Computer Science Journal of Moldova, vol.**20**, no.3(60), 2012

Basics of Intensionalized Data: Presets, Sets, and Nominats

Mykola Nikitchenko, Alexey Chentsov

Abstract

In the paper we consider intensional aspects of the notion of data. We advocate an idea that traditional set-theoretic platform should be enhanced with new data structures having explicit intensional component. Among such data we distinguish the notions of preset and nominat. Intuitively, presets may be considered as collections of "black boxes", nominats may be considered as collections of "grey boxes" in which "white boxes" are names and "black boxes" are their values, while sets may be treated as collections of "white boxes". We describe intensions and properties of the introduced notions. We define operations over such data as functions computable in a special intensionalized sense.

Keywords: Set theory, alternative set theories, notion intension, intensionality, presets, nominats, computability, intensionalized computability.

1 Introduction

Formal methods of software development require precise specifications of the system under construction. Such specifications are usually grounded on set-theoretic platform [1]. For example, well-known B Method [2] and Z Notation [3] declare that they are based on Zermelo-Fraenkel set theory (ZF theory).

The set-theoretic platform is understandable, elaborated, and powerful formalism for describing systems and investigating their properties. Its expressive power is confirmed by the fact that main parts of mathematics can be presented in a unified form within set theory

[1]. But at the same time this power is often excessive and cumbrous. Therefore there were various attempts to restrict classical set theory or even to construct alternative set theories. These attempts were inspired both by immanent development of set theory and by its application for problem domains. Some of these proposals will be considered in section 5 devoted to related work.

Our approach for constructing modified set theory aims to support the software development process which usually starts from abstract system specification and proceeds to concrete implementation. At the abstract levels many system components are described only partially thus objects under investigation are underdetermined. In this case many conventional properties of sets may fail. In particular this concerns the extensionality principle. Recall that this principle is supported by the very first axiom of set theory – the extensionality axiom: two sets are equal if they consist of the same elements [1]. But now we can see more and more facts when a pure extensional orientation becomes restrictive for further development of computer science, artificial intelligence, knowledge bases, and other disciplines dealing with the notions of data, information, and knowledge. Therefore it seems reasonable to enhance extensional definitions of the notion of set and its derivatives (such as *data* and *function*) with *intensional* components. In a broad sense the intension of a notion means properties which specify that notion, and the extension means objects which fall under the notion, i.e. have the properties specified by the notion intension. The *intension/extension* dichotomy was studied primarily in logic, semiotics, and linguistics; we advocate more active investigations of this dichotomy in computer science too. In this paper we continue our investigations on intensionality of basic computer science notions initiated in [4]. Being oriented on computer science, we are inspired by mathematical constructivism with its emphasis on finiteness of objects and constructions. Therefore we restrict our considerations to 1) intensionalized data with finite structure, and 2) computable (in the intensionalized sense) operations over such data.

The rest of the paper is structured in the following way. In section 2 we introduce the general idea of intensionalized data and intuitively

define intensions of objects which can be considered as collections of elements. In section 3 intensions (properties and operations) of special collections called presets, sets, and nominats, are described. In section 4 formal definitions of intensions of collections that have finite structure are given. Based on these definitions, special computability of function over intensionalized data with finite structure is defined; computable functions over presets, sets, and nominates are specified. Section 5 is devoted to related work. In conclusions we summarize obtained results and discuss directions for future work.

2 Intensionalized data

Considering computer science notions in integrity of their intensional and extensional aspects we obtain new possibilities to define more first-level notions as basic notions of mathematical formalisms. Here we will focus on the notion of data trying to transform set theory to a *theory of intensionalized data*. Such data can be considered as certain objects with prescribed intensions. This idea is similar to the notion of typed data, but the latter is usually understood in the extensional sense while we aim to emphasize intensional features of data. The first steps in developing the notion of intensionalized data were made in [5, 6].

The main difficulty in constructing theories of intensionalized data is concerned with the definition of data intension. We start with intuitive understanding of intensions, and then construct their formal explications. We will move from abstract understanding of data to their more concrete representations.

At the most abstract level of consideration data are understood as some objects. Objects can be considered as *unstructured* (as *wholes* with intension I_W) or as *structured* (with *parts*, intension I_P). An object with the intension I_W can be regarded as a "black box" (intuitively it means that nothing is "visible", and therefore nothing is known about the object, intension I_{WB}) or as a "white box" (everything is "visible" and recognizable, intension I_{WW}). An intermediate intension is denoted by I_{WBW} ("black" or "white box").

To come to richer intensions we should treat objects as structured

(with intension I_P). We start with simple structures: all parts of an object are identified and fixed. In this case each part can be regarded as a whole. Relations within the object are also identified and fixed. The above specification of object structure permits to call it *hard structure*. Thus, we divide intension I_P into two subintensions I_{PH} and I_{PS} specifying objects with hard and soft structures respectively.

We continue with I_{PH} concretization caused by possible relationships between object parts. Such relationships are classified along the line *tight–loose*. Loose relationships mean that parts are not connected with each other (intension I_{PHL}); tight relations mean that parts are connected (intension I_{PHT}). In this paper we will primarily consider objects with intension I_{PHL}. In this case such objects are called *collections*; their parts are called *elements*. Empty collection is denoted in a traditional way as $\emptyset$.

Considering elements as unstructured wholes, we can treat them with intensions of "black" and/or "white boxes". Thus, three new intensions stem from this: I_{PHLB}, I_{PHLW}, and I_{PHLBW}.

Objects with intension I_{PHLB} should be regarded as collections of "black boxes". Such objects we call *presets*. Collections of "white boxes" (intension I_{PHLW}) are called *explicit multisets*; if repetition of elements is not allowed then we obtain *explicit sets*. Collections with intension I_{PHLBW} contain "black" and "white" elements (*mixed presets*).

A collection of playing cards is a good example for the introduced notions. Each playing card has two sides: the face and the back. Normally, the backs of the cards should be indistinguishable (identical). As to the faces of the cards, they may all be unique, or there can be duplicates. If all cards of a collection are placed face down on the table (are "black boxes"), then such collection is a preset. If some cards are exposed (placed face up on the table) while others are not exposed (placed face down), then we obtain a mixed preset. If all cards are exposed (are "white boxes"), then we get an explicit multiset, or a set if duplicates are not allowed.

We will make here one more concretization of intension I_{PHLB}. Under this concretization we treat each element as constructed of a

"white box" and "black box". The "white box" is considered as a name of the "black box"; thus, the "black box" is the value of this name. We call such collections *nominats* (from Latin *nomen* – name) and denote a corresponding intension as I_{ND}.

A good example of nominats is a collection of addressed envelopes. The address ("white box") written on an envelope may be considered as a name of the letter ("black box") inserted (placed) into the envelope.

Nominats are a special case of nominative data [7]. It is important to admit that nominative data can model the majority of data structures used in computer science [7, 8].

Thus, we propose to introduce additionally to the notion of set the above specified notions: presets, mixed presets, and nominats as the basic mathematical notions. These notions are enriched with intensional components and are non-extensional. Please also note that these notions are related with each other, say, sets and nominates can be treated as concretizations of presets.

To realize the idea of introducing these notions we have to describe their intensions in more detail.

