Journal of Universal Computer Science, vol. 19, no. 10 (2013), 1375-1395
submitted: 17/8/12, accepted: 27/5/13, appeared: 28/5/13 © *J.UCS*

Term Satisfiability Problem for Two-Element Algebras is in QL or is NQL-Complete

Tomasz A. Gorazd
(Jagiellonian University, Kraków, Poland
gorazd@tcs.uj.edu.pl)

Jacek Krzaczkowski
(Maria Curie-Skłodowska University, Lublin, Poland
krzacz@umcs.lublin.pl)

Abstract: We show that the term satisfiability problem for finite algebras is in NQL. Moreover we present a complete classification of the computational complexity of the term satisfiability problem for two-element algebras. We show that for any fixed two-element algebra the problem is either in QL or NQL-complete.

We show that the complexity of the considered problem, parameterized by an algebra, is determined by the clone of term operations of the algebra.

Key Words: computational complexity, solving equations, algebra, clones, SAT
Category: F.1.3, F.2.2

1 Introduction

Solving equations and systems of equations is one of the oldest mathematical problems. Nevertheless, there are many new results connected with solving equations published in the last years (see [Barrington et al. 2000], [Broniek 2006], [Goldmann and Russel 2002], [Gorazd and Krzaczkowski 2011], [Gorazd and Krzaczkowski 2010], [Horváth 2011], [Horváth and Szabó 2006], [Horváth and Szabó 2011], [Horváth and Szabó 2012], [Klíma et al. 2007], [Larose and Zádori 2006], [Schwarz 2004]). In this paper we consider the term satisfiability problem TERM-SAT($\mathbf{A}$) and the polynomial satisfiability problem POL-SAT($\mathbf{A}$). We are interested in the computational complexity of these problems for two-element algebras.

P and NP are probably the most often considered classes in the computational complexity theory. It is used to assume that P is a class of “easy” problems. But is a problem solvable in the time $O(n^{100})$ really “easy”? Is an algorithm working in the time $O(n^{100})$ really computable? The class of “easy” problems is often associated with a complexity class that differs from P. A few results on this topic can be found in [Creignou 1995], [Dewdney 1982], [Gräedel 1990], [Grandjean 1994], [Gurevich and Shelah 1989], [Schnorr 1978]. One of the considered classes is the class of problems solvable in quasilinear time i.e. in the time $O(n \log^k n)$, where k is a constant. By analogy with the classes P and

NP we consider the classes QL and NQL, where QL and NQL are the classes of problems computable in quasilinear time on a deterministic Turing machine and a non-deterministic Turing machine, respectively. The NQL-completeness is defined by using the deterministic quasilinear time many-one reduction. It was shown in [Schnorr 1978] that SAT is NQL-complete, but on the other hand it is known that not every NP-complete problem belongs to NQL [Cook 1972]. In this paper we show that for any finite algebra $\mathbf{A}$ the problems TERM-SAT($\mathbf{A}$) and POL-SAT($\mathbf{A}$) are in NQL. Moreover, we completely classify the complexity of these problems for two-element algebras.

It is well known that if P is not equal to NP, then in NP there are problems which are neither in P nor NP-complete. Nevertheless, there are large natural classes of problems contained in NP such that any problem from these classes is either in P or NP-complete. One such a class is the constraint satisfaction problems on two and three element sets [Schaefer 1978], [Bulatov 2002]. Another example is TERM-SAT over the two-element algebras [Gorazd and Krzaczkowski 2011] and the preprimal algebras [Gorazd and Krzaczkowski 2010].

Similarly to the case of P and NP we do not know if QL=NQL. If QL is not equal to NQL then there are problems in NQL which are neither in QL nor NQL-complete. We prove that for any two-element algebra the problem TERM-SAT is either in QL or NQL-complete. It may not seem surprising but it requires completely other proofs then those presented in [Gorazd and Krzaczkowski 2011].

One of the interesting questions about problems connected with solving equations is whether for a given algebra $\mathbf{A}$ their complexity depends only on $Clo(\mathbf{A})$ (the clone of term operations of an algebra $\mathbf{A}$, see [McKenzie et al. 1987]). In the general case, it is not true:

Example 1. *Let $(S_3, \cdot)$ be the three-element symmetric group, and let*

$$s(x, y, z, w) = x \cdot [[[x, y], z]], w]^{-1},$$

where $[x, y] = x^{-1} \cdot y^{-1} \cdot x \cdot y$.

- $Clo(S_3, \cdot) = Clo(S_3, \cdot, s)$,
- POL-SAT$(S_3, \cdot)$ *is in P [Horváth and Szabó 2006],*
- POL-SAT$(S_3, \cdot, s)$ *is NP-complete (P. M. Idziak's result, the proof is published in [Gorazd and Krzaczkowski 2010]).*

Algebras with the clone of term operations equal to the clone of the alternating group $\mathbf{A_4}$ are another example where the computational complexity of POL-SAT does not depend on the clone of the algebra's term operations [Horváth and Szabó 2011], [Horváth and Szabó 2012]. In such a context

G.Horváth and C.Szabó consider so called "extended solvability problem". They ask how computational complexity of POL-SAT($\mathbf{A}$) changes if we allow using operations from the arbitrary finite subset of $Clo(\mathbf{A})$ in equations.

On the other hand, for the two-element algebras the computational complexity of TERM-SAT (up to polynomial time reduction) depends only on the clone of term operations of the related algebras (see [Gorazd and Krzaczkowski 2011], [Gorazd and Krzaczkowski 2010]). The main conclusion of the results presented in this paper is that for any two-element algebras $\mathbf{A}$ and $\mathbf{B}$ such that $Clo(\mathbf{A}) = Clo(\mathbf{B})$ if TERM-SAT($\mathbf{A}$) is NQL-complete (in QL) then TERM-SAT($\mathbf{B}$) is also NQL-complete (in QL).

The paper is organized as follows. Section 1 is an introduction to the subject. Section 2 contains the basic definitions needed in the next sections. In Section 3 we define and analyze the classes QL and NQL. Moreover, we present there some specific problems that we encounter when studying QL and NQL and describe the assumptions we make. Sections 4 and 5 contain two main results of this paper. In Section 4 we prove that TERM-SAT for finite algebras is in NQL and in Section 5 we characterize the computational complexity of TERM-SAT and POL-SAT for two element algebras in terms of QL and NQL. Finally, in Section 6 we propose some possibilities for further research.

2 Definitions

A *language* (or *type*) of algebras is a set $\mathcal{F}$ of function symbols with a nonnegative integer assigned to each member of $\mathcal{F}$. This integer is called the arity of $f \in \mathcal{F}$. The subset of n-ary function symbols of $\mathcal{F}$ is denoted by $\mathcal{F}_n$.

An *algebra* of type $\mathcal{F}$ is an ordered pair $\mathbf{A} = (A, F)$, where A is a nonempty set (called *universe*) and F is a family of finitary operations on A indexed by the language $\mathcal{F}$ such that for any n-ary symbol $f \in \mathcal{F}$ there is an n-ary operation $f^{\mathbf{A}}$ on A. The $f^{\mathbf{A}}$'s are called *fundamental operations of* $\mathbf{A}$. If $F = \{f_1, \ldots, f_n\}$ it is customary to write $(A, f_1, \ldots, f_n)$ rather than (A, F). In this paper we consider only algebras with finite universes. Notice that the set of fundamental operations may not be finite.

