Tạp chí Tin học và Điều khiển học, T. 19, S. 3 (2003), 201–216

SOME CHARACTERIZATIONS OF COMPUTATION AND COMPLEXITY OVER THE REAL NUMBERS AND OTHER ALGEBRAIC STRUCTURES

TRAN THO CHAU

National University Hanoi

Abstract. In this paper, the BSS model of computation over the reals and other rings as well as a more general model of computation over arbitrary algebraic structures are introduced and discussed. Some crucial results concerning computability and omputational complexity within both frameworks are given and explained.

Tóm tắt. Trong bài báo này, chúng tôi tổng quan một số đặc trưng của hai mô hình tính toán: mô hình tính toán của Blum-Shub-Smale trên số thực (và cả trên các vành), và một mô hình tổng quát hơn về sự tính toán trên các cấu trúc đại số bất kỳ. Một số kết quả quan trọng liên quan đến khả năng tính toán và độ phức tạp tính toán theo hai mô hình nói trên được đưa ra và nghiên cứu.

1. INTRODUTION

In 1989, L. Blum, M. Shub and S. Smale [4] introduced a model for computations over the real numbers (and other rings as well) which is now usually called a BSS machine. One motivation for this comes from scientific computation. In the use of the computer, a reasonable idealization measures the cost of operations and tests independent of the size of the number. This contrasts to the usual theoretical computer science picture which takes into account the number of bits of the operands. Another motivation is to bring the theory of computation into the domain of analysis, geometry and topology. The mathematics of these subjects can then be used in the systematic analysis of algorithms. A novelty of the approach of Blum, Shub and Smale is that their model is uniform (for all input–lengths) whereas the notions explored in algebraic complexity (straight–line, programs, arithmetic circuits, decision trees) are typically non–uniform. One of the main purposes of the BSS approach was to create a uniform complexity theory dealing with problems having an analytical and topological background, and to show that certain problems remain hard even if arbitrary reals are treated as basic entities.

Many basic concepts and fundamental results of classical computability and complexity theory reappear in the BSS model: the existence of universal machines, the classes $P_{\mathbb{R}}$ and $NP_{\mathbb{R}}$ (real analogues of P and NP) and the existence of $NP_{\mathbb{R}}$–complete problems. Of course these notions appear in a different form, with a strong analytic flavour: typical examples of undecidable, recursive enumerable sets are complements of certain Julia sets, and the first problem that was shown to be $NP_{\mathbb{R}}$–complete is the question whether a given multivariate polynomial of degree four has a real root [4]. In the Boolean parts all problems in the class $NP_{\mathbb{R}}$ are decidable within single exponential time (but this is not as trivial as in the classical case), the P_W versus NP_W question is solved in Koiran's model [17] and the $P_{\mathbb{R}}$ versus $NP_{\mathbb{R}}$ question is one of the major open problems [11, 3].

Based on the computation model introduced in [12] for string functions over single sorted, total algebraic structures, A. Hemmerling [12] studies some basic features of a general theory of computability. The concept generalizes the Blum–Shub–Smale setting of computability over the reals and other rings. The concept of $\mathcal{S}$–computability of string functions over the universe of the structure $\mathcal{S}$ is defined and shown to be general enough to include classical recursion theory. Moreover, nondeterministic computations of two kinds are considered, namely by nondeterministic branching within program and by guessing of elements of the universe [13]. In this generalization is gives correspondingly general results of NP–completeness including both Cook's basic theorem on the NP–completeness of SAT and Meggido's generalization of the completeness results by Blum–Shub–Smale.

The principal technique used for demonstrating that two problems are related is that of "reducing" one to the other, by giving a constructive transformation that maps any instance of the first problem into an instance of the second one. Such a transformation provides the means for converting any algorithm that solves the second problem into the corresponding algorithm for solving the first problems.

The foundations for the theory of NP–completeness were laid in a paper of Stephen Cook, presented in 1971, entitled "The Complexity of Theorem Proving Procedures" [5]. In this brief but elegant paper Cook did several important things:

• Signifiance of "polynomial time reducibility", that is, reductions for which the required transformation can be executed by a polynomial time algorithm.

• On the class NP the decision problem can be solved in polynomial time by a nondeterministic computer.

• One particular problem in NP, called the "satisfiability" problem, has the property that every other problem in NP can be polynomially reduced to it. If the satisfiability problem can be solved with a polynomial time algorithm, then so can every problem in NP, and if any problem in NP is intractable, then the satisfiability problem also must be intractable. Thus, in a sense, the satisfiability problem is the "hardest" problem in NP.

This survey is organized as follows: in section 2 the BSS model over the real numbers is introduced together with the basic definitions, notions, and some results concerning the complexity theory over $\mathbb{R}$ and variations of the BSS model (additive, linear, weak machines). Section 3 generalizes the BSS model over the reals to the model over arbitrary structures, and it gives correspondingly general results concerning computability theory and relationship between the complexity classes.

2. THE BLUM - SHUB - SMALE MODEL

The computational model of the Blum-Shub-Smale is starting with defining as well as introducing the main related complexity theoretical concepts.

Notation: -$\mathbb{R}^\infty := \bigcup\limits_{k\in\mathbb{N}} \mathbb{R}^k$

- A point $y = (y_1, y_2, \dots) \in \mathbb{R}^\infty$ satisfies $y_k = 0$ for k sufficiently large.

2.1. Definitions

Definition 2.1. Let $Y \subset \mathbb{R}^\infty$. A BSS-machine M over $\mathbb{R}$ with *admissible* input set Y is given by a finite set I of instructions labelled by $0, 1, \dots, N$. A *configuration* of M is a quadruple $(n, i, j, x) \in I \times \mathbb{N} \times \mathbb{N} \times \mathbb{R}^\infty$. Here n denotes the currently executed instruction, i and j are used as addresses (copy–registers) and x is the actual content of the registers of M. The *initial* configuration of M's computation on input $y \in Y$ is $(1, 1, 1, length(y), y)$.

If $n = N$ and the actual configuration is (N, i, j, x), the computation stops with output x.

The instructions M can perform are of the following types:

• *Computation*:

– Data computations: $n : x_s \longleftarrow x_k \circ_n x_l$, where $\circ_n \in \{+, -, *, /\}$ or $n : x_s \longleftarrow \alpha$ for some constant $\alpha \in \mathbb{R}$. The register x_s will get the value $x_k \circ_n x_l$ or α resp. All other register–entries

remain unchanged. The next instruction will be $n+1$.
– Index computations: $i \longleftarrow i+1$ or $i \longleftarrow 1$; $j \longleftarrow j+1$ or $j \longleftarrow 1$.

• *Branch*: n : if $x_0 \geq 0$ goto $\beta(n)$ else goto $n+1$. According to the answer of the test the next instruction is determined (here $\beta(n) \in I$). All other registers are not changed.

• *Copy*: $n : x_i \longleftarrow x_j$, i. e. the content of the "read"–register is copied into the "write"-register. The next instruction is $n+1$. All other registers remain unchanged. All α appearing among the computation–instructions built up the (finite) set of machine constants of M.

Remark 2.1.

– The kind of operations allowed depends on the underlying structure. A branch $x \geq 0$? for example does only make sense in ordered set. The copy–registers and –instruction are neccessary in order to deal with arbitrary long inputs from $\mathbb{R}^\infty$. The way of changing the entries in the copy–register the ("addressing") seems to be rather restrictive apart from the fact that there is no indirect addressing. However it is general enough for our purposes, see remark 2.2.

