Pure and Applied Mathematics Journal
2019; 8(1): 1-9
http://www.sciencepublishinggroup.com/j/pamj
doi: 10.11648/j.pamj.20190801.11
ISSN: 2326-9790 (Print); ISSN: 2326-9812 (Online)

Algebra of Countably Functions and Theorems of Completeness

Maydim Malkov

Department of Mathematics, Research Center for Artificial Intelligence, Moscow, Russia

Email address:
mamalkov@gmail.com

To cite this article:
Maydim Malkov. Algebra of Countably Functions and Theorems of Completeness. *Pure and Applied Mathematics Journal.* Vol. 8, No. 1, 2019, pp. 1-9. doi: 10.11648/j.pamj.20190801.11

Received: February 15, 2019; **Accepted**: March 18, 2019; **Published**: April 9, 2019

Abstract: An algebraic approach to the theory of countable functions is given. Compositions (superpositions) of functions are used instead of recursions. Arithmetic and analytic algorithms are defined. All closed sets are founded. Mathematically precise definitions of logic algorithms with quantifiers of existence and universality are given. Logic algorithm for fast-growing function is built as example. Classification of functions is given. There are non-computable functions. These functions are fictitious (useless) and their set is continual. The set of computable functions is countable. Incompleteness of disjunction and negation, conjunction and negation, of Pierce, Sheffer and diagonal Webb functions is proved. The completeness of the set of one-place functions and any all-valued essential function (Slupecki theorem) is proved for computable functions. Existence of generators of all computable functions is proved too.

Keywords: Discrete Functions, Complete Sets of Functions, Algebra Countably-valued Functions, Logic Programming, Theory of Algorithms

1. Introduction

The algebras of compositions (superpositions) of multi-valued, countably-valued, and real functions were defined mathematically precise by A. I. Mal'cev [1]. He called these algebras iterative and pre-iterative Post algebras. More precisely, iterative algebras can be called Jablonski algebras, since the results obtained by S. V. Yablonski [2] are constructed in these algebras. Pre-iterative algebra is more precisely called Post algebra, since E. L. Post [3] used pre-iterative algebra. Later Post algebra was ignored. In particular, only Jablonski algebra was used in the D. Lau [4] monograph devoted to algebras of multi-valued functions.

Further, only Post algebra is used.

The algebra of countably-valued functions differs significantly from the algebra of multi-valued functions. In particular, the completeness of disjunction and negation, as well as conjunction and negation, exists in all algebras of multi-valued functions, but does not exist in the algebra of countably-valued functions.

The algebra of countably-valued functions contains the theory of algorithms, in particular, the theory of computable functions. But this algebra radically changes the theory of computable functions. In this algebra recursions are absent, but all recursive functions can be constructed. The composition algebra are more powerful than the recursion algebra.

In addition to compositions, there are many other closure operations, the main purpose of which is to enlarge closed sets. This integration is intended to reduce the family of closed sets of functions, since this family is very large in the algebra of multi-valued functions, and especially in the algebra of countably-valued functions. The most powerful integration of closed sets is provided by FE -operations of the closure [5]. In particular, in the algebra of two-valued functions there exists a countable family of sets closed by compositions, and only 2 sets closed by FE -operations.

Building of classification of sets closed by operations other than compositions is inefficient. It is more efficient to confine some upper levels of classification by compositions[1]. In particular, the number of maximal sets in a two-valued algebra is three (two sets are fictitious).

[1] In [6], the hyper-continuation of maximal sets in a countable-valued logic is proved, but this hyper-continuation arises due to fictitious sets, including sets of non-computable functions. After removing fictitious sets, the number of other sets is countable.

The number of works devoted to recursions is very large, but the number of works devoted to compositions in algebras of countable-valued functions is insignificant. These are works of A. I. Mal'cev [6] on one-place functions, A. Salomaa [7, 8] on completeness, G. P. Gavrilov [9, 10] on completeness, and S. S. Marchenkov [11] on expressibility.

Complete functions (complete generators), i.e. computable functions generating all other computable functions, are generally accepted to call Sheffer functions. But the term "complete" functions is more convenient, since it is more correct to call the negation of a conjunction *Scheffer's function*. The negation of disjunction is generally called Webb function, but it is more correct to call this negation *Pierce function*. Webb function is more correctly to call its *diagonal function*: $We(x,x) = x+1, We(x_1,x_2 \neq x_1) = 0$. All these functions are not complete in the algebra of countable functions. But new complete functions exist in this algebra (A. Salomaa [7] argued that complete functions do not exist in this algebra).

2. Algebra of Countable-Valued Functions

The algebra of countably-valued functions is one of algebras of compositions (Post algebras). The definition of algebras of compositions was given by A. I. Mal'cev. This definition has the following form for algebras of countable-valued functions.

Definition 1. *Post algebra* $\mathrm{P_N}$ is

$$\mathrm{P}_N = (P_N; \Omega)$$

where carrier P_N is the set containing all countably-valued functions, N is the set of natural numbers (the set of values of functions), Ω is the set of basic operations of the algebra.

The set Ω consists of the following operations on functions:

(1) ζ is a cyclic permutation of variables, at which the first variable becomes last, then the numbering of the variables is corrected;

(2) τ - permutation of two variables, first and last; this and previous operations give any permutation of variables;

(3) $\rhd$ - identification (equality) of the first two variables;

(4) $*$ - substitution the first variable of one function by another function.