For a language $\mathcal{F}$ and a set of variables X ($|X| = \omega$) we define $T(X)$, the set of terms of type $\mathcal{F}$, as the smallest one, such that

- $X \cup \mathcal{F}_0 \subseteq T(X)$
- If $t_1, \ldots, t_n \in T(X)$ and $f \in \mathcal{F}_n$, then the "string" $f(t_1, \ldots, t_n) \in T(X)$

If for an algebra $\mathbf{A}$ of type $\mathcal{F}$ we additionally admit all constant operation symbols on A while building terms, we get the *polynomials* of $\mathbf{A}$.

Definition 2. The **composition of terms** $t(x_1, \dots, x_k)$, $w_1(x_1^1, \dots, x_{n_1}^1)$, $\dots$, $w_k(x_1^k, \dots, x_{n_k}^k)$ is the term obtained by replacing each occurrence of x_i in the term t by the term $w_i(x_1^i, \dots, x_{n_i}^i)$.

We denote the composition of these terms by $t(w_1(x_1^1, \dots, x_{n_1}^1), \dots, w_k(x_1^k, \dots, x_{n_k}^k))$.

In some cases it is suitable to think about a term as a tree. We define a *subtree* of a tree T to be a tree consisting of a node in T and all of its descendants in T. An *internal node* of a tree is a node which have at least one child.

Definition 3. The **tree of a term** t is a tree T whose nodes are labeled by function symbols and variables occurring in the term t and defined in the following way:

– if t is a variable x then T is a tree containing one node labeled by x,

– if t is in the form $f(t_1, \dots, t_k)$, then the root of T is labeled by f and the subtrees of the root, from left to right, are the trees of terms $t_1,\dots,t_k$ respectively.

For a tree T we define $Sub(T)$ as the set of all proper subtrees of T that contain at least two nodes and whose roots are children of the root of T.

In a similar way, we can define the composition of polynomials and the tree of a polynomial.

If $\mathbf{A}$ is an algebra of type $\mathcal{F}$, then with terms and polynomials we can associate operations on the set A in an obvious way. If t is a term (polynomial), in which only the (distinct) variables from $\{x_1, \dots, x_n\}$ appear, then $t^{\mathbf{A}}(x_1, \dots, x_n)$ describes the corresponding n-ary term (polynomial) operation. We denote the set of all term operations of $\mathbf{A}$ by $Clo(\mathbf{A})$ (and the set of polynomial operations of $\mathbf{A}$ by $Pol(\mathbf{A})$). Observe that this set is a *clone of operations* on A, i.e. sets of operations on A, closed under composition, and containing the projection operations $\pi_i^n(x_1, \dots, x_n) = x_i$.

The notion of frugal terms and polynomials plays an important role in our proofs of the NQL-completeness.

Definition 4. Let $t(x_1, \dots, x_k)$ be a term in the language $\mathcal{F}$ and $\mathbf{A}$ be an algebra of type $\mathcal{F}$. We call this term frugal in the algebra $\mathbf{A}$ iff any x_i such that $t^{\mathbf{A}}$ depends on it occurs in t only once. Frugal polynomials are defined analogously.

When we write that $t(x_1, \dots, x_k)$ is a frugal term (polynomial) we mean that $t^{\mathbf{A}}$ depends on every variable $x_1, \dots, x_k$ and that it is possible there are other variables in t on which $t^{\mathbf{A}}$ does not depend. It does not lead any confusion in the latter arguments and algorithms. An operation f will be called frugally definable in algebra $\mathbf{A}$ if f is equal to $t^{\mathbf{A}}$ for a term t frugal in $\mathbf{A}$.

The main problem considered is TERM-SAT:

Definition 5. For an algebra $\mathbf{A}$ the term satisfiability problem, TERM-SAT($\mathbf{A}$), is the decision problem with

Instance: A pair of terms (s,t) with the tables of the fundamental operations of $\mathbf{A}$ corresponding to all function symbols occurring in s and t.

Question: Does there exist a substitution of variables of s and t by elements of A such that the corresponding values of $s^{\mathbf{A}}$ and $t^{\mathbf{A}}$ are the same?

If in the above definition we put polynomials instead of terms we obtain the definition of POL-SAT. The tables of fundamental operations in the instances of these problems are important because we also consider algebras with infinitely many operations. Notice that the set of operations of an algebra may not be recursive. In the consequence if we did not have the tables of the fundamental operations as a part of the instance, TERM-SAT or POL-SAT for such algebras would be undecidable. For algebras of finite types it does not matter if the tables of fundamental operation occurring in terms or polynomials are a part of the input or not. In both cases, for such algebras TERM-SAT and POL-SAT have the same asymptotic complexity because all the tables of operations may be a part of the algorithm.

We do not exactly consider the problem of solving equation (i.e. looking for a solution). However, using the proofs from the paper, it is not hard to see that for a given two-element algebra, the problem of solving equations and the corresponding satisfiability problem (e.g. TERM-SAT, POL-SAT etc.) belong to the corresponding functional and decisional complexity classes.

In the next section we show a quasilinear time deterministic algorithm solving the following problem:

Definition 6. For the algebra $\mathbf{A}$, we call POL-VAL($\mathbf{A}$) the problem with

Instance: A polynomial p that does not contain any variable together with the tables of fundamental operations of $\mathbf{A}$ corresponding to all function symbols occurring in p.

Question: What is the value of $p^{\mathbf{A}}$?

The fact that we can solve the above problem in quasilinear time is often used in proofs in this paper. The problem POL-VAL is of course a generalization of the well known CIRCUIT VALUE problem.

We will make an essential use of the Post classification for clones on the two-element set [Post 1941]. The Hasse diagram of the order on the set of such clones is presented in the following figure. We use the original Post notation for the clones on the set $2 = \{0, 1\}$, however, we recall them here for the reader's convenience.