– In the initial configuration on input y, the length is included, since it cannot be seen from the register contents if y terminates with some components equal to 0.

Now to any BSS–machine M over Y there corresponds in a natural way the function φ_M computed by M. It is a partial function from Y to $\mathbb{R}^\infty$ and is given as the result of M's computation on an input $y \in Y$.

Definition 2.2. Let $A \subset B \subset \mathbb{R}^\infty$ and M be a BSS–machine over B.

a) The *output–set* of M is the set $\varphi_M(B)$. The *halting–set* of M is the set of all inputs y for which $\varphi_M(y)$ is defined.

b) A is called *recurcively enumerable* over B iff A is the output–set recursive of a BSS–machine over B. (If $B = \mathbb{R}^\infty$, A is simply called *recursively enumerable.*)

c) A pair (B, A) is called a *decision problem.* It is said *decidable* iff there exists a BSS-machine $\tilde{M}$ with admissible input set B such that $\varphi_{\tilde{M}}$ is the characteristic function of A in B. In that case $\tilde{M}$ *decides* (B, A).

As can be seen easily (B, A) is decidable iff A and $B \setminus A$ are both halting sets over B.

Definition 2.3. **a)** For $x \in \mathbb{R}^\infty$ such that $x = (x_1, \ldots, x_k, 0, 0, \ldots)$ it is

$$size(x) := k.$$

b) Let M be a BSS–machine over $Y \subset \mathbb{R}^\infty$, $y \in Y$. The *running time* of M on y is defined by

$$T_M(y) := \begin{cases} \text{number of operations executed by } M \text{ on input } y, & \text{if } \varphi_M(y) \text{ is defined,} \\ \infty & \text{else.} \end{cases}$$

The first important differences to classical complexity theory of the above definition are that this one.

• states that any real number – independently if its magnitude – is considered as entity.

• defines the cost of any basic operation to be 1–no matter about the operands.

Definition 2.4. Let $A \subset B \subset \mathbb{R}^\infty$.

a) A *decision problem* (B, A) *belongs to* class $P_\mathbb{R}$ (*deterministic polynomial time)* iff there exists a BSS–machine M with admissible input–set B and constants $k \in \mathbb{N}$, $c \in \mathbb{R}$ such that M decides (B, A) and

$$\forall y \in B(T_M(y) \leq c \cdot \text{size}(y)^k).$$

b) (B, A) *belongs to* class $NP_{\mathbb{R}}$ (*nondeterministic polynomial time*) iff there exists a BSS–machine M with admissible input–set $B \times \mathbb{R}^{\infty}$ and constants $k \in \mathbb{N}$, $c \in \mathbb{R}$ such that the following conditions hold:

1) $\varphi_M(y, z) \in \{0, 1\}$
2) $\varphi_M(y, z) = 1 \Longrightarrow y \in A$
3) $\forall y \in A \exists z \in \mathbb{R}^{\infty} \left(\varphi_M(y, z) = 1 \text{ and } T_M(y, z) \leq c \cdot \text{size}(y)^k\right)$.

Herein, the input part z can be considered as a "guess", and this will be done in some informal descriptions of $NP_{\mathbb{R}}$ algorithms.

c) (B, A) *belongs to* class $co - NP_{\mathbb{R}}$ iff $(B, B \setminus A) \in NP_{\mathbb{R}}$.

Remark 2.2

• The class $NP_{\mathbb{R}}$ would not be changed if a more general way of addressing is used in the definition of BSS–machine.

• In a similar way as above one defines further complexity classes, for examples $EXP_{\mathbb{R}}$ and $NEXP_{\mathbb{R}}$ (here the running time is bounded to be single–exponential).

Examples

1) The computation of the greatest integer in x, for $x \geq 0$ in $\mathbb{R}$ (see Figure 1.)

Input $x \in \mathbb{R}$ as the second coordinate of a point in $\mathbb{R}^2$ with first coordinate 0

Replace (k, x) by $(k + 1, x - 1)$

Output k, the first coordinate

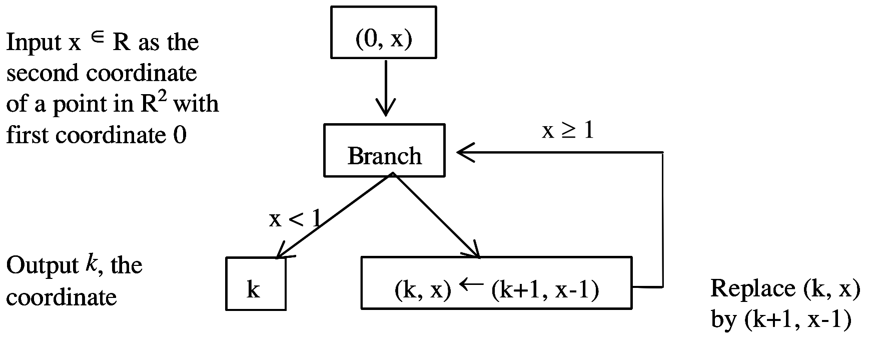
Input x ∈ R as the second coordinate of a point in R² with first coordinate 0
(0, x)
Branch
x ≥ 1
x < 1
Output k, the coordinate
k
(k, x) ← (k+1, x-1)
Replace (k, x) by (k+1, x-1)

Figure 1

2) Let $S \subset \mathbb{Z}^+$, the positive integers. We construct a machine M_S over $\mathbb{R}$ that "decides" S. That is, for each input $n \in \mathbb{Z}^+$, M_S outputs 1 (yes) if $n \in S$ and 0 (no) if $n \notin S$. M_S has a built-in constant $s \in \mathbb{R}$ defined by its binary expansion

$$s = \cdot s_1 s_2 \ldots s_n \ldots, \text{ where } s_n = \begin{cases} 1, & \text{if } n \in S \\ 0, & \text{otherwise.} \end{cases}$$

M_S with its built-in constant s, plays a role analogous to an "oracle" for a Turing machine that answers queries "Is $n \in S$?" at a cost of $n \log n$ (see Figure 2.)

Definition 2.5. Let $n \in \mathbb{N}$ and $S \subset \mathbb{R}^n$. Then the set S is *semialgebraic* if S is the set of elements in $\mathbb{R}^n$ that satisfy a finite system of polynomial equalities and inequalities over $\mathbb{R}$, or equivalently if S is finite union of subsets of the form:

$$\{y \in \mathbb{R}^n : f(y) = 0 \wedge g_1(y) > 0, \ldots, g_r(y) > 0\},$$

where $f, g_1, \dots g_r \in \mathbb{R}[x_1, x_2, \dots, x_n]$ are polynomials.

The class $P_{\mathbb{R}}$ can be considered as a theoretical formalization of the problems being efficiently solvable. The running time increases only polynomially with the problem size. The nondeterminism in b) of the definition refers to the vector z. The $NP_{\mathbb{R}}$–machine is not allowed to answer "yes" if the input does not belong to A and for each $y \in A$ there must be a "guess" z that proves this fact in polynomial time. It is evident that $P_{\mathbb{R}} \subset NP_{\mathbb{R}}$. The question $P_{\mathbb{R}} \neq NP_{\mathbb{R}}$ can be considered as the main unsolved problem in real complexity theory and the analogue to the classical P versus NP in the Turing theory. The difference between $P_{\mathbb{R}}$ and $NP_{\mathbb{R}}$ is the difference of fast proof–finding versus fast proof–checking [23].

