D. Kalociński
M. Wrocławski

Generalization of Shapiro's theorem to higher arities and non-injective notations

Abstract. We work under the framework of Stewart Shapiro according to which computations are performed directly on numerals (syntactic objects) rather than numbers. The interpretation is assigned to numerals via a notation. Shapiro showed that a total unary function (unary relation) is computable in every injective notation if and only if it is an almost constant or almost identity function (finite or co-finite set). We generalize this result by providing a logical characterization of the class of relations and partial functions of arbitrary finite arities computable in every injective notation. A parallel generalization is given for the notion of learnability in place of computability. Finally, we characterize the class of relations and total functions of arbitrary finite arities which are computable in every notation (be it injective or not).

Keywords: computability, learnability, notations, natural numbers, definability

1. Introduction and overview

Philosophical analyses of the concept of computation often involve, more or less explicity, the syntactical layer of numerals and the semantical layer of abstract natural numbers (see, e.g., [1, 2, 10, 4, 11, 9]). This distinction is justified by the intuition, already present in foundational works on computability [16, **?**], that algorithms are performed directly on strings of symbols, while the notion of computing on numbers is established indirectly by assuming an appropriate mapping from strings to numbers. Stewart Shapiro provided a mathematically precise formalization of this idea in the concept of a notation which he considered to be an injective function from a recursive set of numerals to natural numbers [13]. Each notation determines the class of functions and relations computable in it. Shapiro showed that, in general, computability of a number-theoretic function or relation is not an invariant property across all notations. Classical computability theory may be viewed as a theory of partial functions and relations computable in an acceptable notation. According to Shapiro, a notation is acceptable if the successor function is computable in it.

Presented by **Name of Editor**; *Received* December 1, 2005

While searching for the characterization of the class of acceptable notations, Shapiro proved that the only total unary functions (and unary relations) computable in every notation are almost constant and almost identity functions (finite and co-finite sets). This result might be of interest in itself, independetly from the analyses related to notations' acceptability. It leads to the following general question: what is the class of partial functions and relations computable in every notation? This question asks for a characterization of a very strong form of computability which may be called notationally-invariant computability. Shapiro's theorems provide a partial solution of this problem. In this article we present two groups of results all of which extend Shapiro's results to functions and relations of arbitrary arities. Specifically, we investigate the following two problems.

PROBLEM 1. *What is the class of k-ary partial functions and relations computable in every injective notation, for $k = 1, 2, \ldots$?*

This is the original Shapiro's problem, extended to arbitrary finite arities and partial functions. Some initial observations regarding it can be found in [7]. The problem is tackled in Section 3 where we give a logical characterization of the class in question. First, we obtain a solution for total functions and relations. It involves some model theory and an involved computability-theoretic construction. Next, a parallel characterization is extracted from the previous one for the notion of learnability instead of computability. Interestingly, notationally-invariant computability turns out to coincide with notationally-invariant learnability. The latter solution is used to obtain the full answer by covering k-ary partial functions as well.

PROBLEM 2. *What is the class of k-ary total functions and relations computable in every notation, for $k = 1, 2, \ldots$?*

This problem generalizes Problem 1 by making space for noninjective notations. Under a noninjective notation some numbers have many names. It turns out that this minor change is not irrelevant (for more on computability in notations in this less restricted sense, see [17, 18, 19]). In Section **??**, using a long inductive argument, we demonstrate that the class in question consists of constant functions and projections.

We wrap up our work in Section 5 which touches upon some philosophical aspects of present results and open questions.

In addition to the original article of Shapiro, additional results regarding notations were proven in [**?**], [**?**], [**?**]. Somewhat similar questions were considered in [5]. The issue was also subject to philosophical discussions, such as [10], [11], [4], [9]. More on computability in notations for natural numbers may be found in [**?**, **?**, **?**, 7].

2. Background

Familiarity with basic concepts from model theory and recursive function theory is assumed. We shortly review notions that are necessary to follow the proofs. For more information, we refer the reader to textbooks [3, 12, 15].

The set of natural numbers is denoted by $\mathbb{N}$. Σ stands for a finite alphabet, and Σ^* for the set of all words over Σ (i.e., finite sequences of elements of Σ). Lowercase Greek letters $\alpha, \beta, \ldots$, possibly with subscripts, refer to words. A characteristic function of $R \subseteq E^k$ is denoted by χ_R. We shall sometimes confuse a relation R with its characteristic function χ_R which makes writing $R(x_1, \ldots, x_k) = 1$ or $R(x_1, \ldots, x_k) = 0$ meaningful. Finite lists $v_1, v_2, \ldots, v_k$ of elements of a given set are sometimes abbreviated using vector notation $\vec{v}$.

Let $\Gamma_0^{(k)}, \Gamma_1^{(k)}, \ldots$ be a canonical enumeration of all partial recursive Turing functionals from $\mathcal{P}(\mathbb{N}) \times \mathbb{N}^k$ to $\mathbb{N}$, $k = 1, 2, \ldots$. For technical details regarding this approach, see, e.g., [12, §9.2]. We write $\Gamma_n^{(k)X}(x_1, \ldots, x_k)$ instead $\Gamma_n^{(k)}(X, x_1, \ldots, x_k)$ or simply $\Gamma_n^X(x_1, \ldots, x_k)$, which clearly assumes k as the arity. $\Gamma_n^X(x_1, \ldots, x_k)$ may be thought of as the value returned by the computation of the program P_n on input $x_1, \ldots, x_k$ relative to the oracle X. The list $\Gamma_0^{(k)\emptyset}, \Gamma_1^{(k)\emptyset}, \ldots$ provides an enumeration of all k-ary partial recursive functions which we denote by $\Phi_0^{(k)}, \Phi_1^{(k)}, \ldots$. A k-ary relation $A \subseteq \mathbb{N}^k$ is recursive (decidable, computable) if there exists e such that $\Phi_e^{(k)} = \chi_R$. A set $A \subseteq \mathbb{N}$ is computably enumerable (c.e.) if A is the domain of some partial recursive $\Phi_e^{(1)}$. Equivalently, $A \subseteq \mathbb{N}$ is c.e. if $A = \emptyset$ or there exists a (total) unary recursive function Φ such that $A = \{\Phi(n) : n \in \mathbb{N}\}$. Such a function Φ exists for each nonempty c.e. set and is usually referred to as a recursive (or computable) enumeration of A (since one can effectively enumerate values $\Phi(0), \Phi(1), \ldots$ which comprise the whole A). Existence of c.e. uncomputable sets is a folklore. Note that each infinite c.e. set A can be effectively enumerated without repetitions, i.e. there exists an injective recursive enumeration of A. We say a partial function $f : \mathbb{N}^k \to \mathbb{N}$ is computable (recursive) in X, in symbols $f \leq X$, if $f = \Gamma_e^X$, for some e. We say A is computable in B, in symbols $A \leq B$, if $A = \Gamma_e^B$, for some e. We say A is Turing equivalent to B, in symbols $A \equiv B$, if $A \leq B$ and $B \leq A$. The set $\{B : A \equiv B\}$ is called the (Turing) degree of A. The degree of the halting problem $K = \{e : \Phi_e(e) \downarrow\}$ is denoted by $0'$.

Definition 2.1 ([6, 8]). A function $f : \mathbb{N}^k \to \mathbb{N}$ is learnable if there exists a uniformly computable family of computable functions $\{f_t\}_{t\in\mathbb{N}}$ such that for every $\vec{n} = n_1, \ldots, n_k$:

$$f(\vec{n}) = \lim_{t\to\infty} f_t(\vec{n}). \tag{1}$$

A relation $R \subseteq \mathbb{N}^k$ is learnable if χ_R is learnable.

Intuitively, a function is learnable if, for each given input, one can successively make effective guesses regarding its values in a way that the guesses finally settle down on the correct value.

The following classical result provides an important link between learnability and Turing reducibility.

Theorem 2.2 (Limit Lemma [14]). *$f : \mathbb{N}^k \to \mathbb{N}$ is learnable if and only if $f \leq 0'$. Similarly, a relation $A \subseteq \mathbb{N}^k$ is learnable if and only if $A \leq 0'$.*

By means of coding, the above notions (of a partial recursive function, decidability, learnability, computable enumerability, recursive enumeration) can be extended to other domains, in particular to any recursive $S \subseteq \Sigma^*$.