3 Intensions of presets, sets, and nominats

Data intensions specify properties of corresponding data. Operations over such data should be defined in such a way that they use only those possibilities that are prescribed by the intension. In this paper we introduce the notions of "weak" operation, operation with copying, and "strong" operation. For weak operations it is allowed to construct the result of these operations using only those data components that are present in the input data; for operations with copying it is also allowed to make copies of existing components; and for strong operations it is additionally possible to generate new components. For example, the card game players are not allowed to make copies of cards or generate new cards; thus, they must use only weak operations. In computer science we also meet situations when we usually do not have possibilities to copy existing objects (say, for hardware components) or have such possibilities (say, for software components) or even have tools to

produce new objects. These situations correspond to weak operations, operations with copying, and strong operations respectively.

3.1 Preset intension

Intuitively, presets can be understood as collections of externally undistinguishable objects (elements) which have hidden content.

One more example of presets is a collection of tickets of an instant lottery. The surfaces of tickets should be covered by opaque material making them "black boxes" that hide the content of tickets.

Having this example in mind we can specify our understanding of presets by the following intuitive properties:

- each element of a preset is some whole;
- elements are separated from one another;
- elements are independent of one another, i.e., close relations between them are absent;
- all elements "are available", i.e., each element can be obtained for processing;
- exhaustive processing of all elements of a preset is possible;
- elements do not vary until it is explicitly mentioned (the law of identity of elements).

Let us admit that these properties are very weak and do not specify membership relation, so, given a preset and an element, it is not possible to say whether this element belongs to the preset. Also the equality relation is not specified. It is possible to have many hidden equal elements (duplicates) in a preset, thus, extensionality axiom is not valid. These properties of presets have a negative character restricting possibilities for processing of presets. But what operations for preset processing are available?

Analysis of the above formulated properties leads to the conclusion that the following operations are allowed for presets with the intension I_{PHLB}:

- union $\cup$ which given presets pr_1 and pr_2 yields a new preset consisting of elements of pr_1 and pr_2 ;
- nondeterministic choice *ch* which given a preset *pr* yields some element *e* of *pr*;
- nondeterministic choice with deletion *chd* which given a preset *pr* yields some element *e* of *pr* and a preset pr' without this element;
- empty function $\bar{\emptyset}$ which given a preset *pr* yields an empty preset $\emptyset$;
- cardinality operation *card* which given a preset *pr* yields the number of elements in *pr*.

The above defined operations *conform to the intension* (respect the intesion) I_{PHLB} (are *preset-conforming* operations). It means that during their execution these operations will not require additional information hidden in "black boxes" thus they use only that information which is prescribed by the intension. According to this, the intersection of presets is not available, contrary to set theory.

Still, the idea of a preset says that elements contain some hidden content; therefore operations working with this content are also required. The most natural of such operations is *open* operation. Given a preset *pr* this operation constructs a multiset *ms* which consists of "white box" elements that are content of the elements of the initial preset. We use multisets here because cardinality of *pr* and *ms* are to be the same. It means that duplicates should be preserved. The *open* operation does not conform to the intension I_{PHLB} because it opens "black boxes". Therefore, theory of presets should contain two parts: one part describes operations that conform to the intension I_{PHLB} while the other part specifies more powerful operations which can change intensions of preset elements.

3.2 Set intension

The notion of set can be considered as the "final" concretization of the notion of preset. The main new feature of sets is that their ele-

ments are considered as "white boxes", thus no hidden information is present. From this follows that elements are "recognizable" and can be compared upon distinction and equality. Therefore, to the previously formulated properties of presets (to the preset intension) we add the following new property:

- each element of a set is "recognizable" and can be checked upon distinction and equality with any other element.

Usually this property is formalized via set membership relation $\in$. From this follows that we additionally have new operations for set processing, for example, intersection and difference of sets. Still, the powerset operation will be not considered here as it should have possibility to construct copies of elements.

Set intension I_{PHLW} will also be denoted as I_{ST}. As the notion of set is well studied we will not go further into detail of set properties and operations.

3.3 Nominat intension

Intuitively, a nominat can be considered as a concretization of a preset in which each element consists of "white box" and "black box". To make this abstract consideration more concrete we should involve practical observations which permit to say that the "white box" can be considered as a *name* of the "black box"; and their relation is a *naming* (*nominative*) relation. In Slavic languages the term 'nominat' has two different meanings: a naming expression or a value of such expression; thus our treatment unites these meanings, because nominat is a unity of names and values. Nominats are also called *flat nominative data* [7].

Nominats have the dual nature: first, they may be considered as certain collections of elements; second, they may be considered as functions due to relation that connects names and their values.

Traditionally, notations of functional style are chosen to represent nominats. For example, a nominat with names $v_1, \ldots, v_n$ and values $a_1, \ldots, a_n$ respectively, is denoted by $[v_1 \mapsto a_1, \ldots, v_n \mapsto a_n]$. If values themselves are nominats, then we get the notion of *hierarchic nominats*

(*hierarchic nominative data*); for example

$$[v_1 \mapsto [u_1 \mapsto b_1, \ldots, u_k \mapsto b_k], \ldots, v_n \mapsto [t_1 \mapsto c_1, \ldots, t_m \mapsto c_m]]$$

is a 2-level nominat.

It is important to admit that nominats can model the majority of data structures used in computer science [7]. For example, a set $\{e_1, \ldots, e_m\}$ can be represented as $[1 \mapsto e_1, \ldots, 1 \mapsto e_m]$, where 1 is a standard name which has different values $e_1, \ldots, e_m$; a tuple $(e_1, \ldots, e_m)$ can be represented as $[1 \mapsto e_1, \ldots, m \mapsto e_m]$ with $1, \ldots, m$ as standard names; a sequence $\langle e_1, \ldots, e_m \rangle$ can be represented as $[1 \mapsto e_1, 2 \mapsto [\ldots, 2 \mapsto [1 \mapsto e_m, 2 \mapsto \emptyset] \ldots]]$, where 1, 2 are standard names.

The main new operations over nominats are the following:

- *naming* $\Rightarrow v$ (with name $v \in V$ as a parameter) which given a value a yields a nominat $[v \mapsto a]$;
- *denaming* $v\Rightarrow$ (partial multivalued operation with name $v \in V$ as a parameter) which given a nominat d yields a value of v in d if it exists;
- *checking* $v!$ (with name $v \in V$ as a parameter) which given a nominat d yields d if the value of v exists in d; or yields $\emptyset$ if such a value does not exist;
- *overriding* ∇ which given two nominats d_1 and d_2 yields a new nominat d consisting of named values of d_2 and those of d_1, the names of which do not occur in d_2.

These operations *conform to the intension* I_{ND} (are *nominat-conforming* operations). Thus, these operations are allowed for nominats processing.

Now we will describe briefly the distinctions between the notions of set and nominat as mathematical primitives. To do this, various criteria can be used. First, nominats, contrary to sets, have hidden content. This permits to make their further concretizations not possible

for sets. Second, nominats have functional "spirit" of naming relation simplifying nominat processing. We will illustrate this statement by the following observations. We start with the notion of ordered pair (a, b) that can be defined as nominat $[1 \mapsto a, 2 \mapsto b]$ where 1 and 2 are standard names. The notion of ordered pair in set theory has many definitions:

- $(a, b) = \{\{\{a\}, \emptyset\}, \{\{b\}\}\}$ – Norbert Wiener, 1914;
- $(a, b) = \{\{a, 1\}, \{b, 2\}\}$ – Felix Hausdorff, 1914 (1 and 2 are two distinct objects different from a and b);
- $(a, b) = \{\{a\}, \{a, b\}\}$ – Kuratowski, 1921;
- etc.

