \{0, 1\}^*$ such that for all $x \in \{0, 1\}^*$:

$$x \in L_1 \text{ if and only if } f(x) \in L_2.$$

An important complexity class is *NP–complete* [8]. A language $L_1 \subseteq \{0, 1\}^*$ is *NP–complete* if:

- $L_1 \in NP$, and
- $L' \leq_p L_1$ for every $L' \in NP$.

If $L_1$ is a language such that $L' \leq_p L_1$ for some $L' \in$ *NP–complete*, then $L_1$ is *NP–hard* [5]. Moreover, if $L_1 \in NP$, then $L_1 \in$ *NP–complete* [5]. A principal *NP–complete* problem is $SAT$ [6]. An instance of $SAT$ is a Boolean formula $\phi$ which is composed of:

1. Boolean variables: $x_1, x_2, \ldots, x_n$;
2. Boolean connectives: Any Boolean function with one or two inputs and one output, such as $\wedge$(AND), $\vee$(OR), $\neg$(NOT), $\Rightarrow$(implication), $\Leftrightarrow$(if and only if);
3. and parentheses.

A truth assignment for a Boolean formula $\phi$ is a set of values for the variables in $\phi$. A satisfying truth assignment is a truth assignment that causes $\phi$ to be evaluated as true. A formula with a satisfying truth assignment is a satisfiable formula. The problem $SAT$ asks whether a given Boolean formula is satisfiable [6]. We define a $CNF$ Boolean formula using the following terms:

A literal in a Boolean formula is an occurrence of a variable or its negation [5]. A Boolean formula is in conjunctive normal form, or $CNF$, if it is expressed as an AND of clauses, each of which is the OR of one or more literals [5]. A Boolean formula is in 3-conjunctive normal form or $3CNF$, if each clause has exactly three distinct literals [5].

For example, the Boolean formula:

$(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$

is in $3CNF$. The first of its three clauses is $(x_1 \vee \neg x_1 \vee \neg x_2)$, which contains the three literals $x_1$, $\neg x_1$, and $\neg x_2$. Another relevant *NP–complete* language is $3CNF$ satisfiability, or $3SAT$ [5]. In $3SAT$, it is asked whether a given Boolean formula $\phi$ in $3CNF$ is satisfiable.

A logarithmic space Turing machine has a read-only input tape, a write-only output tape, and read/write work tapes [16]. The work tapes may contain at most $O(\log n)$ symbols [16]. In computational complexity theory, $L$ is the complexity class containing those decision problems that can be decided by a deterministic logarithmic space Turing machine [14]. $NL$ is the complexity class containing the decision problems that can be decided by a nondeterministic logarithmic space Turing machine [14].

A logarithmic space transducer is a Turing machine with a read-only input tape, a write-only output tape, and read/write work tapes [16]. The work tapes must contain at most $O(\log n)$ symbols [16]. A logarithmic space transducer $M$ computes a function $f : \Sigma^* \rightarrow \Sigma^*$, where $f(w)$ is the string remaining on the output tape after $M$ halts when it is started with $w$ on its input tape [16]. We call $f$ a logarithmic space computable function [16]. We say that a language $L_1 \subseteq \{0,1\}^*$ is logarithmic space reducible to a language $L_2 \subseteq \{0,1\}^*$, written $L_1 \leq_l L_2$, if there exists a logarithmic space computable function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$x \in L_1$ *if and only if* $f(x) \in L_2$.

The logarithmic space reduction is frequently used for $L$ and $NL$ [14]. A Boolean formula is in 2-conjunctive normal form, or $2CNF$, if it is in $CNF$ and each clause has exactly two distinct literals. There is a problem called $2SAT$, where we asked whether a given Boolean formula $\phi$ in $2CNF$ is satisfiable. $2SAT$ is complete for $NL$ [14]. Another special case is the class of problems where each clause contains $XOR$ (i.e. exclusive or) rather than (plain) $OR$ operators. This is in $P$, since an $XOR\ SAT$ formula can also be viewed as a system of linear equations mod 2, and can be solved in cubic time by Gaussian elimination [12]. 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 $L$ [2], [15].

We can give a certificate-based definition for $NL$ [3]. The certificate-based definition of $NL$ assumes that a logarithmic space Turing machine has another separated read-only tape [3]. On each step of the machine the machine's head on that tape can either stay in place or move to the right [3]. In particular, it cannot reread any bit to the left of where the head currently is [3]. For that reason this kind of special tape is called "read once" [3].