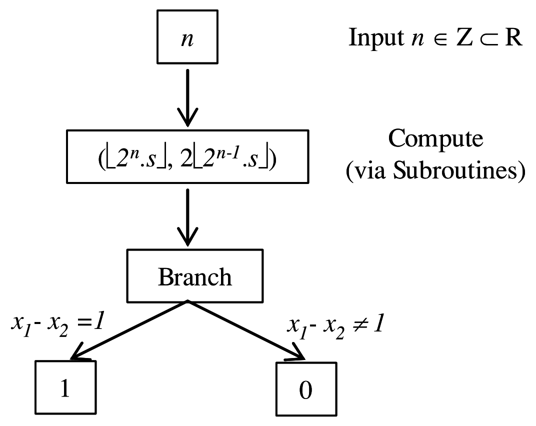
n
Input $n \in Z \subset R$
$(\lfloor 2^n.s \rfloor, 2\lfloor 2^{n-1}.s \rfloor)$
Compute (via Subroutines)
Branch
$x_1 - x_2 = 1$
$x_1 - x_2 \neq 1$
1
0

Figure 2

Definition 2.6. Let (B_1, A_1), (B_2, A_2) be decision problems.

a) (B_2, A_2) is *reducible in polynomial time* to (B_1, A_1) iff there exists a BSS–machine M over B_2 such that $\varphi_M(B_2) \subset B_1$, $\varphi_M(y) \in A_1 \iff y \in A_2$ and M works in polynomial time.

Notation: $(B_2, A_2) \leq_{\mathbb{R}} (B_1, A_1)$.

b) $(B_1, A_1) \in NP_{\mathbb{R}}$ is $NP_{\mathbb{R}}$–complete iff (B_1, A_1) is universal in $NP_{\mathbb{R}}$ w. r. t. $\leq_{\mathbb{R}}$ (i. e. any Problem in $NP_{\mathbb{R}}$ is reducible to it in polynomial time).

c) $(B_1, A_1) \in co - NP_{\mathbb{R}}$ is $co - NP_{\mathbb{R}}$–complete iff it is universal w. r. t. $\leq_{\mathbb{R}}$ in $co - NP_{\mathbb{R}}$.

Remark 2.3

Complete problems are essential for complexity classes because they represent the hardest problems in it. As in classical theory, the relation $\leq_{\mathbb{R}}$ is both transitive and reflective. This implies $P_{\mathbb{R}} = NP_{\mathbb{R}}$ iff it exists a $NP_{\mathbb{R}}$–complete problem in class $P_{\mathbb{R}}$.

Example 2.1. For $k \in \mathbb{N}$ consider the sets

$$F^k := \{f | f \text{ polynomial in } n \text{ unknowns with real coefficients } , deg(f) \leq k, n \in \mathbb{N}\}$$
$$F^k_{zero} := \{f \in F^k | f \text{ has a real zero } \}, \text{ and}$$
$$F^k_{zero,+} := \{f \in F^k | f \text{ has a real zero with all components being nonnegative } \},$$

where a polynomial is represented as an element of $\mathbb{R}^\infty$ in the following way:

The polynomial $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ of degree ≤ 4 is *powerfreely represented* in $\mathbb{R}^\infty$ as $(4, n)$ followed by a sequence of (α, a_α) where $\alpha = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$, $\alpha_i \in [0, \dots, n]$, $\alpha_i \leq \alpha_{i+1}$ and $a_\alpha \in \mathbb{R}$. The pair (α, a_α) stands for the monomial $a_\alpha x_{\alpha_1} x_{\alpha_2} x_{\alpha_3} x_{\alpha_4}$, with $x_0 = 1$ to allow for terms of degree less than 4. These (α, a_α) are supposed ordered by the lexicographic order on the α. Thus $f(x) = \sum\limits_{\alpha} a_\alpha x_{\alpha_1} x_{\alpha_2} x_{\alpha_3} x_{\alpha_4}$. Note that f can be considered as a polynomial on $\mathbb{R}^\infty$

which does not depend on x_i for $i > n$. For each degree $d \in \mathbb{Z}^+$, it is clear how to generalize this description to get the *powerfree representation* in $\mathbb{R}^\infty$ of polynomials $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ of degree $\leq d$.

The both decision problems (F^k, F^k_{zero}) and $(F^k, F^k_{zero,+})$ belong to $NP_\mathbb{R}$ for all $k \in \mathbb{N}$ by guessing a (nonnegative) zero x, pluging it into f and evaluating $f(x)$.

Remark 2.4

– The semi-algebraic subsets of $\mathbb{R}$ are finite unions of intervals-bounded or unbounded, open, closed (including single point sets), or half open [4].

– The decision problems $(\mathbb{R}, \mathbb{Q})$, $(\mathbb{R}, \mathbb{Z})$, $(\mathbb{R}, \mathbb{N})$ and $(\mathbb{Q}, \mathbb{Z})$ do not belong to the class $P_\mathbb{R}$ (see [21]).

– The problem $(\mathbb{R}, \mathbb{Q})$ ist not decidable (see [21]); $(\mathbb{R}, \mathbb{Z})$, $(\mathbb{R}, \mathbb{N})$, $(\mathbb{Q}, \mathbb{Z})$ are decidable.

2.2. Basic complexity results

We come back to Cook's fundamental theorem: the honor of being the "first" NP–complete problem goes to a decision problem from Boolean logic, which is usually referred to as the SATISFIABILITY problem (SAT for short).

Theorem 2.1. (Cook's Theorem [9]) *SATISFIABILITY is NP–complete.*

The experience can still narrow the choices down to a core of basic problems that have been useful in the past. Even though in theory any known NP–complete problem can serve just as well as any other for proving a new problem NP–complete, in practice certain problems do seem to be much better suited for this task. The following six problems are among those that have been used most frequently and these six can serve as a "basic core" of known NP–complete problems for the beginner (see [9]): Now we want return to continue the work on problems like $P_\mathbb{R}$ versus $NP_\mathbb{R}$. Obviously if problems in $NP_\mathbb{R}$ would not be decidable then it would not make sense to speak about their complexity. Moreover, it is important to know whether $NP_\mathbb{R}$–complete problems exist and how they look like. We know that proving completeness results for decision problems in principle is possible by reducing known complete problems to those in question. Nevertheless it remains the task to find a "first" complete problem. This is one of the major results in [3].

Theorem 2.2. (Blum–Shub–Smale [3])

a) *For any $k \geq 4$ the problem (F^k, F^k_{zero}) is $NP_\mathbb{R}$–complete.*

b) *All problems in class $NP_\mathbb{R}$ are decidable in single exponential time.*

Part a) is proved by an adaption of Cook's famous NP–completeness result for the 3–Satisfiability problem in Turing theory to the BSS–model. The decidability of problems in $NP_\mathbb{R}$ is much harder to show than in discrete complexity theory. The problem is closely connected with so called quantifier–elimination over real closed fields: The problem whether a $f \in F^k$ has a zero can be formulated via the first–order formula

$$\exists x_1 \ldots \exists x_n f(x_1, \ldots, x_n) = 0.$$

Remark 2.5. If $P_\mathbb{R} \neq NP_\mathbb{R}$ is assumed then $k = 4$ is a sharp bound in Theorem 2.2. This is a result by Triesch [31], who proved (F^k, F^k_{zero}) belong to $P_\mathbb{R}$ for $k = 1, 2, 3$.