It seems that these definitions do not look fully adequate to the intuitive notion of ordered pair, because they require detailed analysis of bracket structure (Wiener's definition), or are restrictive (Hausdorff's definition), or collapse to singleton $\{\{a\}\}$ when $a = b$ (Kuratovski's definition). It is interesting to admit that in *Principia Mathematica* the notion of ordered pair was considered as primitive, and even N. Bourbaki took the same position. So, introduction of special primitives like ordered pairs (and nominats in our case) is not a new idea.

Concerning further relationships of ordered pairs and tuples with nominats, we would like to emphasize that nominats are more adequate to computer science practice than tuples. To make this claim more understandable, let us consider questions of operating with tuples and nominats. Indeed, given two tuples $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_n)$ we can combine them practically only as concatenation $(a_1, \ldots, a_m, b_1, \ldots, b_n)$ or $(b_1, \ldots, b_n, a_1, \ldots, a_m)$. But concatenation is a coarse operation that ignores possible coincidence of some values from $\{a_1, \ldots, a_m, b_1, \ldots, b_n\}$ representing the same attributes. Thus, we are forced to make finer combinations of $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_n)$ manually that complicates processing of such data. Instead of this data structure (tuples) we propose to consider nominats. In this case we have more natural combining operations, for example, given nominats $[x \mapsto 7, y \mapsto 5, z \mapsto 8]$ and

$[t \mapsto 7, u \mapsto 5, x \mapsto 8]$ we obtain $[y \mapsto 5, z \mapsto 8, t \mapsto 7, u \mapsto 5, x \mapsto 8]$ as their overriding combination (cf. with combination of tuples $(7, 5, 8)$ and $(7, 5, 8)$). Also, other combining operations can be defined. This richness of combining operations simplifies processing of nominats compared with tuples. The reason of this is that the abstraction level of "position" in a tuple is lower than that of "name" in a nominat since position depends more strongly on other positions than a name depends upon other names. Thus, operating with names (with nominats) is more "soft" with respect to data transformations. The above considerations shortly argue in favour of using nominats as one more basic data structure in computer science.

Properties of intensionalized data and operations over them were discussed in this section informally. To make the proposed approach more precise we need formal definitions of these notions.

4 Formal definitions of intensionalized data

To give formal definitions of intensionalized data we will use reduction methods. Roughly speaking it means that given data class D with intension I_D, we construct a reduction procedure to some data class D' that has an understandable and well studied intension. Also, operations over D will be reduced to operations over D'. In our case we will use several reduction steps.

Still, this idea is difficult to be realized if no restrictions are imposed on intension I_D. Taking into consideration that computer science is the intended application domain for intensionalized data, we restrict ourselves to data having finite structures (intension I_{PHF}) and to operations that are computable in a special intensionalized sense. Note that this intension is subintension of I_{PH}; thus, data with intension I_{PHF} can have loose relations between their components (intension I_{PHL}), or can have tight relations (intension I_{PHT}), for example, in finite lists their components are tightly related.

In the sequel we will use the following notations for classes of functions from D to D':

- $D \xrightarrow{p} D'$ – the class of partial single-valued functions;
- $D \xrightarrow{b} D'$ – the class of total single-valued bijective functions;
- $D \xrightarrow{m} D'$ – the class of partial multi-valued functions. Function f is *multi-valued* (non-deterministic) if being applied to the same input data d it can yield different results during different applications to d (and possibly be also undefined);
- $D \xrightarrow{t} D'$ – the class of total functions. Function f is total if the value f on d is always defined;
- $D \xrightarrow{i} D'$ – the class of injective functions. A multi-valued function is *injective*, if it yields different values on different arguments. The inverse of injective function is a single-valued function;
- $D \xrightarrow{\nu} D'$ – the class of total multi-valued injective functions.

4.1 Intensionalized data with finite structures

Let D be a class of data with intension I_D. Assume that we treat data of D as finite structured data. Our intuitive understanding of such a data is the following: any such data d consists of several basic (atomic) components $b_1, \ldots, b_m$, organised (connected) in a certain way. If there are enumerably many different forms of organisation, each of these data can be represented in the (possibly non-unique) form $(k, \langle b_1, \ldots, b_m \rangle)$, where k is the *data code* and the sequence $\langle b_1, \ldots, b_m \rangle$ is the *data base*. Data of this form are called *natural data* [9]. More precisely, if B is any class and Nat is the set of natural numbers, then the class of *natural data* over B is the class $Nat(B) = Nat \times B^*$. An implicit assumption is that the code represents 1) all information that can be "extracted" from those elements of B which are contained in d, and 2) interrelations between such elements. (This will be discussed in more detail in the next subsection.) These properties specify a fixed intension of natural data: they have the form $(k, \langle b_1, \ldots, b_m \rangle)$ where k is a natural number and $\langle b_1, \ldots, b_m \rangle$ is a list of elements treated as "black boxes". As finite

structured data can have different representations, we should use total multi-valued injective functions for constructing such representations.

Note that we use the term 'class' for collections of intensionalized data; term 'set' is used for collections, the intensions of which are subintensions of sets.

Now we are ready to give the formal definition of a class of intensionalized data with some intension I_D which is a subintension of I_{PHF}. A class D is called a class *of finite structured data*, if a class B and a total multi-valued injective mapping $nat\colon D \xrightarrow{\nu} Nat(B)$ are given. This mapping *nat* is called the *naturalization* mapping. Naturalization mapping is actually an *analysing* mapping: it finds in a data d its components and their interrelations according to the properties of data prescribed by its intension. Dually to *nat* we introduce *denaturalization* mapping *denat* which reconstructs (*synthesizes*) data of class D from natural data. For simplicity's sake we assume that $denat = nat^{-1}$. Denaturalization mapping is a partial single-valued mapping. Naturalization and denaturalization mapping are also called *concretization* and *abstraction* mappings respectively.

Example 1 (naturalization mapping for a class B of basic elements)**.** As nothing is known about elements of B, we treat such elements as "black boxes"; therefore B is a preset with intension I_{PHLB}. Thus, we define $nat_B\colon B \xrightarrow{t} Nat(B)$ to be such mapping that $nat_B(b) = (0, \langle b \rangle)$ for any b of B. It means that nothing is known about b (its code is 0) and b has no parts except itself (its base is $\langle b \rangle$).

Example 2 (naturalization mapping for the set Nat of natural numbers)**.** These numbers are treated as "white boxes" without parts. Thus, we define $nat_{Nat}\colon Nat \xrightarrow{t} Nat(B)$ to be such mapping that $nat_{Nat}(n) = (n, \langle\rangle)$ for any n of Nat. It means that n is known (its code is n) and n has no parts (its base is empty sequence $\langle\rangle$).

Example 3 (naturalization mapping for an enumerated set S)**.** The set S is considered as enumerated set (has the intension of enumerated set) if a bijective mapping $u\colon Nat \xrightarrow{b} S$ is given. In this case we define

$nat_S\colon S \xrightarrow{t} Nat(B)$ to be such mapping that $nat_S(e) = (u^{-1}(e), \langle\rangle)$ for any e of S.