A subalgebra of Post algebra is an algebra, basic set of which is a set of functions closed by compositions, and the basic operations are compositions.

Further, subalgebras of Post algebra are called algebras. Unless otherwise stated, all functions are everywhere defined.

All algebras have the same set of operations. So, any algebra is defined by its carrier. So, it is generally accepted to identify algebra and its carrier. In particular, the intersection of two algebras means an algebra, the set of functions of which is the intersection of the sets of functions of these two algebras. The intersection of two algebras is an algebra, the union of two algebras may not be an algebra. But the union of two algebras generates an algebra containing them (and containing not only them).

The algebra of countable functions refuses recursions. But any recursive functions can be generated by any complete functions. The theory of algorithms is described below from the point of view of Post algebra.

3. Algorithms

3.1. Arithmetic and Analytic Algorithms

Algorithms are arithmetic and analytic.

The exact definition of algorithms that compute functions and relations is given in 4.2 (Definition 11). The exact definition of all other algorithms does not exist by means of logic language. Conditionally, an algorithm is a program presented by a programming language.

Arithmetic algorithms are designed to calculate functions and relations. Relations are sets, and all sets are relations (in Post algebra). All functions are relations too.

Relations and functions are represented by tables and are denoted by $r(x_1,\ldots,x_n)$ and $f(x_1,\ldots,x_n)$, where r and f are table names, $x_1,\ldots,x_n$ are column names.

Arithmetic algorithms are also denoted by $r(x_1,\ldots,x_n)$ and $f(x_1,\ldots,x_n)$, where r and f are names of the algorithms, x_i are free variable of formulas fulfilled by algorithms. This designation coincides with the designation of relations, but it is always clear from the text what is a name of algorithm and what is a name of relation or function.

Analytic algorithm analyzes arithmetic algorithms, or other analytic algorithms. This analysis is needed to identify algorithms which work without stopping, i.e. infinitely long for calculating some line in table of function or relation. In particular, analytic algorithm can find that the arithmetic algorithm works without stopping to find a nonzero result of the Fermat grate equation.

Analysis of algorithms is performed by compiler and interpreter of any programming language (compiler or interpreter analyzing the Fermat equation are not created and are unlikely to be created). An algorithm is executed only after a positive analysis, but this does not guarantee that the algorithm is non-stopping. There are theorems proving that there are algorithms that work with stopping, but they are not detected by analysis.

Further, only arithmetic algorithms are considered, but it is assumed that analysis of these algorithms can preserve non-stopping.

Next, construction of some classification of relations will be done. These relations are called sets.

3.2. Sets

Further, sets are one-place relations. But the obtained results are valid for other sets too. Sets have complements.

Definition 2. A family of sets is called *closed* if each set of the family contains a complement of this set.

Classes of sets are next: decidable, almost decidable, partially decidable, (computably) enumerable, partially enumerable, and non-computable. Any set belongs to only one of these classes and is called by name of the class. An analysis

of an algorithm, which calculates a set, may not determine which class contains this set, even though this set belongs to only one of these classes.

Definition 3. A set is *decidable* if algorithm calculates elements of this set in order of increase of element values.

It is impossible to prove by calculations that a set is decidable, since it can be proved only after infinite number of calculations. It is possible to prove this by analysis of the algorithm calculated this set.

Lemma 1. A closure of a decidable set is a decidable set.

Proof. The algorithm computing a decidable set calculates simultaneously the complement of this set. The elements of this complement are the numbers, which precede the first element of the decidable set and which are between two elements of this set. Therefore, the algorithm calculates the elements in order of increasing their values.

Definition 4. A set is *almost decidable* if an algorithm calculates elements of this set in order of increasing values of these elements only after a certain number of elements. We denote this number by m.

It is impossible to calculate m. For this you need to calculate all elements. Therefore, m can be determined by analyzing the algorithm. Knowledge of m is necessary only for calculating the complement of this set. This complement is a partially decidable set (PDS).

Definition 5. A set is *partially decidable* if an algorithm calculates elements of this set with errors, and with corrections of all of them by finite number of times.

A PDS can be obtained by calculating the complement of the almost decidable set. A PDS is empty until the first element of the almost decidable set. Then this PDS contains all elements preceding the first element. After calculating the second element, this PDS contains all elements preceding the second element except the first element. So, this PDS contains all elements preceding any next element, except for elements of the almost decidable set. If a next element of almost decidable has not increasing value, then this element is removed from the PDS, i.e. this value is erroneous.

A PDS is calculated with errors up to a certain number of elements of almost decidable set, after which all erroneous elements of PDS are deleted. This number is m. Further, all calculated elements are not erroneous.

Any set is a PDS if it is calculated with errors and their deleting only up to a certain element. After reaching this element, all other elements are calculated without errors. So, a PDS can be calculated independently, without simultaneously calculating an almost decidable set.

Lemma 2. The closure of an almost decidable set is this set and its PDS.

Proof. It was shown above that complement of an almost decidable set is a PDS. It remains to prove that complement of any PDS is the almost decidable set.

This complement occurs simultaneously with the calculation of the PDS. The complement is empty until the first element is calculated. After this, the complement contains all elements preceding the first element. With each next element of the PDS, the complement contains all the preceding elements, except for elements of the PDS. If a next element of the PDS is erroneous, then it is added to the complement.

Erroneous elements do not occur after reaching element with number m. Further, the complement contains elements only with increasing values. Therefore, the complement of the PRM is an almost decidable set.

