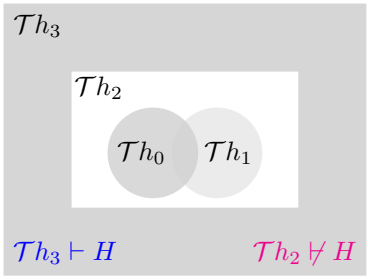
SpecVer & LEM'12 projects XX (2018) 1–31
Dombrova Research

A new proof of Euclid's algorithm

Andrzej Salwicki
salwicki@mimuw.edu.pl, Dombrova Research
Partyzantow 19, 05-092 Lomianki, POLAND

Abstract. Our main result is a new proof of correctness of Euclid's algorithm E. The proof is conducted in algorithmic theory of natural numbers $\mathcal{T}h_3$. 1° We are constructing a formula H that expresses the halting property of the algorithm. 2° Accordingly, we analyze the structure of the algorithm (our proof makes no references to the computations of the algorithm). 3° Our proof makes use of inference rules of *calculus of programs* (i.e. algorithmic logic), 4° The only formulas accepted without proof are the axioms of the program calculus and axioms of algorithmic theory of natural numbers $\mathcal{T}h_3$.
We consider also other theories: first-order theory of natural numbers $\mathcal{T}h_0$, algorithmic theory of addition $\mathcal{T}h_1$, algorithmic theory of addition and multiplication $\mathcal{T}h_2$. It is easy to observe that $\mathcal{T}h_0 \subsetneq \mathcal{T}h_2, \mathcal{T}h_1 \subsetneq \mathcal{T}h_2, \mathcal{T}h_2 \subsetneq \mathcal{T}h_3$.



We complete our result showing that the theorem on correctness of Euclid's algorithm can not be proved in any of theories $\mathcal{T}h_1, \mathcal{T}h_2, \mathcal{T}h_0$.

Keywords: program calculus, algorithmic theory, halting property, algorithm's correctness

1. Introduction

History of Euclid's algorithm is over 2300 years long. We shall analyze the algorithm in its simplest form.

$$\underbrace{\left\{\begin{array}{l}\textbf{while } n \neq m \textbf{ do}\\ \quad \textbf{if } n > m \textbf{ then } n := n - m \textbf{ else } m := m - n \textbf{ fi}\\ \textbf{od}\end{array}\right\}}_{Euclid's\ algorithm} \tag{E}$$

A computation of the algorithm, for initial values n_0 and m_0 results in the sequence of states.
If a state satisfies the condition $n = m$ then the algorithm stops and n is the greatest common divisor of n_0 and m_0. Otherwise a subtraction is performed, changing the value of variable n or m.
Euclid's algorithm is used in solutions of many problems of programming and mathematics. It is applied in various data struct ures. In some structures (e.g. the ring of integers, the field of rational numbers) the algorithm stops and in others (e.g. the field of real numbers, the planar geometry) some of its execution are finite, some others can be prolonged *ad infinitum*.
The traditional proof of the theorem that the Euclid's algorithm halts and computes the greatest common divisor of two natural numbers is commonly accepted. We shall have a closer look at it later. The proof uses some assumptions and bases on the notion of execution of the algorithm. Therefore some semantics is involved in the proof.
We are arguing that it is possible to present a purely syntactic proof of halting property of the algorithm.

To begin with, we need a formula that expresses the stopping property of Euclid's algorithm. In 1967 Erwin Engeler c.f. [Eng67], found that, for every deterministic while program M there is an infinite disjunction (of open formulas) which does express the halting property of the program. In the case of Euclid's algorithm the halting formula may look as follows

$$\forall_{n\neq 0}\forall_{m\neq 0}\left(\begin{array}{l} n = m \vee\\ n = 2m \vee 2n = m \vee\\ n = 3m \vee 2n = 3m \vee 3n = 2m \vee m = 3n \vee\\ \ldots\end{array}\right) \tag{H1}$$

Shortly after, we proposed to extend the language of first-order formulas by algorithms and algorithmic formulas, c.f. [Sal70], see also section II.3 of [MS87]. Our proposal is based on the remark that the program itself is a finite expression.
The formula (H) below is the halting formula of the Euclid's algorithm.

$$\underbrace{\forall_{n\neq 0}\,\forall_{m\neq 0}\left\{\begin{array}{l}\textbf{while } n \neq m \textbf{ do}\\ \textbf{if } n > m\\ \quad \textbf{then } n := n - m\\ \quad \textbf{else } m := m - n\\ \textbf{fi}\\ \textbf{od}\end{array}\right\}(n = m)}_{halting\ formula\ of\ Euclid's\ algorithm} \tag{H}$$

Formula (H) reads informally: *for every* $n > 0$ *and* $m > 0$*, algorithm (E) halts and the result of its computation satisfies formula* $n = m$. Hence, formula expresses the stop property of the program E.
Obviously, there are many formulas equivalent to the halting formula (H). Look at following formula

$$\underset{n\neq 0, m\neq 0}{\forall} \bigcup \left\{ \begin{array}{l} \textbf{if } n > m \\ \quad \textbf{then } n := n - m \\ \quad \textbf{else } m := m - n \\ \textbf{fi} \end{array} \right\} (n = m) \qquad \text{(H')}$$

Formula (H') contains an *iteration quantifier* $\bigcup$ and reads informally: *for every* $n > 0$ *and* $m > 0$*, there exists an iteration of the program*
$\{$**if** $n > m$ **then** $n := n - m$ **else** $m := m - n$ **fi** $\}$ *such that formula* $n = m$ *is satisfied afterwards*, i.e. the formula
$\{$**if** $n > m$ **then** $n := n - m$ **else** $m := m - n$ **fi**$\}^i(n = m)$ holds .
The section 2 brings more information on algorithmic formulas and their semantics.
Note, there is no first-order formula that expresses the same meaning as formulas (H1), (H) or (H') do.

Do we need a new proof?

This text is addressed to programmers and computer scientists as well as to mathematicians. We are going to convince you that:

- Proving properties of programs is like proving mathematical theorems. One needs: axioms, calculus of programs and well defined language. In other words, programmers, makers of specifications (of software), verifiers of software properties, should accept algorithmic theories as the workplace. Also mathematicians will find many open problems in algorithmic theories of traditional data structures. For them also calculus of programs may be an interesting workplace.
  Such a theory contains classical theorems i.e. first-order formulas and algorithmic theorems as well. In the proofs of theorems on certain programs we can use earlier theorems on other programs (and classical theorems also). In the proofs of some first-order theorems we can use some facts about programs. Section 6 gives a flavor of such a theory, to be developed.

- Developers of algorithmic theories of numbers, of graphs, etc. should accept programs as "first class citizens" of the languages of these theories. Moreover, the language should contain formulas that express the semantical properties of programs. And, naturally, the reasoning should be done in a richer calculus of programs that contains the predicate calculus as its subset.