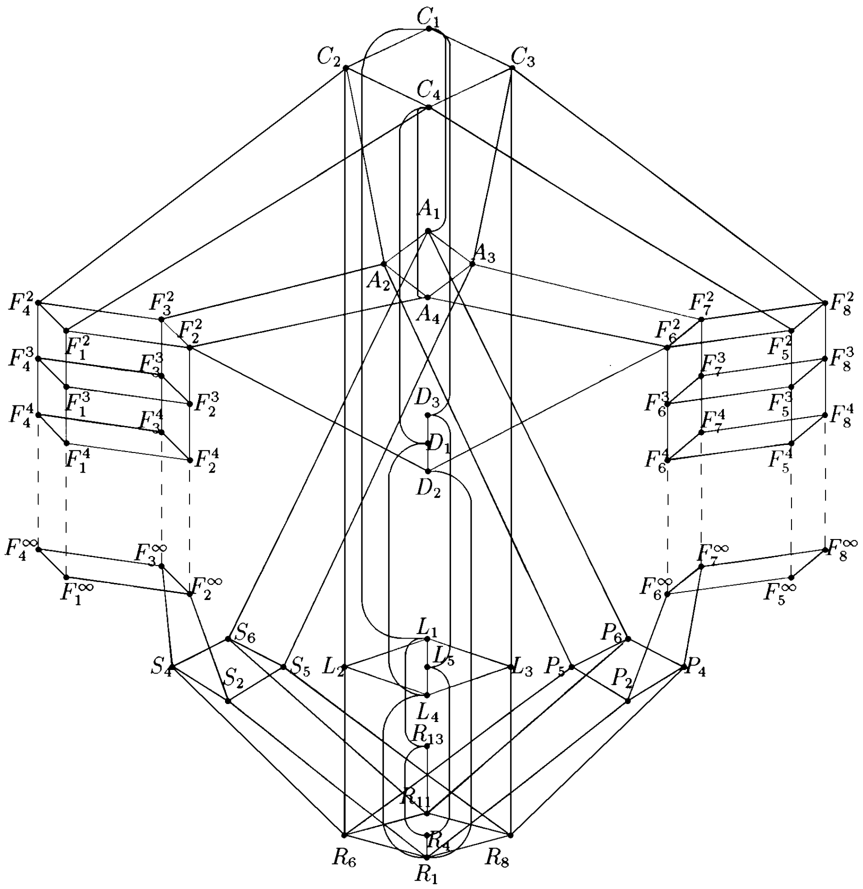

Figure 1: *The lattice of clones on the two-element set.*

$\mathbf{C_1} = (2, \wedge, \vee, \neg)$	$\mathbf{C_3} = (2, -, \vee)$	$\mathbf{C_4} = (2, \vee, ki)$	
$\mathbf{A_1} = (2, \wedge, \vee, 0, 1)$	$\mathbf{A_3} = (2, \wedge, \vee, 0)$	$\mathbf{A_4} = (2, \wedge, \vee)$	
$\mathbf{D_3} = (2, d, \neg)$	$\mathbf{D_1} = (2, d, +_3)$	$\mathbf{D_2} = (2, d)$	
$\mathbf{L_1} = (2, +, \neg)$	$\mathbf{L_3} = (2, +)$	$\mathbf{L_5} = (2, +_3, \neg)$	$\mathbf{L_4} = (2, +_3)$
$\mathbf{F_8^m} = (2, -, d_m)$	$\mathbf{F_8^\infty} = (2, -)$	$\mathbf{F_7^m} = (2, ka, d_m, 0)$	$\mathbf{F_7^\infty} = (2, ka, 0)$
$\mathbf{F_6^m} = (2, ka, d_m)$	$\mathbf{F_6^\infty} = (2, ka)$	$\mathbf{F_5^m} = (2, ki, d_m)$	$\mathbf{F_5^\infty} = (2, ki)$
$\mathbf{P_6} = (2, \wedge, 0, 1)$	$\mathbf{P_5} = (2, \wedge, 1)$	$\mathbf{P_4} = (2, \wedge, 0)$	$\mathbf{P_2} = (2, \wedge)$
$\mathbf{R_{13}} = (2, \neg, 0)$	$\mathbf{R_4} = (2, \neg)$	$\mathbf{R_{11}} = (2, 0, 1)$	$\mathbf{R_8} = (2, 0)$
$\mathbf{R_1} = (2)$			

where

$x - y = x \wedge \neg y,$
$ki(x, y, z) = x \wedge (y \to z),$
$ka(x, y, z) = x \wedge (y \vee z),$
$+_3(x, y, z) = x + y + z \ \ (\, mod\, 2),$
$d_m(x_0, \ldots, x_m) = \bigvee_{i=0}^{m}(x_0 \wedge \ldots \wedge x_{i-1} \wedge x_{i+1} \wedge \ldots \wedge x_m), \quad m \geq 2,$
$d = d_2.$

They describe the central and right hand side parts of the diagram. The clones $C_2, A_2, L_2, R_6, F_i^\alpha$ and S_i are dual to $C_3, A_3, L_3, R_8, F_{i+4}^\alpha$ and P_i, respectively, in the sense that an operation $\mathbf{f}$ is dual to $\mathbf{g}$ if $\mathbf{f}(x_1, \ldots x_k) = \neg \mathbf{g}(\neg x_1, \ldots, \neg x_k)$, i.e. the clone X is dual to Y if the map $\neg : 2 \longrightarrow 2$ is an isomorphism of $(2, X)$ onto $(2, Y)$.

A function $f : 2^n \to 2$ is called

- monotone iff $(\forall_{0 \leq i < n}\ a_i \leq b_i) \Rightarrow f(a_0, \ldots, a_{n-1}) \leq f(b_0, \ldots, b_{n-1})$.
- 0-valid iff $f(0, \ldots, 0) = 1$
- 1-valid iff $f(1, \ldots, 1) = 1$

The symmetric group $\mathbf{S_3}$ and alternating group $\mathbf{A_4}$ are examples which show that the computational complexity of a problem parameterized by an algebra $\mathbf{A}$ may not be determined by $Clo(\mathbf{A})$.

Definition 7. We say that for a clone $\mathcal{C}$ a problem is NQL-complete (in QL) if for any algebra $\mathbf{A}$ such that $\mathcal{C} = Clo(\mathbf{A})$ the problem is NQL-complete (in QL).

In this paper we prove that for any clone on the two-element set the problems TERM-SAT and POL-SAT are either NQL-complete or in QL.

3 QL and NQL

We say that an algorithm works in quasilinear time if for an input of size n the algorithm works in time $O(n \log^k n)$, where k is a constant. We define QL (NQL) as a class of problems solvable in quasilinear time on the multitape deterministic (non-deterministic) Turing machines. We define NQL-completeness using reductions computable on a multitape deterministic Turing machine in quasilinear time. These classes do not depend on the number of tapes and the size of the alphabet provided that there are at least two tapes and two alphabet symbols.

In [Schnorr 1978] Schnorr proved the following important theorem:

Theorem 8. *SAT is NQL-complete.*

It is a straightforward matter to see that 3-SAT is also NQL-complete.

The easy conclusion of Theorem 8 is the following:

Corollary 9. *Every NQL-complete problem is NP-complete.*

Observe that if $NQL = QL$ then $NP = P$. From the results obtained by Cook [Cook 1972] we have that NQL is a proper subclass of NP.

When we consider QL and NQL we have to be more careful than in the case of P and NP. For example in contrast to the polynomial time case the classes of problems solvable on the random access machine and the deterministic Turing machine in quasilinear time are probably not equal. So we cannot use in our algorithms the random access memory instead of the Turing machine's tape. On the other hand, it may be shown using technic from [Gurevich and Shelah 1989] and [Angluin and Valiant 1979] that NQL is equal to the class of problems solvable in quasilinear time on non-deterministic random access machinie.

All algorithms in this paper are parameterized by a finite algebra $\mathbf{A}$ of type $\mathcal{F}$. The universe of this algebra will be the set $A = \{a_1, \ldots, a_n\}$, the elements will be ordered in the natural way.

The alphabet of the Turing machine will be the following:

- $0, 1$ for the encoding numbers
- brackets,
- characters for each element of the set A,
- two symbols for the designation names of functions and variables. The particular function (variable) name will be represented by the special symbol with a unique number after it,
- some additional characters used for separation of elements on the tape.

We assume that the alphabet is ordered as given above.