Definition 6. The set is (computably) *enumerate* if the algorithm calculates its elements without order in values down to infinity.

An example of an enumerable set is a set of values of a function that non-monotonically increases up to its variable to be infinite.

There are partial enumerable sets (PES).

Definition 7. A set is *partial enumerable* if the algorithm calculates its elements with errors and then deleting all of them only by infinitely many times.

A PES can be obtained by calculating the complement of an enumerable set. This PES is empty until the first element of the enumerable set. After that, the PES contains all elements preceding the first element. And the PES contains all elements presiding any next element of the enumerable set, except for elements of the enumerable set. If a next value of an element is not increasing, then this element is removed from the PES, i.e. this element was erroneous.

Any set is a PES if its calculation creates erroneous elements but deletes all of them only by infinitely times. This means that a PCP can be calculated independently, without simultaneously calculating the enumerable set.

Lemma 3. The closure of an enumerable set is complete.

Proof. The proof repeats the proof of the previous lemma.

It was shown above that a complement of enumerable set is a PES. It remains to prove that the complement of any PES is an enumerable set.

The complement is calculated simultaneously with the calculation of the PES. The complement is empty before calculating the first element of the PES. After this complement contains all elements preceding the first element. And this complement contains all elements preceding the any next elements except elements of PES. If a next element of the PES is erroneous, then this element is added to the supplement. In this case, elements of the complement have increase and decrease values. So, the complement is indeed enumerable, without erroneous elements.

Definition 8. A set is *computable* if there is an algorithm for calculating all its elements.

This definition is different from the generally accepted. It is generally accepted that a set is computable (algorithmically decidable) if it is decidable. An enumerable set is algorithmically undecidable, since finding out an element to belong to the set leads to non-stop calculating, if this element does not belong to the set. But analysis of the algorithm does not allow to fulfill the algorithm. And any algorithm must be analyzed before computing, even this algorithm is decidable.

This definition of computability is not constructive, since an algorithm is considered computable if it works one hundred years. In addition, compilers and interpreters do not perform a

complete analysis of algorithms.

Theorem 1. A closure of decidable, almost decidable, and enumerable sets contains all computable sets.

Proof. All elements of a computable set are calculated in any sequence of their values, and all erroneous elements are excluded. In accordance with the lemmas shown above, the closure of first 5 classes of sets is the same class of sets.

The family of computable sets is countable, since the set of algorithms is countable. All other sets are non-computable, and their family is continual, since the family is a family of all subsets of the natural set.

A similar definition of computability exists for functions.

Definition 9. A function is *computable* if there is an algorithm for calculating it.

This definition holds for partial functions too. The algorithm for calculating these functions must calculate its domain. Otherwise, the algorithm will be blocked as a result of analysis.

Functions, like relations, are also decidable, almost decidable, partially decidable, enumerable, partially enumerable, and non-computable if they are such relations. Any function belongs to only one of these classes of functions.

The set of computable functions is countable, the set of non-computable functions is continual.

4. Logic Algorithms

4.1. Definition

The exact definition of an algorithm is reduced to the definition of a logic algorithm.

Logic algorithms are built according to the rules of logic.

Each calculated object must have a definition. In logic, the definition of an object is given by a logic formula, a special case of which is equality.

Equalities are the simplest definitions of functions. The left part of equality is a function in the form $f(x_1,\dots,x_n)$, where f is the name of the function. The right part is an arithmetic expression.

An example of definition by equality is: $Ac(x_1,x_2) = Fast_{x_1}(2,x_2)$, where Ac is an Ackermann function, $Fast_{x_1}$ is a fast-growing function, the definition of which already exists. The variables in the left part of this equality can be replaced by values, for which the function should be calculated. It is more correct to represent $Fast_{x_1}(2,x_2)$ as $Fast(x_1,2,x_2)$, but it is generally accepted to use the subscript in definition of $Fast$.

Other definitions of function use logic formulas. For example,

$$Fast_m(x_1,x_2) = x_0 \rightleftarrows m = 0 \wedge x_0 = x_1 + x_2 \vee$$

$$\vee\, m = 1 \wedge x_0 = x_1 \cdot x_2 \vee m > 1 \wedge (x_2 = 0 \wedge x_0 = 1 \vee x_2 = 1 \wedge x_0 = x_1 \vee \vee\, x_2 > 1 \wedge\ x_0 = Fast_{m-1}\,(x_1, Fast_m(x_1, x_2 - 1)))$$

Simpler example is definition of Fibonacci functions:

$$Fi(x_1) = x_0 \rightleftarrows x_1 = 0 \wedge x_0 = 0 \vee x_1 = 1 \wedge$$

$$\wedge\, x_0 = 1 \vee x_1 \geq 2 \wedge x_0 = Fi(x_1 - 1) + Fi(x_1 - 2)$$

The relations are defined as

$$r(x_1,\dots,x_n) \rightleftarrows \text{logic formula without } \rightleftarrows$$

It must be used the lower part of $\rightleftarrows$:

$$r(x_1,\dots,x_n) \leftarrow \text{logic formula without } \rightleftarrows$$

Then the left part of the definition is true if the right part is true. It is enough to calculate relations and functions.

The upper part in $\rightleftarrows$ means that truth of the left part implies truth of the right part. This is not applicable for calculations, since a calculation is possible if truth of the left part follows from truth of the right side.