Example 4 (naturalization mapping for the class B^* of finite sequences over preset B)**.** For any element $\langle b_1, \ldots, b_n\rangle$ of B^* we know its structure (which is a list of length n), but we know nothing about elements of B. Thus, we define $nat_{B^*}\colon B^* \xrightarrow{t} Nat(B)$ to be such mapping that $nat_{B^*}(\langle b_1, \ldots, b_n\rangle) = (n, \langle b_1, \ldots, b_n\rangle)$.

The above given definitions may be considered as a special formal definition of intension I_D: given a finite structured data class D a pair (B, nat) is called *naturalized intension* of D; a tuple $(D, (B, nat))$ is called a *naturalized class of intensionalized data*.

Still, these definitions which reduce intuitive understanding of data of D to $Nat(B)$ lack precise description of their intensions because we did not define operations over D and over $Nat(B)$; in other words, we do not have complete description of the intensions of these classes. As mentioned earlier, we are oriented on mathematical constructivism, thus, we will treat operations over D and over $Nat(B)$ as computable in a special sense. Computabillity considered here is called weak natural computability.

4.2 Weak natural computability over intensionalized data

To formalize operations that conform to data intensions we will use a special computability called *intensionalized* computability. This computability will be reduced in several steps to traditional computability of n-ary functions defined on integers or strings. Traditional computability may be called Turing computability. In the light of our investigations traditional computability does not pay much attention to the variety of data intensions, because it concentrates on computability over integers (or strings) which have fixed intensions.

The idea behind intensionalized computability is the following: for data processing it is allowed to use only those operations that conform

to their intensions. Thus, intensionalized computability is *intensionally restricted computability*. In fact, such computability is a *relative computability* – relative to data intensions.

Defining this computability we follow [5] with several modifications: 1) we define computability for functions of the type $D \xrightarrow{m} D'$ instead of $D \xrightarrow{m} D$, 2) we consider weak computability (without copying) instead of computability with copying.

Introduction of naturalization mapping is a crucial moment for defining intensionalized computability. This mapping is regarded as a formalization of data intension; and this enables us to explicate an intuitive notion of intensionalized computability over D with intension I_D via formally defined *weak natural computability* over D. The latter is then reduced to a new special computability over $Nat(B)$ called *weak code computability*. To define this type of computability we should recall that natural data has a fixed intension under which the code collects all known information about data components and their interrelations, and the base is treated as a list of "black boxes". Thus, weak code computability should be independent of any specific manipulation (processing) operations of the elements of B and can use only information that is explicitly exposed in the natural data. The only explicit information is the data code and the length of the data base. Therefore in code computability the data code plays a major role, while the elements of the data base virtually do not affect the computations. These elements may be only used to form the base of the resulting data. To describe the code of the resulting data and the order in which elements of the initial base are put into the base of resulting data, a special function of type $Nat^2 \xrightarrow{m} Nat \times Nat^*$ should be defined. Such a function is called *weak index-computable*. These considerations lead to the following definition.

A multi-valued function $g\colon Nat(B) \xrightarrow{m} Nat(B)$ is called weak code-computable if there exists a weak index-computable multi-valued function $h\colon Nat^2 \xrightarrow{m} Nat \times Nat^*$ such that for any k, m from Nat, $b_1, \ldots, b_m$ from B, $m \geq 0$, we have $g(k, \langle b_1, \ldots, b_m \rangle) = (k', \langle b_{i_1}, ..., b_{i_l} \rangle)$ if and only if $h(k, m) = (k', \langle i_1, \ldots, i_l \rangle)$, $1 \leq i_1 \leq m$, $\ldots$, $1 \leq i_l \leq m$, $l \geq 0$, and all indexes $i_1, \ldots, i_l$ are distinct. If one of the indexes

$i_1, \ldots, i_l$ lies outside the interval $[1, m]$, or there are equal indexes in the sequence $i_1, \ldots, i_l$, or $h(k, m)$ is undefined, then $g(k, \langle b_1, \ldots, b_m \rangle)$ is also undefined.

In other words, in order to compute g on $(k, \langle b_1, \ldots, b_m \rangle)$, we have to compute h on (k, m), generate a certain value $(k', \langle i_1, \ldots, i_l \rangle)$, and then try to form the value of the function g by selecting the components of the sequence $\langle b_1, \ldots, b_m \rangle$ pointed to by the indexes $i_1, \ldots, i_l$.

This definition actually completes our formalization of the natural data intension because it specifies operations over natural data as weak code-computable.

Note that weak computability defined here differs from the computability with copying defined in [4, 5] by the requirement that all evaluated indexes should be distinct.

It is clear that index computability of $h\colon Nat^2 \xrightarrow{m} Nat \times Nat^*$ may be reduced by traditional methods of recursion theory to conventional computability of a certain function $r\colon Nat \xrightarrow{m} Nat$.

We are ready now to give the formal definition of a weak natural computable function.

Let $(D, (B, nat))$ and $(D', (B, nat'))$ be naturalized classes of intensionalized data (w.l.o.g. we treat these classes as based on one class B). A function $f\colon D \xrightarrow{m} D'$ is called *weak natural computable* (with respect to naturalized intensions (B, nat) and (B, nat')) if there is a weak code-computable function $g\colon Nat(B) \xrightarrow{m} Nat(B)$ such that $f = denat' \circ g \circ nat$.

This definition completes our formalization of the data intension of the class D because it gives possibility to formalize operations over D as weak natural computable.

Thus, intensionalized computability has been defined via a sequence of the following reductions: intensionalized computability – weak natural computability – weak code computability – weak index computability – partial recursive computability. Analysing the definitions we can also conclude that weak natural computability is a generalization (relativization) of enumeration computability. In fact, for $B = \emptyset$ weak code computability is reduced to partial recursive computability on Nat, and weak natural computability is reduced to enumeration com-

putability [10]. Therefore, the notions of weak code and weak natural computability defined above are quite rich.

In the sequel weak natural computability will also be denoted as wn-computability.

Example 5 (wn-computability over preset B)**.** The naturalization mapping nat_B was defined in Example 1. To define the complete class of wn-computable functions over $(B, (B, nat_B))$ of type $B \xrightarrow{m} B$, we have to describe all weak index-computable function of the type $h\colon Nat^2 \xrightarrow{m} Nat \times Nat^*$. It is easy to understand that under the naturalization mapping nat_B we need to know the results of weak index-computable function only on the element $(0, 1)$. On this input data a weak index-computable function can 1) yield $(0, \langle 1 \rangle)$, 2) yield a value distinct from $(0, \langle 1 \rangle)$, or 3) be undefined. For cases 2) and 3) the denaturalization mapping will be undefined. This induces the following functions of type $B \xrightarrow{m} B$: 1) the identity function *id*, 2) the everywhere undefined function *und*, and 3) the multi-valued (non-deterministic) function *und-id* such that *und-id*(d) is equal to d or is undefined. Actually it means that the following result was proved: *the complete class of weak natural computable partial multi-valued functions over preset B consists of functions und, id, and und-id.* In other words, the three functions defined above are the only computable functions over "black box" intensionalized data.

Example 6 (wn-computability over the set Nat of natural numbers)**.** The naturalization mapping nat_{Nat} was defined in Example 2. Under this naturalization we are interested in weak index-computable functions defined on the sets of elements of the form $(n, 0)$. This set is isomorphic to Nat. Thus (as expected), the set of all wn-computable functions over Nat is exactly the set of all partial recursive functions.