We believe that by constructing a new proof of correctness of Euclid's algorithm we shall gain a new insight into the nature of (algorithmic) theory of numbers. It is commonly accepted that algorithms play an important role in this theory. We need the tools adequate to the structure of analyzed texts. By this we mean that 1° the algorithms should form a third subset of the set WFF, besides the sets of terms and formulas, 2° the semantical properties of algorithms should be expressed by the formulas, 3° these formulas should be the subject of studies having as aim their proof or a counterexample.
We expect that the proofs will be inter-subjective ones. It means, that everyone reading a proof will

necessarily agree with the arguments. Finally, such a proof should be analyzable by a proof-checker [1]. The fact of incompleteness of first-order theory of natural numbers should not be used as an indulgence for our laziness.

The notion of halting formula may cause some doubts. Informally, a halting formula of an algorithm K is any formula that expresses the stopping property of the algorithm. Let $\mathcal{C}$ be a class of similar data structures. Let $\mathcal{L}$ be a language of formulas. We shall accept the following

Definition 1.1. Let K be a program. A *halting formula* of the program K is any formula $H(K) \in \mathcal{L}$ such that, for every data structure $\mathbb{A} \in \mathcal{C}$ the following conditions are equivalent

(*i*) all executions of the program K in data structure $\mathbb{A}$ are finite,

(*ii*) the formula $H(K)$ is valid in structure $\mathbb{A}$. □

We start with a suggestion: to analyze algorithms one should extend the language of mathematical theory. It means that the set of well formed expressions of a language of a theory, should contain the properly defined set of algorithms (or programs) aside of sets of terms and formulas. See figure 1. In algorithmics the natural numbers and Euclid's algorithm play a significant role. (In the most used programming languages one encounters the structure of unsigned integers, i.e. natural numbers.) Moreover, in some programming languages we find a class[2] named Int. The details of implementation can be hidden (even covered by patents). In such a case one may doubt, whether the class Int is a proper implemementation of integers. One may ask what are the properties of the operations defined by the class. In the appendix A we show a class *Cn* that satisfies the axioms of the theory of addition. We are also showing that for some objects of class *Cn* as arguments, the computations of Euclid's algorithm need not to be finite. Therefore, it seems important that a theory in which we shall conduct the correctness proof of Euclid's algorithm will not have non-standard models, see sections 3, 4,5,6. Hence, we are seeking for a categorical axiomatisation of natural numbers. One can choose among: weak second order logic, logic of infinite disjunctions $\mathcal{L}_{\omega_1\omega}$ [Eng67, Kar64] and algorithmic logic $\mathcal{AL}$[MS87]. We prefer algorithmic logic for it has a system of axioms and inference rules. Moreover in the language of algorithmic logic we have ready formulas that express the semantical properties of programs, such as termination, correctness, etc. The next question which appears is: should we look for a new set of axioms of natural numbers or perhaps the set proposed by Peano will do. We shall consider three algorithmic theories and will find which of three is suitable for conducting the correctness proof of Euclid's algorithm. The result of comparison is presented in the table 1.

The Euclid's algorithm is very important for mathematicians as well as for programmers. There is no doubt on it. However, the proofs of correctness of this algorithm do not satisfy us. Why?

One can split the goal of proving the correctness of an algorithm A with respect to the given precondition α and postcondition β onto two subgoals: 1) to prove that if some result exists then it satisfies the postcondition β, and 2) to prove that if the arguments satisfy the precondition α then the computation of the algorithm terminates. The first subgoal is easier. In the case of Euclid's algorithm, it suffices to remark that a common divisor of two numbers n and m is also a common divisor of n and the difference

[1] proof-checkers of algorithmic proofs do not exists yet

[2] class is a kind of program module, cf. Appendix A

Table 1. Which theory allows to prove the halting formula H of Euclid's algorithm?

Th	**Language, Logic**	**Axioms**	**Is there a proof?**
$\mathcal{T}h_0$	$\mathcal{L}$ – 1-st order	Peano	*No* - The formula H does not appear in the language of first-order Peano Arithmetic. There is nothing to prove. See also the stronger Fact 5.4.
$\mathcal{T}h_1$	$\mathcal{L}_A$ – algorithmic	Presburger	*No* - the halting formula is independent from the axioms of this theory. A programmable counterexample is presented. See 4.2.
$\mathcal{T}h_2$	$\mathcal{L}_A$ – algorithmic	Peano	*No* - the halting formula is independent from the axioms of this theory. See 5.3.
$\mathcal{T}h_3$	$\mathcal{L}_A$ – algorithmic	Algorithmic Arithmetic	*Yes* - there exists a proof. See theorem 7.3.

$n - m$. From this the partial correctness of Euclid's algorithm

$$\left\{\begin{array}{l} n := n_0;\ m := m_0; \\ \textbf{while } n \neq m \textbf{ do} \\ \quad \textbf{if } n > m \textbf{ then } n := n - m \textbf{ else } m := m - n \textbf{ fi} \\ \textbf{od} \end{array}\right\}$$

w.r.t. the pair of formulas $(n_0 > 0 \wedge m_0 > 0)$ and $(n = gcd(n_0, m_0))$ follows immediately.
The second subgoal is more difficult. We require that a proof of the correctness starts with axioms (either axioms of logic or axioms of natural numbers) and uses the inference rules of program calculus *(i.e.* algorithmic logic) to deduce some intermediate formulas and to terminate with the halting formula. All the proofs we know, do not satisfy this requirement. Let us take an example. In some monographs on theoretical arithmetic the proof goes as follow: 1°some intermediate formulas are proven, 2°a remark is made that any descending sequence is finite, 3°therefore for any natural numbers n and m the computation of Euclid's algorithm is finite and brings the $gcd(n, m)$.
Let us remark that in a proof like mentioned above :

(*i*) In a non-standard model of theory of addition there are infinite, descending sequences, c.f. Appendix A. One has to assume that the algorithm works in the standard model of natural numbers. However, no elementary theory can guarantee that every of its models is isomorphic to the standard one.

(*ii*) The proof analyzes some sequences of numbers saying this is an execution sequence of the algorithm. It would be preferable if the proof restrained from referring to semantics.

2. A short exposition of calculus of programs

We shall discuss three algorithmic theories $\mathcal{T}h_1, \mathcal{T}h_2, \mathcal{T}h_3$. All three theories have the same formalized language. All theories share the same consequence operation that is determined by the axioms and inference rules of program calculus, see appendix B. The theories have different sets of specific axioms.

All three algorithmic theories $\mathcal{T}h_1, \mathcal{T}h_2, \mathcal{T}h_3$ share the same language $L = \langle A, \mathcal{WFF} \rangle$. The alphabet A has the following subsets: set of functors $\Phi = \{s, P, +, *, \dot{-}\}$, set of predicates $\Theta = \{=, <\}$, set of logical operators $\{\wedge, \vee, \Rightarrow, \neg\}$, set of program operators {:=, ;, **while, if** }, and auxiliary symbols, parentheses and others. The alphabet A contains also the set of variables.
Language L is an extension of the language of theory $\mathcal{T}h_0$, it admits programs and has a larger set of formulas.