As an instance of TERM-SAT we have two terms which are written on the first tape. We assume that the tables of fundamental operations we need are written on the second tape. For a k-ary fundamental function f we encode its table in the following way: as the first symbol we write k and then

- if $k = 0$ then f is a constant operation and its value is written on the tape,
- if $k > 0$ then the table consists of recursively written tables of operation $f(a_1, x_2, \ldots, x_k)$,..., $f(a_n, x_2, \ldots, x_k)$ but without information about their arity.

It seems that the purpose of describing operations by their tables is to increase the size of the input but in the average case it is impossible to find significantly shorter description of operations. Notice that the number of different k-ary operations on set A is $|A|^{\left(|A|^k\right)}$ and hence we need $O(\log |A|^{\left(|A|^k\right)}) = O(|A|^k)$ bits to be able to describe every such an operation. Moreover, it easy to see that the average number of bits needed to describe k-ary operations on set A is $O(|A|^k)$, too. For comparison, note that the size of table of a k-ary operation on set A is $|A|^k$.

Note that we consider also algebras with an infinite number of fundamental operations but we use the Turing machine over a finite alphabet. It implies that we need more than one character to write the function symbols and in consequence, the length of function symbols may be arbitrarily large. The similar situation is with variables. We assume that the function symbols (variables) consist of unique numbers of them and a special character used only in the function symbols (variables). We could assume that the function symbols and variables are numbered consecutively but this is not needed.

In this paper we will frequently use the notion of a *token*. The meaning of this notion depends on the context. Typically, a token may be a character but also a variable or a function symbol. Sometimes it means a function symbol with its arguments too. This will be clear from the context. The idea is that we may divide terms in many ways, and tokens are parts of terms obtained during such a process. We often use this concept when sorting elements on a tape. In this case we will only describe the main tokens, the others will be the remaining elements of the alphabet. We order tokens in such a way that the main tokens are the first ones.

One important fact we use is that one can sort elements written on the Turing machine's tape in quasilinear time. For example, it is quite easy to implement the mergesort algorithm working in deterministic quasilinear time on the Turing machine (see [Schnorr 1978], Program p_1). Observe that the size of the input is not the number of sorted tokens but the number of characters written on the tape. One may ask if the fact that the sorted elements may be arbitrarily large increases the complexity of sorting. Fortunately, in our case the size of the sorted tokens does not affect the computational complexity of sorting. In general, for some complicated comparing functions, this may be a problem.

Note that in some algorithms we sort parts of a term. We will need to separate sorted tokens in order to be able to check easily where they start and end. Of course, this can be done in quasilinear time using an additional character. We will ignore such problems in implementations in order not to complicate the algorithms. Also when describing the Turing machines we will not talk about the additional tapes we need for sorting. The reader can easily complete all the algorithms.

For convenience we suppose that on the first tape after every function symbol there is written the arity of the symbol. On can easily convert the standard input to this form in quasilinear time using the methods presented in the following sections.

4 Term-Sat is in NQL

First we will prove that TERM-SAT($\mathbf{A}$) for any finite algebra $\mathbf{A}$ is in NQL. The main problem in the proof is that we also allow algebras with an infinite set of operations. We note on the other hand, when we consider algebras only of finite type, the tables of operations are in fact a part of the machine, not of the input. In this case the proof is very easy. Our proof for algebras of arbitrary type will be a corollary of two lemmas.

Lemma 10. *Let* $\mathbf{A}$ *be a finite algebra.*
There is a deterministic Turing machine which for a given frugal polynomial $t(x_1,\dots,x_k)$ *and polynomials* $w_1(x_1^1,\dots,x_{n_1}^1)$, $\dots$, $w_k(x_1^k,\dots,x_{n_k}^k)$ *over* $\mathbf{A}$ *creates their composition – i.e. the polynomial* $t(w(x_1^1,\dots,x_{n_1}^1),\dots,w(x_1^k,\dots,x_{n_k}^k))$ *– in quasilinear time.*

Proof. We show the deterministic Turing machine M composing the given polynomials over $\mathbf{A}$ in quasilinear time. We may assume that machine M has three main tapes. On the first one there is the polynomial t. On the second tape there are polynomials $w_1(x_1^1,\dots,x_{n_1}^1)$, $\dots$, $w_k(x_1^k,\dots,x_{n_k}^k)$ ordered by their indexes. The third tape is an auxiliary one.

Notice that the naive algorithm which just rewrites the polynomial t and substitutes its variables by appropriate polynomials works in quadratic time. It is because the variables in the polynomial t may occur in different orders and on a Turing machine to find the polynomial w_i we have to shift the machine head step by step. This can take linear time. The main idea used in this proof is the same as that used by Schnorr [Schnorr 1978] to prove that SAT is in NQL.

The main tokens on the first tape will be the names of variables ordered by their indexes. Let n be the number of tokens. The machine M writes on the third tape the numbers from 1 to n. In the next step M sorts tokens on the first tape. While sorting M applies the same transformation simultaneously to the corresponding numbers on the third tape. After that the machine M substitutes every occurrence of any variable on the first tape by corresponding polynomial from the second one (every occurrence of x_i should be replaced by $w_i(x_1^i,\dots,x_{n_i}^i)$). It possible to do the substitution in linear time using one auxiliary tape.

The last thing the machine M has to do is to sort the numbers on the third tape applying the same transformation simultaneously to the corresponding tokens on the first tape. The main tokens on the first tape will be the polynomials $w_1(x_1^1,\dots,x_{n_1}^1)$, $\dots$, $w_k(x_1^k,\dots,x_{n_k}^k)$. Now, the first tape contains the polynomial $t(w(x_1^1,\dots,x_{n_1}^1),\dots,w(x_1^k,\dots,x_{n_k}^k))$. Because of the frugality of t the final term is not longer than the input and one can see that M works in quasilinear time.

Notice that if the polynomial t in the Lemma 10 were not frugal then the size of the composition of polynomials can be even quadratic in the size of the input. Obviously, in such a case there is no quasilinear algorithm composing polynomials.

Corollary 11. *Let t be a term over $\mathbf{A}$. There is a deterministic Turing machine which substitutes all variables of t by given values from A in quasilinear time.*

Proof. This comes directly from the proof of Lemma 10. Notice that because the length of the elements of A is equal to 1 we do not need the assumption about the frugality of t.

To prove the second lemma mentioned above we will need the following observation:

Observation 12. *Let $\mathcal{T}$ be a set of trees closed under formation of subtrees and let $s : \mathcal{T} \to \mathbb{N}$ be defined as follows:*

- $s(T) = 1$ *if t has at most one internal node,*
- $s(T) = max\{s(X) : X \in Sub(T)\}$ *if there is exactly one tree R such that* $s(R) = max\{s(X) : X \in Sub(T)\}$,
- $s(T) = max\{s(X) : X \in Sub(T)\} + 1$ *if there are at least two trees R_1, $R_2 \in Sub(T)$ such that $R_1 \neq R_2$ and* $s(R_1) = s(R_2) = max\{s(X) : X \in Sub(T)\}$.

Then $s(T)$ is $O(\log_2 |T|)$, where $|T|$ is the number of nodes of tree T.