Example 7 (wn-computability over the enumerated set S)**.** The naturalization mapping nat_S was defined in Example 3. Under this naturalization we are again interested in weak index-computable functions defined on the sets of elements of the form $(n, 0)$, $n \in Nat$. This set is isomorphic to Nat. Thus (as expected), the wn-computability over S coincides with the enumeration computability over S [10].

Example 8 (wn-computability over the class B^*)**.** The naturalization mapping nat_{B^*} was defined in Example 4. Under this naturalization we are again interested in weak-index computable functions defined on the sets of elements of the form (n, n) with results of the form $(k, \langle i_1, \cdots, i_k\rangle)$, where $k \in Nat$, $k \leq n$, $1 \leq i_1 \leq n$, ..., $1 \leq i_k \leq n$, and all indexes $i_1, \cdots, i_k$ are distinct. One among such functions is a function h_{tail} such that $h_{tail}(n, n) = (n-1, \langle 2, \ldots, n\rangle)$. It means that *tail* operation (such that $tail(\langle b_1, \ldots, b_n\rangle) = \langle b_2, \ldots, b_n\rangle$, $n > 0$) is wn-computable. Note that doubling operation $doubl(\langle b_1, \ldots, b_n\rangle) = (\langle b_1, \ldots, b_n, b_1, \ldots, b_n\rangle)$ is not wn-computable.

Having defined the notion of natural computability, we can now check whether operations over intensionalized data (presets, sets, and nominats) intuitively defined in the previous section indeed conform to the corresponding intensions.

As domains and ranges of operations can be constructed with the help of Cartesian product, now we will give definition of the intension for such a product. Let $(D_1, (nat_1, B))$, ..., $(D_n, (nat_n, B))$ be naturalized classes of intensionalized data. Then naturalization mapping $nat_{D_1\times\cdots\times D_n}\colon D_1 \times \cdots \times D_n \xrightarrow{\nu} Nat(B)$ is defined as follows (d_1 is of D_1, ..., d_n is of D_n):

$$\begin{aligned} nat_{D_1\times\cdots\times D_n}(d_1, \ldots, d_n) = \\ (c(n, c(c(k_1, l_1), c(\ldots c(c(k_{n-1}, l_{n-1}), c(k_n, l_n))\ldots)))), \\ \langle b_{11}, \ldots, b_{1l_1}, \ldots, b_{n1}, \ldots, b_{nl_n}\rangle), \end{aligned}$$

where $nat_j(d_j) = (k_j, \langle b_{j1}, \ldots, b_{jl_j}\rangle)$, $1 \leq j \leq n$; c is a pairing function that uniquely encodes two natural numbers into a single natural number, say, the Cantor pairing function.

The idea behind this definition is simple: given a tuple $(d_1, \ldots, d_n)$ we first find naturalizations $nat_j(d_j) = (k_j, \langle b_{j1}, \ldots, b_{jl_j}\rangle)$, then construct the code of the resulting natural data by encoding codes and lengths of tuple components, and at last we construct the base by concatenating components' bases.

4.3 Computability of preset operations

First, we should define naturalization mapping for presets. Let B be a class of elements and $PreF(B)$ be a class of finite presets with elements of B. Naturalization mapping $nat_{PS}\colon PreF(B) \xrightarrow{\nu} Nat(B)$ is defined as follows: given a preset pr with elements $e_1, \ldots, e_n$ function nat_{PS} on pr can yield any natural data of the form $(n, \langle e_{i_1}, \ldots, e_{i_n}\rangle)$, where $e_{i_1}, \ldots, e_{i_n}$ is a permutation of $e_1, \ldots, e_n$.

Example 9 (wn-computability of choice function $ch\colon PreF(B) \xrightarrow{m} B$)**.** The naturalization mapping nat_{PS} was defined in this section and the mapping nat_B was defined in Example 1. For choice operation ch a weak index-computable multi-valued function $h_{ch} : Nat^2 \xrightarrow{m} Nat \times Nat^*$ is defined by the formula: $h_{ch}(n, n) = (0, \langle i\rangle)$, where $1 \leq i \leq n$. This function is obviously Turing computable; therefore ch is wn-computable.

Example 10 (wn-computability of union $\cup\colon PreF(B)^2 \xrightarrow{t} PreF(B)$)**.** Let preset pr_1 of $PreF(B)$ consists of elements $b_1, \ldots, b_n$ and preset pr_2 of $PreF(B)$ consists of elements $e_1, \ldots, e_m$; $nat_{PS}(pr_1) = (n, \langle b_{i_1}, \ldots, b_{i_n}\rangle)$ and $nat_{PS}(pr_2) = (m, \langle e_{j_1}, \ldots, e_{j_m}\rangle)$, where b_{i_1}, $\ldots$, b_{i_n} and e_{j_1}, $\ldots$, e_{j_m} are permutations of b_1, $\ldots$, b_n and $e_1, \ldots, e_m$ respectively. According to the definition of naturalization of Cartesian product, we obtain the following natural data for the pair (pr_1, pr_2): $(c(2, c(c(n, n), c(m, m))), \langle b_{i_1}, \ldots, b_{i_n}, e_{j_1}, \ldots, e_{j_m}\rangle)$. Index-computable function $h_\cup$ such that

$$h_\cup(c(2, c(c(n, n), c(m, m))), n + m) = (n + m, \langle 1, 2, \ldots, n + m\rangle)$$

is partial recursive and determines wn-computable binary function. It is clear that the result does not depend on permutations of b_1, $\ldots$, b_n and $e_1, \ldots, e_m$, thus obtained union function is single-valued.

In the same way we can prove that other operations over presets defined in section 3 are computable. Thus, the following statement is valid.

Proposition 1. *The following operations over presets: union* $\cup$*, choice ch, nondeterministic choice with deletion chd, empty function* $\bar{\emptyset}$*, and cardinality card, are weak natural computable.*

It means that these operations conform to preset intension.

Actually, using such techniques we can formally describe all preset-conforming operations of different types. For example, any preset-conforming operation *op* of type $PreF(B) \xrightarrow{m} Nat$ can be represented as a composition of a certain multi-valued partial recursive function $na\colon Nat \xrightarrow{m} Nat$ and a cardinality operation *card*, thus, $op = na \circ card$.

We can also prove that some operations, say, intersection of two presets, are not preset-conforming operations.

4.4 Computability of set operations

Set intensions assume that elements of sets are "white boxes". The naturalization approach requires that such elements can be encoded. It means that we should consider B as an enumerated set $B = \{b_0, b_1, \dots\}$. Thus, bijective enumeration function $u : Nat \xrightarrow{b} B$ is given. Let $SetF(B)$ be a class of finite sets with elements of enumerated set B.

We can define two naturalization mappings: weak and strong.

The weak naturalization mapping $nat_{SFW} : SetF(B) \xrightarrow{\nu} Nat(B)$ is defined as follows: given a set s with elements $e_1, \dots, e_n$, mapping nat_{SF} on s, can yield any natural data of the form $(k, \langle e_{i_1}, \dots, e_{i_n}\rangle)$, where $k = c(n, c(k_1, \dots c(k_n, 0)\dots))$, $u(k_1) = e_{i_1}, \dots, u(k_n) = e_{i_n}$; and $e_{i_1}, \dots, e_{i_n}$ is a permutation of $e_1, \dots, e_n$.

