L Versus Parity-L

Frank Vega

May 7, 2020

L versus parity-L

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

Abstract

A major complexity classes are L and $\oplus L$ (parity-L). A logarithmic space Turing machine has a read-only input tape, a write-only output tape, and some read/write work tapes. The work tapes may contain at most $O(\log n)$ symbols. L is the complexity class containing those decision problems that can be decided by a deterministic logarithmic space Turing machine. The complexity class $\oplus L$ has the same relation to L as $\oplus P$ does to P. Whether $L = \oplus L$ is a fundamental question that it is as important as it is unresolved. We prove there is a complete problem for $\oplus L$ that can be logarithmic space reduced to a problem in L. In this way, we demonstrate that $L = \oplus L$.

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

Keywords and phrases complexity classes, logarithmic space, XOR-3SAT, XOR-2SAT, reduction

1 Results

▶ **Definition 1.** ***XOR–3SAT***

INSTANCE: A natural number n and a Boolean formula ϕ that is the conjunctions of a set C of clauses $c_1, \ldots, c_m$, where each c_i consists of the EXCLUSIVE OR (denoted $\oplus$) of three literals and ϕ contains n variables represented by a unique positive integer between 1 and n just similar to the DIMACS representation [3].

QUESTION: Is it the case that ϕ is satisfiable?

REMARKS: XOR–3SAT is complete for $\oplus L$ [2].

▶ **Definition 2.** ***XOR–2SAT***

INSTANCE: A Boolean formula ψ that is the conjunctions of a set C of clauses $c_1, \ldots, c_m$, where each c_i consists of the EXCLUSIVE OR (denoted $\oplus$) of two literals and the ψ variables are represented by a unique positive integer just similar to the DIMACS representation [3].

QUESTION: Is it the case that ψ is satisfiable?

REMARKS: XOR–2SAT is in L [1], [4].

A logarithmic space transducer is a Turing machine with a read-only input tape, a write-only output tape, and some read/write work tapes [5]. The work tapes must contain at most $O(\log n)$ symbols [5]. A logarithmic space transducer M computes a function $f : \Sigma^* \to \Sigma^*$, where $f(w)$ is the string remaining on the output tape after M halts when it is started with w on its input tape [5]. We call f a logarithmic space computable function [5]. 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 $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

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

▶ **Theorem 3.** *$L = \oplus L$.*

Proof. From the Definition 1, we assume that each variable in the Boolean formula ϕ of n variables in *XOR–3SAT* is represented by a unique positive integer between 1 and n. The negative literals are represented as the negative value of each variable value, such that the negative literal of the variable a is $-a$. In order to make the reduction, we need to create the variables inside of a Boolean formula in *XOR–2SAT* and therefore, we use the function h such that

$$h(x) = \textbf{if } (x > 0) \textbf{ return } (2 \times x) \textbf{ else return } (-2 \times x + 1).$$

Moreover, from a tuple (x, y) of two integers, we denote the function g such that

$$g((x, y)) = \textbf{if } (x < y) \textbf{ return } (x, y) \textbf{ else return } (y, x)$$

where g sorts the elements of a tuple (x, y) of two integers. Furthermore, from a tuple (x, y) of two positive integers, we denote the function v such that

$$v((x, y)) = ((x + y)^2 + 3 * x + y)/2$$

where v returns a unique integer for a tuple (x, y) of two positive integers. Finally, we denote a clause that contains n-integers $a_1, a_2, \ldots, a_n$ as the function $c(a_1, a_2, \ldots, a_n)$ just similar to the DIMACS representation [3]. The logarithmic space reduction is described in the pseudo code Algorithm 1.