The reader familiar with the algorithmic logic [MS87] can safely skip the rest of this section. For the convenience of other readers we offer a few words on the calculus of programs and in the Appendix B we are listing axioms and inference rules of the calculus.
A formalized logic $\mathcal{L}$ is determined by its language L and the syntactic consequence operation C, $\mathcal{L} = \langle L, C \rangle$. How to describe the difference between first-order logic FOL and algorithmic logic AL? The language of algorithmic logic is a superset of the language of first-order logic, it is also a superset of deterministic while programs, moreover, it includes algorithmic formulas and is closed by the usual formation rules. In the language of AL we find all well formed expressions of FOL. The alphabets are similar. However, the language of AL contains programs and the set of formulas is richer than the set of first-order formulas.

As you can see the language $\mathcal{WFF}_{AL}$ contains programs. Moreover, the set of formulas $\mathcal{F}_{AL}$ is a proper superset of the set of first-order formulas $\mathcal{F}_{FOL}$.

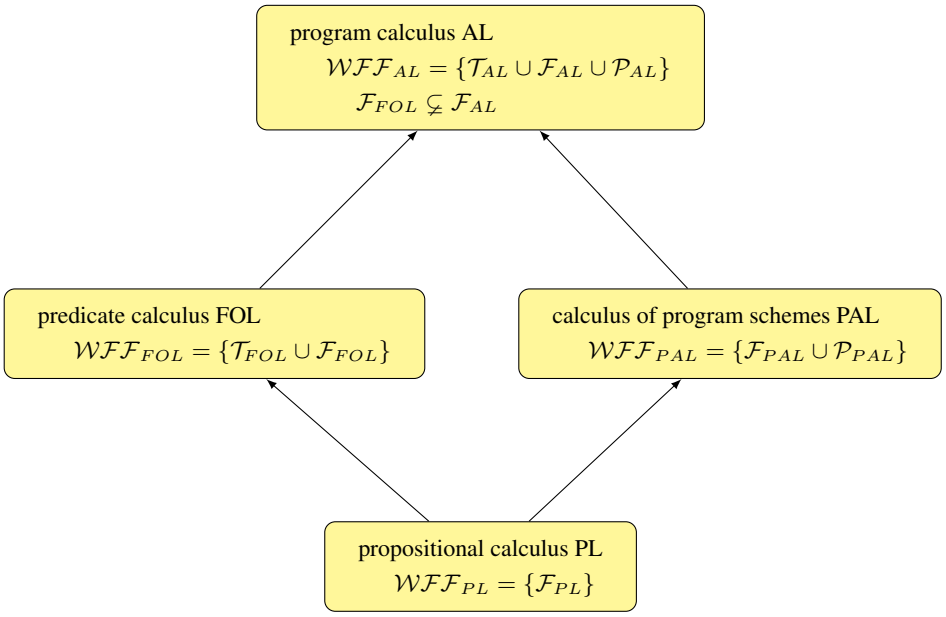


Figure 1. Comparison of logical calculi w.r.t. their $\mathcal{WFF}$ sets

The set $\mathcal{WFF}_{AL}$ of well formed expressions is the union of three sets: set of terms (programmers

may say, set of arithmetical expressions), set of formulas (i.e. set of boolean expressions) and the set of programs.

Definition 2.1. The set of *terms* is the least set of expressions T such that

- each variable x is an element of the set T,
- if an expression τ belongs to the set T, then the expressions $s(\tau),\ P(\tau)$ belong to the set T,
- if expressions τ and σ belong to the set T, then the expressions $(\tau + \sigma),\ (\tau * \sigma),\ (\tau \dot{-} \sigma)$ belong to the set T. □

The set of formulas we describe in two steps.

Definition 2.2. The set of *open formulas* is the least set F_O of expressions such that

- if expressions τ and σ are terms, then the expressions $(\tau = \sigma),\ (\tau < \sigma)$ are open formulas,
- if expressions α and β are open formulas, then the expressions $(\alpha \wedge \beta)\ (\alpha \vee \beta),\ (\alpha \Rightarrow \beta),\ \neg\alpha$ are open formulas. □

Definition 2.3. The set of *programs* (in the language of theories $\mathcal{T}h_1, \mathcal{T}h_2, \mathcal{T}h_3$) is the least set $\mathcal{P}$ of expressions, such that

- If x is a variable and an expression τ is a term, then the expression $x := \tau$ is a program. (Programs of this form are called assignment instructions. They are atomic programs.)
- if expressions K and M are programs, then the expression $\{K;\ M\}$ is a program,
- if expression γ is an open formula and expressions K and M, are programs, then the expressions **while** γ **do** M **od** and **if** γ **then** K **else** M **fi** are programs. □

We use the braces { } to delimit a program.

Definition 2.4. The set of *formulas* is the least set of expressions F such, that

- each open formula belongs to the set F,
- if an expression K is a program and an expression α is a formula, then the expression $K\,\alpha$ is a formula,
- if an expression K is a program and an expression α is a formula, then expressions $\bigcup K\,\alpha$ and $\bigcap K\,\alpha$ are formulas,
- if an expression α is a formula, then the expressions $\forall_x\,\alpha$ and $\exists_x\,\alpha$ are formulas,
- if expressions α and β are formulas, then the expressions $(\alpha \wedge \beta)\ (\alpha \vee \beta),\ (\alpha \Rightarrow \beta),\ \neg\alpha$ are formulas. □

Following Tarski we associate to each well formed expression of the language a mapping. The meanings of terms and open formulas is defined in a classical way. Semantics of programs requires the notion of computation (i.e. of execution). For the details consult [MS87]. Two facts would be helpful in reading further:

- The meaning of an algorithmic formula $K\alpha$ in a data structure $\mathfrak{A}$ is a function from the set of valuations of variables into two-element Boolean algebra B_0 defined as follow

$$(K\alpha)_{\mathfrak{A}}(v) \stackrel{df}{=} \begin{cases} \alpha_{\mathfrak{A}}(K_{\mathfrak{A}}(v)) & \text{if the result } K_{\mathfrak{A}}(v) \text{ of computation} \\ & \text{at initial valuation } v \text{ is defined,} \\ \textbf{false} & \text{otherwise i.e. if the computation} \\ & \text{of program } K \text{ fails or loops endlessly.} \end{cases}$$

  This explains why the formula H expresses the halting property of the program E.
  Define $K^i\alpha$ by induction: $K^0\alpha = \alpha$ and $K^{i+1}\alpha = KK^i\alpha$.
  We read the formula $\bigcup K\,\alpha$ as *there exists an iteration of program K such that $K^i\alpha$ holds*, and $\bigcap K\,\alpha$ means *for each iteration of program K formula $K^i\alpha$ holds*.
  The signs $\bigcup$ and $\bigcap$ are *iteration quantifiers.* The meaning of these formulas is defined as follow.

$$(\bigcup K\alpha)_{\mathfrak{A}}(v) \stackrel{df}{=} l.u.b.\ \{(K^i\alpha)_{\mathfrak{A}}(v)\}_{i\in N}$$
$$(\bigcap K\alpha)_{\mathfrak{A}}(v) \stackrel{df}{=} g.l.b.\ \{(K^i\alpha)_{\mathfrak{A}}(v)\}_{i\in N}$$

- The calculus of programs i.e. algorithmic logic, enjoys the property of completeness. For the theorem on completeness consult [MS87].

