Fast Quantifier Elimination Means P = NP

Mihai Prunescu[1,2]

[1] "Dr. Achim Hornecker — Software-Entwicklung und I.T.-Dienstleistungen", Freiburg im Breisgau, Germany
[2] Institute of Mathematics "Simion Stoilow" of the Romanian Academy, Bucharest, Romania

Abstract. The first part is a survey of Poizat's theory about fast elimination of quantifiers and the P = NP question according to the unit-cost model of computation, as developed along the book [7]. The second part is a survey of the structure with fast elimination constructed by the author in [9].

1 Introduction

In [9] a structure with fast elimination is constructed. Here I intend to recall the whole context of this construction and to explain why it is said that the structure constructed there satisfies P = NP for the unit-cost model of computation over algebraic structures. The construction itself will be also shortly presented here, but I will emphasize exactly the steps which hasn't been presented with too much details in the cited paper. With this occasion I shall try to answer to some frequently asked questions. This extended abstract should be seen as a complement to [9].

The notation used is the standard notation for mathematical logic. Bold-faced letters like $\boldsymbol{u}$ denote tuples $(u_1, u_2, \ldots, u_n)$.

2 Machines, Circuits and Existential Formulas with Parameters

The unit-cost complexity over algebraic structures was born with the paper of L. Blum, M. Shub and S. Smale dedicated to unit-cost computations over the ordered field of the reals. The approach presented here belongs to B. Poizat and was developed by him along the lines of the book "Les petits cailloux", [6].

Let L be an abstract finite signature for algebraic structures. L consists in a set of constant-symbols (c_i), a set of relation-symbols (R_j) with arities (n_j) and a set of operation-symbols (f_k) with arities (m_k). We fix an L-structure S. The interpretation of L in S shall be done using the same symbols. (We do not make a notational difference between symbol and interpretation here.)

Definition 1. A Turing machine working over the L-structure S is a multi-tape Turing machine with finitely many states and the following ideal abilities:

- The machine works with a possibly infinite alphabet consisting in the elements of S. In other words, every element of S has a unique name. This name can be written in one cell of the machine as component of the input or as result of a computation. During a computation, the name of a given element may arrise several times in different cells. Equivalently, you can think about the underlying set of S like about a (possibly infinite) alphabet used by a Turing machine.
- Let $x_i \in S$ be the content of a cell of the tape number i which is read at this moment by the head H_i. Following program lines can occur: stop; H_i+, H_i- to move a head on a tape; $x_i := f(x_k, x_l, \dots, x_s)$, where f is a function symbol in L or the identity (this means $x_i := x_j$); if $R(x_k, x_l, \dots x_s)$ then continue with state q, else continue with state q', where R is a relation symbol in L or the equality; if H_i is reading an empty cell then continue with state q, else continue with state q'.
- Any such step is performed in a unit of time.

This will be simply called a machine over S.

To say that "a cell is empty" is the same as saying "a cell contains a blank symbol" — I will not insist here on this. The multi-tape formulation given here is directly used used by Poizat in order to prove his Theorem 1. One can define the notion of computability over algebraic functions using only one-tape Turing machines: if L is finite and all relations (functions) have a finite arity, then there is a translation of multi-tape Turing machines in one-tape Turing machines, that does not change the defined class P. Indeed, if k is the number of tapes, consider the $mk + i$-th cell of the one-tape machine to be the m-th cell of the i-th tape ($m \in \mathbb{Z}$, $0 \leq i < k$), and multiply the number of states with k.

Definition 2. A problem is a subset of $S^* := \coprod_{n \in \mathbb{N}} S^n$ seen as set of finite inputs which are accepted by a machine over S. For an input $\boldsymbol{x} \in S^*$ let $|\boldsymbol{x}|$ be its length. Note: the symbol $\coprod$ used here is meant as disjoint union. There are no identifications between S^n and factors of S^m when $n \leq m$.

Definition 3. For an L-structure S we define the complexity class $\mathrm{P}(S)$ as the set of all problems over S accepted by deterministic machines over S in polynomial time. This means the following: there is a polynomial $p(n)$ with natural coefficients such that for all inputs $\boldsymbol{x} \in S^*$ the decision is taken in less that $p(|\boldsymbol{x}|)$ units of time.