4.2. Skolem Formulas

There are several ways to represent any logic formula. But the Skolem formula eliminates this uncertainty.

The reduction to this formula is carried out in 7 steps:

(1) reduction to the form with negations only of relations;
(2) reduction to the prefix form, in which all quantifiers are at the beginning of the right part of definition;
(3) deletion of existential quantifiers with replacing their variables by Skolem functions, the variables of the universal quantifiers are denoted by x with subscript, the Skolem functions are denoted by a with subscript;
(4) deletion of universal quantifiers; their variables are absent in the left part;
(5) replacing $\rightleftarrows$ with $\leftarrow$;
(6) reduction of the right part to disjunctive normal form;
(7) reduction to several definitions, the right side of which is a conjunct of disjunctive normal form; this is possible because a disjunctive normal form is true if any of its conjuncts are true, i.e. the truth of any conjunct entails truth of the left part.

Definitions in this form are called *rules* or *clauses*. A rule is represented as

$$r(x_1,\dots,x_n) \leftarrow r_1 \wedge \dots \wedge r_m$$

where all r_i are relations or negations of relations.

Definition 10. *Logic algorithm* is a set of rules that are a logic definition of a calculated object in Skolem form.

Now one can give a precise definition of the algorithm.

Definition 11. *Algorithm* (arithmetic algorithm) is a program for calculating functions or relations, if the program can be presented as a logic program.

A program for calculating functions or relations presented in other programming languages may not be algorithm if this program calculates an object that does not have a logic definition. In other cases, a program is algorithm.

4.3. Functions $Fast_m$

The definition of $Fast_m(x_1,x_2)$ presented above generates the next rules.

$$Fast_0(x_1,x_2) = x_1 + x_2$$

$$Fast_1(x_1, x_2) = x_1 \cdot x_2$$

$$Fast_m(x_1, 0) = x_0 \leftarrow m > 1 \wedge x_0 = 1$$

$$Fast_m(x_1, 1) = x_0 \leftarrow m > 1 \wedge x_0 = x_1$$

$$Fast_m(x_1, x_2) = x_0 \leftarrow m > 1 \wedge x_2 > 1 \wedge$$

$$\wedge x_0 = Fast_{m-1}(x_1, Fast_m(x_1, x_2 - 1))$$

These rules are an algorithm. The execution of this algorithm is given by the rule

$$Fast_{\dot{m}}(\dot{x}_1, \dot{x}_2)$$

where $\dot{m}, \dot{x}_1, \dot{x}_2$ are concrete values of m, x_1, x_2.

Calculations are performed in the next sequence

$$Fast_{\dot{m}}(\dot{x}_1, \dot{x}_2) = Fast_{\dot{m}-1}(\dot{x}_1, Fast_{\dot{m}}(\dot{x}_1, \dot{x}_2 - 1)) =$$

$$= Fast_{\dot{m}-1(\dot{x}_1, Fast_{\dot{m}-1}(\dot{x}_1, Fast_{\dot{m}}(\dot{x}_1, \dot{x}_2 - 2)))=\ldots=}$$

$$= Fast_{\dot{m}-1}(\dot{x}_1, Fast_{\dot{m}-1}(\dot{x}_1, Fast_{\dot{m}-1}($$

$$\dot{x}_1, Fast_{\dot{m}-1}(\ldots(Fast_{\dot{m}-1}(\dot{x}_1, \dot{x}_1))))))$$

Next, $Fast_{\dot{m}-1}(\dot{x}_1, \dot{x}_1)$ with a lower value $m = \dot{m} - 1$ is calculated.

Let the value of m be decreased to 3. Then

$$Fast_3(\dot{x}_1, \dot{x}_1) = Fast_2(\dot{x}_1, Fast_3(\dot{x}_1, \dot{x}_1 - 1)) =$$

$$= Fast_2(\dot{x}_1, Fast_2(\dot{x}_1, Fast_3(\dot{x}_1, \dot{x}_1 - 2))) =\ldots=$$

$$= Fast_2(\dot{x}_1, Fast_2(\dot{x}_1, Fast_2($$

$$\dot{x}_1, Fast_2(\ldots(Fast_2(\dot{x}_1, \dot{x}_1))))))$$

If $m = 2$, then

$$Fast_2(\dot{x}_1, \dot{x}_1) = Fast_1(\dot{x}_1, Fast_2(\dot{x}_1, \dot{x}_1 - 1)) =$$

$$= Fast_1(\dot{x}_1, Fast_1(\dot{x}_1, Fast_2(\dot{x}_1, \dot{x}_1 - 2))) =\ldots=$$

$$= Fast_1(\dot{x}_1, Fast_1(\dot{x}_1, Fast_1(\dot{x}_1, Fast_1($$

$$\ldots(Fast_1(\dot{x}_1, \dot{x}_1))))))) = \dot{x}_1^{\dot{x}_1}$$

I.e. $Fast_2(\dot{x}_1, \dot{x}_1) = x_1^{x_1}$. Substituting this result into the previous expression with $m = 2$, we get $Fast_3(\dot{x}_1, \dot{x}_1) = \dot{x}_1^{\dot{x}_1^{\dot{x}_1}}$. Further substitutions lead to increase the number of powers for one.

5. Classification of Functions and Algebras

In any theory, the classification of its objects is a main problem.

First, classification of all objects is constructed. This classification allows to detect of fictitious (useless) objects. Then more deeper classification is built for only useful objects.