Proof. We will show that $s(T) \leq \log_2 |T| + 1$. The proof will be by induction on the structure of T. The case where T has at most one internal node is obvious. Now it is enough to consider the last two cases of the definition of s. In the second case we have that $s(T) = s(R) \leq log_2|R| + 1 \leq log_2|T| + 1$. In the third case we assume without a loss of generality that $|R_1| \leq |R_2|$. Hence, we have that

$$s(T) = s(R_1) + 1 \leq \log_2 |R_1| + 1 + 1 = \log_2 |R_1| + \log_2 2 + 1 =$$

$$= \log_2 (2 \cdot |R_1|) + 1 \leq \log_2 |T| + 1.$$

The second lemma needed to prove that TERM-SAT($\mathbf{A}$) for any finite algebra $\mathbf{A}$ is in NQL is the following:

Lemma 13. *Let $\mathbf{A}$ be a finite algebra. Then the problem* POL-VAL(A) *is solvable in quasilinear time.*

Proof. We will show a quasilinear time algorithm computing the value of a given polynomial without any variable in it. Our algorithm will step by step compute values of unbranched paths in the tree of the considered polynomial as long as it computes the value of the polynomial. An *unbranched path* of a polynomial is a maximal subtree of the tree of the polynomial such that every node of this subtree has at most one child which is not a leaf.

Let p be the polynomial given in the input. We will use five main tapes. We assume that the polynomial p is on the first tape and the tables of fundamental operations of $\mathbf{A}$ corresponding to all function symbols occurring in p are on the second tape. We use the standard lexicographic order when sorting tokens.

To make it clearer how our algorithm works we present it through the example of the unbranched path:

$$f(a_1, g(a_2, h(a_1)), a_3), \tag{1}$$

where f, g, h are the function symbols and $A = \{a_1, a_2, a_3\}$.

Algorithm 14.

1. Repeat steps (2)-(8) until the value of p is computed.
2. Rewrite all unbranched paths from the first to the third tape in the following way: for every internal node of a path write the function symbol from the node with all possible values of children. Rewrite nodes contained in the same path in the order from the top to the bottom. Rewrite paths in the order of their occurrence in p. For example (1) we will obtain:

   $$f(a_1, a_1, a_3), f(a_1, a_2, a_3), f(a_1, a_3, a_3), g(a_2, a_1), g(a_2, a_2), g(a_2, a_3), h(a_1)$$

3. Now on the third tape the tokens are the occurrences of function symbols together with their arguments. Let m be the number of tokens. Write on the fourth tape the numbers from 1 to m.
4. Sort tokens on the third tape in the lexicographic order and apply the same transformation simultaneously to the corresponding numbers on the fourth tape.
5. Substitute tokens on the third tape by their values.
6. Restore the original order of tokens of the third tape by simultaneous sorting numbers on the fourth tape.

   Suppose that for the path in the example (1) we obtain the values:

   $$a_3, a_3, a_1, a_1, a_3, a_1, a_2$$

7. Compute values of all paths and write them on the fifth tape. For the path in the example (1) this will be the value a_1.

8. Substitute every unbranched path in the polynomial on the first tape by its value.

It is obvious that the above algorithm computes the value of p. The problem is if the computations could be done in quasilinear time. First observe that the length of the symbols on the third tape in step (2) is at most linearly bigger than the size of the polynomial written on the first tape. It is because the algebra is a part of the algorithm not a part of the input. Hence, it is clear that steps (3), (4) and (6) may be computed in quasilinear time. To compute step (5), it is enough to read the second and third tapes once. Hence, we have that step (5) may be done in quasilinear time. Step (7) can be done in linear time reading the values on the fourth tape from the end to the beginning. Now step (8) can by done in linear time if, for example, in step (2) we write the positions of the occurrences of unbranched paths on an auxiliary tape.

Step (2) may be computed in quasilinear time using an algorithm reading the polynomial from its end to the beginning with the auxiliary tape as a stack. The algorithm stores on the stack the read constants and function symbols with the information if a given symbol belongs to an unbranched path. If the algorithm reaches a function symbol then it gets and removes arguments of this function from the stack. Next, the algorithm pushes on the stack the function symbol with the information if it belongs to an unbranched path. Function symbols belonging to some unbranched paths are written on the third tape together with all possible values of their arguments.

It remains to show that steps (2)-(8) of Algorithm 14 will be computed at most $O(\log n)$ times, where n is the length of p. Note that the value of every unbranched path of p is computed in one turn of the loop. Hence, steps (2)-(8) will be computed $s(T)$ times, where s is the function from Observation 12 and T is the tree of the polynomial p. To complete the proof consider that from Observation 12 $s(T)$ is $O(\log n)$.

Now we are ready to prove the main theorem in this section.

Theorem 15. *For any finite algebra* $\mathbf{A}$ *the problem* TERM-SAT($\mathbf{A}$) *is in NQL.*

Proof. The following algorithm running on a non-deterministic Turing machine checks if a given equation over $\mathbf{A}$ has a solution:

Algorithm 16.

1. Write on the auxiliary third tape a list of values of variables (in a non-deterministic way).
2. Substitute variables occurring in terms of the equation by the values written on the third tape.

3. Compute values of polynomials obtained in step (2).

4. If the values computed in step (3) are equal return YES. Otherwise return NO.

The fact that the above algorithm solves TERM-SAT(**A**) on a non-deterministic Turing machine is obvious. The only thing we need to justify is that steps (2) and (3) may be computed in quasilinear time. We have it from Corollary 11 and Lemma 13.

The immediate corollary of the above theorem is that POL-SAT(**A**) is in NQL for any finite algebra **A**.

5 TERM-SAT and Two-Element Algebras

In this section we show the classification of two-element algebras according to the computational complexity of TERM-SAT. We will use the same type of Turing machines as before but we will consider only algebras on the set $\{0, 1\}$.

First we start with algebras for which TERM-SAT is NQL-complete :

Lemma 17. *For a function $f : 2^n \to 2$ the following are equivalent:*

(i) *f is affine, i.e. $f \in Clo(\mathbf{L_1})$*

(ii) *for any pairwise different $\overline{a_0}, \overline{a_1}, \overline{a_2}, \overline{a_3} \in 2^n$ which differ at most on positions $0 \leq i \neq j \leq n-1$, we have one of the following:*
- $f(\overline{a_0}) = f(\overline{a_1}) = f(\overline{a_2}) = f(\overline{a_3})$ *or*
- $f(\overline{a_g}) = f(\overline{a_h}) \neq f(\overline{a_k}) = f(\overline{a_l})$, *where* $\{g, h, k, l\} = \{0, 1, 2, 3\}$

Proof. $(i) \Rightarrow (ii)$

It is enough to observe, that $f(x_0 \ldots x_{n-1}) = \sum_{i \in I} x_i + c$, for some set $I \subseteq \{0 \ldots n-1\}$ and $c = 0$ or 1. Then the proof is obvious. $(ii) \Rightarrow (i)$

First, we show that for any $0 \leq i \leq n-1$ exactly one of the following holds:

(a) $\forall_{(a_0 \ldots a_{n-1}) \in 2^n}\ f(a_0 \ldots a_{i-1}, 0, a_{i+1} \ldots a_{n-1}) = f(a_0 \ldots a_{i-1}, 1, a_{i+1} \ldots a_{n-1})$

(b) $\forall_{(a_0 \ldots a_{n-1}) \in 2^n}\ f(a_0 \ldots a_{i-1}, 0, a_{i+1} \ldots a_{n-1}) \neq f(a_0 \ldots a_{i-1}, 1, a_{i+1} \ldots a_{n-1})$

Suppose, $f(a_0, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_{n-1}) \neq f(a_0, \ldots, a_{i-1}, 1, a_{i+1}, \ldots, a_{n-1})$ for some $i, a_0, \ldots, a_{i-1}, a_{i+1}, \ldots, a_{n-1}$.