Definition 4. A problem $B \subset S^*$ is said to belong to the class $\mathrm{NP}(S)$ if and only if there is a problem A in $\mathrm{P}(S)$ and a polynomial $q(n)$ such that for all $\boldsymbol{x} \in S^*$:

$$\boldsymbol{x} \in \mathrm{B} \leftrightarrow \exists\ \boldsymbol{y} \in S^* \quad |\,\boldsymbol{y}\,| = q(|\boldsymbol{x}|) \ \wedge\ \boldsymbol{xy} \in \mathrm{A}.$$

By $\boldsymbol{xy}$ we mean the concatenation of the strings $\boldsymbol{x}$ and $\boldsymbol{y}$.

At this point I must make some commentaries about parameters. In the literature unit-cost Turing machines are allowed to contain a finite tuple of elements of the structure, and to use them in the computations. The classical notation for the complexity classes with allowed parameters is P and NP. For the situation described here, where a finite signature is fixed and the machines are not allowed to contain other parameters than the fixed interpretation of the given constants, the classical notation is P^0 and NP^0. I like to work with the definitions and the notations as given here because I consider them more clear and more resonant with the model-theoretic point of view. It is worth to remark that by writing down the things in this way, I didn't really introduce a restriction. As an anonymous referee pointed out *using the classical notation*: if a structure has $\mathrm{P}^0 = \mathrm{NP}^0$ then it has P = NP, because all tuples in a problem can be completed with the tuple of suplementary parameters. On the other hand, if the structure has P = NP, one can expand the structure with a finite number of constants such that the new one has $\mathrm{P}^0 = \mathrm{NP}^0$: the new constants are the parameters used to solve some NP-complete problem over the structure. This notion is explained in the sequel, together with the equivalent of the work of Cook for the classes P and NP over a an L-structure S.

Definition 5. Suppose from now on that the language L contains at least two constants, which will be called 0 and 1. An L-circuit is a finite directed graph without cycles. The vertices of the graph are called gates, and the directed edges are called arrows. There are input-gates, constant-gates, operation-gates, relation-gates, selection-gates and output-gates; at least one input-gate and one output-gate must be present. We call fan-in of a gate the number of arrows going into the gate. The fan-out is the number of arrows going from the gate outside. All gates have an unbounded fan-out. The gates input and output elements of S.

- An input-gate has fan-in 1. It just copies the input element and sends it along the outgoing arrows.
- A constant-gate has fan-in 0. It sends copies of the corresponding (in S interpreted) constant along the outgoing arrows.
- Operation-gates and relation-gates have a fan-in equal to the arity of the corresponding operation (relation). The operation-gate for f computes the value $f(x_1, \dots, x_s) \in S$. The relation-gate checks if the relation $R(x_1, \dots, x_s)$ is true and outputs the constant 1 then, else it outputs the constant 0.
- The selection-gate has fan-in 3 and computes the function $s(x, y, z)$, where $s(0, y, z) = y$, $s(1, y, z) = z$ and $s(x, y, z) = x$ if $x \neq 0$ and $x \neq 1$.
- In the case of the so-called decision circuits there is only one output gate that outputs 0 or 1.

The complexity-measure of a circuit C is its number of gates $|C|$.

Considering the circuits to be compressed first-order formulas, they are a good instrument for making concepts like problem or complexity class independent of a special type of computing device.

Theorem 1. *Let M be a machine with k tapes over S, working in a bounded time $\leq t(n)$, where $t(n) \geq n$ is a function of the length $|x|$. Then there is a* *recursive sequence* $(C_n(x_1, \dots, x_n))$ *with* $|C_n| \leq t(n)^{k+1}$*, such that* $C_n(\boldsymbol{x})$ *gives for input* $\boldsymbol{x}$ *of length* n *the same result as the machine* M*, and* C_n *are uniformly constructed by a classical Turing machine in polynomial time* $p(n)$*.*

Definition 6. The satisfiability problem for L-circuits with parameters in S:
Given a string $\boldsymbol{wa} \in S^*$, such that the subword $\boldsymbol{w}$ is a binary word made up by the special boolean constants $0, 1 \in S$ and encoding a decision L-circuit $C(\boldsymbol{x}, \boldsymbol{y})$;
It is asked if there is $\boldsymbol{b} \in S^*$ of appropriate length, such that

$$(S, \boldsymbol{a}, \boldsymbol{b}) \models C(\boldsymbol{a}, \boldsymbol{b}) = 1.$$

Don't wonder about our use of the symbol "models" ($\models$) in this context. As already said, circuits are first-order formulas written down compactly.

It follows directly from the theorem that the satisfiability problem for L-circuits with parameters in S is NP(S)-complete. This problem belongs to P(S) if and only if S has the property P = NP.