Consider a first-order language with variables $V = \{x_1, x_2, \ldots\}$, individual constants $C = \{c_1, c_2, \ldots\}$ and no functional or relational symbols, except the logical predicate $=$. The set of formulae is defined in a standard way. If we write $\varphi(z_1, z_2, \ldots, z_k)$, all the free variables of φ occur among $z_1, z_2, \ldots, z_k$. A model for our language is a pair $\mathcal{M} = (M, \{c_i^M : i \in \mathbb{N}\})$, where M is a non-empty set and $c_i^M \in M$ is the distinguished element named by c_i, for $i \in \mathbb{N}$. Any function $a : V \to M$ is called an assignment. The satisfaction relation $\models$ between a model $\mathcal{M}$, formula φ and an assignment a, is defined in a standard way. We write $\mathcal{M} \models \varphi[a]$ to say that φ is satisfied in $\mathcal{M}$ under the assignment a. Given a formula $\varphi(z_1, \ldots, z_k)$, it is customary to write $\mathcal{M} \models \varphi[a_1, \ldots, a_k]$ where a_i is understood as the element assigned to z_i. Let $\mathcal{M} = (M, \{c_i^M : i \in \mathbb{N}\})$, $\mathcal{M}' = (M', \{c_i^{M'} : i \in \mathbb{N}\})$ be models for our language. $\mathcal{M}$ and $\mathcal{M}'$ are said to be isomorphic if there exists a bijection $h : M \to M'$ such that for every constant $c, c' \in C$, $h(c^M) = c^{M'}$. If $h : M \to M'$ is an isomorphism between $\mathcal{M}$ and $\mathcal{M}'$ then for every formula $\varphi(x_1, \ldots, x_k)$ and for every $\vec{a} = a_1, \ldots, a_k \in M$, $\mathcal{M} \models \varphi[\vec{a}]$ iff $\mathcal{M}' \models \varphi[h(\vec{a})]$.

2.1. Notations for natural numbers

Definition 2.3 ([19]). Let Σ be a finite alphabet. (S, σ) is a notation for $\mathbb{N}$ if $S \subseteq \Sigma^*$ is computable and $\sigma : S \to \mathbb{N}$ is onto. If σ is one-one, then the notation (S, σ) is called injective.

Our approach is less restrictive than that of Shapiro [13] who used the term *notation* to mean *injective notation* in the sense of Definition 2.3. More about this less restricted notion may be found in [17, 18, 19].

Elements of S are referred to as numerals. We abbreviate *notation for* $\mathbb{N}$ to *notation* since we do not consider notations for other domains. The standard notation and standard numerals is understood as the usual decimal notation and its numerals, respectively. When we refer to numerals rather than numbers, we put bars over them: $\overline{n}$ is the standard decimal numeral for the number n. However, in a non-standard notation it may represent a different number, or it may not be a valid numeral at all.

Definition 2.4. Let (S, σ) be a notation and let $f : \mathbb{N}^n \to \mathbb{N}$. Let f^σ be the class of all functions $F : S^n \to S$ such that for any $\alpha_1, ..., \alpha_n, \beta \in S$ the following condition is satisfied:

$$F(\alpha_1, ..., \alpha_n) = \beta \implies f(\sigma(\alpha_1), ..., \sigma(\alpha_n)) = \sigma(\beta).$$

Each function from the class f^σ is called a notation for f in (S, σ). If any of these functions is computable, we say that f^σ is computable or that f is computable in (S, σ).

Obviously, if a notation is ambiguous, then there are multiple functions in f^σ. It is possible that some of them are computable, and some are not.

There is going to be a certain ambiguity when we talk of computing f^σ. Unless explicitly stated otherwise, it shall be synonymous with computing any function from the class f^σ. However, sometimes, when a concrete function from this class has already been specified, it can refer to computing this specific function.

Definition 2.5. Let (S, σ) be a notation and let $R \subseteq \mathbb{N}^n$. Let $R^\sigma \subseteq S^n$ be defined in the following way:

$$(\alpha_1, ..., \alpha_n) \in R^\sigma \iff (\sigma(\alpha_1), ..., \sigma(\alpha_n)) \in R,$$

for all $\alpha_1, ..., \alpha_n \in S$. We say that R is computable in (S, σ) if R^σ is computable.

The above definitions might be summarized as follows. Computability of a number-theoretic function (or relation) in a notation is understood as the existence of a program which acts on numerals and outputs numerals (or truth values) such that the underlying referents agree with the function (relation) being computed.

DEFINITION 2.6 ([7]). Let (S, σ) be an injective notation. A function $f : \mathbb{N}^k \to \mathbb{N}$ is learnable in (S, σ) if f^σ is learnable. Similarly, a relation $R \subseteq \mathbb{N}^k$ is learnable in (S, σ) if R^σ is learnable.

The following proposition shows that a direct analogue of uniform computability could be used to define learnability in an injective notation.

PROPOSITION 2.7. *Let $f : \mathbb{N}^k \to \mathbb{N}$ be a function and let (S, σ) be an injective notation. f is learnable in (S, σ) iff there exists a function $g : \mathbb{N}^{k+1} \to \mathbb{N}$, computable in (S, σ), such that, for every $\vec{n} \in \mathbb{N}^k$: $f(\vec{n}) = \lim_{t\to\infty} g(t, \vec{n})$.*

PROOF. Let $\gamma_1, \gamma_2, \ldots$ be any recursive enumeration of S without repetitions. Let $g : \mathbb{N}^{k+1} \to \mathbb{N}$ be a function and $\{g_t\}_{t\in\mathbb{N}}$ a family of functions $g_t : S^k \to S$ satisfying, for every $t \in \mathbb{N}, \alpha_1, \ldots, \alpha_k, \gamma \in S$:

$$g(\sigma(\gamma_t), \sigma(\alpha_1), \ldots, \sigma(\alpha_k)) = \sigma(\gamma) \iff g_t(\alpha_1, \ldots, \alpha_k) = \gamma. \tag{2}$$

Observe that g is computable in (S, σ) iff g_t are uniformly computable. The final observation is that for every $\vec{n} \in \mathbb{N}^k, \vec{\alpha} \in S^k$ such that $\sigma(\vec{\alpha}) = \vec{n}$:

$$\lim_{t\to\infty} g(t, \vec{n}) \simeq \lim_{t\to\infty} g_t(\vec{\alpha}), \tag{3}$$

where $\simeq$ holds iff either both sides are undefined or both are defined and equal. ■

3. Computability across injective notations

In this section we solve

PROBLEM 1. *What is the class of k-ary partial functions and relations computable in every injective notation, for $k = 1, 2, \ldots$?*

Let us recall Shapiro's theorems which provide a partial solution with regard to unary total functions and relations.

THEOREM 3.1 (Shapiro [13]). *The only unary functions computable in every injective notation are almost constant and almost identity functions.*

THEOREM 3.2 (Shapiro [13]). *The only subsets of $\mathbb{N}$ whose characteristic functions are computable in every injective notation are finite and cofinite sets.*

It might be tempting to conjecture that similar statements are true for functions and relations of higher arities. However, this is not the case, as demonstrated by Proposition 3.4.

Definition 3.3. We say that a function $f : \mathbb{N}^k \to \mathbb{N}$ is almost constant if there exists $y \in \mathbb{N}$ such that $f(x_1, \dots, x_k) = y$ holds for all but finitely many tuples $x_1, \dots, x_k$. Similarly, a function $f : \mathbb{N}^k \to \mathbb{N}$ is said to be almost projection if there exists i such that $1 \leq i \leq k$ and $f(x_1, \dots, x_k) = x_i$ holds for all but finitely many tuples $x_1, \dots, x_k$.

Proposition 3.4. *There exists a function which is computable in every injective notation but is neither almost constant nor almost projection.*

Proof. Consider the function $f : \mathbb{N}^2 \to \mathbb{N}$ defined by:

$$f(x_1, x_2) = \begin{cases} 0 & \text{if } x_1 = 0 \vee x_2 = 0, \\ 2 & \text{if } x_1 \neq 0 \wedge x_2 \neq 0. \end{cases} \tag{4}$$

It is easy to show that f is computable in any injective notation (S, σ). To prove that f is not almost constant, one has to demonstrate that for every y and every finite $F \subseteq \mathbb{N}^2$ there exists $(x_1, x_2) \notin F$ such that $f(x_1, x_2) \neq y$. Similarly, to prove that f is not almost a projection, one has to demonstrate that for every $i \in \{1, 2\}$ and every finite $F \subset \mathbb{N}^2$ there exists $(x_1, x_2) \notin F$ such that $f(x_1, x_2) \neq x_i$. The continuation is left to the reader. ■

Generalizing Shapiro's theorem to arbitrary functions and relations requires a slightly different perspective which is suggested by the form of equation (4). Namely, the right approach is to consider the class of functions and relations on natural numbers which are definable in a certain weak sense.

Consider the first-order language with no functional or relational symbols, except the logical predicate $=$, with $x_1, x_2, \dots$ as variables and $\underline{0}, \underline{1}, \dots$ as constants. Let $\mathcal{N} = (\mathbb{N}, c^{\mathbb{N}})$ be a model, where $c^{\mathbb{N}}$ is the interpretation function for constants defined by $c^{\mathbb{N}}(\underline{n}) = n$, for every $n \in \mathbb{N}$. We say that a formula $\phi(x_1, \dots, x_k, y)$ defines a function $f : \mathbb{N}^k \to \mathbb{N}$ in $\mathcal{N}$ if for all $n_1, \dots, n_k, m \in \mathbb{N}$: $\mathcal{N} \models \phi(x_1, \dots, n_k, m)[n_1, \dots, n_k, m] \iff f(n_1, \dots, n_k) = m$.

Definition 3.5. We say that $f : \mathbb{N}^k \to \mathbb{N}$ is qf-definable in $\mathcal{N}$ if there exists a quantifier-free formula ϕ in the language $\mathcal{L}$ defining f in $\mathcal{N}$.

Below we define an equivalence relation over tuples of an arbitrary set. This notion, although slightly technical, will allow us to encapsulate certain cumbersome details in Lemmas 3.7 and 3.9 which will be used in the proof of Theorem **??**.