▶ **Definition 2.** *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 $p : \mathbb{N} \rightarrow \mathbb{N}$ such that for every $x \in \{0,1\}^*$,*

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

*where by $M(x,u)$ we denote the computation of $M$ where $x$ is placed on its input tape and $u$ 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 [3]. $M$ is called a logarithmic space verifier [3].*

We state the following Hypothesis:

▷ Hypothesis 3. Given a language $L_1 \in L$, there is a language $L_2$ in *NP–complete* with a deterministic Turing machine $M$, where:

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

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 $L$.

From the early days of automata and complexity theory, two different models of Turing machines are considered, the offline and online machines [10]. Each model has a read-only input tape and some work tapes [10]. The offline machines may read their input two-way while the online machines are not allowed to move the input head to the left [10]. In the terminology of the (generalized) Turing machine models are called two-way and one-way Turing machines respectively [10].

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 [9]. 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 [9]. Furthermore, they have proven that $1NL \subseteq L$ if and only if $L = NL$ [9].

▶ **Theorem 4.** *If the Hypothesis 3 is true, then $L = NL$ and $P = NP$ cannot be both false at the same time.*

**Proof.** We can simulate the computation $M(w, u) = y$ in the Hypothesis 3 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 $u$ [14]. 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 [14].

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 output 1 when $w \in L_2$. Consequently, $M''$ is a nondeterministic logarithmic space Turing machine which decides the language $L_2$. 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 [14]. 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 $M'$. Therefore, $L_2$ is in $NL$ and thus, $L_2 \in P$ due to $NL \subseteq P$ [14]. If any single *NP–complete* problem can be solved in polynomial time, then $P = NP$ [5]. 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 of $L \subset 1NL$.

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 [9]. If this way, if $L \neq NL$, then $L \subset 1NL$ by contraposition [14]. Since we already obtained that $P = NP$ under the assumption of $L \subset 1NL$, therefore if $L \neq NL$, then $P = NP$. Hence, if $P \neq NP$, then $L = NL$ by contraposition [14]. Consequently, $L = NL$ and $P = NP$ cannot be both false at the same time. ◀

## 5 Results

We show a previous known *NP–complete* problem:

▶ **Definition 5.** ***MONOTONE NAE 3SAT***

*INSTANCE: A Boolean formula $\phi$ in $3CNF$ such that each clause has no negation variables.*

*QUESTION: Is there a truth assignment for $\phi$ such that each clause has at least one true literal and at least one false literal?*

*REMARKS: This is equivalent to the special case of the NP–complete problem known as SET SPLITTING when the sets in the input have exactly three elements and therefore, MONOTONE NAE 3SAT $\in$ NP–complete [6].*

We define a new problem:

▶ **Definition 6.** ***MINIMUM EXCLUSIVE-OR 2-SATISFIABILITY***

*INSTANCE: A positive integer $K$ and a Boolean formula $\phi$ that is an instance of XOR 2SAT such that each clause has no negation variables.*

*QUESTION: Is there a truth assignment in $\phi$ such that at most $K$ clauses are unsatisfiable?*

*REMARKS: We denote this problem as $MIN \oplus 2SAT$.*

▶ **Theorem 7.** *$MIN \oplus 2SAT \in$ NP–complete.*

**Proof.** It is trivial to see $MIN \oplus 2SAT \in NP$ [14]. Given a Boolean formula $\phi$ in $3CNF$ with $n$ variables and $m$ clauses such that each clause has no negation variables, 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 $\phi$, where $x$, $y$ and $z$ are positive literals, in the following formula:

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

We can see $P_i$ has at most one unsatisfiable 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 $\psi$ as the conjunction of the $P_i$ formulas for every clause $c_i$ in $\phi$, such that $\psi = P_1 \wedge \ldots \wedge P_m$. Finally, we obtain that

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

Consequently, we prove *MONOTONE NAE 3SAT* $\leq_p MIN \oplus 2SAT$ where we already know the language *MONOTONE NAE 3SAT* $\in$ *NP–complete* [6]. To sum up, we show $MIN \oplus 2SAT \in$ *NP–hard* and $MIN \oplus 2SAT \in NP$ and thus, $MIN \oplus 2SAT \in$ *NP–complete*. ◀

▶ **Theorem 8.** *There is a deterministic Turing machine $M$, where:*

$$MIN \oplus 2SAT = \{w : M(w, u) = y \textit{ for some string } u \textit{ such that } y \in \textit{XOR 2SAT}\}$$