We now come to the most delicate point of the reduction: from the satisfaction of circuits to the satisfaction of first-order formulae.

Definition 7. The satisfiability problem for quantifier-free L-formulae with parameters in S:
Given a string $\boldsymbol{wa} \in S^*$, such that the subword $\boldsymbol{w}$ is a binary word made up by the special boolean constants $0, 1 \in S$ and encoding a quantifier-free L-formula $\varphi(\boldsymbol{x}, \boldsymbol{y})$;
It is asked whether there is a $\boldsymbol{b} \in S^*$ of appropriate length, such that

$$(S, \boldsymbol{a}, \boldsymbol{b}) \models \varphi(\boldsymbol{a}, \boldsymbol{b}).$$

At first sight there is no big difference between this satisfiability problem and the satisfiability problem concerning circuits. In fact, there is an important difficulty here. It is true that every circuit is logically equivalent with a quantifier-free formula, *but the translation might not be possible in polynomial time!*

Example. (Poizat) Let S be a structure possessing an associative addition denoted by $+$ and let $C_n(x, y)$ be the circuit $x \Rightarrow + \Rightarrow + \dots \Rightarrow + \rightarrow = \leftarrow y$ containing n addition-gates and the gate $=$ that checks the equality. $C_n(x, y) = 1$ is equivalent with the formula $x + x + \dots + x = y$ with $2^n - 1$ additions.

In the case of this circuit,

$$C(x, y) = 1 \leftrightarrow \exists\, \boldsymbol{z} \quad z_1 = x + x \wedge z_2 = z_1 + z_1 \wedge \dots \wedge y = z_n + z_n.$$

This existential formula has a length which is linear in n. Even if we use more symbols for the indices in order to write them using some finite alphabet, it will have at most a quadratic length. If we forget the quantifiers and we look at the quantifier-free conjunction with parameters x and y, this quantifier-free formula is satisfied if and only if the circuit was satisfied by x and y.

Following this idea, we modify the definition for the satisfiability problem for quantifier-free formulae with parameters in S. This is just an equivalent definition, and not a new problem.

Definition 8. The satisfiability problem for quantifier-free L-formulae with parameters in S:
Given a string $\boldsymbol{wa} \in S^*$, such that the subword $\boldsymbol{w}$ is a binary word made up by the special boolean constants $0, 1 \in S$ and encoding a quantifier-free L-formula $\varphi(\boldsymbol{x}, \boldsymbol{y})$;
It is asked whether

$$(S, \boldsymbol{a}) \models \exists\, \boldsymbol{y}\ \varphi(\boldsymbol{a}, \boldsymbol{y}).$$

The satisfaction of quantifier-free formulae with parameters in S is the same thing as the truth of existential formulae with parameters in S.

Theorem 2. *The satisfiability problem for quantifier-free formulae with parameters in S is complete for the class* NP*(S). Consequently, this problem belongs to* P*(S) if and only if S has the property* P = NP.

Proof. The problem is evidently in NP by guess and check. To prove the NP-completeness, we interpret the satisfiability problem of L-circuits with parameters in S in the satisfiability problem for formulae. Let $(C(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{a})$ be an instance for the circuit-problem. For each gate in C which is not an input-gate, a constant-gate or the output-gate, consider a new variable z_{gate}. The following holds:

$$\exists\, \boldsymbol{y}\ C(\boldsymbol{a}, \boldsymbol{y}) = 1 \leftrightarrow \exists\, \boldsymbol{y}\ \exists\, z_{\text{gate}_1} \ldots \exists\, z_{\text{gate}_k}$$

$$\bigwedge_{\text{all gates}} z_{\text{gate}} = \text{gate(predecessor gates)} \wedge \text{output} = 1.$$

This gives a quantifier-free formula of a length which is polynomially bounded in the length of the circuit and which is satisfiable if and only if the circuit is satisfiable. □

Definition 9. The structure S is said to allow elimination of quantifiers if for every formula $\varphi(\boldsymbol{x})$ with quantifiers and no other free variables as in the tuple $\boldsymbol{x}$ there is a quantifier-free formula $\psi(\boldsymbol{x})$ such that:

$$S \models \forall\, \boldsymbol{x}\ (\varphi(\boldsymbol{x}) \leftrightarrow \psi(\boldsymbol{x})).$$