Definition 3.6. Let E be any set and let $F \subset E$. Let $\vec{a}, \vec{b} \in E^k$. We say that $\vec{a}$ and $\vec{b}$ are of the same F-type (in symbols: $\vec{a} \sim_F \vec{b}$) if:

1. For every $i \in \{1, 2, \dots, k\}$: $a_i \in F \iff b_i \in F$.
2. For every $i \in \{1, 2, \dots, k\}$: $a_i \in F \implies a_i = b_i$.
3. For every $i, j \in \{1, 2, \dots, k\}$: if $a_i \notin F$ and $a_j \notin F$ then $[a_i = a_j \iff b_i = b_j]$.

For $E = \mathbb{N}$ and $F = \{0, 1, \dots, c-1\}$ we shall write $\sim_c$ instead of $\sim_F$.

Given two sets, E_1, E_2, sets $F_1 \subseteq E_1$, $F_2 \subseteq E_2$ and a bijection $h : F_1 \to F_2$ we say that $\vec{a} = (a_1, a_2, \dots, a_k) \in (E_1)^k$ is of the same h-type as $\vec{b} = (b_1, b_2, \dots, b_k) \in (E_2)^k$ (in symbols: $\vec{a} \sim_h \vec{b}$) if $h(\vec{a})$ is of the same F_2-type as $\vec{b}$, where $h(\vec{a}) = (h(a_1), h(a_2), \dots, h(a_k))$. Other way of saying this is that $h^{-1}(\vec{b})$ is of the same F_1-type as $\vec{a}$.

Intuitively, $\vec{a}, \vec{b} \in E^k$ are of the same F-type if: the elements from F occurring in $\vec{a}$ are the same as the elements from F occurring in $\vec{b}$; moreover, these elements are positioned in exactly the same way in $\vec{a}$ and $\vec{b}$; finally, positions at which elements from $E - F$ occur, must satisfy the same equalities. Here are some examples of true statements: $(1,2,3,4,5,6,5) \sim_4 (1,2,3,5,7,8,7)$, $(1,2,3,4,5,6,7) \not\sim_4 (1,2,6,5,4,8,7)$, $(1,2,3,4,5,6,7) \not\sim_4 (1,2,2,5,4,8,7)$, $(1,2,3,4,5,6,7) \not\sim_4 (1,2,3,5,4,8,5)$.

Clearly, given any set E, $F \subseteq E$, $\sim_F$ is an equivalence relation on E^k, for every $k \in \mathbb{N}$.

Lemma 3.7. *Let E be a set and let $F \subset E$ be finite. Then $E^k/\sim_F$ is finite.*

Proof. Finiteness of $E^k/\sim_F$ is obvious. Out of curiosity, for infinite E and $|F| = n > 0$: $|E^k/\sim_F| = \Sigma_{i=0}^{k} \binom{k}{i} \cdot n^i \cdot B_{k-i}$, where B_j is the Bell number which counts the number of partitions of the set of cardinality j. In the above expression, i is the number of indices among $1, 2, \dots, k$ at which an element from F occurs. There are n^i possible ways of assigning elements from F to given i indices. Let J be the rest of $k-i$ indices at which elements from $E - F$ occur. There is one-one correspondence between partitions of J and the number of ways of choosing at which indices among J the same elements of $E - F$ should be placed. ■

Lemma 3.8. *If $\phi(x_1, \dots, x_k)$ is a quantifier-free formula, $k \geq 0$ and $c = \max\{n : \underline{n}$ occurs in $\phi\}$, then $\mathcal{N} \models \phi[\vec{a}] \iff \mathcal{N} \models \phi[\vec{b}]$, for every $\vec{a}, \vec{b} \in \mathbb{N}^k$ such that $\vec{b} \sim_c \vec{a}$.*

Proof. Let $\phi(x_1, \dots, x_k)$ be a quantifier-free formula, $k \geq 0$, let $c = \max\{n : \underline{n}$ occurs in $\phi\}$ and let $\vec{a}, \vec{b} \in \mathbb{N}^k$ such that $\vec{b} \sim_c \vec{a}$. We show that $\mathcal{N} \models \phi[\vec{a}] \iff \mathcal{N} \models \phi[\vec{b}]$. It suffices to prove that, for every atomic formula ψ in ϕ, $\mathcal{N} \models \psi[\vec{a}] \iff \mathcal{N} \models \psi[\vec{b}]$.

Let ψ be an atomic formula in ϕ.

Suppose $\psi := (\underline{d} = \underline{d'})$, for some $d, d' \leq c$. The following are equivalent: $\mathcal{N} \models (\underline{d} = \underline{d'})[\vec{a}]$, $d = d'$, $\mathcal{N} \models (\underline{d} = \underline{d'})[\vec{b}]$.

Suppose $\psi := (x_i = \underline{d})$, for some $i \in \{1, 2, \ldots, k\}$ and $d \leq c$. If $a_i \leq c$, then $b_i = a_i$ because $\vec{a} \sim_c \vec{b}$. On the other hand, if $a_i > c$ then, by $\vec{a} \sim_c \vec{b}$, $b_i > c$ and thus both $a_i = d$ and $b_i = d$ are false. Hence, the following are equivalent: $\mathcal{N} \models (x_i = \underline{d})[\vec{a}]$, $a_i = d$, $b_i = d$, $\mathcal{N} \models (x_i = \underline{d})[\vec{b}]$.

Suppose $\psi := (x_i = x_j)$, for some $i, j \in \{1, 2, \ldots, k\}$. We will show that $a_i = a_j \iff b_i = b_j$. We prove left-to-right implication, because the reverse implication is symmetric. Assume $a_i = a_j$. Suppose $a_i \leq c$. Then $b_i = a_i$. Moreover, since $c \geq a_i = a_j$, we also have $b_j = a_j$. Hence $b_i = a_i = a_j = b_j$. Now, suppose that $a_i > c$. Then $b_i > c$, $a_j > c$ and $b_j > c$. By the definition of $\sim_c$ the equivalence $a_i = a_j \iff b_i = b_j$ holds. Again, the following are equivalent: $\mathcal{N} \models (x_i = x_j)[\vec{a}]$, $a_i = a_j$, $b_i = b_j$, $\mathcal{N} \models (x_i = x_j)[\vec{b}]$. ■

LEMMA 3.9. *$f : \mathbb{N}^k \to \mathbb{N}$ is qf-definable in $\mathcal{N}$ iff there exists $c \in \mathbb{N}$ such that for every $\vec{x} \in \mathbb{N}^k$, $f \restriction [\vec{x}]_{\sim_c}$ is constant or a projection.*

PROOF. We start with the left-to-right implication. Let $f : \mathbb{N}^k \to \mathbb{N}$ be qf-definable in $\mathcal{N}$ by $\phi(x_1, \ldots, x_{k+1})$. Let $c = \max\{n : \underline{n} \text{ occurs in } \phi\}$. Let $\vec{n} = n_1, \ldots, n_k \in \mathbb{N}^k$ and let $m = f(\vec{n})$.

First, suppose m occurs in $\vec{n}$ and $m \leq c$. We show that $f \restriction [\vec{n}]$ is constant. Let $\vec{n}' \sim_c \vec{n}$. Clearly, m must occur in $\vec{n}'$ in exactly the same places as in $\vec{n}$. Therefore, $(\vec{n}', m) \sim_c (\vec{n}, m)$. Since $\mathcal{N} \models \phi[\vec{n}, m]$, we also have $\mathcal{N} \models [\vec{n}', m]$, by Lemma 3.8, and thus $f(\vec{n}) = m$ so $f \restriction [\vec{n}]_{\sim_c}$ is constant.

Second, suppose that m occurs in $\vec{n}$ and $m > c$. We show that $f \restriction [\vec{n}]$ is a projection. Let $\vec{n}' \sim_c \vec{n}$. Choose $i \in \{1, 2, \ldots, k\}$ such that $n_i = m$. Observe that we have $(\vec{n}', n'_i) \sim_c (\vec{n}, n_i)$. But $\mathcal{N} \models \phi[\vec{n}, n_i]$ holds, so by Lemma 3.8 $\mathcal{N} \models \phi[\vec{n}', n'_i]$ holds as well. Therefore, $f(\vec{n}') = n'_i$ which proves that $f \restriction [\vec{n}]_{\sim_c}$ is a projection.

Third, we show that if m does not occur in $\vec{n}$ then $m \leq c$. For the sake of contradiction, suppose that m does not occur in $\vec{n}$ but $m > c$. Let $\vec{n}' \sim_c \vec{n}$. We have $(\vec{n}, m) \sim_c (\vec{n}, m')$, for every $m' > c$ such that $m' \notin \{p : p > c \wedge p \in \vec{n}\}$. But then $\mathcal{N} \models \phi[\vec{n}, m']$, for each such m', which is impossible because f is a function.

Fourth, we show that if m does not occur in $\vec{n}$ and $m \leq c$ then $f \restriction [\vec{n}]$ is constant. Let $\vec{n}' \sim_c \vec{n}$. Then m does not occur in $\vec{n}'$. Hence, $(\vec{n}, m) \sim_c (\vec{n}', m)$. Since $\mathcal{N} \models \phi[\vec{n}, m]$, we also have $\mathcal{N} \models \phi[\vec{n}, m]$ by Lemma 3.8, so $f(\vec{n}') = m$ and thus $f \restriction [\vec{n}]_{\sim_c}$ is constant.