The idea behind this definition is very simple: we encode the cardinality of s and numbers of its elements according to the enumeration function; as to the base we include in it all elements of s. (The definition may be simpler if we take into account ordering of elements induced by enumeration function, cf. with definitions in the next subsection.)

The strong naturalization mapping $nat_{SFS}\colon SetF(B) \xrightarrow{\nu} Nat(B)$ is defined as follows: given a set s with elements $e_1, \dots, e_n$, mapping nat_{SFS} on s, can yield any natural data of the form $(k, \langle\rangle)$, where

$k = c(n, c(k_1, \dots c(k_n, 0) \dots))$, $u(k_1) = e_{i_1}, \dots, u(k_n) = e_{i_n}$; and $e_{i_1}, \dots, e_{i_n}$ is a permutation of $e_1, \dots, e_n$. The base of the obtained natural data is empty.

The difference between these naturalization concerns the possibility of producing new elements. In the first case this is not allowed because code-computable functions construct the base of result using only the base of initial natural data. In the second case we can evaluate any code and then (using denaturalization mapping) produce any elements of S.

These naturalization mappings define different intensions of the class $SetF(B)$.

Example 11 (wn-computability of intersection $\cap$: $SetF(B)^2 \xrightarrow{t} SetF(B)$)**.** Let set s_1 of $SetF(B)$ consists of elements $b_1, \dots, b_n$ and set s_2 consists of elements $e_1, \dots, e_m$; $nat_{SFW}(s_1) = (q_1, \langle b_{i_1}, \dots, b_{i_n} \rangle)$ and $nat_{SFW}(s_2) = (q_2, \langle e_{j_1}, \dots, e_{j_m} \rangle)$, where $q_1 = c(n, c(k_1, \dots c(k_n, 0) \dots))$, $u(k_1) = b_{i_1}, \dots, u(k_n) = b_{i_n}$; $q_2 = c(m, c(r_1, \dots c(r_m, 0) \dots))$, $u(r_1) = e_{j_1}, \dots, u(r_m) = e_{j_m}$; $b_{i_1}, \dots, b_{i_n}$ and $e_{j_1}, \dots, e_{j_m}$ are permutations of $b_1, \dots, b_n$ and $e_1, \dots, e_m$ respectively. According to the definition of naturalization of Cartesian product, we obtain the following natural data for the pair (s_1, s_2):

$$(c(2, c(c(q_1, n), c(q_2, m))), \langle b_{i_1}, \dots, b_{i_n}, e_{j_1}, \dots, e_{j_m} \rangle).$$

How to define an index-computable function $h_\cap$? The following algorithm can be proposed. First, the code $c(2, c(c(q_1, n), c(q_2, m)))$ should be analyzed and all pairs (k_i, r_j) such that $k_i = r_j$ should be identified. Then a list of their positions (say, in s_1) should be formed (this list is a list of indexes in the result of index-computable function). At last, a code of the result should be evaluated; in this code we include the numbers (under naturalization mapping) of the elements of the intersections and its cardinality. Defined function is partial recursive, the results do not depend on permutations of the elements of the initial sets. So, intersection is wn-computable.

In the same way we can prove that conventional operations over sets are computable. Thus, the following statement is valid.

Proposition 2. *The following operations over sets: union $\cup$, intersection $\cap$, difference $\setminus$, choice ch, nondeterministic choice with deletion chd, empty function $\bar{\emptyset}$, and cardinality card are weak natural computable.*

So, we have proved that these operations conform to set intension. But some operations over sets, say powerset operation, are not wn-computable. Still, this operation is natural computable with copying.

4.5 Computability of nominat operations

Nominat intensions assume that elements of a nominat are constructed of names ("white boxes") and their values ("black boxes"). Thus, naturalization mapping of nominats is constructed of naturalizations for names and values. We assume that the set of names $V = \{v_0, v_1, \dots\}$ is enumerated by bijective enumeration function u (see the previous subsection). It is also reasonable to choose a strong naturalization mapping nat_{SFS} because normally any name can be generated. As to values, we assume that they are elements of a preset (with intension I_{PHLB}).

Let $NomF(V, B)$ be a class of finite nominats constructed over V and B and $nm = [v_1 \mapsto b_1, \dots, v_n \mapsto b_n]$ be a nominat of this class. W.l.o.g. we can assume that names are ordered according to their numbers with respect to enumeration mapping, that is $u^{-1}(v_1) < u^{-1}(v_2) < \dots < u^{-1}(v_n)$. Under this assumption weak (with respect to B) naturalization mapping $nat_{NMW} \colon NomF(V, B) \xrightarrow{t} Nat(B)$ is defined as follows: given a nominat $nm = [v_1 \mapsto b_1, \dots, v_n \mapsto b_n]$ mapping nat_{NMW} on nm yields natural data of the form $(k, \langle b_1, \dots, b_n\rangle)$, where $k = c(n, c(k_1, \dots c(k_n, 0) \dots))$, $u(k_1) = v_1, \dots, u(k_n) = v_n$.

Example 12 (wn-computability of denaming $v\Rightarrow \colon NomF(V, B) \xrightarrow{m} B$)**.** The naturalization nat_{NMW} has been defined just now and the mapping nat_B was defined in Example 1. For denaming function $v \Rightarrow$ a weak index-computable multi-valued function $h_{v\Rightarrow} \colon Nat^2 \xrightarrow{m} Nat \times Nat^*$ is defined by the formula: $h_{v\Rightarrow}(c(n, c(k_1, \dots c(k_n, 0) \dots)), n) = (0, \langle k_i\rangle)$, where $1 \le i \le n$, $u(k_i) = v$; in other cases the value is

undefined. This function is obviously Turing computable; therefore $v\Rightarrow$ is wn-computable.

In the same way we can prove that other operations over nominats defined in the previous section are computable. Thus, the following statement is valid.

Proposition 3. *The following operations over nominats: naming $\Rightarrow v$, denaming $v\Rightarrow$, checking $v!$, and overriding ∇ are weak natural computable.*

As to computability with copying, in [4, 9] several theorems were proved that may be considered as descriptions of complete classes of natural computable (with copying) functions over various kinds of intensionalized data, and hierarchic nominats, in particular.

Summing up, we can say that proposed naturalization approach permits to define preset-, set-, and nominat-conforming operations (for finite collections), thus giving possibility for further development of the theory of intensionalized data.

5 Related work

The notion of data, being one the main notion of computer science, has many aspects, definitions, and explications. The analysis of such diversity of data concepts is worth a special investigation the authors plan to fulfill in forthcoming papers. In this paper, oriented on enhancement of the notion of set, we will consider only those works that are related to set theory variations.

Set theory, being a primary foundation for mathematical research, has been debated for decades. Paradoxes, controversies and inconsistencies with mathematical practice in some areas have led to multiplicity of set theories as well as rise of quite uncommon alternative theories. The approaches used by different "schools of thoughts" can be classified by many criteria like extensionality, kind of logic employed, intensionality, finiteness, well-foundedness, characteristics of membership relation, predicativity, incompleteness of knowledge, information

hiding, etc. Most "radical" departures from standard set theory concern base logic. Less radical ones modify or reject some principle of ZFC through system of axioms or more informally.