3. Elementary Peano arithmetic $\mathcal{T}h_0$

This first-order theory has been studied by many researchers. Many variants of the theory are known c.f. [Grz74]. The specific axioms of theory $\mathcal{T}h_0$ can be found in section 5.

It is obvious that the language of the theory does not contain algorithms, and hence it does not contain algorithmic formulas. Therefore the halting formula (H) of Euclid's algorithm can not be formulated nor proved in first-order theory of natural numbers with the axioms of Peano.

Still, a problem arises, is there a first-order formula, theorem of theory $\mathcal{T}h_0$, equivalent to the formula (H)?

Question 1. *Is there a first-order formula $\psi(m,n)$, such that, for every data structure A in C,*
(i) it expresses the following property: the computation of Euclid's algorithm for arguments m and n is finite – *in this case formula $\psi(m,n)$ evaluates to **true**,*
(ii) the formula $\psi(m,n)$ rejects the infinite computations in the data structure, – *i.e. it assumes the value **false** if the computation on arguments m, n can be prolonged* ad infinitum, *and*
(iii) the formula $\forall_m\forall_n\,\psi(m,n)$ is a theorem of Peano arithmetic.

An answer to the question is in section 5.

4. Algorithmic theory of addition $\mathcal{T}h_1$

We consider an algorithmic theory, henceforth its language contains programs and algorithmic formulas. Note, all axioms of this theory are first-order formulas!

Definition 4.1. The set of *specific axioms* of the theory $\mathcal{T}h_1$ consists of the following formulas:

$$\forall_x\, s(x) \neq 0 \tag{1}$$
$$\forall_x \forall_y\, s(x) = s(y) \Rightarrow x = y \tag{2}$$
$$\forall_x\, x + 0 = x \tag{3}$$
$$\forall_x \forall_y\, x + s(y) = s(x + y) \tag{4}$$
$$x < y \Leftrightarrow \exists_z y = x + s(z) \tag{5}$$
$$P(0) = 0 \tag{6}$$
$$P(s(x)) = x \tag{7}$$
$$z \dot{-} 0 = z \tag{8}$$
$$z \dot{-} s(x) = P(z \dot{-} x) \tag{9}$$

and an infinite set of formulas built in accordance with the following scheme of induction:

$$\Phi(x/0) \wedge \forall_x\big(\Phi(x) \Rightarrow \Phi(x/s(x))\big) \Rightarrow \forall_x\, \Phi(x) \tag{10}$$

The last line is a scheme of infinitely many axioms. It is the scheme of induction. The expression Φ denotes an arbitrary first-order formula with a free variable x. The expression $\Phi(x/0)$ denotes a formula resulting from the expression Φ by the replacement of all free occurrences of variable x by constant 0. Similarly, the expression $\Phi(x/s(x))$ is the formula that results from Φ by the simultaneous replacement of all free occurrences of variable x by the term $s(x)$. □

Our set of axioms differs insignificantly from those considered by Presburger. cf. [Pre29, Sta84]. Observe, that the formula H satisfies requirements *(i)* and *(ii)* of Question 1.

Fact 4.1. The formula H is not a theorem of the theory $\mathcal{T}h_1$.

$$\mathcal{T}h_1 \nvdash \mathrm{H}$$

Proof:
The formula H is falsifiable in a non-standard model $\mathfrak{N}$ of theory $\mathcal{T}h_1$, cf. Appendix A. By completeness of algorithmic logic it follows that the formula is not a theorem of algorithmic theory $\mathcal{T}h_1$. □

Fact 4.2. The formula H is independent of axioms 1 - 10.

Remark that there exist a programmable model of this theory.

5. Algorithmic Peano arithmetic – theory $\mathcal{T}h_2$

The language of this theory contains additional functor * of multiplication.

Definition 5.1. The set of axioms of the next theory $\mathcal{T}h_2$ consists of formulas 1 - 9 and two formulas 11, 12 that define the operation of multiplication. Moreover, the set of axioms contains all the formulas built in accordance with scheme of induction,

$$\forall_x\, x * 0 = 0 \tag{11}$$

$$\forall_x \forall_y\, x * s(y) = (x * y) + x \tag{12}$$

scheme of induction:

$$\Phi(x/0) \wedge \forall_x \big(\Phi(x) \Rightarrow \Phi(x/s(x))\big) \Rightarrow \forall_x\, \Phi(x)$$

As in the preceding section, we shall limit the scheme of induction: the formula $\Phi(x)$ must be a first-order formula. □

Note, that all axioms of theory $\mathcal{T}h_2$ are first-order formulas.

Fact 5.1. The theory $\mathcal{T}h_2$ has (at least) two non-isomorphic models. One is the standard model $\mathfrak{N}_0$ of theory $\mathcal{T}h_0$, another is a non-standard model $\mathfrak{N}$ of the same theory.

Let us remark that there is no recursive (i.e. programmable) non-standard model of the theory,c.f. [Ten59]. Despite the fact, that we extended the language adding the operator of multiplication and the set of axioms adding the definition of the operation of multiplication, the new theory does not contain a theorem on correctness of Euclid's algorithm. It is so because, in the non-standard model $\mathfrak{N}$ Euclid's algorithm has infinite computations for non-standard elements.
We recall, that the formula H satisfies requirements *(i)* and *(ii)* of Question 1.

Fact 5.2. The formula H is not a theorem of the theory $\mathcal{T}h_2$.

$$\mathcal{T}h_2 \nvdash \mathrm{H}$$

Fact 5.3. The formula H is independent of axioms 1 - 12.

Now, we are able to answer the question 1 of section 3. Observe that the theory $\mathcal{T}h_2$ is richer and stronger that the theory $\mathcal{T}h_0$. It is richer for its set of WFF is a proper superset of WFF set of theory $\mathcal{T}h_0$. It is stronger for it is based on bigger set of logical axioms and inference rules. Both theories have the same set of specific axioms, namely the axioms of Peano arithmetic. Both theories share the same models. Note, if the richer theory $\mathcal{T}h_2$ does not contain a theorem on correctness of Euclid's algorithm, how can such a theorem appear in the poorer and weaker theory $\mathcal{T}h_0$? We shall write down this observation as the following

Fact 5.4. Among theorems of first-order Peano arithmetic there is no formula that expresses the same meaning as formula H.