For equivalent definitions, we may require this only for formulae that are logically equivalent with prenex existential formulae, or even for formulae that are logically equivalent with formulae containing only one existential quantifier. Summing up all results got so far, we conclude:

Theorem 3. *The L-structure S has the property* P = NP *if and only if there is a polynomial-time machine over S which transforms all formulae $\exists\, \boldsymbol{y}\ \varphi(\boldsymbol{x}, \boldsymbol{y})$ in a circuit $C(\boldsymbol{x})$ such that:*

$$\forall\, \boldsymbol{x}\ (\exists\, \boldsymbol{y}\ \varphi(\boldsymbol{x}, \boldsymbol{y}) \leftrightarrow C(\boldsymbol{x}) = 1).$$

Proof. The structure has P = NP if and only if the decision problem for existential formulae with parameters in S is in P(S). Using Poizat's Theorem 1, there is a polynomial-time constructible sequence of circuits (C_n) such that for inputs $\boldsymbol{wa}$ of length n,

$$\exists\ \boldsymbol{y}\ \varphi(\boldsymbol{a}, \boldsymbol{y}) \leftrightarrow C_n(\boldsymbol{wa}) = 1.$$

and the binary word $\boldsymbol{w}$ encodes the existential formula $\exists\ \boldsymbol{y}\ \varphi(\boldsymbol{x}, \boldsymbol{y})$. Now fix the existential formula and let $C(\boldsymbol{x})$ be the circuit $C_n(\boldsymbol{w}, \boldsymbol{x})$. This means that the input gates corresponding to the subword $\boldsymbol{w}$ are replaced with constant-gates giving the corresponding booleans, and the input gates for the subword $\boldsymbol{a}$ remain free input gates.

For the other direction, recall that the satifiability of existential formulas with parameters in the structure is an NP-complete problem for this model of computation. If this problem lies in P, then P = NP. □

Definition 10. We say that the L-structure S has fast quantifier-elimination if it satisfies the condition occurring in Theorem 3.

In particular, all structures with P = NP allow quantifier-elimination.

Remarks. There are maybe some points which need a special emphasis:

- The model of computation is very different from the classical one. In particular, the computational devices are ideal, working with arbitrary structure elements, as for example real or complex numbers, and the structures are in general infinite. The property P = NP has to be consequently understood.
- On the other hand, if this theory is applied for some finite structure, one gets back the familiar classes P and NP from the classical theory of complexity.
- The notion "fast quantifier-elimination" is also slightly different from the similar notions used in the literature. Peoples tend to understand that the equivalent quantifier-free formula has to be short. This condition would be too strong for our purpose. Here the equivalent quantifier-free circuit has to be small, although the equivalent quantifier-free formula might be long.
- There are several results giving exponential lower bounds for the quantifier-elimination for the field of complex numbers or for the ordered field of real numbers. The known results are not sufficient for proving that those structures satisfy P $\neq$ NP! Even the exponential lower bound for purely existential formulae over the complex numbers is too weak: although the equivalent quantifier-free formula has exponential length, it is still not proved that there is no circuit of polynomial length which is equivalent with that formula!
- Last but not least: Theorems 1, 2 and 3 have been proved by Poizat in [6] at the pages 109, 149 and 156. The reader has observed that I have slightly modified the statements, because I have deleted all about supplementary parameters in machines or circuits. For this I have introduced the constant-gates.

3 A Structure with Fast Quantifier-Elimination

In the past section we shortly presented Poizat's theory about arbitrary algebraic structures that satisfy the condition P = NP. In this section we will come in contact with the structure with fast quantifier-elimination constructed by the author in [9], proving that we don't deal with the empty class.

For a short historical account: in [7] Poizat discussed the possibility of constructing a structure with P = NP and proposed some approaches; the most concrete proposal was to define a consistent truth-predicate over Malcev's freely generated tree-algebra. In [6] he produced a truth-predicate over an algebra with unary operations only, that will be presented below. His predicate V encodes the truth of existential formulae with only one free variable, in a way that to all formula $\exists \boldsymbol{y}\ \varphi(x, \boldsymbol{y})$ there is a term $\tau_\varphi(x)$ such that $\forall x\ (\exists \boldsymbol{y}\ \varphi(x, \boldsymbol{y}) \leftrightarrow V(\tau_\varphi(x)))$. Because of the unary functions, we cannot have more than one free variable in a term. The construction of his predicate is rather difficult. In [4] Hemmerling, working with a similar underlying algebra, doubled the length of the binary words satisfying some PSPACE-complete predicate in order to make it sparse (see definition below). The doubling technic is used also in the approach of Gaßner, [3]. She doesn't use the classical properties of PSPACE-complete predicates (to manifest P = NP for computations with oracles) and tried a direct construction of a structure with P = NP by encoding machine instances. The machine-oriented approaches look however very difficult; it is always quite hard to write down all conditions to be checked for a such construction. In [9] the author combined Poizat's truth-predicate, the model-theoretic view about effective quantifier elimination as described above and the doubling technic. In fact the construction is based on the following rules:

1. As in Poizat's case, a general Elimination Lemma for unary structures with generic predicates.
2. By the Elimination Lemma, the satisfaction of $\exists \boldsymbol{y}\ \varphi(\boldsymbol{x}, \boldsymbol{y})$ depends only of some local information on $\boldsymbol{x}$, which is encoded by a quantifier-free formula $\beta(\boldsymbol{x})$ of polynomial length.
3. The predicate V will encode the truth value of a special kind of $\forall\exists$-sentences.

3.1 Preliminaries

The Elimination Lemma is used in [9] without proof, so it shall be proved here. The other lemmas are quoted from [9] with some hints of proof.

Definition 11. Let $\mathcal{R}$ be an infinite set of elements called roots. The set M is the algebra freely generated by $\mathcal{R}$ with two independent unary successor operations, s_0 and s_1 such that: all elements $x \in M$ are terms in some $r \in \mathcal{R}$ and two elements x and y are equal if and only if they are the same term of the same root. The set of elements generated by a given $r \in \mathcal{R}$ is called a block. We add a unary predecessor operation p such that $p(x) = x$ defines the set of all roots and for all x, $p(s_0(x)) = p(s_1(x)) = x$. We add also a constant a to be interpreted by a fixed root and a unary predicate V called also colour, which will be constructed later. Elements x with $V(x)$ are called black, the other are called white. Our structure is (M, s_0, s_1, p, V, a) but shall be refered to as (M, V).

Definition 12. A triangle of height n is a conjunctive formula $T(x)$ as follows: For all $2^{n+1}-1$ terms (using only the successors s_0 and s_1) $t(x)$ of length $\leq n$ exactly one of the atomic formulas $V(t(x))$ or $\neg V(t(x))$ occurs in the conjunction. No other atomic formula does occur in the conjunction $T(x)$. There are exactly $2^{2^{n+1}-1}$ possible triangles of height n.

Definition 13. For a tuple $\boldsymbol{z} \in M$ we call m-neighborhood of $\boldsymbol{z}$ a conjunction of the following formulas: the formulas $T_{2m}(p^m(z_i))$ with $i = 1, \ldots, k$; and the formulae $p(y) = y$, $p(y) \neq y$, $y = y'$, $y \neq y'$, for all terms y, y' occurring in the triangles above, and exactly those equalities and negated equalities that are realized by the tuple $\boldsymbol{z}$ in M. If the tuple consists of only one element, we speak about an individual neighborhood.

Definition 14. The predicate V is called generic if it satisfies the following condition G:

$$G: \text{ if } (M,V) \text{ realizes some finite individual neighborhood } \mathcal{N}(x)$$

$$\text{then } (M,V) \text{ realizes } \mathcal{N}(x) \text{ infinitely many times.}$$

A structure (M,V) that is an infinite disjoint union of identically coloured blocks has always a generic predicate.

Definition 15. Let us use the alphabet of 15 letters $\forall$, $\exists$, x, $'$,), (, $\neg$, $\vee$, $\wedge$, s_0, s_1, p, $=$, V, a for writing down formulae. Different variables are built by x and $'$ like: x, x', x'', ... We denote by $|\varphi(\boldsymbol{x})|$ the length of a formula $\varphi(\boldsymbol{x})$ as word over this alphabet.

Lemma 1. *Let (M,V) be a structure consisting of a disjoint union of (not necessarily identic) blocks such that V is a generic predicate. Consider a formula $\psi(\boldsymbol{x})$ which is logically equivalent with a prenex $\exists$-formula. Let $|\psi(\boldsymbol{x})| = n$. Then there is a quantifier-free formula $\lambda(\boldsymbol{x})$ such that $M \models \forall\, \boldsymbol{x}\ \psi(\boldsymbol{x}) \leftrightarrow \lambda(\boldsymbol{x})$. Moreover, all the terms in $\boldsymbol{x}$ and a occurring in $\lambda(\boldsymbol{x})$ have length smaller than $2n$, and the formula $\lambda(\boldsymbol{x})$ depends only on the list of all isomorphism-types of individual $2n$-neighborhoods occurring in M.*

Consequently, in order to decide if a tuple $\boldsymbol{z} \in M$ satisfies this $\psi(\boldsymbol{x})$, we must know the $2n$-neighborhood of the tuple $(\boldsymbol{z}, a)$ and the list of isomorphism-types of individual $2n$-neighborhoods which are realized in M.