Take a tuple $(b_0, \ldots, b_{i-1}, b_{i+1}, \ldots, b_{n-1})$ which differs from $(a_0, \ldots, a_{i-1}, a_{i+1}, \ldots, a_{n-1})$ exactly on one position.

Now, from (ii) we have $f(b_0 \ldots b_{i-1}, 0, \ldots b_{n-1}) \neq f(b_0 \ldots b_{i-1}, 1, \ldots b_{n-1})$. Notice, that we can reach any element of 2^{n-1} from $(a_0, \ldots, a_{i-1}, a_{i+1} \ldots, a_{n-1})$, step by step, by changing only one position in every step. Hence, for all tuples $(c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n) \in 2^{n-1}$ we have that $f(c_0 \ldots c_{i-1}, 0, c_{i+1}, \ldots c_{n-1}) \neq f(c_0 \ldots c_{i-1}, 1, c_{i+1}, \ldots c_{n-1})$

Let I be the set of all $0 \leq i \leq n-1$ such that (b) is true. It is easy to see that $f(x_0 \ldots x_{n-1}) = \sum_{i \in I} x_i + f(0, \ldots, 0)$.

Lemma 18. *For an algebra* $\mathbf{A}$ *such that* $\wedge, \neg \in Clo(\mathbf{A})$ *and* $\wedge$ *is frugally definable over* $\mathbf{A}$*, the problem* TERM-SAT($\mathbf{A}$) *is NQL-complete.*

Proof. It is clear that $\vee$ and the constant 1 are also definable over $\mathbf{A}$. There is an obvious reduction from 3-SAT to TERM-SAT($\mathbf{A}$), which transforms the formula

$$(x_1^1 \vee x_2^1 \vee x_3^1) \wedge \ldots \wedge (x_1^k \vee x_2^k \vee x_3^k),$$

where x_i^j are the variables or negations of variables, into the equation

$$(x_1^1 \vee \ x_2^1 \vee x_3^1) \wedge \ldots \wedge (x_1^k \vee x_2^k \vee x_3^k) = 1.$$

In order to obtain a reduction to TERM-SAT($\mathbf{A}$) we have to express the last equation in the language of $\mathbf{A}$. First in the preprocessing we find a term defining the constant 1 and the term $v(x_1^1, x_2^1, x_3^1)$ such that $v^{\mathbf{A}}(x_1^1, x_2^1, x_3^1) = x_1^1 \vee x_2^1 \vee x_3^1$.

Let $c_2(x_1, x_2)$ be a term over $\mathbf{A}$ which frugally defines $x_1 \wedge x_2$. Now we are looking for a term $c_k(x_1, \ldots x_k)$ over $\mathbf{A}$ defining the function $x_1 \wedge \ldots \wedge x_k$. Notice that k is not known in the preprocessing.

One can see that if we have a frugal term c_i representing the conjunction of i variables, we can obtain a frugal term representing the conjunction of $2i$ variables by applying Lemma 10 to the terms c_2, c_i, c_i. In this way, in quasilinear time we can obtain the term $c_k(x_1, \ldots x_k)$.

In the last step of our reduction we apply Lemma 10 for terms $c_k(x_1, \ldots x_k)$, $v(x_1^1, x_2^1, x_3^1) \ldots v(x_1^k, x_2^k, x_3^k)$.

It turns out that there are only two clones for which TERM-SAT is NQL-complete. The first one is the clone of all operations on the two-element set.

Lemma 19. TERM-SAT *for the clone* $Clo(2, \wedge, \neg)$ *is NQL-complete.*

Proof. Let $\mathbf{A} = (2, F)$ and $Clo(\mathbf{A}) = Clo(2, \wedge, \neg)$. Thus, there exist $f_u, f_p \in F$ such that f_u is not monotone and f_p is not affine ($Clo(\mathbf{A}) \nsubseteq Clo(\mathbf{A_1})$ and $Clo(\mathbf{A}) \nsubseteq Clo(\mathbf{L_1})$).

Because f_u is not monotone there are $(a_0, \ldots, a_i, a_{i+1}, \ldots, a_{n_u-1}) \in 2^{n_u-1}$ such that $f_u(a_0, \ldots, a_i, 1, a_{i+1}, \ldots, a_{n_u-1}) < f_u(a_0, \ldots, a_i, 0, a_{i+1}, \ldots, a_{n_u-1})$. It is easy to see that $\neg(x) \equiv f_u(a_0, \ldots, a_i, x, a_{i+1}, \ldots, a_{n_u-1})$ (obviously $1, 0 \in Clo(\mathbf{A})$) and in consequence $\neg$ is frugally definable over $\mathbf{A}$.

On the other hand, because of the fact that f_p is not affine, from Lemma 17 there are pairwise different $\overline{b_1}, \overline{b_3}, \overline{b_3}, \overline{b_4} \in 2^{n_p}$ which differ at most on the positions $0 \leq i \neq j \leq n_p$ and $f_p(\overline{b_1}) = f_p(\overline{b_3}) = f_p(\overline{b_3}) \neq f_p(\overline{b_4})$.

Now, consider $F(x, y) = f_p(b_0, \ldots, b_{i-1}, x, b_{i+1}, \ldots, b_{j-1}, y, b_{j+1}, \ldots, b_{n_p-1})$. It is easy to see that $F(x, y)$ is equivalent to one of the following functions $x \vee y$, $x \wedge y$, $\neg x \vee \neg y$, $\neg x \wedge \neg y$, $\neg x \vee y$, $\neg x \wedge y$, $x \vee \neg y$ or $x \wedge \neg y$. The term $F(x, y)$ is of course a frugal one.

To complete the proof, observe that using $F(x, y)$ and $\neg$ we can create a frugal term defining $\wedge$. From this fact based on Lemma 18 we have that TERM-SAT($\mathbf{A}$) is NQL-complete.

The second clone for which TERM-SAT is NQL-complete is the clone D_3

Lemma 20. TERM-SAT *for the clone* $Clo(2, d, \neg)$ *is NQL-complete.*

Proof. Let $\mathbf{A}$ be an algebra with $Clo(\mathbf{A}) = Clo(\mathbf{D_3})$. If to the basic operations of $\mathbf{A}$ we add the constant operation 1 we obtain a new algebra $\mathbf{A'}$ such that $Clo(\mathbf{A'}) = Clo(2, \wedge, \neg)$.

Call $\neg$ a term in the language of $\mathbf{A}$ representing negation.

We will define a reduction from 3-SAT to TERM-SAT($\mathbf{A}$). Let S be a 3-SAT instance. First, we verify whether S is a tautology or not. This can be easily done in quasilinear time, for example, by checking if any of the clauses can be false. We reduce every tautology to the equation $x = x$.