The objects of algebra of countably-valued functions are functions and subalgebras of this algebra.

Let's start with classification of subalgebras. This classification contains several levels.

The top level contains algebra P_N. By definition, it is a subalgebra too.

The second level contains subalgebras of computable and non-computable functions. The subalgebra of non-computable functions is fictitious (useless in P_N); therefore, a further classification is constructed only for subalgebra of computable functions.

The third level contains the countable set of subalgebras U_i $(0 \le i \le \infty)$, where i is the minimal number of functions generated algebras of the subalgebra. There are [12] subalgebras not generated by functions $(i = 0)$ and subalgebras generated by an infinite set of functions $(i = \infty)$.

The subalgebra U_∞ contains continuum of algebras, algebras U_i with finite i contain countable set of algebras.

Below it will be shown that all algebras U_i (except U_1) are fictitious (useless). Therefore, further classification is a classification of the subalgebra U_1. But this classification is isomorphic to the classification of computable functions. Further, classification of functions is built.

The class of computable functions is on the second level of function classification. The class of non-computable functions are on the second level too. The class of all functions is on the top level.

Further, classification is constructed for computable functions.

The classification of computable functions is isomorphic to the classification of the subalgebra U_1.

Each function generates some algebra by compositions. There are much functions that generate the same algebra. This set is a class of the classification, since any function belongs to only one class which is generated by these functions. Therefore, classes do not intersect.

Each class of functions belongs to unique algebra which is generated by these functions. This algebra belongs to U_1, since U_1 contains all algebras generated by a single function. Consequently, the classification of the subalgebra U_1 is isomorphic to the classification of computable functions. So there is a one-to-one correspondence between these classifications.

Not all algebras are generated by a single function. There are algebras generated by only a few functions. These algebras are fictitious (useless) from the point of view of the classification of functions (the classification of functions is the main problem of the algebra of countably-valued functions).

Fictitious objects are deleted in any theory. Therefore, fictitious algebras are deleted. The study of their properties and construction of further classification are useless. The set of algebras contained in fictitious subalgebras is continual (due to U_∞). The set of algebras contained in U_1 is countable. Therefore, the statement in [12] is erroneous. This statement says that properties of algebras of multi-valued functions are significantly different from properties of the algebra of two-valued functions due to continuity. Fictitious objects are removed from any theory, so their properties are not

interesting.

Further the classification of functions is continued.

The classification of functions in Post algebras uses a composition of functions. Therefore, 5 classes of functions: decidable, almost decidable, partially decidable, enumerable, and partially enumerable, are not included in this classification. The function of one of these classes can generate functions of other classes by compositions, in particular, by identification of variables.

The third level of classification of functions contains classes called maximal. The set of maximal classes is countable. As in algebras of multi-valued functions, these classes form 3 families (I. G. Rosenberg [13] called 6 families, but 3 of them are fictitious; in the algebra of two-valued functions, 2 such families are fictitious [14]). Non-fictitious families are families of self-dual (auto-dual) functions, functions that preserve nontrivial equivalence relations, and functions that preserve central relations.

A deeper classification of functions is difficult because of infinite sets of classes at almost every level of classification.

All functions in Post algebras are everywhere defined. But a partial function can have the single way to be continued to an everywhere defined function. Therefore, the classification of partial functions coincides with the classification of everywhere defined functions.

6. Incompleteness Theorems

A function containing $max(x_1, x_2)$, $min(x_1, x_2)$ and $x + 1$ is represented further by a logic formula with $max(x_1, x_2)$ replaced by $x_1 \vee x_2$, $min(x_1, x_2)$ by $x_1 \wedge x_2$ and $x + 1$ by $\overline{x}$. This image of functions is more familiar.

Completeness of theorems in the countably-valued algebra differs significantly from completeness of these theorems in a multi-valued algebra.

Theorem 2**.** In P_N the sets $\overline{x}, x_1 \vee x_2$ and $\overline{x}, x_1 \wedge x_2$ do not generate all functions.

Proof. It is enough to restrict the proof to generating one-place functions.

The negation $\overline{x} = x + 1$ generates one-place functions without values 0.

The function $x_1 \vee x_2$ generates only one-place function x. Substitutions of the negation into $x_1 \vee x_2$ and of $x_1 \vee x_2$ into the negation generate only one-place function $\overline{x}$.

The function $x_1 \wedge x_2$ generates only one-place function x. Substitution $x_1 \wedge x_2$ into the negation generates only one-place function $\overline{x}$. Substitution of the negation into $x_1 \wedge x_2$ generates only one-place function $f(0) = 0, f(x \neq 0) = \overline{x}$.

One place functions with several values 0 are not generated. □

Corollary 1. The sets $\overline{x}, x_1 \vee x_2$ and $\overline{x}, x_1 \wedge x_2$ are not complete.

Proof. These sets do not generate all functions.

This is in contrast to multi-valued algebras in which these sets are complete.

Theorem 3. The Pierce ($\overline{x_1 \vee x_2}$), Sheffer ($\overline{x_1 \wedge x_2}$) and diagonal Webb ($x + 1$ if $x_1 = x_2 = x$, 0 if $x_1 \neq x_2$) functions are not complete.

Proof. The Pierce and Sheffer functions are not complete, since negations of $x_1 \vee x_2$ and of $x_1 \wedge x_2$ do not contain values 0. The diagonal Webb function is not complete since this function generates only one-place functions $x + m$with $1 \leq m < infty$.