We start with variations caused by set theory paradoxes. If U is a set-theoretic universe then it should satisfy equation that can be stated in the abstract form as $\mathcal{P}^*(U) \simeq U$. In order to avoid paradoxes, $\mathcal{P}^*(U)$ cannot be powerset of U but rather collection of some distinguished subsets of U. The solution of this equation $\mathcal{U} = \langle U, f\rangle$, where $f\colon U \simeq \mathcal{P}^*(U)$, is called Frege structure. It determines abstract set-theoretic universe where membership relation is interpreted as follows: $u \in_{\mathcal{U}} v$ iff $u \in f(v)$.

Conventional remedy to paradoxes was in limitation of the cardinality of the sets. This limitation was quite restrictive turning ZF theory (with axiom of foundation) into the theory of small and iterative sets [11]. Some alternative theories do not reject ZF completely but rather look for extensions of ZF that avoid paradoxes by other means than limitation of size.

In [12] class of subsets $\mathcal{P}^*(U)$ is selected from topological considerations to be either open or closed subsets of topological space U. Moreover bijection in this case can be required to be homeomorphism.

A few alternatives (in order to avoid Russell's paradox) are based on modification of the concept of (co-)extension. Formalizing notion of 'partial information' in [13] a concept of partial set was proposed. Though partial set extension and coextension are disjoint, they do not necessarily cover the universe. The theory of partial sets introduces new primitive operators $\notin$, $\neq$. Construction of sets and abstraction axioms are allowed only for formulas without negations – positive formulas. Extensionality principle cannot be used to identify partial sets (it is possible to express positively negative properties). Intensionality can be used instead implying some sort of set naming and pure term models [14].

Positive sets can be seen as simplification of partial sets (though have their own motivation) [15]. In this case operators $\notin$, $\neq$ and abstractors are dropped while extensionality is restored. This theory has models known as 'hyperuniverses' constructed using topological set-

theoretic structures [16]. [17, 18] studied the first-order generalization of positive sets theory known as GPK^+_∞. In this theory additionally axiom of infinity and existence of least set that contains "extension" for given (arbitrary) formula (closure principle) are postulated. This theory disproves axiom of choice and class of its hereditary well-founded sets interprets ZF. Some peculiar constructions are possible in GPK^+ models like self containing singleton (auto-singleton) [19].

Paradoxical set theory is another consistent theory without extensionality. It is dual to theory of partial sets. In it set extension and coextension are not necessarily disjoint but cover the universe [20]. Analogously set theory HF (Hyper-Frege) is counterpart to the GPK^+ [21]. Its models are built on the same bases as GPK. Stronger theory HF_∞ (with axiom of infinity) is capable of interpreting ZF.

In double extension set theory to avoid classical paradoxes the concept of extension was bifurcated [22]. There are two membership relations $\in$, $\in'$. Extensionality axiom for this theory is formulated as follows: $\forall z(z \in x \leftrightarrow z \in' y) \rightarrow x = y$. Some analog of infinite ordinal is possible to construct in this theory without explicitly stating axiom of infinity. Also it is possible to interpret ZF in some form in the theory [23]. Serious shortcoming of this theory is lack of proof of its consistency.

Rough set theory [24] presumes incomplete knowledge which is formalized using equivalence relation of indiscernibility. Based on this approach, [11] proposed generalized Proximal Frege Structures which are universes of sets with additional modal operators. This gives prospects for axiomatic modal set theory.

Another line of research is related to category theory. Category theory emphasizes external properties of objects. Concept of morphism or function is abstract and primitive in category theory and is not reduced to sets. Typically objects of a category are instances of the structure of certain kind, and morphisms are structure-preserving functions [25, 26]. Structure of objects and properties of morphisms are described in terms of other objects and morphisms only.

Sets together with functions between sets form a category. It is possible to give purely category-theoretic characterization to this cat-

egory which leads to a concept of elementary topos. Toposes can be provided with internal language which is very similar to that of set theory and can be interpreted inside the topos in category-theoretic terms [27]. Thus topos may be regarded as a mathematical domain of discourse or "world" in which mathematical concepts can be interpreted and mathematical constructions performed [28]. This idea was further developed in local set theory [28].

Frege structure can be considered in categorical framework. In Heyting categories (some generalization of toposes) it is possible to introduce the notion of "smallness" defining sets. If such category has a powerclass functor of subsets then its free algebras are models of set theory [29, 30]. Membership relation in these models is determined algebraically. Field, known as algebraic set theory, researches some aspects of set theory through these models. Primarily intuitionistic ZF theory is targeted. But models of other set theories can be constructed by the same algebraic method simply varying particular category and notion of "smallness".

As a contrary Lawere advocates that set theory should not be based on membership but rather on isomorphism-invariant structures. He proposed an Elementary Theory of the Category of Sets (ETCS) for this purpose [31, 32]. Objects of ETCS are abstract sets. In short, abstract set is an assemblage of featureless but distinct "dots". From technical standpoint ETCS is non-degenerate well-pointed topos with natural numbers object for which axiom of choice holds. It is argued that strong case can be made for ETCS logical and conceptual autonomy [33].

Martin-Löf type theory emphasizes constructivity [34]. It follows Curry-Howard correspondence to represent propositions as sets thus interpreting predicate logic. Sets also can be seen as problem descriptions. The equality between sets is intensional which means it is definitional or syntactical. Theory has formal language that is used as programming language, specification language and programming logic. Axiom of choice is provable in Martin-Löf type theory [35] while in constructive or intuitionistic set theory it implies the law of excluded middle.

The admissible set theory [36] aims to present a weaker axiomatic system more adequate for processing of finite domains. Additionally this theory includes basic elements (praelements).

Now we would like to say a few words about the term 'preset'. Probably Bishop [37] was the first who introduced this term. Toby Bartels explains that for Bishop a preset is like a set without an equality relation; conversely, a set is a preset equipped with an equality relation. This understanding stems from Bishop's three steps definition of a set: you should first state how to construct an element of the set; then you should describe how to prove that two elements are equal; and at last you should prove that this (equality) relation is reflexive, symmetric, and transitive. If you only do the first step, then you don't have a set, according to Bishop; you only have a preset. A given preset may define many different sets, depending on the equality relation. From this follows that a membership relation is defined for Bishop's presets, but extensionality axiom fails. Thus, our understanding of presets is weaker and different from Bishop's treatment.

Such numerous examples (of course, not exhaustive) of set theory variations give good evidence that many scientists are aware of restrictedness of traditional set theory. We argue for intensional approach to constructing set theory variants. We also emphasize constructiveness of such variants through explicit computability aspects.

Summing up, we would like to admit that the proposed notions of preset and nominat differ from the conventional notion of set in several aspects: from the one side, theories of presets and nominats are weaker than conventional set theory, in particular, extensionality fails, also membership relation and equality are not definable; but on the other hand, these notions seem to be more adequate to computer science domain because operations are defined as computable in a special intensionalized sense, presets and nominats are constructed over basic elements (praelements) which may have hidden content, from this stems a possibility to change levels of abstraction of data consideration (up to non-wellfoundedness). Still, investigation on the topic should be continued in order to establish more precise relations between theories under investigations.