Some more completeness results: • (QS, QS_{yes}), does a system of quadratic polynomials have a common zero ($NP_\mathbb{R}$–complete [3])

• Is a semi–algebraic set, given by polynomials of degree at most d, non–empty ($NP_\mathbb{R}$–complete for $d \geq 2$ [3])

• $(F^k, F^k_{zero,+})$ ($NP_{\mathbb{R}}$–complete for $k \geq 4$ [16])

• Is a semi–algebraic set, given by polynomials of degree at most d, convex ($co-NP_{\mathbb{R}}$–complete for $d \geq 2$ [5])

Especially with respect to classical complexity theory, $(F^2, F^2_{zero,+})$ is interesting. This bounds the complexity of many important combinatorial problems if considered over the reals, for examples, 3SAT, HC, Traveling Salesman are all reducible to it in polynomial time. Starting with the first master problem according to Theorem 2.2. a) one can hopefully reach many new NP–completeness problems those are used most frequently too.

2.3. Relations with weak–BSS–models and linear additive machines

Let M be a BSS–machine with real constants $x_1, \ldots, x_s$. For any input–size n, M realizes an algebraic computation tree. If any node ν is passed by M during this computation, the value computed by M up to this node is of the form $f_\nu(\alpha_1, \ldots, \alpha_s, x_1, \ldots, x_n)$ where $f_\nu \in Q(\alpha_1, \ldots, \alpha_s, x_1, \ldots, x_n)$ is a rational function with rational coefficients only. And now the *weak cost* of the according operation is fixed as maximum of $deg(f_\nu)$ and the maximum height of all coefficients of f_ν (here the height of a rational $\frac{p}{q}$ is given by $\lfloor \log(|p|+1) + \log(|q|) \rfloor$).

Definition 2.7. ([14]) The *weak BSS–model* is given as the BSS–model together with the weak cost–measure, i. e. the weak running time of a BSS–machine M on input $x \in \mathbb{R}^\infty$ is the sum of the weak costs related to all operations M performs until $\varphi_M(x)$ is computed.

Weak deterministic and non–deterministic polynomial time as well as weak polynomial time reducibility are defined in a straightforward manner (and denoted by P_W, NP_W etc.)

Definition 2.8. ([2,14])

a) Let $\mathcal{C}$ be a complexity class over the reals, the *boolean part* $BP(\mathcal{C})$ of $\mathcal{C}$ denotes

$$\{L \cap \{0,1\}^* : L \in \mathcal{C}\}.$$

b) ([2,7]). If C is a non–deterministic complexity class, then $DigC$ (*digital* C) denotes the subclass of those problems in C which membership can be established by guessing only elements from $\{0,1\}^*$ (for example $DigNP_{\mathbb{R}}, DigNP_W$).

Definition 2.9

– Any class of functions from $\mathbb{N} \longrightarrow \Sigma^*$ call these functions *advice functions.*

– Let $\mathcal{C}$ be a class of sets, and let $\mathcal{F}$ be a class of advice functions. The class $\mathcal{C}/\mathcal{F}$ is the class of all the sets B for which there exists a set $A \in \mathcal{C}$ and a function $f \in \mathcal{F}$ such that

$$B = \{x | \langle x, f(|x|)\rangle \in A\}.$$

The classes obtained in this way are known as *nonuniform* classes.

Theorem 2.3. ([14])$BP(P_W) = P/poly.$

Definition 2.10. If the set of operations in the BSS–model is reduced to addition/substraction or to linear operations (i. e. addition/substraction and scalar multiplications with a fixed finite set of reals) we get *additive* resp. *linear* BSS–machines. If only test–operations "$x = 0$? " can be performed we get BSS–machines *branching on equality.*

Notation: The kind of branching is denoted as *upper index* whereas the kind of model is indicated as *lower index*, for example:

$$P^{=}_{lin}, \quad P^{\leq}_{add}, \quad P^{\leq}_{lin}, \text{etc.}$$

Theorem 2.4. ([14]) $P_W \subset P_{\mathbb{R}} \subset NP_{\mathbb{R}} = NP_W.$

This theorem means that the P_W versus NP_W question is solved in Koiran's model, and the equation $NP_{\mathbb{R}} = NP_W$ is interesting that the full non–determinism is not stronger that the weak one. Moreover, Cucker, Shub and Smale show that the problems $(F^4, F^4_{Zero}$ and (QS, QS_{yes}) are complete also in the weak setting, i.e. w.r.t. *weak* in polynomial reductions.

Remark 2.6. The problem *Knapsack* belongs to $DigNP^{=}_{W} \setminus P^{=}_{W}$ (see [6]).

Considering the order–free linear resp. additive BSS–modell the P versus NP question can be answered.

Theorem 2.5. ([15,17,18])

a) $P^{=}_{lin} \neq NP^{=}_{lin}$ and $P^{=}_{add} \neq NP^{=}_{add}$.

b) $DigNP^{\leq}_{lin} = NP^{\leq}_{lin}$, $DigNP^{=}_{lin} = NP^{=}_{lin}$, $DigNP^{\leq}_{add} = NP^{\leq}_{add}$, and $DigNP^{=}_{add} = NP^{=}_{add}$.

3. A MODEL OF COMPUTATION OVER ALGEBRAIC STRUCTURES

We know that the classical complexity theory based on the model of Turing machine does not immediately deal with functions or decision problems over the natural numbers or the integers. More precisely, it considers the digital encodings of those functions or problems. This means that it deals with strings over finite alphabets which possibly represent numbers. The "genuine" complexity of number problems has been scarely investigated so far (see [11,19]). This ideas of the model of computation with respect to structures of finite signatures have already been outlined and used by J. Goode [7] and B. Poizat [21]. That confirms once again the importance of this approach.

Definitions

Definition 3.1. Let $\mathbb{N}_+$ the set of all positive integers. An *algebraic structure* is a quadruple $\mathcal{S} = \langle S; (c_i : i \in I_C); (R_i : i \in I_R); (F_i : i \in I_F)\rangle$, where • S is a nonempty set, called the *universe* of $\mathcal{S}$ • $(c_i : i \in I_C)$ is (possibly empty) family of *base constants*, i. e. $c_i \in \mathcal{S}$ for all $i \in I_C$ • $(R_i : i \in I_R)$ is a family of the *base relations*, thus $R_i \subseteq S^{k_i}$, with some *arity* $k_i \in \mathbb{N}_+$ for all $i \in I_R$ • $(F_i : i \in I_F)$ is a family of the *base functions*, each with some *arity* $l_i \in \mathbb{N}_+$. The triple $\sigma = \langle I_C; (k_i : i \in I_R); (l_i : i \in I_F)\rangle$ is called the signature of $\mathcal{S}$. It is said to be *finite* if all the *index sets* I_C, I_R, I_F are finite.

Examples

$\mathcal{B} = \langle \{0,1\}; 0,1; =; +, -, *, /\rangle$-	the binary field
$\mathcal{N} = \langle \mathbb{N}; 0; =; succ\rangle$-	the Peano structure of natural numbers
$\mathcal{A} = \langle \mathbb{N}; 0,1; =; +, *\rangle$-	the structure of elementary arithmetic
$\mathcal{Z} = \langle \mathbb{Z}; 0,1; \leq; +, -, *\rangle$-	the ordered ring of integer
$\mathcal{R} = \langle \mathbb{R}; 0,1; \leq; +, -, *, /\rangle$-	the ordered field of reals
$\mathcal{G} = \langle G; e; =; \cdot, ^{-1}\rangle$-	arbitrary groups
$\mathcal{V} = \langle V; 0; =; (\sigma : r \in \mathbb{R}), +\rangle$-	a linear vectorspace $(\sigma_r(x) = r \cdot x)$
$\mathcal{R}_{lin} = \langle \mathbb{R}; 0,1; =; (\mu_r : r \in \mathbb{R}), +\rangle$-	the reals as linear space$(\mu_r(x) = r \cdot x)$
$\mathcal{R}_{lin} = \langle \mathbb{R}; 0,1; \leq; (\mu_r : r \in \mathbb{R}), +\rangle$-	the ordered reals as linear space

To get a total operation of division is always assumed $s/0 = 0$.
The first six examples are structures of finite signatures, and the remaining three have infinite signatures.