Proof. This is Poizat's "Lemme d'élimination" proved in [7] for one free variable. Let $\psi(\boldsymbol{z})$ be logically equivalent with $\exists\, \boldsymbol{y}\ \varphi(\boldsymbol{y}, \boldsymbol{z})$. The quantifier-free formula φ is put in disjunctive normal form. All conjunctions are shorter than n and the existential block commutes with the big disjunction. Working with equations, we write a conjunction in the form:

$$\exists\, \boldsymbol{y}\ \varphi_0(\boldsymbol{z}) \wedge \varphi_1(\boldsymbol{z}, \boldsymbol{y}) \wedge \lambda_1(y_1) \wedge \ldots \wedge \lambda_k(y_k).$$

Here, $\varphi_1(\boldsymbol{z}, \boldsymbol{y})$ is a conjunction of negated equalities of the form $t_1(x_i) \neq t_2(y_i)$ and $t_1(y_i) \neq t_2(y_j)$, and all terms appearing in the whole formula have lengths

$\leq 2n$. Because of the genericity of V we can always satisfy the inequalities, provided that the formulas $\lambda_i(y_i)$ are realizable in M. This can be decided if we know the list of isomorphism-types of individual $2n$-neighborhoods realized in M. The conjuction is then equivalent over M with:

$$\varphi_0(\boldsymbol{z}) \wedge \exists\, y_1\ \lambda_1(y_1) \wedge \ldots \wedge \exists\, y_k\ \lambda_k(y_k).$$

In the case that some $\exists\, y_l \lambda_l(y_l)$ is not consistent, or just not compatible with the list of individual $2n$-neighborhoods realized in the structure, all the conjuction disappears. In the contrary case, the whole conjunction is equivalent with the quantifier free formula $\varphi_0(\boldsymbol{z})$. □

Definition 16. The predicate V is called sparse if it satisfies the following condition:

$$\forall\, x\ \ [V(x) \ \ \rightarrow \ \ \exists\, n \in \mathbb{N}\ \ \exists \boldsymbol{\varepsilon} \in \{0,1\}^n\ \ \exists r\ \ x = s_1^n s_0 s_{\varepsilon_1} \ldots s_{\varepsilon_n}(r) \ \wedge\ p(r) = r\,].$$

The sparse predicates are very useful: they allow a small list of isomorphism types of individual neighborhoods and for all elements, the corresponding individual neighborhood has a succint description.

Lemma 2. *Suppose that the predicate V is sparse. For all $x \in M$ the following holds: if x is at distance $> 3m$ from its root, then the individual m-neighborhood of x contains at most one black point, which is of the form $s_1^n p^m(x)$ with $0 \leq n \leq 2m$.*

Consequently, there is a unit-cost algorithm such that for input $x \in M$ and $m \in \mathbb{N}$ it constructs a quantifier-free formula $\beta(x)$ which determines the individual m-neighborhood of x up to isomorphism. The algorithm works in time $O(m)$.

Proof. A remark on the first part: it is easy to see that if an m-neighborhood of x contains two black points ore more, then x is at a distance $\leq 3m$ from the root.

The algorithm starts by exploring the $3m$ ancestors of x. If one finds a root, the formula $\beta(x)$ gives x as an s_i-term of the root and the information if this root is the constant a or not. If one doesn't find the root and there is no black point in the m-neighborhood, the algorithm gives x as an s_i-term of his $3m$-th ancestor, the information that this ancestor is not a root, and a new symbol Σ meaning "white neighborhood". If one doesn't find the root and there is a unique black point in the m-neighborhood of x, instead of Σ write down the formula $V(b)$ where the black point b is given as term in x. □

Lemma 3. *If V is sparse, there is a unit-cost algorithm such that for input consisting of a tuple $\boldsymbol{x} \in M$ of length k and $m \in \mathbb{N}$ it constructs a quantifier-free formula $\beta(\boldsymbol{x})$ which determines the m-neighborhood of $\boldsymbol{x}$ up to isomorphism. The algorithm works in time $O(mk^2)$ and the length of $\beta(\boldsymbol{x})$ is $O(mk)$.*