We proceed to the proof of the right-to-left implication. Assume that there exists $c \in \mathbb{N}$ such that for every $\vec{n} \in \mathbb{N}^k$, $f \restriction [\vec{n}]_{\sim_c}$ is constant or a projection. Choose an appropriate c. By Lemma 3.7, $\mathbb{N}^k/\sim_c$ is finite. Let $\mathbb{N}^k/\sim_c = \{N_1, N_2, \ldots, N_p\}$. Observe that for each N_i there exists a formula $\phi_i(x_1, \ldots, x_k)$ such that $N_i = \{\vec{n} \in \mathbb{N}^k : \mathcal{N} \models \phi_i(x_1, \ldots, x_k)[\vec{n}]\}$. Now, if $f \restriction N_i$ is constant and $f(\vec{n}) = d$ for all $\vec{n} \in N_i$, let $\psi_i := (\phi_i \Rightarrow x_{k+1} = \underline{d})$. If $f \restriction N_i$ is a projection, i.e. for some $j \in \{1, 2, \ldots, k\}$, $f(\vec{n}) = n_j$, for all $\vec{n} \in N_i$, let $\psi_i := (\phi_i \Rightarrow x_{k+1} = x_j)$. Finally, let $\phi(x_1, x_2, \ldots, x_{k+1}) := \psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_p$. Formula ϕ is quantifier-free and defines f in $\mathcal{N}$. ■

THEOREM 3.10. *The class of total k-ary functions computable in every injective notation is the class of total k-ary functions which are qf-definable in $\mathcal{N}$.*

PROOF. Let $f : \mathbb{N}^k \to \mathbb{N}$. For $k = 1$, the theorem holds by Lemma 3.9 and Theorem 3.1. It remains to prove that the theorem holds for $k > 1$. Let $k > 1$.

First, we show the right-to-left implication. Let f be qf-definable by $\phi(x_1, \ldots, x_k, y)$. Let (S, σ) be an injective notation. Let $\beta_1, \beta_2, \ldots$ be a computable enumeration of S without repetitions. Let $\mathcal{S} = (S, c^S)$ be a model where the interpretation function $c^S : \{\underline{0}, \underline{1}, \ldots\} \to S$ is defined by $c^S = \sigma^{-1}c^{\mathbb{N}}$.

The program computing f in (S, σ) looks as follows: given $\alpha_1, \ldots, \alpha_k \in S$, compute $j :=$ the least i such that $\mathcal{S} \models \phi[\alpha_1, \ldots, \alpha_k, \beta_i]$ and return β_j.

First, observe that, given arbitrary $\gamma_1, \ldots, \gamma_k, \gamma \in S$, one can compute whether $\mathcal{S} \models \phi[\gamma_1, \ldots, \gamma_k, \gamma]$ holds. One has to calculate the truth values of all atomic formulae occurring in $\phi[\gamma_1, \ldots, \gamma_k, \gamma]$. This is effective because each atomic formula in $\phi[\gamma_1, \ldots, \gamma_k, \gamma]$ is of the form $\delta = \eta$, for some strings $\delta, \eta \in S$. The rest follows from the computability of the truth value functions.

It remains to show that the above program stops when fed with an arbitrary k-tuple $\alpha_1, \ldots, \alpha_k \in S$ and outputs β such that $f(\sigma(\alpha_1), \ldots, \sigma(\alpha_k)) = \sigma(\beta)$. It is easy to verify that σ is an isomorphism between $\mathcal{S}$ and $\mathcal{N}$. It follows that for all $i \in \mathbb{N}$:

$$\mathcal{S} \models \phi[\alpha_1, \ldots, \alpha_k, \beta_i] \iff \mathcal{N} \models \phi[\sigma(\alpha_1), \ldots, \sigma(\alpha_k), \sigma(\beta_i)]. \tag{5}$$

Because ϕ defines a function, there is precisely one number m such that $\mathcal{N} \models \phi[\sigma(\alpha_1), \ldots, \sigma(\alpha_k), m]$. Let $\beta = \sigma^{-1}(m)$. By (5), $\mathcal{S} \models \phi[\alpha_1, \ldots, \alpha_k, \beta_i]$ holds iff $\beta_i = \beta$. Hence, checking the condition $\mathcal{S} \models \phi[\alpha_1, \ldots, \alpha_k, \beta_i]$ for $i = 1, 2, \ldots,$ the first encountered i for which $\mathcal{S} \models \phi[\alpha_1, \ldots, \alpha_k, \beta_i]$ holds will be such that $\beta_i = \beta$. Hence, the program stops, returns β which is the right output since $f(\sigma(\alpha_1), \ldots, \sigma(\alpha_k)) = \sigma(\beta)$.

We proceed to the left-to-right implication. Suppose f is not qf-definable. We will construct a notation (T, τ) in which f is not computable. Let $T \subseteq \Sigma^*$

be an infinite computabe set. Let $\beta_0, \beta_1, \dots$ be an enumeration of T without repetitions.

Construction. We proceed in stages. At stage $n+1$ we have a finite injection $\tau_n : [0, l_n] \to T$ which we extend to a finite injection $\tau_{n+1} : [0, l_{n+1}] \to T$, $l_n \leq l_{n+1}$. τ_{n+1} is chosen so that the following requirement is satisfied:

R_n : $\Phi_n^{(k)}$ does not compute f in any injective notation (T, τ') such that τ_{n+1}^{-1} is an initial segment of τ'.

Moreover, the extensions are chosen in a way that, by the end of the construction, each element of T occurs in some τ_n. The final notation is defined by $\tau = \bigcup_{n \in \mathbb{N}} \tau_n^{-1}$. Since k is fixed, we shall write Φ_n instead $\Phi_n^{(k)}$.

Stage 0. $\tau_0 = \beta_0$.

Stage $n+1$. Let $s = |\tau_n| - 1$. We say that x is fresh if $x > s$. A tuple $\vec{x} \in \mathbb{N}^k$ is said to be fresh if some number occurring in $\vec{x}$ is fresh. Similarly, we say that a numeral α is fresh if α does not occur in τ_n. A tuple $\vec{\alpha} = \alpha_1, \dots, \alpha_k$ is said to be fresh if some numeral occurring in α is fresh.

1. Check whether Φ_n is partial. If not, go to 2.
   Since Φ_n is partial, R_n is satisfied because f is total and thus Φ_n cannot compute f. Set $\tau_{n+1} = \tau_n$ and go to the next stage.

2. We work under the assumption that Φ_n is total.

   Check whether there exist $\vec{i} = i_1, \dots, i_k \leq s$ such that $f(\vec{i}) > s$.

   If not, go to 3. Otherwise, choose $\vec{i} = i_1, \dots, i_k \leq s$ such that $f(\vec{i}) > s$. Set $\beta := \Phi_n(\tau_n(i_1), \dots, \tau_n(i_k))$. If β occurs in τ_n, then R_n is satisfied, set $\tau_{n+1} = \tau_n$ and go to the next stage. Otherwise β is fresh. Choose the least $j > s$ such that $j \neq f(\vec{i})$ and set $\tau_{n+1} = \tau_n B \beta$, where B is a list of distinct fresh numerals, each $\neq \beta$, chosen in a way to guarantee that $\tau_{n+1}(j) = \beta$. Go to the next stage.

3. We work under the assumption that Φ_n is total and for every $\vec{i} = i_1, \dots, i_k \leq s$ we have $f(\vec{i}) \leq s$.

   Check whether there exist $\vec{i} \leq s$ such that $\Phi_n(\tau_n(i_1), \dots, \tau_n(i_k)) \neq \tau_n(f(\vec{i}))$.

   If not, go to 4. Otherwise, R_n is satisfied, set $\tau_{n+1} = \tau_n$ and go to the next stage.

4. We work under the assumption that Φ_n is total, for every $\vec{i} = i_1, \dots, i_k \leq s$ we have $f(\vec{i}) \leq s$ and $\Phi_n(\tau_n(i_1), \dots, \tau_n(i_k)) = \tau_n(f(\vec{i}))$.

> Check whether there exists a fresh tuple $\vec{\alpha} = \alpha_1, \ldots, \alpha_k$ such that $\beta := \Phi_n(\vec{\alpha})$ is fresh and $\beta \notin \vec{\alpha}$.

If not, go to 5. Otherwise, let $\delta_1, \ldots, \delta_p$ be the list of all distinct fresh numerals occurring in $\vec{\alpha}$. Define the function $i(\delta_j) = s{+}j$, for $j = 1, 2, \ldots, p$. Moreover, for each non-fresh numeral γ from $\vec{\alpha}$, define $i(\gamma)$ to be the position of γ in τ_n. Now, compute the value $y_0 = f(i(\alpha_1), \ldots, i(\alpha_k))$. If $y_0 \leq s{+}p$, set $\tau_{n+1} = \tau_n \delta_1 \ldots \delta_p \beta$, thus satisfying R_n. Otherwise, let j be the least number such that $j > s{+}p$ and $j \neq y_0$. Set $\tau_{n+1} = \tau_n \delta_1 \ldots \delta_p B \beta$, where B is a list of new numerals (i.e. fresh and not occurring in $\vec{\alpha}$ and different from β) such that the position of β in τ_{n+1} is j. This way R_n becomes satisfied. Go to the next stage.