If S is not a tautology we compute the reduction from the proof of Lemma 19 to obtain the term $t'(x_1, \ldots, x_k)$ over $\mathbf{A'}$ which is equal to 1 iff S is true (for the same valuation of variables). Next in $t'(x_1, \ldots, x_k)$ we replace every occurrence of constant 1 with a new variable u. We call the new term $t(x_1, \ldots, x_k, u)$. Now we reduce S to the following instance of TERM-SAT($\mathbf{A}$)

$$t(x_1, \ldots, x_k, u) = t(x'_1, \ldots, x'_k, \neg u), \tag{2}$$

where $\{x_1, \ldots, x_k\} \cap \{x'_1, \ldots, x'_k\} = \emptyset$.

We have to show that the procedure described above is in fact a reduction.

One can see that all the above operations can be done in the quasilinear time and that the size of $t(x_1, \ldots, x_k, u)$ is at most quasilinearly larger than that of S.

Observe that for every term g in the language of $\mathbf{A}$ we have

$$\neg g(x_1, \ldots, x_m) = g(\neg x_1, \ldots, \neg x_m) \tag{3}$$

If S is not satisfiable then $\forall_{(a_1,\ldots,a_k)\in 2^k}\ t^{\mathbf{A}}(a_1, \ldots, a_k, 1) = 0$ and therefore from (3) we have that $\forall_{(a_1,\ldots,a_k)\in 2^k}\ t^{\mathbf{A}}(a_1, \ldots, a_k, 0) = 1$. Consequently, (2) cannot be satisfiable.

Conversely, assume that S is satisfiable and not a tautology.

There are $(a_1,\ldots,a_k),(b_1,\ldots,b_k)\in 2^k$ such that $t^{\mathbf{A}}(a_1,\ldots,a_k,1)=1$ and $t^{\mathbf{A}}(b_1,\ldots,b_k,1)=0$. Now, using (3), we have that

$$t^{\mathbf{A}}(a_1,\ldots,a_k,1)=t^{\mathbf{A}}(\neg b_1,\ldots,\neg b_k,0)$$

Therefore (2) is satisfiable.

Before the proof of the main theorem of this section we have to consider affine algebras, one of the classes of algebras for which TERM-SAT is in QL.

Lemma 21. *Let $\mathcal{C}$ be a clone on the two-element set such that $\mathcal{C}\subset Clo(2,+,\neg)$. Then* TERM-SAT *for the clone $\mathcal{C}$ is in QL.*

Proof. Let $\mathbf{A}=(A,F)$ with $Clo(\mathbf{A})=\mathcal{C}$.

Every operation $f\in Clo(2,+,\neg)$ may be expressed in the form:

$$f(x_1,\ldots,x_k)=\sum_{i\in I_f}x_i+c_f, \tag{4}$$

where $c_f\in\{0,1\}$ and I_f is a set of indexes of arguments on which f depends.

The plan of the proof is the following. First we show how to obtain in quasilinear time an expression in form (4) from the table of a given operation. Next we show how to transform a given term over $\mathbf{A}$ into the form (4). Finally, we show a quasilinear time algorithm determining whether or not a given equation between the expressions in form (4) has a solution.

Observe that to transform the table of a function into an expression in form (4) we need only to obtain a set I_f and the constant c_f. Fortunately, a k-ary operation f depends on the i-th argument iff the first and the $(1+2^{k-i})$-th elements of the table of the operation f are different. Additionally, the first element of the table is equal to the constant c_f from expression (4). So to obtain the set I_f and the constant c_f, it is enough to read the table once and make k comparisons. This is an easy consequence of the fact that f may be expressed in form (4).

Consequently we may assume that instead of the table for any basic operation f we have the set I_f and the constant c_f written on the second tape.

Now we need an algorithm which transforms a given term into form (4). The algorithm we show uses six main tapes. Tapes one and two contain the input. Tape three contains variables which have a chance to occur in the resulting expression and tape four is used in the computation of c_f. On tape five the algorithm writes the output. The sixth tape is used as a stack. As tokens in this algorithm we mean the function symbols and the variables. For simplicity of the algorithm we assume that terms are written without brackets.

The main idea of our algorithm is that we traverse the tree of a given term. For every node we skip subtrees on which the operation corresponding to the node does not depend. We use a stack instead of the recursion. The algorithm is quite similar to the classical one computing the value of the expression written in the Reverse Polish Notation.

Algorithm 22.

1. Rewrite the given term in such a way that after every function symbol f there is written the set I_f and the constant c_f (this may be done in a similar way as the composition of polynomials in the proof of Lemma 10).
2. Push on the stack the symbol 1.
3. Read the next token from tape 1 (denote it by d). If there is no token to read go to step 9.
4. Read and pop a symbol from the stack (denote it by s).
5. If $s = 1$ and $d = x$ is an variable then add x to the list on tape 3 and go to step 3.
6. If $s = 1$ and $d = f$ is a function symbol then
   (a) Read c_f and add it (modulo 2) to the value written on tape 4. If tape 4 is empty then just write c_f on it.
   (b) Push on the stack the values of the indicator function of the set I_f in the order from the last to the first argument.
   (c) Go to step 3.
7. If $s = 0$ and d is a variable then go to step 3.
8. If $s = 0$ and d is a function symbol then move the pointer on tape 1 after the last argument of this symbol and go to step 3.
9. Sort variables written on tape 3.
10. Write on tape 5 the symbol of every variable which occurrs on tape 3 an odd number of times.
11. Add on the end of tape 5 the value from tape 4 (if any value is written there).

The above algorithm is obviously computable in quasilinear time. For a given term it returns on tape 5 the list of variables on which the corresponding function depends. At the end of the list it writes the value of the constant c_f of formula (4).

The following algorithm checks in quasilinear time whether or not the given equation between terms over $\mathbf{A}$ has a solution.

Algorithm 23.

1. For the two terms from the input write on tapes 2 and 3 respectively the lists of the variables on which they depend and the corresponding constants from formula (4). Use Algorithm 22.
2. If the constants on both tapes obtained in step (1) are equal return YES
3. If the constants on both tapes obtained in step (1) are not equal then check if on tapes 2 and 3 there are exactly the same variables.
   (a) If there are the same variables then return NO.
   (b) Otherwise return YES.

Now, we are ready to show the whole classification of the computational complexity of TERM-SAT for two-element algebras.

Theorem 24. *Let $\mathcal{C}$ be a clone on a two-element set.*

TERM-SAT *for the clone $\mathcal{C}$ is NQL-complete if and only if one of the following holds*

– $\mathcal{C} = Clo(2, \wedge, \neg) = Clo(\mathbf{C_1})$

– $\mathcal{C} = Clo(2, d, \neg) = Clo(\mathbf{D_3})$

Otherwise, TERM-SAT *for C is in QL.*

Proof. From Lemmas 19 and 20 we know that if $\mathcal{C} = Clo(2, \wedge, \neg)$ or $\mathcal{C} = Clo(2, d, \neg)$ then TERM-SAT for $\mathcal{C}$ is NQL-complete.