7. Slupetski Theorem

7.1. Theorem

The Slupetski theorem holds in P_N.

Theorem 4. Let the algebra have countable computable functions, let the set contain all one-place functions and any two-place all-valued essential function X. Then this set is complete.

Proof. J. Slupetski used the Lukasiewicz function Lu. In modern notation, this function for finite k is

$$Lu(x_1, x_2) = S(U_{0,0}^{a_{0,0}}(x_1, x_2), (S(U_{0,1}^{a_{0,1}}(x_1, x_2), (\dots ($$

$$S(U_{0,k-1}^{a_{0,k-1}}(\dots \dots (S(U_{k-1,0}^{a_{k-1,0}}(x_1, x_2), (\dots (S(U_{k-1,k-2}^{a_{k-1,k-2}}($$

$$x_1, x_2), S(U_{k-1,k-1}^{a_{k-1,k-1}}(x_1, x_2))))\dots)))\dots \dots)))))))$$

where S and $U_{i_1,i_2}^{a_{i_1,i_2}}$ are: $S(0, x) = x, S(x, 0) = x, U_{i_1,i_2}^{a_{i_1,i_2}}(i_1, i_2) = a_{i_1,i_2},\ x_1 \neq i_1 \vee x_2 \neq i_2 \rightarrow U_{i_1,i_2}^{a_{i_1,i_2}}(x_1, x_2) = 0$. Constants a_{i_1,i_2} are any, functions S and $U_{i_1,i_2}^{a_{i_1,i_2}}$ are generated by X and one-place functions.

Any function $Y(x_1, x_2) = Lu(x_1, x_2)$ if $a_{i_1,i_2} = Y(i_1, i_2)$.

Function Lu does not exist if $k = \infty$. But the new function Lu exists:

$$Lu(x_1, x_2) = S(U_{l(0),r(0)}^{a_{l(0),r(0)}}(x_1, x_2), (S(U_{l(1),r(1)}^{a_{l(1),r(1)}}(x_1, x_2),$$

$$(\dots (S(U_{l(k(k+1)/2),r(k(k+1)/2)}^{a_{l(k(k+1)/2),r(k(k+1)/2)}}(x_1, x_2)))\dots))))$$

where l and r are the left and right parts of Cantor's number of the pair (i_1, i_2) in $U_{i_1,i_2}^{a_{(i_1,i_2)}}$.

Then any function $Y(x_1, x_2) = Lu(x_1, x_2)$ if $a_{l(i),r(i)} = Y(l(i), r(i))$.

If $x_1 \leq l(k), x_2 \leq r(k)$ then

$$Lu(x_1, x_2) = S(U_{l(0),r(0)}^{a_{l(0),r(0)}}(x_1, x_2), (S(U_{l(1),r(1)}^{a_{l(1),r(1)}}(x_1, x_2),$$

$$(\dots (S(U_{l(k),r(k))}^{a_{l(k),r(k)}}(x_1, x_2)))\dots))))$$

since $S(U_{l(i),r(i))}^{a_{l(i),r(i)}}(x_1, x_2) = 0$ if $i > k$ and $x_1 \leq l(k), x_2 \leq r(k)$.

If $k = \infty$ then

$$Lu(x_1, x_2) = S(U_{l(0),r(0)}^{a_{l(0),r(0)}}(x_1, x_2), ($$

$$S(U_{l(1),r(1)}^{a_{l(1),r(1)}}(x_1, x_2), \dots)))$$

This formula has infinite number of parenthesis. But the

next formula has finite number of them:

$$Lu(\dot{x}_1,\dot{x}_2)=S(U_{l(0),r(0)}^{a_{l(0),r(0)}}(\dot{x}_1,\dot{x}_2),($$

$$S(U_{l(1),r(1)}^{a_{l(1),r(1)}}(\dot{x}_1,\dot{x}_2),\dots,S(U_{l(c),r(c)}^{a_{l(c),r(c)}}(\dot{x}_1,\dot{x}_2)))))$$

where $\dot{x}_1,\dot{x}_2$ are concrete values, and c is a Cantor's number of $(\dot{x}_1,\dot{x}_2)$. The formula holds since $S(U_{l(i),r(i)}^{a_{l(i),r(i)}}(\dot{x}_1,\dot{x}_2)=0$ if $i>c$.

Then $Y(\dot{x}_1,\dot{x}_2)=Lu(\dot{x}_1,\dot{x}_2)$ for any function Y if $a_{l(i),r(i)}=Y(l(i),r(i))$.

It is necessary to prove that S and $U_{i_1,i_2}^{Y(i_1,i_2)}$ are generated by the function X and one-place functions.

7.2. Function S

I. Slupetsky built this function in k steps. This does not apply to $k=\infty$. Therefore a new construction is given. This construction is more simple.

The construction starts with $S(0,0)$. Since the function X is all-valued, there are a_1 and a_2 such that $X(a_1,a_2)=0$. Then $S(0,0)=X(a_1,a_2)$.

There are b_1 and b_2 with $X(b_1,b_2)=1$. Therefore, $S(0,1)=X(b_1,b_2)$. And so on.

The function S has the next definition:

$$S(x_1,x_2)=x_0\rightleftarrows x_1=0\wedge$$

$$\wedge(\exists(a_1,a_2)\,X(a_1,a_2)=x_2\wedge x_0=X(a_1,a_2))\vee$$

$$\vee x_2=0\wedge\exists(b_1,b_2)\,X(b_1,b_2)=x_1\wedge x_0=X(b_1,b_2)\vee$$

$$\vee x_1\neq 0\wedge x_2\neq 0\wedge x_0=X(x_1x_2)$$

So X generated S.