5. We work under the assumption that Φ_n is total, for every $\vec{i} = i_1, \ldots, i_k \leq s$ we have $f(\vec{i}) \leq s$ and $\Phi_n(\tau_n(i_1), \ldots, \tau_n(i_k)) = \tau_n(f(\vec{i}))$, for every fresh $\vec{\alpha}$ we always have $\Phi_n(\vec{\alpha}) = \beta$, where β is not fresh or $\Phi_n(\vec{\alpha}) = \alpha_i$, for some $i \in \{1, 2, \ldots, k\}$.

   > Check whether there exists fresh $\vec{\alpha}$, fresh $\vec{x}$ of the same τ_n-type as α and $i \in \{1, 2, \ldots, k\}$ such that $\Phi_n(\vec{\alpha}) = \alpha_i$ and $f(\vec{x}) \neq x_i$.[1]

   If not, go to 6. Otherwise, choose $\vec{\alpha}$, $\vec{x}$ and i accordingly. For each $\alpha_i \in \vec{\alpha}$, if α_i is fresh, place it at position x_i in τ_{n+1}. If some positions $< \max\{x_i : \alpha_i \text{ is fresh}\}$ remain unfilled in τ_{n+1}, fill them with distinct new numerals (i.e. fresh and not occurring in $\vec{\alpha}$). Clearly, this way R_n is satisfied. Go to the next stage.

6. We work under the assumption that Φ_n is total, for every $\vec{i} = i_1, \ldots, i_k \leq s$ we have $f(\vec{i}) \leq s$ and $\Phi_n(\tau_n(i_1), \ldots, \tau_n(i_k)) = \tau_n(f(\vec{i}))$, for every fresh $\vec{\alpha}$ we always have $\Phi_n(\vec{\alpha}) = \beta$, where β is not fresh or $\Phi_n(\vec{\alpha}) = \alpha_i$, for some $i \in \{1, 2, \ldots, k\}$ and for every fresh $\vec{\alpha}$, fresh $\vec{x}$ of the same τ_n-type as $\vec{\alpha}$ and $i \in \{1, 2, \ldots, k\}$: $\Phi_n(\vec{\alpha}) = \alpha_i$ implies $f(\vec{x}) = x_i$.

   > Check whether there exists fresh $\vec{\alpha}$, $\vec{x}$ of the same τ_n-type as $\vec{\alpha}$ such that $\Phi_n(\vec{\alpha}) = \beta$ and $f(\vec{x}) \neq y$, where y is the position of β in τ_n.

   If not, go to the next stage. Otherwise, choose $\vec{\alpha}, \vec{x}$ accordingly. For each $\alpha_i \in \vec{\alpha}$, if α_i is fresh, place it at position x_i in τ_{n+1}. If some positions $< \max\{x_i : \alpha_i \text{ is fresh}\}$ remain unfilled in τ_{n+1}, fill them with distinct new numerals (i.e. fresh and not occurring in $\vec{\alpha}$). This way R_n gets satisfied. Go to the next stage.

[1] Notice that the list τ_n is treated as a bijection between positions $0, 1, \ldots, s$ and the set of numerals occurring τ_n.

This ends the construction.

Verification Every stage of the construction is based on the checklist of six questions, each equipped with the description of how to react to a positive answer. If the answer to at least one question from the checklist is positive, R_n is immediately satisfied and this fact is verified in the description of the construction. Hence, if there is no stage at which all questions give negative answer, then we are done because every R_n is satisfied. Since, by the definition of τ, τ_{n+1}^{-1} is an initial segment of τ, Φ_n does not compute f in (T, τ).

It remains to show that at every stage, at least one question from the checklist gives positive answer. Suppose it is not the case. Let $n+1$ be a stage such that all questions from the checklist give negative answer. Let $s = |\tau_n| - 1$. We show that f is qf-definable in $\mathcal{N}$. By Lemma 3.9 it suffices to show that for every $\vec{z} \in \mathbb{N}^k$, $f \restriction [\vec{z}]_{\sim_s}$ is constant or a projection.

Given $\vec{z}$ with all elements $\leq s$, this is obvious, because $[\vec{z}]_{\sim_s}$ is a singleton. Let $\vec{z}$ be given such that at least one of its elements is $> s$. Then $\vec{z}$ is fresh. Let $\vec{\gamma}$ be of the same τ_n-type as $\vec{z}$. Since the 5th question from the checklist gives negative answer, the following holds: for every fresh $\vec{\alpha}$, $\Phi_n(\vec{\alpha}) = \beta$, where β is not fresh, or $\Phi_n(\vec{\alpha}) = \alpha_i$, for some $i \in \{1, 2, \ldots, k\}$. We consider two cases.

First, suppose that $\Phi_n(\vec{\gamma}) = \delta$, where δ is not fresh. The negative answer to the 6th question from the checklist states that for every fresh $\vec{\alpha}$, $\vec{x}$ of the same τ_n-type as $\vec{\alpha}$, if $\Phi_n(\vec{\alpha}) = \beta$ and β is not fresh then $\beta = \tau_n(f(\vec{x}))$. Hence, $\delta = \tau_n(f(\vec{z}))$. Let $\vec{z}' \sim_s \vec{z}$. Then $\vec{z}'$ is of the same τ_n-type as $\vec{\gamma}$. Again, by the negative answer to the 6th question from the checklist, $\delta = \tau_n(f(\vec{z}'))$. Since τ_n is injective, $f(\vec{z}) = f(\vec{z}')$. Therefore, $f \restriction [\vec{z}]_{\sim_s}$ is constant.

Second, suppose that $\Phi_n(\vec{\gamma}) = \gamma_i$, for some $i \in \{1, 2, \ldots, k\}$. If γ_i is not fresh, then the previous case applies. Assume γ_i is fresh. The negative answer to the 6th question from the checklist states that for every fresh $\vec{\alpha}$, fresh $\vec{x}$ of the same τ_n-type as $\vec{\alpha}$ and $i \in \{1, 2, \ldots, k\}$: $\Phi_n(\vec{\alpha}) = \alpha_i$ implies $f(\vec{x}) = x_i$. Hence, $f(\vec{z}) = z_i$. Let $\vec{z}' \sim_s \vec{z}$. $\vec{z}'$ is of the same τ_n-type as $\vec{\gamma}$. Again, by the negative answer to the 6th question from the checklist, $f(\vec{z}') = z_i'$. Therefore, $f \restriction [\vec{z}]_{\sim_s}$ is a projection.

By Lemma 3.9, f is qf-definable in $\mathcal{N}$ which is impossible because we have assumed otherwise. This ends to proof of the fact that at each stage, at least one of the questions from the checklist must give positive answer. Hence, the proof of the theorem is complete.

■

THEOREM 3.11. *The class of k-ary relations computable in every injective notation is the class of k-ary relations qf-definable in $\mathcal{N}$.*

PROOF. Let $R \subseteq \mathbb{N}^k$. Observe that $\chi_=$ is computable in every injective notation. By Theorem 2.9. from [19], given an injective notation (S, σ), R is computable in (S, σ) iff χ_R is computable in (S, σ). Therefore, the following are equivalent: R is computable in every injective notation, χ_R is computable in every injective notation, χ_R is qf-definable in $\mathcal{N}$, R is qf-definable in $\mathcal{N}$. ■

3.1. Extension to partial functions

Theorems 3.10 and 3.11 answer Problem 1 with regard to total k-ary functions and relations. In this section, we provide a full answer for arbitrary partial k-ary functions.

DEFINITION 3.12. Let (S, σ) be an injective notation. We say $A \subseteq \mathbb{N}^k$ is computably enumerable (c.e.) in (S, σ), if $A = \emptyset$ or there exists a total function $f : \mathbb{N} \to \mathbb{N}^k$ such that f is computable in (S, σ) and $A = \{f(n) : n \in \mathbb{N}\}$.

The following proposition shows that basic characterizations of c.e. sets remain valid under injective notations.

PROPOSITION 3.13. *Let (S, σ) be an injective notation and let $A \subseteq \mathbb{N}^k$. The following are equivalent:*

1. *A is c.e. in (S, σ).*
2. *A is the domain of a partial k-ary function computable in (S, σ).*
3. *There exists a relation $R \subseteq \mathbb{N}^{k+1}$ such that R is computable in (S, σ) and $A = \{(n_1, \ldots, n_k) : \exists n_0 R(n_0, n_1, \ldots, n_k)\}$.*

PROOF. Let (S, σ) be an injective notation, $A \subseteq \mathbb{N}^k$.

(1 $\Rightarrow$ 2) Choose $f : \mathbb{N} \to \mathbb{N}^k$ computable in (S, σ) such that $A = \{f(n) : n \in \mathbb{N}\}$. Let $\beta_1, \beta_2, \ldots$ be a computable enumeration of S without repetitions. The following program computes in (S, σ) a partial k-ary function whose domain is A. On input $\alpha_1, \alpha_2, \ldots, \alpha_k \in S$, for $i = 1, 2, \ldots$, compute $(\alpha'_1, \alpha'_2, \ldots, \alpha'_k) := f(\beta_i)$ and if $\alpha_1 = \alpha'_1, \alpha_2 = \alpha'_2, \ldots, \alpha_k = \alpha'_k$, then stop.