*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 $(\psi, K)$ for $MIN \oplus 2SAT$ when $\psi$ 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 $\psi$ 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 a valid 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 $m$. While we read the elements of the array $A$, we remove the clauses from the instance $(\psi, K)$ for $MIN \oplus 2SAT$ just creating another instance $\phi$ for *XOR 2SAT* where the Boolean formula $\phi$ does not contain the $K$ different indexed clauses $\psi$ represented by the numbers in $A$. Therefore, we obtain the array $A$ should be valid according to the Theorem 8 when:

$$(\psi, K) \in MIN \oplus 2SAT \textit{ if and only if } \phi \in \textit{XOR 2SAT}.$$

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 $\psi$ 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 a valid certificate and also remove the $K$ different clauses from the Boolean formula $\psi$ when we write the final clauses to the output. This logarithmic space verification will be the Algorithm 1. We assume whether a value does not exist in the array $A$ into the cell of some position $i$ when $A[i] = undefined$. In addition, we reject immediately when the following comparisons

$$A[i] \leq max \vee A[i] < 1 \vee A[i] > m$$

hold at least into one single binary digit. Note, in the loop $j$ from $min$ to $max - 1$, we do not output any clause when $max - 1 < min$. ◀

▶ **Theorem 9.** *The Hypothesis 3 is true.*

**Proof.** This is a consequence of Theorems 7 and 8. ◀

▶ **Theorem 10.** *$L = NL$ and $P = NP$ cannot be both false at the same time.*

**Proof.** This is a direct consequence of Theorems 4 and 9. ◀

## 6 Materials and Methods

This work is implemented into a Project programmed in Scala [17]. In this Project, we use the Assertion on the properties of the instances of each problem and the Unit Test for checking the correctness of every reduction [17]. We need to install JDK 8 in order to test the Scala Project [13]. In addition, we need to install SBT to run the unit test (we could run the unit test with the **sbt test** command) [13].

### References


1 Scott Aaronson. P $\stackrel{?}{=}$ NP. *Electronic Colloquium on Computational Complexity, Report No. 4*, 2017.

2 Carme Álvarez and Raymond Greenlaw. A Compendium of Problems Complete for Symmetric Logarithmic Space. *Computational Complexity*, 9(2):123–145, 2000. `doi:10.1007/PL00001603`.

3 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

**Algorithm 1** Logarithmic space verifier

```
/*A valid instance for MIN ⊕ 2SAT with its certificate*/
procedure VERIFIER((ψ, K), A)
    /*Initialize minimum and maximum values*/
    min ← 1
    max ← 0
    /*Iterate for the elements of the certificate array A*/
    for i ← 1 to K + 1 do
        if i = K + 1 then
            /*There exists a K + 1 element in the array*/
            if A[i] ≠ undefined then
                /*Reject the certificate*/
                return 0
            end if
            /*m is the number of clauses in ψ*/
            max ← m + 1
        else if A[i] = undefined ∨ A[i] ≤ max ∨ A[i] < 1 ∨ A[i] > m then
            /*Reject the certificate*/
            return 0
        else
            max ← A[i]
        end if
        /*Iterate for the clauses of the Boolean formula ψ*/
        for j ← min to max − 1 do
            /*Output the indexed j clause in ψ*/
            output " ∧ c_j"
        end for
        min ← max + 1
    end for
end procedure
```

**4** Stephen A. Cook. The P versus NP Problem, April 2000. In Clay Mathematics Institute at `http://www.claymath.org/sites/default/files/pvsnp.pdf`.

**5** Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

**6** Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W. H. Freeman and Company, 1 edition, 1979.

**7** William I. Gasarch. Guest column: The second P $\stackrel{?}{=}$ NP poll. *ACM SIGACT News*, 43(2):53–77, 2012.

**8** Oded Goldreich. *P, NP, and NP-Completeness: The basics of computational complexity*. Cambridge University Press, 2010.

**9** Juris Hartmanis and S Mahaney. Languages Simultaneously Complete for One-Way and Two-Way Log-Tape automata. *SIAM Journal on Computing*, 10(2):383–390, 1981.

**10** Martin Kutrib, Julien Provillard, György Vaszil, and Matthias Wendlandt. Deterministic One-Way Turing Machines with Sublinear Space. *Fundamenta Informaticae*, 136(1-2):139–155, 2015.

**11** Richard J. Lipton. Efficient checking of computations. In *STACS 90*, pages 207–215. Springer Berlin Heidelberg, 1990. `doi:10.1007/3-540-52282-4_44`.

**12** Cristopher Moore and Stephan Mertens. *The Nature of Computation*. Oxford University Press, 2011.

**13** Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala: Updated for Scala 2.12*. Artima Incorporation, USA, 3rd edition, 2016.

**14** Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

**15** Omer Reingold. Undirected Connectivity in Log-space. *J. ACM*, 55(4):1–24, September 2008. `doi:10.1145/1391289.1391291`.

**16** Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

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