• *Computability* in S means the computability of (partial) k–arity functions $\varphi : S^k \longrightarrow S$.

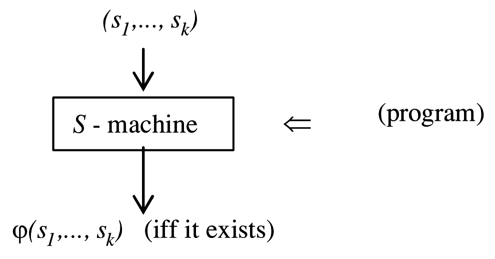

Figure 3

The Figure 3 illustrates the underlying basic idea of so–called *finite algorithmic procedures.*

The input $(s_1, s_2, \ldots, s_k)$ is given to a "machine" which works according to a certain program and the output is the value of the function iff the function defined.

• A *(deterministic) program* is a sequence $\mathcal{P} = (B_0, B_1, \ldots B_N)$ of instructions B_λ which act on finitely many of the variables (registers) $x_0, x_1, \ldots, x_n, \ldots$. Usually the instructions can be:

assignments:	$x_j := C_i$	$(i \in I_C)$
	$x_j := F_i(x_{j_1}, \ldots, x_{j_{l_i}})$	$(i \in I_F)$
	$x_j := x_{j_1}$	
branchings	*if* $R_i(x_{j_1}, \ldots, x_{j_{k_i}})$ *then goto* λ_1	$(i \in I_R)$
stops:	halt	

• *Computability over* $\mathcal{S}$ means the computability of (partial) string functions $\varphi : S^+ \longrightarrow S^+$, where $S^+ = S^* \cup \{\Lambda\}$ with the empty string Λ is not allowed to occur in the course of the computations. The basic idea in Figure 6 remains unchanged essentially, but input and output are strings now. It gives the differential kinds for the access of arbitrary many variables. Here the approach avoids such a direct reference to natural numbers, and we use finitely many pointers which act like the heads of a Turing machine on the current string of data.

For an illustration of data handling by the machine (see Figure 4).

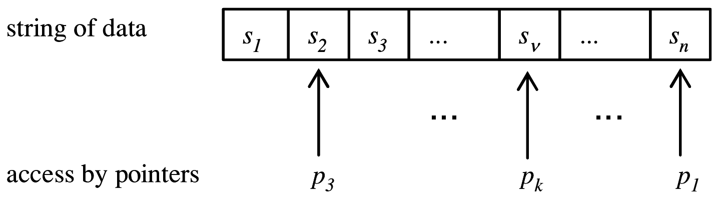

Figure 4

An $\mathcal{S}$–program will use finitely many *pointer variables* p_j $(j = 1, \ldots, k)$ which point to elements of the current string w, i. e. they have values from $\{1, 2, \ldots, \mathit{length}(w)\}$.

The three types of *atomic pointer expressions*:

$$p_j = p_{j'}; \quad (1) \quad r - \mathit{end}(p_j); \quad (2) \quad l - \mathit{end}(p_j); \quad (3)$$

for $1 \leq j, j' \leq k$, where $p_j = p_{j'}$ is true iff both pointer variables have the same current value; $r - \mathit{end}(p_j)$ (resp. $l - \mathit{end}(p_j)$) is true iff p_j points to the *right*– (resp. *left*–) end of the current string.

Data variables are of the form "$p_j \uparrow$" for $j = 1, \ldots, k$ their current values are those elements form S to which the p_j point.

Data terms are inductively defined to be either data variables or constants C_i $(i \in I_C)$, or to have the form $F_i(t_1, \ldots, t_{l_i})$ with a base function F_i $(i \in I_F)$ and data terms $t_1, \ldots, t_{l_i}$.

Atomic data expressions are either equations "$t_1 = t_2$" with data terms t_1 and t_2, or predicative expressions "$R_i(t_1, \ldots, t_{k_i})$" with $i \in I_R$ and data terms $t_1, \ldots, t_{k_i}$.

Definition 3.2. An *$\mathcal{S}$–program* is a finite sequence $\mathcal{P} = (B_0; B_1; \ldots; B_N)$, where $n \in \mathbb{N}$, and the unconditional or conditional instructions B_λ $(\lambda = 0, \ldots, N)$:

a) *Seven unconditional instructions* of the following types $(j = 1, \ldots, k)$:

- assignments: "$p_j \uparrow := t$" with a data term t;
- pointer moves: "$r - move(p_j)$" or "$l - move(p_j)$";
- append instructions: "$r - app(p_j)$" or "$l - app(p_j)$"
- delete instructions: "$del(p_j)$";
- halt instruction: "*halt*";
- jumps: "$goto(m_0, \ldots, m_n)$" with $n, m_0, \ldots, m_n \in \mathbb{N}$;
- guess instructions: "$guess(p_j \uparrow)$";

b) A *conditional instruction* of the form "*if* **Cond** *then* **Inst**"
with an unconditional instruction **Inst** and an atomic (pointer or data) expression **Cond**.

The meanings of all above instructions:

- Assignments are straightforward.
- Stop instructions finish the work of the program.
- A pointer move is performed only if the pointer would not leave the current string by that move, otherwise it doesn't cause an action.
- If pointer p_j occupies the right resp. left end of the current string, the append instruction causes an enlargement of the string (to the right resp. left) by one place which has to be filled with the former rightmost resp. leftmost element, and the pointer p_j has to take this position in the following step. If the pointer doesn't occupy the corresponding end of the current string, the append instruction has no effect.
- Delete instructions causes an action only if the current string has a length ≥ 2 and if the corresponding p_j points to the right or left end of the string. Then this elements has to be removed, and all pointers placed there take the new end elmenent as their positions in the next step.
- Jump instructions causes a jump to one of the instructions whose indices are given in the list of goal labels $(m_0, \ldots, m_n)$.
- Guess instructions replaces the value of the corresponding data variable by an arbitrary element of the universe S.
- In all cases of instructions (excepted jump and stop), after having performed them, the program control continues with the next instruction of the program.
- Finally, the instruction **Inst**, within some conditional instruction as given above, has to be performed iff the condition **Cond** holds with respect to the current values of the involved pointer and data variables, otherwise the program goes to the next instruction.

A *$\mathcal{P}$ – configuration* is a $(k+2)$–tuple $k = (\omega, \lambda, \sigma_1, \ldots, \sigma_k)$, where $\omega \in S^+$ – the current string, $\lambda \in \mathbb{N}$ – instruction counter, $\sigma_i = 1, \ldots, length(\omega)$ $(i = 1, \ldots, k)$, and the current values of the pointers $val_k(p_j) = \sigma_j$ $(j = 1, \ldots, k)$. The values of data variables are $val_k(p_j \uparrow) = \omega[\sigma_j]$

(the σ_j–the element of the string ω).

On this basis, the values of pointer expressions and of data terms, and data expressions with respect to κ can straightforwardly be defined.