(2 $\Rightarrow$ 3) Suppose A is the domain of a partial $g : \mathbb{N}^k \to \mathbb{N}$ computable in (S, σ) by a program P. Let $\beta_1, \beta_2, \ldots$ be a computable enumeration of S without repetitions. Define the relation $R \subseteq \mathbb{N}^{k+1}$ such that for every $\alpha_1, \ldots, \alpha_k \in S$, for every $t \in \mathbb{N}$:

$R(\sigma(\beta_t), \sigma(\alpha_1), \ldots, \sigma(\alpha_k))$ iff P stops in at most t steps on input $\alpha_1, \ldots, \alpha_k$.

Clearly, $A = \{(n_1, \ldots, n_k) : \exists n_0 R(n_0, n_1, \ldots, n_k)\}$ and R is computable in (S, σ).

($3 \Rightarrow 1$) Let $A = \{(n_1, \ldots, n_k) : \exists n_0 R(n_0, n_1, \ldots, n_k)\}$, where R is computable in (S, σ). Consider an effective enumeration of S^{k+1} and once you enumerate $\alpha_0, \alpha_1, \ldots, \alpha_k$ such that $R^\sigma(\alpha_0, \alpha_1, \ldots, \alpha_k)$, output $(\alpha_1, \ldots, \alpha_k)$. ■

Theorem 3.14. *The class of total k-ary functions learnable in every injective notation is the class of total k-ary functions qf-definable in $\mathcal{N}$.*

Proof. The right-to-left implication, holds by Theorem 3.10. The left-to-right implication is obtained by assuming f is not qf-definable and repeating the construction of Theorem 3.10 with each occurrence of Φ_n replaced by $\Gamma_n^{0'}$. Such a construction determines a notation (T, τ) such that f^τ is not computed by any $\Gamma_n^{0'}$ and thus $f^\tau \not\leq 0'$. By the Limit Lemma, f^τ is not learnable, and by Definition 2.6, f is not learnable in (T, τ). ■

Theorem 3.15. *The class of k-ary relations learnable in every injective notation is the class of k-ary relations qf-definable in $\mathcal{N}$.*

Proof. The proof is the same as that of Theorem 3.11. It is necessary to realize the following consequence of Theorem 2.9 in [19]: learnability of $R \subseteq \mathbb{N}^k$ in an injective notation (S, σ) is equivalent to the learnability of χ_R in (S, σ). ■

4. Computability across notations

In this section we solve

Problem 2. *What is the class of k-ary total functions and relations computable in every notation, for $k = 1, 2, \ldots$?*

Results contained in this part of the article were earlier published in the PhD thesis [19].

Theorem 4.1. *The only unary functions computable in every notation are constant and identity functions.*

Proof. It is clear that all constant and identity functions are computable in every notation. We need to prove that only these functions are. Suppose f is neither constant nor identity. Thanks to the theorem 3.1, we only need to consider the following two cases:

1. f is almost constant but not constant,
2. f is almost identity but not identity.

Case 1 We shall construct a notation (S, σ) in which f is not computable. Let S be the standard set of numerals. Let k be the value of f for nearly all arguments and $a_0, ..., a_m$ be all arguments for which f takes values other than k.

Let $A \subseteq N$ be a set not computable in the standard notation. We shall construct σ as follows:

$$\sigma(\overline{0}) = k.$$

Let $b_0, b_1, b_2, ...$ be an enumeration of all numbers from $A \setminus \{0\}$ and $c_0, c_1, c_2, ...$ — an enumeration of all numbers from $(\mathbb{N} \setminus A) \setminus \{0\}$.

For $i \in \mathbb{N}$, to each of the numerals $\overline{b_i}$, function σ shall assign one of numbers $a_0, ..., a_m$ and to each of the numerals $\overline{c_i}$ it shall assign one of numbers from the set $\mathbb{N} \setminus \{a_0, ..., a_m\}$. It shall be done in such a way that function σ shall be surjective and for every positive natural number t: $\sigma(\overline{t}) \neq k$.

This is possible unless k is the only argument for which f assumes a value different from k. We shall deal with this case later.

Suppose that f is computable in (S, σ). We shall show that contrary to our assumption A needs to be computable in the standard notation.

The algorithm provided below only works for $n \neq k$ but that does not need to bother us since the answer for k can be given explicitly.

Let $n \in \mathbb{N} \setminus \{k\}$. We want to know whether $n \in A$. We calculate $f^\sigma(\overline{n})$. The following are equivalent:

1. $f^\sigma(\overline{n}) \neq \overline{0}$,
2. $f(\sigma(\overline{n})) \neq k$,
3. $\sigma(\overline{n}) \in \{a_0, a_1, ..., a_m\}$,
4. $\overline{n} \in \{\overline{b_i} : i \in \mathbb{N}\}$,
5. $n \in \{b_i : i \in \mathbb{N}\}$,
6. $n \in A$.

Analogously, we can prove that $n \notin A \Leftrightarrow f^\sigma(\overline{n}) = \overline{0}$ and we have obtained a contradiction.

To complete the proof of the first case, we need to consider f of the following form:

$$f(n) = \begin{cases} k & \text{if } n \neq k, \\ l & \text{if } n = k, \end{cases}$$

where $k \neq l$. This is because in such a case it is not possible to construct σ in the way described above.

Let $A \subseteq \mathbb{N}$ be a set of natural numbers not computable in the standard notation. Let $a_0, a_1, a_2, ...$ be an enumeration of all numbers from $A \setminus \{0\}$ and $b_0, b_1, b_2, ...$ — an enumeration of all numbers from $(\mathbb{N} \setminus A) \setminus \{0\}$.

We shall construct a notation (S, σ) in which f is not computable. Let S be the standard set of numerals. We shall construct σ as follows:

$$\sigma(\overline{0}) = l,$$

$$\sigma(\overline{a_i}) = k, \text{for all } i \in \mathbb{N},$$

and to all the numerals $\overline{b_0}, \overline{b_1}, \overline{b_2}, ...$ we shall assign all the numbers other than k and l.

Suppose that f is computable in (S, σ). We shall show that contrary to our assumption A needs to be computable in the standard notation.

The algorithm provided below works only for $n \neq 0$ but that does not need to bother us because we can give the answer for 0 explicitly.

Let $n \in \mathbb{N} \setminus \{0\}$. We want to know whether $n \in A$. We calculate $f^\sigma(\overline{n})$. The following are equivalent:

1. $f^\sigma(\overline{n}) = \overline{0}$,
2. $f(\sigma(\overline{n})) = l$,
3. $\sigma(\overline{n}) = k$,
4. $\overline{n} \in \{\overline{a_i} : i \in \mathbb{N}\}$,
5. $n \in \{a_i : i \in \mathbb{N}\}$,
6. $n \in A$.

Analogously, we can prove that $n \notin A \Leftrightarrow f^\sigma(\overline{n}) \neq \overline{0}$ and we have obtained a contradiction.

Case 2 Assume that f is almost identity but not identity. Let $A \subseteq \mathbb{N}$ be a set of natural numbers not computable in the standard notation. Let $a_0, a_1, a_2, ...$ be an enumeration of all numbers from A and $b_0, b_1, b_2, ...$ — an enumeration of all numbers from $\mathbb{N} \setminus A$.

We shall construct a notation (S, σ) in which f is not computable. S shall be the standard set of numerals. Let $c_0, c_1, ..., c_m$ be natural numbers such that $f(c_i) \neq c_i$ for $i = 0, ..., m$ and let them be the only natural numbers with such a property.

Now let us construct σ. To each of the numerals $\overline{a_i}$, σ shall assign one of the numbers c_i and to each of the numerals $\overline{b_i}$, σ shall assign one of the other numbers. This shall be done in such a way that each natural number is assigned to at least one numeral.

Suppose for the sake of contradiction that f is computable in (S, σ). We shall provide an algorithm for A. Let $n \in \mathbb{N}$. We want to know whether $n \in A$. We calculate $f^\sigma(\overline{n})$. For any n, the following conditions are equivalent:

1. $f^\sigma(\overline{n}) \neq \overline{n}$,
2. $\sigma(\overline{n}) \in \{c_0, c_1, ..., c_m\}$,
3. $\overline{n} \in \{\overline{a_i} : i \in \mathbb{N}\}$,
4. $n \in \{a_i : i \in \mathbb{N}\}$,
5. $n \in A$.

Analogously, for every natural number n the following holds:

$$f^\sigma(\overline{n}) = \overline{n} \Leftrightarrow n \notin A.$$

Therefore we have obtained a contradiction.

It follows that the only functions computable in every notation are constant and identity functions.

■

Definition 4.2. A function $f : \mathbb{N}^k \to \mathbb{N}$ shall be called a projection if there exists $i \in \{1, ..., k\}$ such that:

$$\forall_{x_1} ... \forall_{x_k} \; f(x_1, ..., x_k) = x_i.$$

Theorem 4.3. *The only functions $f : \mathbb{N}^k \to \mathbb{N}$ computable in every notation are constant functions and projections.*

Proof. The implication $(\Leftarrow)$ is obvious.