6. Algorithmic theory of natural numbers $\mathcal{T}h_3$

The set of specific axioms of the theory $\mathcal{T}h_3$ contains the following formulas:

$$\forall_x\, s(x) \neq 0 \tag{I}$$

$$\forall_x \forall_y\, s(x) = s(y) \Rightarrow x = y \tag{M}$$

$$\forall_x\, \{y := 0; \textbf{while } y \neq x \textbf{ do } y := s(y) \textbf{ od}\}(x = y) \tag{S}$$

. . . . . .

$$x + y \stackrel{df}{=} \left\{ \begin{array}{l} t := 0; w := x; \\ \textbf{while } t \neq y \textbf{ do } t := s(t); w := s(w) \textbf{ od} \end{array} \right\} w \tag{A}$$

$$x < y \stackrel{df}{\Leftrightarrow} \left\{ \begin{array}{l} w := 0; \\ \textbf{while } w \neq y \wedge w \neq x \textbf{ do} \\ \quad w := s(w) \\ \textbf{od} \end{array} \right\} (w = x \wedge w \neq y) \tag{L}$$

$$P(x) \stackrel{df}{=} \left\{ \begin{array}{l} w := 0; \\ \textbf{if } x \neq 0 \textbf{ then} \\ \quad \textbf{while } s(w) \neq x \textbf{ do } w := s(w) \textbf{ od} \\ \textbf{fi} \end{array} \right\} w \tag{P}$$

$$x \dot{-} y \stackrel{df}{=} \left\{ \begin{array}{l} w := x; t := 0; \\ \textbf{while } t \neq y \textbf{ do } t := s(t); w := P(w) \textbf{ od} \end{array} \right\} w \tag{O}$$

The third of axioms [S] is an algorithmic, (not a first-order), formula. It states that every element is a standard natural number.

In addition to these three formulas [I, M, S] we assume four more axioms [A, P, O, L] that are defining operations: *addition, predecessor, subtraction* (*respectively* $+$, P, $\dot{-}$) and ordering relation $<$. Axioms A, P, O that define operations are shorter versions of longer formulas, e.g. $x + y = z \Leftrightarrow \{t := 0; w := x; \textbf{while } t \neq y \textbf{ do } t := s(t); w := s(w) \textbf{ od}\}(w = z)$. Observe that $z = \{t := 0; w := x; \textbf{while } t \neq y \textbf{ do } t := s(t); w := s(w) \textbf{ od}\}z$ since the variable z does not occur in this program and the program always halt.

This theory $\mathcal{T}h_3$ contains a proof of the halting formula (H) of the Euclid's algorithm (E). Why? 1°) The theory is categorical: *every model* $\mathfrak{M}$ *of this theory is isomorphic to the standard model* $\mathfrak{N}_0$ of natural numbers. Hence the theory is complete. 2°) Computations of Euclid's algorithm in the structure $\mathfrak{N}_0$ are finite. This follows from the traditional proof of correctness of Euclid's algorithm. 3°) Hence, the formula (H) is valid in each model of the theory $\mathcal{T}h_3$. 4°) Therefore, there exists a proof of formula H and the formula is a theorem of the theory $\mathcal{T}h_3$.

We do not stop at the statement that the proof of the halting formula H exists. We believe that in the future attaching proofs of correctness to the software produced will become the norm. Below we are developing the proof [3]. We show its detailed (and formalizable) version.

[3] of the correctness of Euclid's algorithm in algorithmic theory of numbers

- We start by showing that all axioms of the theory $\mathcal{T}h_2$ are theorems of the theory $\mathcal{T}h_3$. Hence $\mathcal{T}h_2 \subsetneq \mathcal{T}h_3$.
- We shall prove also a couple of properties that occur in the traditional proof of Euclid's algorithm. We omit the proofs of those properties of greatest common divisor, that are theorems of the theory $\mathcal{T}h_0$. E.g. the formula $\big((n > m \wedge m > 0) \Rightarrow gcd(n,m) = gcd(n-m,m)\big)$ has a proof in the elementary theory of Peano.
- We deduce halting property of algorithm (E) from the halting property of another program S.

In the following subsection we are proving the scheme of induction.

6.1. Scheme of induction

Lemma 6.1. The following formula is a theorem of the theory $\mathcal{T}h_3$.

$$\mathcal{T}h_3 \vdash \{y := 0\}\bigcup\{y := s(y)\}(x = y)$$

Proof:
The following equivalence is a theorem of algorithmic logic cf. [MS87] p. 62.

$$\vdash \left(\begin{array}{c} \{y := 0; \mathbf{while}\, y \neq x\, \mathbf{do}\, y := s(y)\, \mathbf{od}\}(x = y) \Leftrightarrow \\ \quad \{y := 0\}\bigcup\{\mathbf{if}\, y \neq x\, \mathbf{then}\, y := s(y)\, \mathbf{fi}\}(x = y) \end{array} \right)$$

Another theorem of algorithmic logic is the following equivalence

$$\vdash \left(\begin{array}{c} \{y := 0\}\bigcup\{y := s(y)\}(x = y) \Leftrightarrow \\ \quad \{y := 0\}\bigcup\{\mathbf{if}\, y \neq x\, \mathbf{then}\, y := s(y)\, \mathbf{fi}\}(x = y). \end{array} \right)$$

By propositional calculus we have

$$\vdash \left(\begin{array}{c} \{y := 0; \mathbf{while}\, y \neq x\, \mathbf{do}\, y := s(y)\, \mathbf{od}\}(x = y) \Leftrightarrow \\ \{y := 0\}\bigcup\{y := s(y)\}(x = y). \end{array} \right)$$

By modus ponens we obtain

$$\mathcal{T}h_3 \vdash \{y := 0\}\bigcup\{y := s(y)\}(x = y).$$

□

Lemma 6.2. The following equivalences are theorems of algorithmic logic.

$$\vdash \{y := 0\}\bigcup\{y := s(y)\}\alpha(y) \Leftrightarrow \{x := 0\}\bigcup\{x := s(x)\}\alpha(x)$$
$$\vdash \{y := 0\}\bigcap\{y := s(y)\}\alpha(y) \Leftrightarrow \{x := 0\}\bigcap\{x := s(x)\}\alpha(x)$$

Proof:
Let $\alpha(x)$ be an arbitrary formula with free variable x. The expression $\alpha(y)$ denotes the formula resulting from the formula $\alpha(x)$ by the simultaneous replacement of all free occurrences of the variable x by the variable y. It is easy to remark, that for every natural number $i \in N$ the following formula is a theorem

$$\vdash \alpha(y/s^i(0)) \Leftrightarrow \alpha(x/s^i(0)).$$

By the axiom Ax_{14} of the assignment instruction we obtain another fact , for every natural number $i \in N$ the following formula is a theorem

$$\{y := 0\}\{y := s(y)\}^i \alpha(y) \Leftrightarrow \{x := 0\}\{x := s(x)\}^i \alpha(x)$$

Now, we apply the axiom Ax_{23} and obtain, that for every natural number i the following formula is a theorem.

$$\{y := 0\}\{y := s(y)\}^i \alpha(y) \Rightarrow \{x := 0\}\bigcup\{x := s(x)\} \alpha(x)$$

We are ready to apply the rule R_4. We obtain the theorem

$$\{y := 0\}\bigcup\{y := s(y)\} \alpha(y) \Rightarrow \{x := 0\}\bigcup\{x := s(x)\} \alpha(x).$$

In a similar manner we are proving the other implication and the formula

$$\{y := 0\}\bigcap\{y := s(y)\} \alpha(y) \Leftrightarrow \{x := 0\}\bigcap\{x := s(x)\} \alpha(x).$$

□

In the proof of scheme of induction we shall use the following theorem.