To every $\mathcal{P}$–configuration $\kappa = (\omega, \lambda, \sigma_1, \ldots, \sigma_k)$ obtained from some *initial configuration* $(\omega_0, 0, 1, \ldots, 1)$ by finitely many steps of the program, an instruction B_λ of the program is assigned. Without loss of generality, let B_N be the only stop instruction of the program. Then κ is a *stop configuration* of $\mathcal{P}$ iff $\lambda = N$.

Notation:

- $\kappa \vdash_{\mathcal{P}} \kappa'$ means that the configuration κ' is obtained from κ by executing one step of program $\mathcal{P}$.
- $\vdash^*_{\mathcal{P}}$ denotes the reflexive and transitive hull of relation $\vdash_{\mathcal{P}}$, i.e. $\kappa \vdash^*_{\mathcal{P}} \kappa'$ iff there are an $m \in \mathbb{N}$ and $\mathcal{P}$–configurations $\kappa_0, \kappa_1, \ldots, \kappa_m$ such that $\kappa = \kappa_0$, $\kappa' = \kappa_m$, and $\kappa_i \vdash_{\mathcal{P}} \kappa_{i+1}$ for all $i = 0, 1, \ldots, m-1$.

Finite or infinite sequences $(\kappa_0, \kappa_1, \ldots)$ of configurations such that $\kappa_i \vdash_{\mathcal{P}} \kappa_{i+1}$ are called $\mathcal{P}$*–computations.*

Let $\rho \subseteq S^+ \times S^+$. We say that the program $\mathcal{P}$ *computes* the relation ρ iff

$$\rho = \{(\omega, \omega') : \text{ there is a stop configuration} \kappa = (\omega', N, \sigma_1, \ldots, \sigma_k) \text{ such that } (\omega, 0, 1, \ldots, 1) \vdash^*_{\mathcal{P}} \kappa'\}.$$

Let $W \subseteq S^+$. We say that the program $\mathcal{P}$ *recognizes* W iff $W = \{w :$ there is a finite $\mathcal{P}$–computation which starts with $(\omega, 0, 1, \ldots, 1)$ and terminates with some stop configuration $\}$. In other words, the recognizable sets over $\mathcal{S}$ are the domains computable relations or *halting sets* of $\mathcal{S}$–program.

A set $W \subseteq S^+$ is called *decidable* iff both W and $S^+ \setminus W$ are recognizable by $\mathcal{S}$–programs.

Definition 3.3. – An $\mathcal{S}$-program is said to be *deterministic* (D-program) if it does not contain a guess instruction, and all its jump instructions have just one goal label. D-programs compute only single-valued relations, i. e. (partial) functions.

– A program is *nondeterministic of the first kind* or *binarily nondeterministic* (N_1-program) if it does not contain a guess instruction.

– A program is *nondeterministic of the second kind* or *totally nondeterministic* (N_2-program) if it is an arbitrary program.

Remark 3.1. We shall speak of D–computability and N_i–computability ($i = 1, 2$), and use the related notation with respect to recognitions or decisions.

Example.

Over the structure $\mathcal{N}$ of natural numbers the instruction "*guess*$(p_j \uparrow)$" can equivalently be replaced by

$$p_j \uparrow := 0; L'_0 : goto(L'_1, L'_2), L'_1 : p_j \uparrow := p_j \uparrow +1; goto(L'_0); L'_2 :; .$$

Thus, N_2–program over $\mathcal{N}$ are not more powerful than N_1–programs. The analogue does not hold, however, over the ordered field $\mathcal{R}$ of real numbers. Indeed, the function $\varphi(\omega) = \sqrt{abs(w[1])}$ is not N_1–computable, but it is computed by the following N_2–program:

```
$L_0 : r - move(p_2)$
$if\ not(r - end(p_2))\ then\ goto\ (L_0);$          $\{p_2$ to the right end $\}$
$r - app(p_2); guess(p_2 \uparrow);$
$if\ p_1 \uparrow \leq 0\ then\ p_1 \uparrow := (-1) * p_1 \uparrow;$          $\{p_1 \uparrow := abs(p_1 \uparrow)\}$
```

```
$L_1 : if\, p_2 \uparrow *p_2 \uparrow \neq p_1 \uparrow\ then\, goto\, (L_1);$        { infinite cycle }
$L_2 : del(p_1);\ if\, p_1 \neq p_2\, then\, goto\, (L_2);$        { delete the input $\omega$}
halt.
```

Many authors, like BSS and Friedman–Mansfield, allow the use of arbitrary elements of the universe as constants in their programs. This seems to be quite common with respect to RAM model over the integers. Remark that any constant i can be considered as term "$1+1+\cdots+1$" (i times 1) if $i > 0$, and "$(-1)+(-1)+\cdots+(-1)$" (i times -1) if $i < 0$.

We shall strictly distinguish between $\mathcal{S}-$*programs*, where only the finitely many base constants of the structure are allowed to occur as direct operands, and the $\mathcal{S}-$*quasiprograms* which are analogously defined but allowing arbitrary elements of the universe as direct operands. Those will be denoted as *quasiconstants*.

Now we shall see that the computability of relations by quasiprograms can be characterized by the computability by means of programs.

Lemma 3.1. *A relation $\psi : S^+ \Longrightarrow S^+$ (subset of $S^+ \times S^+$) is computable by a ($D-$, N_1- or N_2-) $\mathcal{S}$–quasiprogram iff there are a relation $\varphi : S^+ \Longrightarrow S^+$ and a string $\omega_0 \in S^*$ such that $\psi = \varphi_{\langle \omega_0 \rangle}$ and φ is computable by an $\mathcal{S}$–program of the same type.*

Notation: A "Q" in the prefix denotes the concepts "Quasiprogram" corresponding the DQ–computability, N_iQ–recognizability, etc.

Lemma 3.2. *For every $x \in \{D, N_1, N_2, DQ, N_1Q, N_2Q\}$, the class of all X–recognizable sets of string is closed under (finite) union and intersection.*

Definition 3.4. An element $s \in S$ is said to be ($\mathcal{S}-$)*constructible* if the total constant functions φ_s with $\varphi_s(\omega) = s$ for all $\omega \in S^+$, is deterministically $\mathcal{S}-$computable.

A string $\omega_0 \in S^+$ is called *constructible* if it consists of constructible elements only.

Lemma 3.3. *Let s_0 be an $\mathcal{S}-$constructible element. Then, for every element $s \in S$, it holds: s is $\mathcal{S}-$constructible iff there is an $\mathcal{S}-$term $t_s(x)$ containing only one individual variable x such that $s = t_s(s_0)$.*

Proposition 3.1. *Let A be a finite set of $\mathcal{S}-$constructible elements, where $card(A) \geq 2$. Then every partial recursive function $\varphi : A^+ \longrightarrow A^+$ is determistically $\mathcal{S}-$computable.*

Definition 3.5. A structure $\mathcal{S}$ is said to be *bipotent* if it contains at least two constructible elements r_0, r_1.

Examples of bipotent structures are the structures with the number domains specifying in section 2, and the nonbipotent structures are groups $\mathcal{G}$ or vector spaces $\mathcal{V}$ in section 2.

Remark 3.2. *Over bipotent structure, one can use auxiliary tracks within string processing in a rather natural way, example: string $\tilde{\omega} = r_{i_1}s_1r_{i_2}s_2\cdots r_{i_n}s_n$ ($i_j \in \{0,1\}$) instead of the current string $\omega = s_1s_2\cdots s_n$.*

Bipotent structures also allow a rather simple *pairing of string*:

$$pair(s_1,\ldots,s_n,s'_1,\ldots,s'_m) =_{df} r_0s_1r_0s_2\ldots r_0s_nr_1s'_1r_0s'_2\ldots r_0s'_m.$$

Obviously, the set $\{pair(w_1,w_2) : w_1,w_2 \in S^+\}$ is $D-$decidable.