Proof. First one writes down the conjunction of the individual formulae β as computed in Lemma 2. Then we observe that:

$$\mathcal{N}_m(x_i) \cap \mathcal{N}_m(x_j) \neq \emptyset \;\leftrightarrow$$
$$\leftrightarrow\; p^m(x_i) \in \{x_j, p(x_j), \ldots, p^{3m}(x_j)\} \vee p^m(x_j) \in \{x_i, p(x_i), \ldots, p^{3m}(x_i)\}.$$

For each pair (i, j) in this situation, write down x_i as a minimal term in x_j. If this doesn't happen, don't write anything. The new symbol Σ may play the role of conjunction of all negated equalities which are true instead. $\square$

Lemma 4. *The number k of different free variables occurring in the formula $\exists \boldsymbol{y}\ \varphi(\boldsymbol{x}, \boldsymbol{y})$ as a word of length n in the 15-letter alphabet satisfies $k(k+1) < 2n$. Consequently, the algorithm given by Lemma 3 for constructing the succint description $\beta(\boldsymbol{x})$ for the neighborhood $\mathcal{N}_{2n}(\boldsymbol{u})$ works in time $O(n^2)$. Moreover, $\beta(\boldsymbol{x})$ as a word in the 15-letter alphabet extended with Σ is shorter than $24n^2$.*

3.2 Construction and Main Result

In order to construct the predicate V we extend the 15-letter alphabet with the symbols Σ and $\rightarrow$ and we fix a coding of these symbols as binary words $\varepsilon_1 \ldots \varepsilon_5 \in \{0, 1\}^5$.

We consider all pairs of formulas $(\beta(\boldsymbol{x}), \psi(\boldsymbol{x}))$ in the language (s_0, s_1, p, a, V) such that:

- $\psi(\boldsymbol{x})$ is logically equivalent with an existential formula $\exists \boldsymbol{y}\ \varphi(\boldsymbol{x}, \boldsymbol{y})$ where $\varphi(\boldsymbol{x}, \boldsymbol{y})$ is quantifier-free. Let n be the length of $\psi(\boldsymbol{x})$ in the 15-letter alphabet.
- $\beta(\boldsymbol{x})$ is a formula produced by Lemma 3 to describe the $2n$-neighborhood $\mathcal{N}_{2n}(\boldsymbol{x})$ for some tuple $\boldsymbol{x}$ of elements in some structure (M, V) consisting of an infinite union of identical blocks, with a root interpreting a and such that V is sparse.

We consider all $\forall\exists$-sentences θ of the form:

$$\forall \boldsymbol{x}\ [\beta(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y}\ \varphi(\boldsymbol{x}, \boldsymbol{y})],$$

together with the existential sentences $\exists \boldsymbol{y}\ \varphi(\boldsymbol{y})$.

Such a sentence θ of length l is encoded by the sequence of letters $\varepsilon_1 \ldots \varepsilon_{5l}$. We define the code:

$$[\theta] := s_1^{t+5l} \circ s_0 \circ s_1^t \circ s_{\varepsilon_{5l}} \circ \ldots \circ s_{\varepsilon_1}(a).$$

Here is t is a natural number such that $t + 5l = 121n^2$. (We use 121 because $121 = 24 \times 5 + 1$.) The elements $[\theta]$ defined here form the set of all codes.

The following Lemma follows by applying Lemma 1 two times successively.

Lemma 5. *Let (M, V) be a structure consisting of an infinite union of copies of a block, so that V is generic and sparse. Consider a sentence $\theta = \forall \boldsymbol{x}\ [\beta(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y}\ \varphi(\boldsymbol{x}, \boldsymbol{y})]$ such that the existential sub-formula has length n. In order to know if θ is true in M it is enough to know the colour of terms $t(a)$ with $|\,t\,| < 2n^2$ and the list of isomorphism-types of individual $4n^2$-neighborhoods realized in M.*

We construct V inductively. The structure (M, V) will consist of an infinite union of identical blocks. We call an element "a black point" if it satisfies the predicate V as constructed up to the given point; the other elements are called white. Let M_0 be the structure that has the following set of black points: $\{ s_1^n s_0^{n+1}(r) \mid r \text{ root} \}$. All this poins are non-codes. M_0 is already a structure with a sparse generic predicate. It ensures the existence of sufficiently many types of individual neighborhoods, even before the construction starts.