Metatheorem 6.1. For every formula α the following formulas are theorems of algorithmic theory of natural numbers.

$$\mathcal{T}h_3 \vdash \forall_x \alpha(x) \quad \Leftrightarrow \quad \{x := 0\}\bigcap\{x := s(x)\} \alpha(x) \tag{13}$$

$$\mathcal{T}h_3 \vdash \exists_x \alpha(x) \quad \Leftrightarrow \quad \{x := 0\}\bigcup\{x := s(x)\} \alpha(x) \tag{14}$$

Proof:
We shall prove the property (14). Let $\alpha(x)$ be a formula.
Every formula of the following form is a theorem of algorithmic logic.

$$\vdash \alpha(x) \Rightarrow \alpha(x) \wedge \{y := 0\}\bigcup\{y := s(y)\}(x = y).$$

This leads to the following theorem of the theory $\mathcal{T}h_3$.

$$\mathcal{T}h_3 \vdash \alpha(x) \Rightarrow \{y := 0\}\bigcup\{y := s(y)\}(\alpha(x) \wedge x = y).$$

In the next step we obtain.

$$\mathcal{T}h_3 \vdash \alpha(x) \Rightarrow \{y := 0\}\bigcup\{y := s(y)\}\alpha(y).$$

Now, we can introduce the existential quantifier into the antecedent of the implication (we use inference rule R6).

$$\mathcal{T}h_3 \vdash \exists_x \alpha(x) \Rightarrow \{y := 0\} \bigcup \{y := s(y)\} \alpha(y).$$

By the previous lemma 6.2 we obtain.

$$\mathcal{T}h_3 \vdash \exists_x \alpha(x) \Rightarrow \{x := 0\} \bigcup \{x := s(x)\} \alpha(x).$$

The proof of other implication as well as of formula (13) is left as an exercise. □

We are going to prove the scheme of induction.

Metatheorem 6.2. Let $\alpha(x)$ denote an arbitrary formula with a free variable x. The formula built in accordance with the following scheme is a theorem of algorithmic theory of natural numbers $\mathcal{T}h_3$.

$$\mathcal{T}h_3 \vdash \Big(\alpha(x/0) \wedge \forall_x \big(\alpha(x) \Rightarrow \alpha(x/s(x))\big)\Big) \Rightarrow \forall_x \alpha(x) \tag{15}$$

Proof:
In the expression below, β denotes a formula, K denotes a program. Each formula of the form

$$\vdash ((\beta \wedge \bigcap K\,(\beta \Rightarrow K\beta)) \Rightarrow \bigcap K\,\beta)$$

is a theorem of calculus of programs, i.e. algorithmic logic (cf.[MS87] p.71(8)).
Hence, every formula of the following form is a theorem of algorithmic logic.

$$\begin{aligned} \vdash ((\alpha(x) \wedge \bigcap \{x := s(x)\}\,(\alpha(x) \Rightarrow \\ \{x := s(x)\}\alpha(x))) \Rightarrow \bigcap \{x := s(x)\}\,\alpha(x)) \end{aligned}$$

We apply the auxiliary inference rule $R2$

$$\frac{\alpha, K\,\mathbf{true}}{K\,\alpha} \tag{R2}$$

(it is easy to deduce this rule from the rule R_2) and obtain another theorem of AL

$$\begin{aligned} \vdash \{x := 0\}((\alpha(x) \wedge \bigcap \{x := s(x)\}\,(\alpha(x) \Rightarrow \\ \{x := s(x)\}\alpha(x))) \Rightarrow \bigcap \{x := s(x)\}\,\alpha(x)) \end{aligned}$$

Assignment instruction distributes over conjunction (Ax15) and implication(cf. [MS87]p.70 formula (4)), hence

$$\begin{aligned} \vdash (((\{x := 0\}\alpha(x) \wedge \{x := 0\} \bigcap \{x := s(x)\}\,(\alpha(x) \Rightarrow \\ \{x := s(x)\}\alpha(x))) \Rightarrow \{x := 0\} \bigcap \{x := s(x)\}\,\alpha(x)) \end{aligned}$$

We apply the axiom of assignment instruction

$$\begin{aligned} \vdash (\alpha(x/0) \wedge \{x := 0\} \bigcap \{x := s(x)\}\,(\alpha(x) \Rightarrow \alpha(x/s(x)))) \Rightarrow \\ \{x := 0\} \bigcap \{x := s(x)\}\,\alpha(x)) \end{aligned}$$

Now, we use the fact that in the algorithmic theory of natural numbers the classical quantifiers and iteration quantifiers are mutually expressive. (cf. formula (13))

$$\vdash (\alpha(x/0) \wedge \underbrace{\{x := 0\}\bigcap\{x := s(x)\}}_{\forall_x} (\alpha(x) \Rightarrow \alpha(x/s(x)))) \Rightarrow \underbrace{\{x := 0\}\bigcap\{x := s(x)\}}_{\forall_x} \alpha(x))$$

and obtain scheme of induction – each formula of the following scheme is a theorem of algorithmic theory of natural numbers $\mathcal{T}h_3$.

$$\mathcal{T}h_3 \vdash (\alpha(x/0) \wedge (\forall x)(\alpha(x) \Rightarrow \alpha(x/s(x)))) \Rightarrow (\forall x)\alpha(x))$$

□

Observe the following useful property of natural numbers. Many proofs use the following lemma.

Lemma 6.3. Let α be any formula. Any equivalence built in accordance to the following scheme is a theorem of theory $\mathcal{T}h_3$

$$\mathcal{T}h_3 \vdash \left\{\begin{array}{l} t := 0; \\ \textbf{while } t \neq s(y) \\ \textbf{do} \\ \quad t := s(t); \\ \textbf{od} \end{array}\right\} \alpha \Leftrightarrow \left\{\begin{array}{l} t := 0; \\ \textbf{while } t \neq y \\ \textbf{do} \\ \quad t := s(t); \\ \textbf{od}; \\ \textbf{if } t \neq s(y) \\ \textbf{then } t := s(t); \\ \textbf{fi} \end{array}\right\} \alpha.$$

Proof:
The proof makes use of the following theorem of AL

$$\vdash \left\{\begin{array}{l} t := 0; \\ \textbf{while } t \neq s(y) \\ \textbf{do} \\ \quad t := s(t); \\ \textbf{od} \end{array}\right\} \alpha \Leftrightarrow \left\{\begin{array}{l} t := 0; \\ \textbf{while } t \neq s(y) \\ \textbf{do} \\ \quad t := s(t); \\ \textbf{od}; \\ \textbf{if } t \neq s(y) \\ \textbf{then } t := s(t); \\ \textbf{fi} \end{array}\right\} \alpha.$$

where α is any formula

and the axiom (M). It suffices to consider the formulas α of the form $\beta \wedge t = s(y)$ without loss of generality. □

The lemma can be formulated in another way: the programs occurring in the lemma 6.3 are equivalent.