6 Conclusions

Set theory is the main formal system that is used for construction of problem domain models. Being well-developed and studied, it gives a powerful mathematical instrument for investigations of models constructed on the set-theoretic platform. But at the same time more and more examples demonstrate that in certain cases set theory is not adequate to problem domain formalization especially when only partial information about domain is available. The reason of this inadequacy lies in the fundamentals of set theory, in particular, in membership relation and extensionality principle. For problem domains with incomplete information a membership relation cannot be defined, also the extensionality principle fails. We propose to consider a weaker "set" theory with explicit intensional component. Such a theory may be called theory of intensionalized data. The first-level notions of this theory are notions of preset, set, and nominat. Presets may be considered as collections of "black boxes", sets may be treated as collections of "white boxes", and nominats are collections of "grey boxes" in which "white boxes" are names and "black boxes" are their values. In the paper we have defined these notions and described their main properties. Being oriented on mathematical constructivism we have defined operations over such data as computable in a special intensionalized sense. Obtained computability has been called weak natural computability. It has been defined via several steps of reduction to conventional Turing computability.

The results presented in the paper can be considered as the initial steps in developing the theory of intensionalized data.

In the forthcoming papers we plan to construct complete classes of weak/strong natural computable functions over classes with different intensions and demonstrate how these notions can be used for describing intensionalized semantics of specification languages and program logics.

References

[1] N. Bourbaki. *Theory of Sets.* Berlin: Springer-Verlag, 2004.

[2] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, 1996.

[3] J.M. Spivey. *The Z Notation: A Reference Manual*, 2nd ed. Prentice Hall, 1992.

[4] N.S. Nikitchenko. *Intensional aspects of the notion of program.* Problems of Programming, No. 3–4 (2001), pp. 5–13. [In Russian]

[5] M.S. Nikitchenko. *Gnoseology-based Approach to Foundations of Informatics.* In: Ermolayev, V. et al. (eds.) Proc. 7-th Int. Conf. ICTERI 2011, Kherson, Ukraine, May 4-7, 2011, CEUR-WS.org/Vol-716, ISSN 1613-0073, pp. 27–40.

[6] M.S. Nikitchenko. *Intensional aspects of main mathematical notions.* In: Contemporary problems of mathematics, mechanics and computing sciences: N.N. Kizilova, G.N. Zholtkevych (eds). Kharkov: Apostrophe Publ. (2011), pp. 183–191.

[7] N.S. Nikitchenko. *A Composition-nominative approach to program semantics.* Technical Report IT-TR 1998-020, Technical University of Denmark, ISSN 1396-1608, 1998.

[8] I.A. Basarab, N.S. Nikitchenko, V.N. Redko. *Composition Databases.* Kiev: Lybid Publ., 1992. [In Russian]

[9] N.S. Nikitchenko. *Abstract computability of non-deterministic programs over various data structures.* In: Perspectives of System Informatics. LNCS, vol. 2244, Berlin: Springer (2001), pp. 471–484.

[10] Yu. L. Ershov. *Enumeration Theory.* Nauka Publ., Moscow, 1977. [In Russian]

[11] P. Apostoli, R. Hinnion, A. Kanda, T. Libert. *Alternative set theories.* In: Philosophy of Mathematics: Irvine A.D. (ed.). Elsevier (2009), pp. 461–491.

[12] O. Esser and T. Libert. *On topological set theory.* Mathematical Logic Quarterly, vol. 51 (2005), pp. 263–273.

[13] P. C. Gilmore. *The consistency of partial set theory without extensionality.* In: Axiomatic Set Theory: Jech, Th., (ed.). American Mathematical Society (1974), pp. 147–153.

[14] R. Hinnion. *Intensional solutions to the identity problem for partial sets.* Reports on Mathematical Logic, 42 (2007), pp. 47–69

[15] R.J. Malitz. *Set theory in which the axiom of foundation fails.* Ph.D. thesis, UCLA, 1976.

[16] M. Forti, R. Hinnion. *The consistency problem for positive comprehension principles.* Journal of Symbolic Logic, 54 (1989), pp. 1401–1418.

[17] O. Esser. *On the consistency of a positive theory.* Mathematical Logic Quarterly, 45, No. 1 (1999), pp. 105–116.

[18] O. Esser. *Une theorie positive des ensembles.* Cahiers du Centre de Logique, 13, Academia-Bruylant, Louvain-la-Neuve (Belgium), 2004.

[19] R. Hinnion. *Stratified and positive comprehension seen as superclass rules over ordinary set theory.* Zeitschrift für mathematische Logik und Grundlagen der Mathematik, 36 (1990), pp. 519–534.

[20] M. Crabbé. *Soyons positifs: la complétude de la théorie näive des ensembles.* Cahiers du Centre de Logique, vol. 7 (1992), pp. 51–68.

[21] T. Libert. *ZF and the axiom of choice in some paraconsistent set theories.* Logic and Logical Philosophy, vol. 11 (2003), pp. 91–114.

[22] A. Kisielewicz. *Double extension set theory.* Reports on Mathematical Logic, 23 (1989), pp. 81–89.

[23] M. Holmes. *The structure of the ordinals and the interpretation of ZF in double extension set theory.* Studia Logica, vol. 79 (2005), pp. 357–372.

[24] Z. Pawlak. *Rough sets*. International Journal of Computer and Information Sciences, vol. 11, No. 5 (1982), pp. 341–356.

[25] S. Awodey. *Category theory*. Oxford: Clarendon Press, 2006.

[26] J. Goguen. *A categorical manifesto*. Mathematical Structures in Computer Science, 1 (1991), pp. 49–67.

[27] P. Johnstone. *Topos theory*. London Mathematical Society Monographs, vol. 10, Academic Press, London, New York, San Francisco, 1977.

[28] J. L. Bell. *Toposes and local set theories: An introduction*. Oxford: Clarendon Press, 1988.

[29] A. Joyal and I. Moerdijk. *Algebraic Set Theory*. Cambridge University Press, 1995.

[30] S. Awodey. *A brief introduction to algebraic set theory*. Bulletin of Symbolic Logic, 14, No. 3 (2008), pp. 281–298.

[31] F. W. Lawvere, R. Rosebrugh. *Sets for Mathematics*. Cambridge University Press, 2003.

[32] J. L. Bell. *Abstract and Variable Sets in Category Theory*. In: What is Category Theory? Polimetrica Publisher, Italy (2006), pp. 9–16.

[33] Ø. Linnebo, R. Pettigrew. *Category Theory as an Autonomous Foundation*. Philosophia Mathematica, vol. 19, No. 3 (2011), pp. 227–254.

[34] B. Nordström, K. Petersson, J. M. Smith. *Programming in Martin-Löf's Type Theory*. Oxford University Press, 1990.

[35] P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, Napoli, 1984.

[36] J.Barwise. *Admissible sets and structures*. Perspectives in Mathematical Logic, Volume 7. Berlin: Springer-Verlag, 1975.

[37] E. Bishop. *Foundations of Constructive Analysis*. New York: McGraw-Hill, 1967.

Mykola Nikitchenko, Alexey Chentsov, Received July 5, 2012

Mykola Nikitchenko
Taras Shevchenko National University of Kyiv
01601, Kyiv, Volodymyrska st, 60
Phone: +38044 2590519
E–mail: *nikitchenko@unicyb.kiev.ua*

Alexey Chentsov
Taras Shevchenko National University of Kyiv
01601, Kyiv, Volodymyrska st, 60
Phone: +38044 2590511
E–mail: *chentsov@ukr.net*