We order the codes in a sequence $[\theta_s]$ according to their length and lexicographically ($s \geq 1$).

Construction step: If the structure M_{s-1} has been constructed, the structure M_s is defined in the following way: the code $[\theta_s]$ is painted in black if and only if $M_{s-1} \models \theta_s$. If this is the case, all the corresponding points in the other blocks become also black. □

Then $M = \lim\limits_{s \to \infty} M_s$.

Lemma 6. *The L-structure (M, V) constructed here has the following properties: the predicate V is generic and sparse, and for all encoded $\forall\exists$ formal L-sentences θ:*

$$(M, V) \models \quad \theta \leftrightarrow V([\theta]).$$

Proof. All structures M_s are generic and sparse, so we can apply Lemma 5 at every step. Consider some step s. The quantifier-free sentence which is equivalent with the encoded sentence depends on terms which are strictly shorter than the code to paint (and so their colour has been already decided). It depends also on the list of isomorphism-types of individual neighborhoods of a relatively small radius. This list does not change anymore, either by painting the new code, nor in the future of the construction. □

Theorem 4. *There is a deterministic unit-cost algorithm able to solve the satisfaction problem for quantifier-free formulae over (M, V) in uniform polynomial time $O(n^2)$ for formulae of length n. Consequently, the structure (M, V) satisfies* P = NP *for the unit-cost model of computation and has fast quantifier-elimination.*

Proof. Consider an input of the form $\boldsymbol{\psi u}$ with $\psi(\boldsymbol{x}) = \exists \boldsymbol{y}\ \varphi(\boldsymbol{x}, \boldsymbol{y})$ pure existential formula of length n and $\boldsymbol{u} \in M$ a tuple of the same length k as the tuple of different free variables $\boldsymbol{x}$. The formula can be encoded using the elements $0 := s_0(a)$ and $1 := s_1(a)$.

Using Lemma 3 we get a quantifier-free formula $\beta(\boldsymbol{u})$ that determines up to isomorphism the $2n$-neighborhood of the tuple $(\boldsymbol{u}, a)$. The algorithm takes time $O(n^2)$ according to Lemma 4. Now construct the following sentence θ:

$$\forall \boldsymbol{x}\ [\beta(\boldsymbol{x}) \to \exists \boldsymbol{y}\ \varphi(\boldsymbol{x}, \boldsymbol{y})].$$

Compute the code $[\theta]$ in M and check if $V([\theta])$ does hold. Recall that in (M, V) the sentence θ does hold if and only if $V([\theta])$ holds.

If θ holds, then $\exists\, \boldsymbol{y}\ \varphi(\boldsymbol{u}, \boldsymbol{y})$. If θ does not hold, then there cannot be any tuple $\boldsymbol{x}$ with $2n$-neighborhood isomorphic with the corresponding neighborhood of $\boldsymbol{u}$ that satisfies $\exists\, \boldsymbol{y}\ \varphi(\boldsymbol{x}, \boldsymbol{y})$. In particular $\exists\, \boldsymbol{y}\ \varphi(\boldsymbol{u}, \boldsymbol{y})$ doesn't hold in M. $\square$

References

1. Lenore Blum, Felipe Cucker, Michael Shub and Steven Smale: Complexity and real computation. Springer Verlag, New York, 1998.
2. Lenore Blum, Michael Shub and Steven Smale: On a theory of computation of complexity over the real numbers. American Mathematical Society Bulletin, 21, 1989.
3. Christine Gaßner: A structure of finite signature with identity relation and with P = NP. Preprints of the University Greifswald number 1, 2, 13, 14 / 2004; 1, 9, 17 / 2005; 1 / 2006 (different versions and expositions).
4. Armin Hemmerling: P = NP for some structures over the binary words. Journal of Complexity, 21, 4, 557 - 578, 2005.
5. Saul Kripke: Outline of a theory of truth. The Journal of Philosophy 72, 19, 690 - 716, 1975.
6. Bruno Poizat: Les petits cailloux. ALEAS, Lyon, 1995.
7. Bruno Poizat: Une tentative malheureuse de construire une structure éliminant rapidement les quanteurs. Lecture Notes in Computer Science 1862, 61 - 70, 2000.
8. Mihai Prunescu: Non-effective quantifier elimination. Mathematical Logic Quarterly 47, (4), 557 - 561, 2001.
9. Mihai Prunescu: Structure with fast elimination of quantifiers. The Journal of Symbolic Logic 71, (1), 321 - 328, 2006.