6.2. Addition

The operation of addition is defined in the theory $\mathcal{T}h_3$ as follows.

Definition 6.1.

$$x + y = z \Leftrightarrow \left\{ \begin{array}{l} t := 0; w := x; \\ \textbf{while } t \neq y \textbf{ do } t := s(t); w := s(w) \textbf{ od} \end{array} \right\} (z = w) \qquad \text{(A)}$$

We have to check whether the definition is correct. It means that we should prove that for any pair of values x, y there exists a result. Moreover, we should prove that the result is unique and that it satisfies the recursive equalities $x + 0 = x$ and $x + s(y) = s(x + y)$. First, we remark that for every x and y the result w of addition is defined.

Lemma 6.4.

$$\mathcal{T}h_3 \vdash \forall_x \forall_y \left\{ \begin{array}{l} t := 0; w := x; \\ \textbf{while } t \neq y \textbf{ do } t := s(t); w := s(w) \textbf{ od} \end{array} \right\} (t = y)$$

Proof:
Proof starts with the axiom (S). Next, we use the following auxiliary inference rule of AL, cf. [MS87] p. 73(19).

$$\frac{\{t := 0;\ \textbf{while } t \neq y \textbf{ do } t := s(t) \textbf{ od}\}\alpha}{\{t := 0;\ \textbf{while } t \neq y \textbf{ do } t := s(t); w := s(w) \textbf{ od}\}\alpha}$$

Thus we proved the following theorem

$$\mathcal{T}h_3 \vdash \{t := 0;\ \textbf{while } t \neq y \textbf{ do } t := s(t); w := s(w) \textbf{ od}\}(t = y)$$

The latter formula can be preceded by the assignment instruction $w := x$ (we use the inference rule R2).

$$\mathcal{T}h_3 \vdash \left\{ \begin{array}{l} w := x; t := 0; \\ \textbf{while } t \neq y \textbf{ do } t := s(t); w := s(w) \textbf{ od} \end{array} \right\} (t = y)$$

In the next step we may interchange two assignment instructions, for they have no common variables.

$$\mathcal{T}h_3 \vdash \left\{ \begin{array}{l} t := 0; w := x; \\ \textbf{while } t \neq y \textbf{ do } t := s(t); w := s(w) \textbf{ od} \end{array} \right\} (t = y)$$

Finally we can add the quantifiers (rule R7) and obtain the thesis of lemma.

$$\mathcal{T}h_3 \vdash \forall_x \forall_y \left\{ \begin{array}{l} t := 0; w := x; \\ \textbf{while } t \neq y \textbf{ do } t := s(t); w := s(w) \textbf{ od} \end{array} \right\} (t = y).$$

□

Our next observation is

Lemma 6.5.

$$\mathcal{T}h_3 \vdash x + 0 = x$$

Proof:
From the definition we have

$$x + 0 = z \Leftrightarrow \left\{ \begin{array}{l} t := 0; w := x; \\ \textbf{while } t \neq 0 \textbf{ do } t := s(t); w := s(w) \textbf{ od} \end{array} \right\} (z = w)$$

We apply the axiom Ax_{21} of while instruction to obtain

$$x + 0 = z \Leftrightarrow \left\{ \begin{array}{l} t := 0; w := x; \\ \textbf{if } t = y \textbf{ then else} \\ \quad \textbf{while } t \neq 0 \textbf{ do } t := s(t); w := s(w) \textbf{ od} \\ \textbf{fi} \end{array} \right\} w$$

Indeed, from the properties of while instruction we obtain the implication.

$$\mathcal{T}h_3 \vdash y = 0 \Rightarrow \left\{ \begin{array}{l} t := 0; w := x; \\ \textbf{while } t \neq y \textbf{ do} \\ \quad t := s(t); w := s(w) \\ \textbf{od} \end{array} \right\} \alpha \Leftrightarrow \{t := 0; w := x; \}\alpha.$$

We conclude that $\mathcal{T}h_3 \vdash x + 0 = x$. □

Our next goal is

Lemma 6.6.

$$\mathcal{T}h_3 \vdash x + s(y) = s(x + y)$$

Proof:
Proof uses the equivalence:

$$\left\{\begin{array}{l} t := 0; \\ w := x; \\ \textbf{while}\, t \neq s(y) \\ \textbf{do} \\ \quad t := s(t); \\ \quad w := s(w) \\ \textbf{od} \end{array}\right\} \alpha \Leftrightarrow \left\{\begin{array}{l} t := 0; \\ w := x; \\ \textbf{while}\, t \neq y \\ \textbf{do} \\ \quad t := s(t); \\ \quad w := s(w) \\ \textbf{od}; \\ \textbf{if}\, t \neq s(y) \\ \textbf{then}\, t := s(t); w := s(w); \\ \textbf{fi} \end{array}\right\} \alpha.$$

The expression α is any formula. The equivalence is an instance of the lemma 6.3.

□

6.3. Definition of relation $<$

Definition 6.2.

$$x < y \stackrel{df}{=} \left\{\begin{array}{l} w := 0; \\ \textbf{while}\, w \neq y \wedge w \neq x\, \textbf{do}\, w := s(w)\, \textbf{od} \end{array}\right\} (w = x \wedge w \neq y).$$

We shall prove the useful property.

Lemma 6.7.

$$\mathcal{T}h_3 \vdash \forall_x \forall_y \, (x < y \vee x = y \vee y < x).$$

Proof:
It follows from the axiom (S), that

$$\mathcal{T}h_3 \vdash \forall_x \forall_y \left\{\begin{array}{l} w := 0; \\ \textbf{while}\, w \neq y \wedge w \neq x \\ \textbf{do}\, w := s(w)\, \textbf{od} \end{array}\right\} \left(\begin{array}{l} w = x \wedge w \neq y \vee w \neq y \wedge \\ w = y \vee x = y \end{array}\right).$$

because, the formula $\big((w = x \wedge w \neq y) \vee (w \neq y \wedge w = y) \vee x = y\big)$ is a theorem of AL and the implication $(w \neq y \wedge w \neq x) \Rightarrow w \neq x$ follows from axiom Ax_6.