Theorem 3.1. ([9]) *A string relation ρ over a structure $\mathcal{S}$ is N_2–computable iff there is a $D-$computable string function φ such that*

$$\rho = \{(w, w') : \textit{ there is a string } a \in S^+ \textit{ such that } \varphi(pair(w, a)) = w'\}.$$

A string relation ρ over a structure $\mathcal{S}$ is N_1-computable iff there is a $D-$computable string function φ such that

$$\rho = \{(w, w') : \textit{ there is a string } a \in \{r_0, r_1\}^+ \textit{ such that } \varphi(pair(w, a)) = w'\}.$$

3.2. Recognizability and related concepts

– As defined in section 2, by $(D-)$*recognizable* sets of strings $W \subseteq S^+$, we understand the domain of $D-$computable string functions $\varphi : S^+ \longrightarrow S^+$, $W = dom(\varphi)$. And the synonymous denotation *halting set* is used.

– An *output set* W is the range of a $D-$computable string function φ, $W = ran(\varphi)$.

– A set $W \subseteq S^+$ is said to be $(\mathcal{S})-$*enumerable* if it is empty or there are an $(\mathcal{S})-$constructible string w_0 and a $D-$computable (partial) string function ψ such that $W = \{\psi^i(w_0) : i \in \mathbb{N}\}$, where ψ^i denotes the i-th iteration of ψ.

This representation of W means that it can be exhausted by an $\mathcal{S}-$effective counting process starting with the constructible string ω_0. ψ is called a *successor function enumerating W*.

Proposition 3.2. ([9])*A structure of finite signature is constructive iff its universe is enumerable.*

Theorem 3.2. ([9]) *For an arbitrary structure $(\mathcal{S})$, the following condition are equivalent:*
a) *The universe of $(\mathcal{S})$ is enumerable.*
b) *Every halting set over $(\mathcal{S})$ is enumerable.*
c) *Every output set over $(\mathcal{S})$ is enumerable.*

Definition 3.6. Let $W \subseteq S^+$. Its *projection* is defined by

$$\Pi(W) =_{df} \{w_1 : \text{ there is a } w_2 \in S^+ \text{ such that } pair(w_1, w_2) \in W\}.$$

Theorem 3.3. ([9]) *Over arbitrary structures, the following conditions are equivalent:*
a) *The classes of all halting sets and of all output sets respectively, coincide.*
b) *The class of all halting sets is closed under projection.*
c) *The class of all halting sets is closed under images of $D-$computable functions.*

Remark 3.3. There exists the structure owning an output set that is not a halting set, for example:

- Let $M = \langle \mathbb{N}; 0, 1; =; * \rangle$, where "*" denotes the multiplication; 0 and 1 are the only constructible elements of the universe of M. Hence, $\mathbb{N}$ is a halting set which is not $M-$enumerable. The set of square numbers $\{k^2 : k \in \mathbb{N}\}$ is an output set, but not a halting set. Indeed, for inputs of length 1: $w = x \in \mathbb{N}$, by the possible promises of conditional expressions with the only variable x, the elements from $\mathbb{N} \setminus \{0, 1\}$ cannot be separated each from the other.

- Over the (unordered) field of real numbers, $\mathcal{R} = \langle \mathbb{R}; 0, 1; =; +, -, *, / \rangle$, the set $\mathbb{R}_+ = \{r : r > 0\} = \{r^2 : r \neq 0\}$ is an output set, but not a halting set.

3.3. Universal programs and some results

Here we suppose that the considered structures $(\mathcal{S})$ have finite signature.

Using two constructible elements r_0 and r_1, every $(\mathcal{S})-$quasiprogram $(\mathcal{P})$ can be encoded in a straightforward manner by a string denoted by $code(\mathcal{P})$. The keywords of the programming language, the technical symbols, base constants, base relations and base functions be encoded by strings from $\{r_0, r_1\}^+$, and indices of pointer variables and the goal labels be binarily encoded over $\{r_0, r_1\}$. The quasi–constants of quasiprograms are encoded by themselves.

It is convenient to use only the track of even–numbered places in strings for these encodings, where as the odd–numbered places are filled by r_0 or r_1 such that the starting places of codes of the syntactic units can uniquely be identified. More precisely, instead of the direct encoding "$s_1 s_2 \dots s_n$" of some syntactic unit u, we use the padded string $code(n) = r_1 s_1 r_0 s_2 \dots r_0 s_n$. Then for quasiprograms $\mathcal{P} = u_1 u_2 \dots u_m$, where u_i are the syntactic units $(i = 1, \dots, m)$, let $code(\mathcal{P}) = code(u_1) \cdot code(u_2) \cdots code(u_m)$. Now, the parts of $code(\mathcal{P})$ which represent the codes of the syntactic units can be identified by a deterministic $\mathcal{S}-$program.

Theorem 3.4. ([9]) There is a $D-$program $\mathcal{U}$ such that, for all $D-$quasiprograms $\mathcal{P}$ and all strings $w \in S^+$:

$$\varphi_{\mathcal{U}}(pair(code(\mathcal{P}), w)) \cong \varphi_{\mathcal{P}}(w).$$

For $i = 1, 2$ there is an N_i-program U_i such that, for all N_i-quasiprograms $\mathcal{P}$:

$$\rho_{\mathcal{P}} = \{(w, w') | (pair(code(\mathcal{P}), w), w') \in \rho_{\mathcal{U}_i}\}.$$

This is proved by applying standard techniques of simulation and programming, as they are well-known from classical computation theory based on the concept of Turing machine. Notice that the concept of $\mathcal{S}-$program corresponds to the notion of multihead Turing machine. Thus, the simulated program $\mathcal{P}$ may use arbitrarily many pointer variables, whereas the universal program $\mathcal{U}$ and $\mathcal{U}_i$ respectively, are equipped with some fixed number of pointers only. Thus, in each step of the simulation, the positions of the $\mathcal{P}-$pointers p_j must be marked by the simulating program $\mathcal{U}_{[i]}$ by means of encodings of p_j at the corresponding places of the (encoding of the) current $\mathcal{P}-$configuration. To compute the values of base functions or base relations, the universal programs can use (finitely many) suitable subroutines. The simulation of nondeterministic steps can analogously be performed by subroutines.

The encoding of pairs of strings is used to define inductively the encoding of $\kappa-$tuples of strings, for $k \in W^+$. Let

$$tuple_1(w) = w, \quad tuple_2(w, w') = pair(w, w'),$$

$$tuple_{k+1}(w_o, w_1, \dots, w_k) = pair(w_0, tuple_k(w_1, \dots, w_k)) \text{ for } k > 1.$$

We simply write $[w_1, \dots, w_k]$ instead of $tuple_k(w_1, \dots, w_k)$ for $k > 1$.

Definition 3.7. A $k-$ary partial function $\varphi : (S^+)^k \longrightarrow S^+$ is said to be (deterministically) *computable* over $\mathcal{S}$ iff the unary string function $\overline{\varphi}$ is deterministically $\mathcal{S}-$computable, where:

$$\overline{\varphi}([w_1, \dots, w_k]) \cong \varphi(w_1, \dots, w_k) \text{ for } w_1, \dots, w_k \in S^+,$$

$$\overline{\varphi}(w) \text{ is undefined for } w \notin \{[w_1, \dots, w_k] : w_1, \dots, w_k \in S^+\}.$$

Notation. $\varphi_w(w')$ instead of $\varphi_{\mathcal{U}}(pair(w, w'))$ with respect to some fixed universal program $\mathcal{U}$ according to Theorem 5.1. Without loss of generality, suppose that $\varphi_{\mathcal{U}}(pair(w, w'))$ is undefined if $w \notin \{r_0, r_1\}^+$.