We shall prove the implication $(\Rightarrow)$ by induction over k. The previous theorem constitutes the base case for this induction. Now suppose that the only functions of k arguments computable in every notation are constant functions and projections. Let $f : \mathbb{N}^{k+1} \to \mathbb{N}$ be a function computable in every notation. We wish to show that it is either constant or a projection. In this proof we shall utilise Lemmas 4.4, 4.5, 4.6 and 4.7, included below.

For every $1 \leq i \leq k+1$ and every $j \in \mathbb{N}$, let us define a function:

$$f_{i,j} : \mathbb{N}^k \to \mathbb{N}$$

such that for all $x_1 ...., x_{i-1}, x_{i+1}, ..., x_{k+1}$:

$$f_{i,j}(x_1 ...., x_{i-1}, x_{i+1}, ..., x_{k+1}) = f(x_1 ...., x_{i-1}, j, x_{i+1}, ..., x_{k+1}),$$

i.e. a function obtained by substituting the value j for the variable x_i in f.

All functions $f_{i,j}$ are computable in every notation because they are obtained by substituting a value for a variable in f, and f is computable in every notations. Therefore, by inductive assumption, each of them is either constant or a projection.

We want to show that f is either a constant function or a projection. We have two cases to consider.

Case 1 Suppose that among all functions $f_{i,j}$ there is at least one projection $f_{i_0,j_0} = x_l$. Then by Lemmas 4.4 and 4.5, every function $f_{i,j}$ is also a projection on the same coordinate x_l, unless $i = l$. Hence, for any $i \neq l$ and any $x_1, ..., x_{k+1}$:

$$f(x_1, ..., x_{k+1}) = f_{i,x_i}(x_1 ...., x_{i-1}, x_{i+1}, ..., x_{k+1}) = x_l.$$

Therefore, f is a projection on x_l.

Case 2 Suppose that all functions $f_{i,j}$ are constant. Then by Lemmas 4.6 and 4.7 all these functions are identical and always equal to the same value c. Hence f is also constant and equal to c.

■

Note that functions $f_{i,j}$ mentioned in subsequent lemmas are those defined in the proof of Theorem 4.3.

LEMMA 4.4. *If f_{i_1,j_1} is a projection on x_l and $i_2 \neq l$, then f_{i_2,j_2} is also a projection.*

PROOF. Suppose to the contrary that the function f_{i_2,j_2} is not a projection, i.e. it is constant. Assume that for all arguments:

$$f_{i_1,j_1}(x_1, ..., x_{k+1}) = x_l$$

and

$$f_{i_2,j_2}(x_1, ..., x_{k+1}) = c.$$

Let us consider the following cases:

Case 1 Suppose that $i_1 \neq i_2$. We know that for every sequence of arguments:

$$f_{i_1,j_1}(x_1, ..., x_{i_1-1}, x_{i_1+1}, ..., x_{i_2-1}, x_{i_2}, x_{i_2+1}, ..., x_{k+1}) = x_l.$$

In particular, for $x_{i_2} = j_2$:

$$f_{i_1,j_1}(x_1, ..., x_{i_1-1}, x_{i_1+1}, ..., x_{i_2-1}, j_2, x_{i_2+1}, ..., x_{k+1}) = x_l.$$

But this is also the value of f_{i_2,j_2}, with j_1 substituted for x_{i_1}. This is however a contradiction since this is always equal to x_l, and we assumed that f_{i_2,j_2} is constant. Therefore, f_{i_2,j_2} must be a projection.

Case 2 Suppose that $i_1 = i_2$. If $j_1 = j_2$, then it is trivial. Hence suppose that $j_1 \neq j_2$.

Let $A \subseteq \mathbb{N}$ be uncomputable. We shall construct a notation (S, σ). Let S be the standard set of decimal numerals and let

$$\sigma(\overline{2n}) = \begin{cases} c & \text{if } n = 0, \\ j_1 & \text{if } n > 0 \land n \in A, \\ j_2 & \text{if } n > 0 \land n \notin A. \end{cases}$$

Assign the remaining numbers to numerals of the form $\overline{2n+1}$ in any injective way.

To obtain a contradiction, we want to construct an algorithm which decides whether $n \in A$. The answer for $n = 0$ is given explicitly as a special case. Assume $n > 0$. Since f is computable in every notation, we compute the value of f^σ, where we substitute the numeral $\overline{2n}$ for x_{i_1} (which is the same variable as x_{i_2}), the numeral $\overline{1}$ — for x_l, and for other variables we substitute any numerals.

Due to the construction of σ and because $n > 0$, the numeral substituted for x_{i_1} represents either j_1 or j_2. If it represents j_1, then f is a projection on x_l and it has to return a numeral which represents the same number as the numeral $\overline{1}$ — hence it has to return the numeral $\overline{1}$, since no other numeral represents the same number. If, on the other hand, the numeral substituted for x_{i_1} represents j_2, then f is a constant function always equal to c and in this case the algorithm returns the numeral $\overline{0}$.

Therefore, if the algorithm returns $\overline{1}$, then $n \in A$. If it returns $\overline{0}$, then $n \notin A$.

■

LEMMA 4.5. *If f_{i_1,j_1} and f_{i_2,j_2} are both projections, then they are projections on the same coordinate.*

PROOF. Suppose that $f_{i_1,j_1} = x_{l_1}$ and $f_{i_2,j_2} = x_{l_2}$ are different projections, i.e. $l_1 \neq l_2$. Let us consider the following cases:

Case 1 Suppose that $i_1 \neq i_2$. We know that for every sequence of arguments:

$$f_{i_1,j_1}(x_1, ..., x_{i_1-1}, x_{i_1+1}, ..., x_{i_2-1}, x_{i_2}, x_{i_2+1}, ..., x_{k+1}) = x_{l_1}.$$

In particular, for $x_{i_2} = j_2$:

$$f_{i_1,j_1}(x_1, ..., x_{i_1-1}, x_{i_1+1}, ..., x_{i_2-1}, j_2, x_{i_2+1}, ..., x_{k+1}) = x_{l_1}.$$

But this is also equal to f_{i_2,j_2}, with j_1 substituted for x_{i_1}, hence it is always equal to x_{l_2}. This is a contradiction since we can substitute different values for x_{l_1} and x_{l_1}.

Case 2 Suppose that $i_1 = i_2$. If $j_1 = j_2$, then it is trivial. Hence suppose that $j_1 \neq j_2$. Let $A \subseteq \mathbb{N}$ be a set uncomputable in the standard notation. We are going to construct a notation (S, σ). Let S be the standard set of numerals and let:

$$\sigma(\overline{2n}) = \begin{cases} j_1 & \text{if } n \in A, \\ j_2 & \text{if } n \notin A. \end{cases}$$

Assign the rest of numbers to the remaining numerals in any injective way.

To obtain a contradiction, we shall construct an algorithm which decides whether $n \in A$. Since f is computable in every notation, we are going to compute f^σ, with $\overline{2n}$ substituted for x_{i_1} (which is equal to x_{i_2}), $\overline{1}$ — for x_{l_1} and $\overline{3}$ — for x_{l_2}. If the output numeral is $\overline{1}$, then the algorithm has computed projection on coordinate x_{l_1}. Then $\sigma(\overline{2n}) = j_1$ and $n \in A$. Analogously, if the output numeral is $\overline{3}$, then $n \notin A$. Hence A is computable in the standard notation and we have obtained a contradiction.

■

LEMMA 4.6. *If $i_1 \neq i_2$ and functions $f_{i_1,j_1} = c_1$ and $f_{i_2,j_2} = c_2$ are constant, then $c_1 = c_2$.*

PROOF. Let these functions be constant and assume values, respectively, c_1 and c_2. Without loss of generality assume that $i_1 < i_2$. We shall show that $c_1 = c_2$. Then for any $x_1, \dots, x_{k+1}$:

$$\begin{aligned} c_1 &= f_{i_1,j_1}(x_1, ..., x_{i_1-1}, x_{i_1+1}, ..., x_{i_2-1}, j_2, x_{i_2+1}, ..., x_{k+1}) \\ &= f(x_1, ..., x_{i_1-1}, j_1, x_{i_1+1}, ..., x_{i_2-1}, j_2, x_{i_2+1}, ..., x_{k+1}) \\ &= f_{i_2,j_2}(x_1, ..., x_{i_1-1}, j_1, x_{i_1+1}, ..., x_{i_2-1}, x_{i_2+1}, ..., x_{k+1}) \\ &= c_2. \end{aligned}$$

■

LEMMA 4.7. *If $f_{i_0,j_0} = c$ is constant, then:*

1. $f_{i_0,j} = c$, *for every j, if all the functions $f_{i,j}$ are constant,*
2. $f_{i_0,j} = j$, *for every j, if at least one function $f_{i,j}$ is a projection.*

PROOF. First note that if $f_{i_0,j_0} = c$ is constant, then $f_{i_0,j}$ must be constant for every j. Otherwise $f_{i_0,j}$ would be a projection, for some j, and then, by Lemma 4.4, f_{i_0,j_0} would also be a projection. That would be a contradiction because f_{i_0,j_0} is constant.