Observe that if $Clo(\mathbf{A}) \neq Clo(2, \wedge, \neg)$ and $Clo(\mathbf{A}) \neq Clo(2, d, \neg)$ then there are four possible cases:

All functions from $\mathcal{C}$ are

– affine, i.e. $\mathcal{C} \subseteq Clo(\mathbf{L_1})$

– monotone, i.e. $\mathcal{C} \subseteq Clo(\mathbf{A_1})$

– 1-valid, i.e. $\mathcal{C} \subseteq Clo(\mathbf{C_2})$

– not 0-valid, i.e. $\mathcal{C} \subseteq Clo(\mathbf{C_3})$

In the first case TERM-SAT is in QL from Lemma 21. The quasilinear time algorithms in the last three cases are obvious.

An easy consequence of the above theorem is the following:

Corollary 25. *Let* $\mathbf{A}$ *be a two-element algebra.*

If $d \in Clo(\mathbf{A})$ and $\neg \in Clo(\mathbf{A})$ then TERM-SAT($\mathbf{A}$) *is NQL-complete. Otherwise,* TERM-SAT($\mathbf{A}$) *is in QL.*

Since for the two-element algebras $\mathbf{A} = (A, F)$ and $\mathbf{A}' = (A, F \cup \{0, 1\})$ $Pol(\mathbf{A}) = Clo(\mathbf{A}')$ and POL-SAT($\mathbf{A}$) is obviously equivalent to TERM-SAT($\mathbf{A}'$) in quasilinear time, we have another easy corollary of Theorem 24.

Theorem 26. *Let A be a two-element algebra and $\mathcal{C} = Pol(\mathbf{A})$.*

If the clone $\mathcal{C} = Pol(2, \wedge, \neg)$ then POL-SAT($\mathbf{A}$) *is NQL-complete. Otherwise* POL-SAT($\mathbf{A}$) *is in QL.*

6 Conclusion

We have characterized the computational complexity of TERM-SAT for two element algebras in terms of QL and NQL. It is natural to ask what the situation looks like if we considered larger algebras. Unfortunately, for such algebras we do not have a similar description of the clone lattice as we have for two element algebras. Additionally, there is a continuum of clones of algebras with three or more elements.

We have shown that for two element algebras TERM-SAT is NP-complete iff it is NQL-complete and TERM-SAT is in P iff it is in QL. Is this also true for larger algebras? It had been mentioned in the introduction that there exist NP-complete problems which are not in NQL. Analysis of the known proofs of NP-completeness of TERM-SAT for different classes of finite algebras indicates that some of these proofs seem to be hard to use when proving NQL-completeness of TERM-SAT. Hence, we state the conjecture that there exists a finite algebra $\mathbf{A}$ such that TERM-SAT($\mathbf{A}$) is NP-complete but not NQL-complete. It seems to us that it is possible to find such an example among the primal algebras, i.e. algebras generating the clone of all operations.

References

[Angluin and Valiant 1979] Dana Angluin and Les Valiant, "Fast Probabilistic Algorithm for Hamiltonian Circuits and Matchings", J. of Computer and system Sciences (1979) 155-193.

[Barrington et al. 2000] D. M. Barrington, P. McKenzie, C. Moore, P. Tesson and D. Thérien, "Equation satisfiability and program satisfiability for finite monoids", Math. Found. Comp. Sci. (2000) 127-181.

[Bulatov 2002] A. Bulatov, "A dichotomy theorem for constraints on a three-element set", Proceedings of the 43rd Symposium on Foundations of Computer Science, IEEE Computer Society, Washington (2002) 649-658.

[Broniek 2006] P.Broniek, "Solving equations over small unary algebras", Discrete Mathematics and Theoretical Computer Science proc. AF (2006) 49-60.

[Cook 1972] S.A.Cook, "A hierarchy for non-deterministic time complexity", Proc Fourth Annual ACM Symp on Theory of Computing, ACM, New York (1972) 187-192.

[Creignou 1995] N.Creignou, "The class of problems that are linearly equivalent to Satisfiability or an uniform method for proving NP-copleteness", Theoretical Computer Science 145 (1995) 111-145.

[Dewdney 1982] A.K.Dewdney, “Linear transformations between combinatorial problems”, Internat. J. Comput. Math. 11 (1982) 91-110.

[Goldmann and Russel 2002] M. Goldmann and A. Russel, “The Complexity of Solving Equations over Finite Groups”, Inf. Comput. 178(1) (2002) 253-262.

[Gorazd and Krzaczkowski 2011] T.Gorazd and J.Krzaczkowsk, “The complexity of problems connected with two-element algebras”, Reports on Mathematical Logic 46 (2011) 91-108.

[Gorazd and Krzaczkowski 2010] T.Gorazd and J.Krzaczkowski, “Term equation satisfiability over finite algebras”, Internat. J. Algebra Comput., 20(8) (2010) 1001-1020.

[Gräedel 1990] E.Gräedel, “On the notion of linear time”, Internat. J. Found. Comput. Sci. 1 (1990) 295-307.

[Grandjean 1994] E.Grandjean, “Linear time algorithms and NP-complete problems”, SIAM J. Comput. 23 (1994) 573-597.

[Gurevich and Shelah 1989] Y.Gurevich and S.Shelah, “Nearly-linear time”, Proc. Logic in Botik’89, Lecture Notes in Computer Science 363, Springer, Berlin (1989) 108-118.

[Horváth 2011] G. Horváth, “The complexity of the equivalence and equation solvability problems over nilpotent rings and groups”, Algebra Universalis 66(4) (2011) 391-403.

[Horváth and Szabó 2006] G.Horváth and C.Szabó, “The complexity of checking identities over finite groups”, Internat. J. Algebra Comput. 16(5) (2006) 931-939.

[Horváth and Szabó 2011] G.Horváth and C.Szabó, “The extended equivalence and equation solvability problems for groups”, Discrete Math. Theor. Comput. Sci., 13(4) (2011) 23-32.

[Horváth and Szabó 2012] G.Horváth and C.Szabó, Equivalence and equation solvability problems for the group A_4. J. Pure Appl. Algebra, 216(10) (2012) 2170-2176.

[Klíma et al. 2007] O.Klíma, P.Tesson and D.Thérien, “Dichotomies in the Complexity of Solving Systems of Equations over Finite Semigroups”, Theory of Computing Systems 40(3) (2007) 263-297.

[Larose and Zádori 2006] B. Larose and L. Zádori, “Taylor terms, constraint satisfaction and the complexity of polynomial equations over finite algebras”, Internat. J. Algebra Comput. 16(3) (2006) 563-581.

[McKenzie et al. 1987] R.McKenzie, G.McNulty and W.Taylor, “Algebras”, Lattices, and Varieties. Vol. I., Mathematics Series, Wadsworth and Brooks/Cole, 1987

[Post 1941] E.Post, “The Two-valued Iterative Systems of Mathematical Logic“ Annals of Mathematics Studies, N.5, Princeton University Press, 1941

[Schaefer 1978] T.J.Schaefer, ”The complexity of satisfiability problems“, Proceedings of the 13th ACM Symposium on Theory of Computing, ACM, New York (1978) 216-226.

[Schnorr 1978] C.P.Schnorr, ”Satisfiability Is Quasilinear Complete in NQL“, J. ACM 25 (1978) 136-145.

[Schwarz 2004] B. Schwarz, ”The Complexity of Satisfiability Problems over Finite Lattices“, Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science 2996, Springer, Berlin (2004) 31-43.