Proposition 3.3. (s-m-n Theorem [9]) *To every $m, n \in \mathbb{N}_+$, there is a deterministically $\mathcal{S}-$computable $(m+1)-$ary total function $\sigma_n^m : (\{r_0, r_1\}^+)^{m+1} \longrightarrow \{r_0, r_1\}^+$ such that for all $w_0, w_1, \dots\ w_m \in \{r_0, r_1\}^+$ and all $w_{m+1}, \dots, w_{m+n} \in S^+$:*

$$\varphi_{w_0}([w_1, \dots, w_m, w_{m+1}, \dots w_{m+n}]) \cong \varphi_{\sigma_n^m(w_0, w_1, \dots, w_m)}([w_{m+1}, \dots, w_{m+n}]).$$

Remark 3.4. The functions σ_n^m are recursive word functions on the alphabet $\{r_0, r_1\}$.

Proposition 3.4. (Recursion Theorem [9]) *Let $n \in \mathbb{N}_+$, and $\varphi : (S^+)^{n+1} \longrightarrow S^+$ be a deterministically $\mathcal{S}-$computable function. There is a string $w_0 \in \{r_0, r_1\}^+$ such that for all $w_1, \dots w_m \in S^+$:*

$$\varphi(w_0, w_1, \dots, w_n) \cong \varphi_{w_0}([w_1, \dots, w_n]).$$

Using this theorem, we have, for example, the following results:

Proposition 3.5. *There exists a deterministically $\mathcal{S}-$computable 2–ary total function φ :*

$(\{r_0, r_1\}^+)^2 \longrightarrow \{r_0, r_1\}^+$ such that for all$w, w' \in \{r_0, r_1\}^+$ and all$w_1, \dots w_m \in S^+$:

$$\varphi_w([w_1, \dots, w_m]) \cdot \varphi_{w'}([w_1, \dots, w_m]) \cong \varphi_{\varphi(w,w')}([w_1, \dots, w_m]).$$

Proposition 3.6. *The function*

$$g(pair(code(\mathcal{P}), [w_1, \dots w_n]) =_{df} \begin{cases} w', & \text{if } \varphi_{code(\mathcal{P})}([w_1, \dots w_n]) \text{ exists and is equal} \\ & \text{to} w' \in S^+ \\ \text{undefined}, & \text{if } \varphi_{code(\mathcal{P})}([w_1, \dots w_n]) \text{ does not exist.} \end{cases}$$

for all $D-$quasiprograms $\mathcal{P}$ and all $w_1, \dots w_m \in S^+$, is $\mathcal{S}-$computable.

Proposition 3.7. (Fixed–Point Theorem [9]) *Let $n \in \mathbb{N}_+$, $\varphi : S^+ \longrightarrow S^+$ be a unary deterministically $\mathcal{S}-$computable function. There is a string $w_0 \in \{r_0, r_1\}^+$ such that for all $w_1, \dots, w_n \in S^+$:*

$$\varphi_{w_0}([w_1, \dots, w_n]) \cong \varphi_{\varphi(w_0)}([w_1, \dots, w_n]).$$

Proposition 3.8. (Rice's Theorem [9]) *Let $\mathcal{F}$ be a set of deterministically $\mathcal{S}-$computable unary partial functions which does not contain all such functions but does contain the empty function $\emptyset$. Then the index set*

$$\mathcal{I}(\mathcal{F}) =_{df} \{w : w \in \{r_0, r_1\}^+ \text{ and } \varphi_w \in \mathcal{F}\}$$

is not deterministically $\mathcal{S}-$recognizable.

REFERENCES

[1] S. Ben-David, K. Meer, C. Michaux, A note on non-complete problems in N_R, *MSRI Preprint No. 1999–008.*

[2] L. Blum, F. Cucker, M. Schub, S. Smale, *Complexity and real computation,* Springer-Verlag, New York, 1998.

[3] L. Blum, M. Schub, S. Smale, On a theory of computation and complexity over the real numbers: NP–completeness, recursive functions and universal machines, *Bull. Amer. Math. Soc.* (21) (1989) 1–46.

[4] S. A. Cook, The complexity of theorem proving–procedures, *Proc. 3rd Ann. ACM Symp. on Theory of Computing, Association for Computing Machinery,* New York (1971) 151–158.

[5] F. Cucker, F. Rossello, On the complexity of some problems for the Blum–Shub–Smale model, *Proc. LATIN '92, Lecture Notes in Computer Science* (583) (1992) 117–129.

[6] F. Cucker, M. Shub, S. Smale, Separation of complexity class in Koiran's weak model, *Theoretical Computer Science* (133) (1994) 3–14.

[7] J. B. Goode, Accessible telephone directories, *J. Symb. Logic* (59) (1994) 91–105.

[8] E. Grädel, K. Meer, Descriptive complexity theory over the real numbers, In 27th Annual *ACM Symp.* on the Theory of Computing (1995) 315–324.

[9] A. Hemmerling, Computability of String functions over algebraic structures, *Math. Logic Quarterly* (44) (1998) 1–44.

[10] Hemmerling, A., *Computability and complexity over strutures of finite type*, E.-M.-Arndt-University Greifswald, Preprint 2 (1995).

[11] Hemmerling, A., On genuine complexity and kinds of nondeterminism, *J. Inform. Process. Cybernet, EIK* **30** (2) (1994) 77–96.

[12] Hemmerling, A., On P versus NP for parameter–free programs over algebraic structures, *Mathematical Logic Quarterly* (47) (2001) 67–92.

[13] Hemmerling, A., On the time complexity of partial real functions, *J. of complexity* (16) (2000) 363–376.

[14] P. Koiran, A weak version of the Blum–Shub–Smale model, *FOCS '93* (1993) 486–495 and Neuro COLT TR Series NC–TR–94–5 (1994).

[15] P. Koiran, Computing over the reals with addition and order, *Theoretical Computer Science* (133) (1994) 35–47.

[16] K. Meer, Computation over $\mathbb{Z}$ and $\mathbb{R}$: a comparison, *Journal of Complexity* (6) (1990) 256–263.

[17] K. Meer, *Komplexitätsbetrachtungen für reelle Maschinenmodelle*, PhD. Dissertation, Verlag Shaker, Aachen, 1993.

[18] K. Meer, C. Michaux, A survey on real structural complexity theory, *Bull. of the Belgian Math. Society,* (1996).

[19] N. Megiddo, Towards a genuinely polynomial algorithm for linear programming, *SIAM J. Comp.* (12) (1983) 347–353.

[20] C. Michaux, P $\neq$ NP over the nonstandard real implies P $\neq$ NP over $\mathbb{R}$, *Theoretical computer Science* (133) (1994) 95–104.

[21] B. Poizat, *Les petit cailloux*, Aleas. Lyon, 1995.

[22] M. Prunescu, P $\neq$ NP for the reals with various analytic functions, *J. of complexity* (17) (2001) 17–26.

[23] E. Triesch, A note on a theorem of Blum, Shub and Smale, *Journal of Complexity* (6) (1990) 166–169.

Received October 6, 2002