Definition of S generates equations

$$S(0,x_2)=x_2,\quad S(x_1,0)=x_1$$

where x_1 and x_2 are values of X. Only this was used in previous subsection.

7.3. Function $U_{i_1,i_2}^{a_{i_1,i_2}}$

The definition of $U_{i_1,i_2}^{a_{i_1,i_2}}$ given by J. Slupetski is applicable to $k=\infty$. But function S_2 used in the definition has definitions non-applicable to $k=\infty$.

The function

$$U_{i_1,i_2}^{a_{i_1,i_2}}(x_1,x_2)=F_{a_{i_1,i_2}}(S_2(E_{i_1}(x_1),E_{i_2}(x_2)))$$

where one-place functions $F_j(0)=j,F_j(x\neq 0)=0$ and $E_j(j)=0,E_j(x\neq j)=1$.

If $S_2(0,0)=0\wedge S_2(1,1)=1$, then

$$U_{i_1,i_2}^{a_{i_1,i_2}}(i_1,i_2)=a_{i_1,i_2},$$

$$x_1\neq i_1\vee x_2\neq i_2\rightarrow U_{i_1,i_2}^{a_{i_1,i_2}}(x_1,x_2)=0$$

Indeed,

$$U_{i_1,i_2}^{a_{i_1,i_2}}(i_1,i_2)=X_{a_{i_1,i_2}}\left(S_2\left(E_{i_1}(i_1),E_{i_2}(i_2)\right)\right)=$$

$$=X_{a_{i_1,i_2}}(S_2(0,0))=X_{a_{i_1,i_2}}(0)=a_{i_1,i_2},$$

$$x_1\neq i_1\vee x_2\neq i_2\rightarrow$$

$$\rightarrow U_{i_1,i_2}^{a_{i_1,i_2}}(x_1,x_2)=X_{a_{i_1,i_2}}\left(S_2\left(E_{i_1}(x_1),E_{i_2}(x_2)\right)\right)=$$

$$=X_{a_{i_1,i_2}}(S_2(1,1))=X_{a_{i_1,i_2}}(1)=0$$

7.4. Function S_2

The Slupecki definition of S_2 is very complex and non-applicable to $k=\infty$ The next definition is very simple:

$$S_2(x_1,x_2)=A_3(X(A_1(x_1),A_2(x_2)))$$

where one-place functions A_1,A_2,A_3 are

$$A_1(0)=a_1,A_1(x\neq 0)=b_1,$$

$$A_2(0)=a_2,A_2(x\neq 0)=b_2,$$

$$A_3(X(a_1,a_2))=0,A_3(x\neq X(a_1,a_2))=1$$

and where $\exists(a_1,a_2,b_1,b_2)\;X(a_1,a_2)\neq X(b_1,b_2)$ holds.

The definition of S_2 generates all equations used in the previous subsection:

$$S_2(0,0)=0,S_2(1,1)=1$$

Indeed,

$$S_2(0,0)=A_3\left(X\left(A_1(0),A_2(0)\right)\right)=A_3\left(X(a_1,a_2)\right)=0,$$

$$S_2(1,1)=A_3(X(A_1(1),A_2(1)))=A_3(X(b_1,b_2))=1$$

7.5. Logic Algorithm

Any definition must not contain dots, except dots for variable number. The next algorithm does not contain dots:

$$Lu(0,x_1,x_2)=U_{l(0),r(0)}^{a_{l(0),r(0)}}(x_1,x_2).$$

$$Lu(j,x_1,x_2)=S(Lu(j-1,x_1,x_2),U_{l(j),r(j)}^{a_{l(j),r(j)}})(x_1,x_2).$$

This is a new function Lu:

$$Lu(j,x_1,x_2)=S(S(\dots S(U_{l(0),r(0)}^{a_{l(0)(x_1,x_2),r(0)}},$$

$$U_{l(1),r(1)}^{a_{l(1),r(1)}}(x_1,x_2),\dots,U_{l(j),r(j)}^{a_{l(j),r(j)}}(x_1,x_2))))$$

at $j\rightarrow\infty$.

Any function $Y(x_1,x_2)=Lu(c,x_1,x_2)$ if c is Cantor number of (x_1,x_2), and if $a_{l(i),r(i)}=Y(l(i),r(i))$.

The functions S and $U_{i_1,i_2}^{a_{i_1,i_2}}$ have the next algorithm.

$$S(x_1,X(a_1,a_2))=X(a_1,a_2)\;S(X(a_3,a_4),x_2)=X(a_3,a_4).$$

$$U_{i_1,i_2}^{a_{i_1,i_2}}(x_1,x_2)=F_{a_{i_1,i_2}}(S_2(E_{i_1}(x_1),E_{i_2}(x_2))).$$

$$F_j(0) = j. \quad F_j(x \neq 0) = 0.$$

$$E_j(j) = 0. \quad E_j(x \neq j) = 1.$$

$$S_2(x_1, x_2) = x_0 \leftarrow X(a_1, a_2) \neq X(a_3, a_4) \wedge$$

$$\wedge x_0 = A_3(X(A_1(x_1), A_2(x_2)))$$

$$A_1(0) = a_1. \quad A_1(x \neq 0) = a_3.$$

$$A_2(0) = a_2. \quad A_2(x \neq 0) = a_4.$$

$$A_3(X(a_1, a_2)) = 0. \quad A_3(x \neq X(a_1, a_2)) = 1.$$

The execution of algorithm of Lu is given by the rule $Lu(\dot{c}, \dot{x}_1, \dot{x}_2$, where $\dot{c}, \dot{x}_1, \dot{x}_2$ are concrete values of c, x_1, x_2. The function X and Y must be given.