Suppose that all the functions $f_{i,j}$ are constant. Consider the function f_{i_1,j_1} such that $i_1 \neq i_0$. By Lemma 4.6, it is also equal to c. Then, for any j we can again apply Lemma 4.6 to f_{i_1,j_1} and $f_{i_0,j}$ and we conclude that $f_{i_0,j} = c$, for every j.

Now suppose that the function f_{i_1,j_1} is a projection on x_l. Then, by Lemmas 4.4 and 4.5, all functions $f_{i,j}$ are projections on x_l unless $i = l$. Since all functions $f_{i_0,j}$ are constant, it follows that $l = i_0$. Then for all $i \neq i_0$ and all j, functions $f_{i,j}$ are projections on x_{i_0}, and for all j, functions $f_{i_0,j}$ are constant and equal to j.

■

DEFINITION 4.8. Let the notation (S, σ) of $\mathbb{N}$ be defined as follows:

The alphabet Σ consists of standard digits $\overline{0}, ..., \overline{9}$, a left bracket (, a right bracket) and a comma. The set of numerals S consists of all inscriptions of the form $(\overline{a}, \overline{b})$, where $\overline{a}$, $\overline{b}$ are standard numerals.

Let $B \subseteq \mathbb{N}$ be a set not computable in the standard notation. We define σ as follows:

$$\sigma((\overline{a}, \overline{b})) = \begin{cases} a & \text{if } b \notin B, \\ a+1 & \text{if } b \in B. \end{cases}$$

LEMMA 4.9. (S, σ) *defined as above is a correct notation for* $\mathbb{N}$.

PROOF. The only condition that might not be obvious is that for every natural number n there is a numeral $(\overline{a}, \overline{b}) \in S$ representing n. Let $n \in \mathbb{N}$. Since B is not computable, it follows that $B \neq \mathbb{N}$. Let $b \in \mathbb{N} \setminus B$. Then $\sigma(\overline{n}, \overline{b}) = n$.

■

LEMMA 4.10. *Let $A \subseteq \mathbb{N}$. Then A is computable in (S, σ) if and only if $A = \emptyset$ or $A = \mathbb{N}$.*

Proof. The implication ($\Leftarrow$) is obvious. We shall prove ($\Rightarrow$). Suppose that $A \neq \emptyset$, $A \neq \mathbb{N}$ and that A is computable in (S, σ).

There must be a number $n \in \mathbb{N}$ such that $n \notin A$ and $n+1 \in A$ (or $n \in A$ and $n+1 \notin A$ — but this case is analogous). For any $a \in \mathbb{N}$ the following conditions are equivalent:

1. $a \notin B$,
2. $\sigma((\overline{n}, \overline{a})) = n$,
3. $\sigma((\overline{n}, \overline{a})) \notin A$.

Analogously, for any $a \in \mathbb{N}$ the following conditions are equivalent:

1. $a \in B$,
2. $\sigma((\overline{n}, \overline{a})) = n+1$,
3. $\sigma((\overline{n}, \overline{a})) \in A$.

Therefore B is computable in the standard notation which constitutes a contradiction with our assumption.

■

Lemma 4.11. *Let $R \subseteq \mathbb{N}^k$. Then R is computable in (S, σ) if and only if $R = \emptyset$ or $R = \mathbb{N}^k$.*

Proof. The implication ($\Leftarrow$) is obvious. We shall prove ($\Rightarrow$). Suppose that $R \neq \emptyset$, $R \neq \mathbb{N}^k$ and that R is computable in (S, σ).

For any $(n_1, ..., n_k), (n'_1, ..., n'_k) \in \mathbb{N}^k$, we shall call them neighbouring elements if they differ only on one coordinate, and on this coordinate they differ only by 1, i.e. if there is $1 \leq i \leq k$ such that $n_i = n'_i + 1$ or $n'_i = n_i + 1$ and for all $1 \leq j \leq k$, if $j \neq i$, then $n_j = n'_j$.

If $R \neq \emptyset$ and $R \neq \mathbb{N}^k$, then there must obviously exist $(n_1, ..., n_k)$ and $(n'_1, ..., n'_k)$ — two neighbouring elements of $\mathbb{N}^k$ such that $(n_1, ..., n_k) \in R$ and $(n'_1, ..., n'_k) \notin R$. Without loss of generality we can assume that $n_1 = n'_1 + 1$, and that $n_j = n'_j$, for $1 < j \leq k$. Let us fix $n_2, ..., n_k$.

Let $C = \{a \in \mathbb{N} : (a, n_2, ..., n_k) \in R\}$. Since C is neither $\emptyset$, nor $\mathbb{N}$, it follows from Lemma 4.10 that C is not computable in (S, σ). Then R is not computable in (S, σ) either. Thus we have obtained a contradiction. Therefore the only relations computable in (S, σ) are $\emptyset$ and $\mathbb{N}^k$.

■

Theorem 4.12. *The only relations on natural numbers whose characteristic functions are computable in every notation are $\emptyset$ and $\mathbb{N}^k$, for $k \in \mathbb{N}$.*

5. Conclusions

It was shown in [**?**, Theorem 5] that described in [**?**, p. 592]

$$\begin{aligned} X &= \{0, \ldots, \omega + 1\}, \\ R' &= \{(p, q) : p, q \leq \omega \text{ and } p > q\}. \end{aligned}$$

Acknowledgements. The first author is funded by the National Science Centre Poland grant no. 2018/31/B/HS1/04018.

References

[1] Benacerraf, Paul, 'What Numbers Could Not Be', *Philosophical Review*, 74 (1965), 1, 47–73.

[2] Benacerraf, Paul, 'Recantation or Any Old ω-sequence Would Do After All', *Philosophia Mathematica*, 4 (1996), 2, 184–189.

[3] Chang, Chen-Chung, and Howard Jerome Keisler, *Model theory*, vol. 73 of *Studies in Logic and the Foundations of Mathematics*, North-Holland Press, Amsterdam, New York, 1973.

[4] Copeland, Brian Jack, and Diane Proudfoot, 'Deviant Encodings and Turing's Analysis of Computability', *Studies in History and Philosophy of Science Part A*, 41 (2010), 3, 247–252.

[5] Godziszewski, Michał Tomasz, and Joel David Hamkins, 'Computable Quotient Presentations of Models of Arithmetic and Set Theory', *arXiv e-prints*, (2017), arXiv:1702.08350.

[6] Gold, E. Mark, 'Limiting Recursion', *Journal of Symbolic Logic*, 30 (1965), 1, 28–48.

[7] Kalociński, Dariusz, and Michał Wrocławski, 'Successor, nontrivial functions and ordering. A study in computability and learnability in notations for natural numbers', , 2020. Submitted to Archive for Mathematical Logic.

[8] Putnam, Hilary, 'Trial and Error Predicates and the Solution to a Problem of Mostowski', *The Journal of Symbolic Logic*, 30 (1965), 1, 49–57.

[9] Quinon, Paula, 'A Taxonomy of Deviant Encodings', in Dirk Nowotka Florin Manea, Russell G. Miller, (ed.), *14th Conference on Computability in Europe, CiE 2018, Lecture Notes in Computer Science*, vol. 10936 LNCS, Springer Verlag, Kiel, 2018, pp. 338–348.

[10] Rescorla, Michael, 'Church's Thesis and the Conceptual Analysis of computability', *Notre Dame Journal of Formal Logic*, 48 (2007), 2, 253–280.

[11] Rescorla, Michael, 'Copeland and Proudfoot on Computability', *Studies in History and Philosophy of Science Part A*, 43 (2012), 1, 199–202.

[12] Rogers, Hartley, Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill series in higher mathematics, McGraw-Hill Book Company, New York, NY, USA, 1967.

[13] Shapiro, Stewart, 'Acceptable Notation', *Notre Dame Journal of Formal Logic*, 23 (1982), 1, 14–20.

[14] Shoenfield, Joseph R, 'On degrees of unsolvability', *Annals of mathematics*, 69 (1959), 644–653.

[15] Soare, Robert I., *Recursively Enumerable Sets and Degrees*, Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[16] Turing, Alan Mathison, 'On computable numbers, with an application to the Entscheidungsproblem', *J. of Math*, 58 (1936), 345-363, 5.

[17] Wrocławski, Michał, 'Representing Numbers', *Filozofia Nauki*, 26 (2018), 4, 57–73.

[18] Wrocławski, Michał, 'Representations of Natural Numbers and Computability of Various Functions', in Florin Manea, Barnaby Martin, Daniel Paulusma, and Giuseppe Primiero, (eds.), *15th Conference on Computability in Europe, CiE 2019, Lecture Notes in Computer Science*, Springer Verlag, 2019.

[19] Wrocławski, Michał, *Representations of Numbers and their Computational Properties*, Ph.D. thesis, 2019. Institute of Philosophy, University of Warsaw, Poland.

Dariusz Kalociński
Institute of Computer Science
Polish Academy of Sciences
ul. Jana Kazimierza 5
Warsaw, Poland
`dariusz.kalocinski@gmail.com`

Michał Wrocławski
Department of Philosophy
University of Warsaw
ul. Krakowskie Przedmieście 3
Warsaw, Poland
`e-mail@xx.yy.zz`