In this reduction, we guarantee that creation of the variables through the function h. In addition, using the functions g and v, we create the literals into the clauses of the final formula ψ in *XOR–2SAT*. In the first step, we create four clauses in *XOR–4SAT* for each clause in ϕ introducing two new variables $p = n + 1$ and $q = n + 2$, where these total generated clauses are satisfied for some truth assignment if and only if the Boolean formula ϕ is satisfiable when the variables $p = n + 1$ and $q = n + 2$ take opposite values. Hence, we output three clauses that are satisfied for some truth assignment if and only if the four generated clauses are satisfied. Certainly, a clause $(x \oplus y \oplus z \oplus w)$ is satisfied for some truth assignment if and only if the clauses $(a_{x \oplus y} \oplus b_{z \oplus w})$, $(a_{x \oplus z} \oplus b_{y \oplus w})$ and $(a_{x \oplus w} \oplus b_{y \oplus z})$ are satisfied for the same truth assignment, where in this case x, y, z and w are literals and the variables $a_{x \oplus y}$, $b_{z \oplus w}$, $a_{x \oplus z}$, $b_{y \oplus w}$, $a_{x \oplus w}$ and $b_{y \oplus z}$ are equals to the values of $(x \oplus y)$, $(z \oplus w)$, $(x \oplus z)$, $(y \oplus w)$, $(x \oplus w)$ and $(y \oplus z)$, respectively. Note, that we use tuples and thus, for that reason, we need to output two additional clauses for each pair of variables including the introduced variables $p = n + 1$ and $q = n + 2$. For these two new clauses in the second step, we guarantee the appropriated assignment for a single variable in ϕ, which means that a literal should have the opposite value of its negation. We could affirm this, because of the clauses $(x \oplus j \oplus \neg x \oplus j)$ and $(x \oplus \neg j \oplus \neg x \oplus \neg j)$ are always satisfied for any truth assignment, where $\neg$ is the *NOT* Boolean function. Finally, in the third step, we guarantee the variables $p = n + 1$ and $q = n + 2$ could take opposite values just making the formula ϕ satisfiable from the first step. Actually, we can assure this, because of the clause $(p \oplus j \oplus q \oplus j)$ is satisfied for some truth assignment if and only if p and q take opposite values. In this way, we create a Boolean formula $\psi \in$ *XOR–2SAT* if and only if $\phi \in$ *XOR–3SAT*. In general, the whole algorithm uses logarithmic space in the work tapes since the new Boolean formula is created in the output tape into a write-only way. Consequently, we obtain *XOR–3SAT* $\leq_l$ *XOR–2SAT*. The single existence of a complete problem in $\oplus L$ that could be logarithmic space reduced to a problem in L is sufficient to show $L = \oplus L$. This work is implemented into a Project programmed in Scala from a GitHub repository [6]. ◀

References

1 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.

Algorithm 1 Logarithmic space reduction from *XOR–3SAT* to *XOR–2SAT*

```
/*A natural number n and a Boolean formula ϕ of an instance from XOR–3SAT*/
procedure REDUCTION(n, ϕ)
    /*Create two new variables*/
    p ← n + 1
    q ← n + 2
    /*First step: Iterate for the clauses in ϕ*/
    for all c(a, b, c) ∈ ϕ do
        /*Convert the clause to four clauses in XOR–4SAT*/
        S ← {c(a, b, c, p), c(a, b, c, −q), c(−a, −b, −c, −p), c(−a, −b, −c, q)}
        for all c(x, y, z, w) ∈ S do
            output c(v(g((h(x), h(y)))), v(g((h(z), h(w)))))
            output c(v(g((h(x), h(z)))), v(g((h(y), h(w)))))
            output c(v(g((h(x), h(w)))), v(g((h(y), h(z)))))
        end for
    end for
    /*Second step: Iterate quadratically from 1 to n + 2*/
    for i ← 1 to q do
        for j ← 1 to q do
            if i ≠ j then
                output c(v(g((h(i), h(j)))), v(g((h(−i), h(j)))))
                output c(v(g((h(i), h(−j)))), v(g((h(−i), h(−j)))))
            end if
        end for
    end for
    /*Third step: The variable p takes the opposite value of q*/
    for j ← 1 to n do
        output c(v(g((h(p), h(j)))), v(g((h(q), h(j)))))
        output c(v(g((h(p), h(−j)))), v(g((h(q), h(−j)))))
        output c(v(g((h(−p), h(j)))), v(g((h(−q), h(j)))))
        output c(v(g((h(−p), h(−j)))), v(g((h(−q), h(−j)))))
    end for
end procedure
```

2 Carsten Damm. Problems complete for $\oplus$ L. In *International Meeting of Young Computer Scientists*, pages 130–137. Springer, 1990.
3 Varisat Manual. DIMACS CNF, May 2020. In Varisat Manual at `https://jix.github.io/varisat/manual/0.2.0/formats/dimacs.html`.
4 Omer Reingold. Undirected Connectivity in Log-space. *J. ACM*, 55(4):1–24, September 2008. `doi:10.1145/1391289.1391291`.
5 Michael Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, 2 edition, 2006.
6 Frank Vega. Sat Solvers, October 2019. In a GitHub repository at `https://github.com/frankvegadelgado/sat`.