Since $\dot{c}, \dot{x}_1 \dot{x}_2$ are any, the Slupetski theorem is proved.

7.6. Completeness at $k = \infty$

It is proved that the set of all one-place functions and a two-place all-valued essential function generates all two-place functions Y. This set generates all multi-place functions since some functions of Y generate all functions. This holds for infinite k too, since two-place functions generating all computable functions exist in P_N. □

8. Complete Generators

A complete generator is a function generating all computable functions by compositions.

Theorem 5. Complete generators exist. The set of complete generators is infinite.

Proof. Two one-place functions $x + 1$ and $x - [\sqrt{x}]^2$ generate all computable on-place functions (R. Robinson theorem [15])[2]. So a two-valued function generates all computable functions if the function generates these two one-place functions.

Let two-place function X have $X(x, x)) = x - [\sqrt{x}]^2$. Let $X(X(x, x), x) = X(x - [\sqrt{x}]^2, x) = x + 1$. And let $X(x_1, x_2) = 0$ if $x_1 \neq x_2 - [\sqrt{x_2}]^2$. This function X generates all computable one-place function.

By analogy, an infinite set of complete generators can be constructed. □

9. Conclusion

A new theory of algorithms has been built. This theory refuses recursions but can generate all recursive functions. 6 classes of sets were found - decidable, almost decidable, partially decidable, enumerable, partially enumerable and non-computable sets. Any set belongs to only one of these classes. The family of the first five classes is closed with respect to the complement operation.

The new mathematically precise definition of a logic algorithm is given. This algorithm solves problems that cannot be solved in other algorithms. In particular, the problems of quantifiers of existence and universality are solved.

In the case of universal quantifier, a rule contains variables in the right part such that the variables absent in the left part. The right part can contain conjuncts which limit the range of these absent values. If the right part is true for some value of one of the absent values, then it is true for all it values of its range. Therefore, it is sufficient to replace all absent values by the minimum value of their range. Further, the rules are executed without absent values and without conjuncts that limit the range.

Theorems of completeness are found which hold in algebras of multi-valued functions and not does not hold in the algebra of countable-valued functions. In particular, these theorems do not hold for disjunctions and negations, for conjunctions and negations, for disjunctions, conjunctions and negations, as well as for the Pierce, Sheffer and diagonal Webb functions.

A new proof of the Slupetsky theorem, applicable to countable-valued functions, was constructed.

The theorem of existence of complete generators in P_N is proved. But concrete complete generator was not found.

[2] In $x - [\sqrt{x}]^2$, x is a real number and $[\sqrt{x}]$ means integer part of $\sqrt{x}$.

References

[1] A. I. Mal'cev, *Iterative Post algebras* (Russian), Novosibirsk, Novosib. gos. un-t (1976).

[2] S. V. Jablonskij, Functional constructions in k-valued logic (Russian), *Tr. mat. inst. Steklova*, 5-142 (1958).

[3] E. L. Post, *Two-valued iterative systems of mathematical logic*, Princeton, Princeton Univ. Press (1941).

[4] D. Lau, *Functions algebra on finite sets*, Berlin, Springer (2006).

[5] S. S. Marchenkov, On FE-precomplete classes of countably valued logic (Russian), *Discrete math.*, (2), 51–57 (2016).

[6] A. I. Mal'cev, Iterative algebras ans Post manifold, (Russian), *Algebra and logic*, (2), 5-24 (1966).

[7] A. Salomaa, On sequences of functions over an arbitrary system, *Annales Universitatis Turkuensis* AI (16), 5–13 (1963).

[8] A. Salomaa, Some analogues of Sheffer functions in infinite-valued logics, *Acta philosophica Fennica*, 227-235 (1963).

[9] G. P. Gavrilov, On functional completeness in countably valued logic, *Problems of cybernetics*, 5–64 (1965).

[10] G. P. Gavrilov, Precomplete classes of parcially countably value logic contained all functions of a single variable (Russian), *Discrete analysis methods in graph theory and logical functions*, 12–24 (1976).

[11] S. S. Marchenkov, On set power of precomplete classes in some classes of countably valued logic functions (Russian), *Problems of cybernetics*, 109–1981 (2015).

[12] Ju. I. Janov, A. A. Muchnik, On existence of k-valued closed classes without finite basis (Russian), *Dokl. Acad. Nauk SSSR*, (1), 44–46 (1959).

[13] IG Rosenberg, Über die functionale Vollständigkeit dem mehrvertigen Logiken von mehreren Verändlichen auf endlichen Mengen, *Rozpravy Cs. Academie Ved, Ser. Math. Nat. Sci.*, 3-93 (1970).

[14] M. A. Malkov, Classification of Boolean functions and their closed sets, *SOP transactions on applied mathematics*, (1), 1-20 (2014).

[15] R. M. Robinson, Primitive recursive functions, *Bull. Amer. Math. Soc.*, bf53, 925-942 (1947).