From the axiom of algorithmic logic Ax_{15} we deduce

$$\mathcal{T}h_3 \vdash \forall_x \forall_y \Big(\left\{ \begin{array}{l} w := 0; \\ \textbf{while}\, w \neq y \wedge w \neq x \\ \textbf{do}\, w := s(w)\, \textbf{od} \end{array} \right\} (w = x \wedge w \neq y)$$

$$\vee \left\{ \begin{array}{l} w := 0; \\ \qquad \textbf{while}\, w \neq y \wedge w \neq x \\ \textbf{do}\, w := s(w)\, \textbf{od} \end{array} \right\} (w \neq y \wedge w = y)$$

$$\vee \left\{ \begin{array}{l} w := 0; \\ \textbf{while}\, w \neq y \wedge w \neq x \\ \textbf{do}\, w := s(w)\, \textbf{od} \end{array} \right\} (x = y) \Big).$$

It is easy to observe, that the first and second line of the above formula are the definitions of relations $x < y$ and $y < x$. We can skip the program in the third line for 1° the program always terminates and 2° the values of variables x and y are not changed by the program[4].
Finally we obtain $\mathcal{T}h_3 \vdash \forall_x \forall_y (x < y \vee x = y \vee y < x)$. □

Lemma 6.8.

$$x < y \Leftrightarrow \exists_z\, y = x + s(z)$$

Proof:
We first recall the axiom (S) of the theory $\mathcal{T}h_3$

$$\mathcal{T}h_3 \vdash \{w := 0; \textbf{while}\, w \neq y\, \textbf{do}\, w := s(w)\, \textbf{od}\}(w = y).$$

From the definition of the predicate $<$ we obtain

$$\mathcal{T}h_3 \vdash x < y \Rightarrow \left\{ \begin{array}{l} w := 0; \\ \textbf{while}\, w \neq y \wedge w \neq x \\ \textbf{do}\, w := s(w)\, \textbf{od} \end{array} \right\} (w = x \wedge w \neq y).$$

Therefore

$$\mathcal{T}h_3 \vdash x < y \Rightarrow \left\{ \begin{array}{l} w := 0; \\ \textbf{while}\, w \neq y \wedge w \neq x \\ \textbf{do}\, w := s(w)\, \textbf{od} \end{array} \right\} (w = y).$$

Since $x \neq y$, hence the assignment instruction `W:=S(W)` will be executed at least once. Speaking more precisely, the former formula is equivalent to the following one by the axiom Ax_{21}.

$$\mathcal{T}h_3 \vdash x < y \Rightarrow \left\{ \begin{array}{l} w := x; w := s(w); \\ \textbf{while}\, w \neq y\, \textbf{do}\, w := s(w)\, \textbf{od} \end{array} \right\} (w = y).$$

□

[4]From the axiom S one can easily deduce the formula $\{w := 0; \textbf{while}\, w \neq y \wedge w \neq x\, \textbf{do}\, w := s(w)\, \textbf{od}\}(w = y \vee w = x)$ and the following formula $(x = k) \Rightarrow \{w := 0; \textbf{while} w \neq y \textbf{do} w := s(w) \textbf{od}\}(x = k)$.

Lemma 6.9.

$$\mathcal{T}h_3 \vdash \forall_x\, x < s(x) \tag{16}$$

$$\mathcal{T}h_3 \vdash (x < y) \Leftrightarrow \left\{ \begin{array}{l} w := x; \\ \textbf{while } w \neq y \\ \textbf{do } w := s(w) \textbf{ od} \end{array} \right\} (w = y) \tag{17}$$

$$\mathcal{T}h_3 \vdash \forall_x \forall_y\, x < y \Rightarrow x + z < y + z \tag{18}$$

6.4. Predecessor

The operation of predecessor is defined by the following axiom P.

Definition 6.3.

$$P(x) \stackrel{df}{=} \left\{ \begin{array}{l} w := 0; \\ \textbf{if } x \neq 0 \textbf{ then} \\ \quad \textbf{while } s(w) \neq x \textbf{ do } w := s(w) \textbf{ od} \\ \textbf{fi} \end{array} \right\} (w) \tag{P}$$

Lemma 6.10.

$$\mathcal{T}h_3 \vdash P(0) = 0 \tag{19}$$

$$\mathcal{T}h_3 \vdash x \neq 0 \Rightarrow s(P(x)) = x \tag{20}$$

$$\mathcal{T}h_3 \vdash x \neq 0 \Rightarrow P(x) < x \tag{21}$$

$$\mathcal{T}h_3 \vdash (x < y) \Leftrightarrow \left\{ \begin{array}{l} w := y; \\ \textbf{while } w \neq x \\ \textbf{do } w := P(w) \textbf{ od} \end{array} \right\} (w = x) \tag{22}$$

$$\mathcal{T}h_3 \vdash P(s(x)) = x \tag{23}$$

$$\text{For every natural number } i \qquad \mathcal{T}h_3 \vdash P^i(s^i(x) = x \tag{24}$$

We shall prove the fundamental property of the predecessor operator.

Theorem 6.3.

$$\mathcal{T}h_3 \vdash \forall_x \{\textbf{while } x \neq 0 \textbf{ do } x := P(x) \textbf{ od}\}(x = 0)$$

Proof:
For every $i \in N$ the following formula is a theorem of AL

$$\forall_x \left(\begin{array}{c} \{y := 0; (\textbf{if } y \neq x \textbf{ then } y := s(y) \textbf{ fi})^i\}(x = y) \Rightarrow \\ \{y := 0; (\textbf{if } y \neq x \textbf{ then } y := s(y) \textbf{ fi})^i\}(x = y) \end{array} \right).$$

We use the scheme of mathematical induction and the lemma 6.10(24), to prove that for every $i \in N$, the following formula is a theorem of the theory $\mathcal{T}h_3$

$$\forall_x \left(\begin{array}{c} \{y := 0; (\mathbf{if}\ y \neq x\ \mathbf{then}\ y := s(y)\ \mathbf{fi})^i\}(x = y) \Rightarrow \\ \{(\mathbf{if}\ x \neq 0\ \mathbf{then}\ x := P(x)\ \mathbf{fi})^i\}(x = 0) \end{array} \right).$$

Note, the antecedent in each implication asserts $x = s^i(0)$, and the successor of the implication asserts $0 = P^i(x)$.
Hence we can apply the axiom Ax_{21} of AL and obtain that for every $i \in N$

$$\mathcal{T}h_3 \vdash \forall_x \left(\begin{array}{c} \{y := 0; (\mathbf{if}\ y \neq x\ \mathbf{then}\ y := s(y)\ \mathbf{fi})^i\}(x = y) \Rightarrow \\ \{\mathbf{while}\ x \neq 0\ \mathbf{do}\ x := P(x)\ \mathbf{od}\}(x = 0) \end{array} \right).$$

Now, we apply the inference rule R_6 to obtain

$$\mathcal{T}h_3 \vdash \forall_x \left(\begin{array}{c} \{y := 0; \mathbf{while}\ y \neq x\ \mathbf{do}\ y := s(y)\ \mathbf{od}\}(x = y) \Rightarrow \\ \{\mathbf{while}\ x \neq 0\ \mathbf{do}\ x := P(x)\ \mathbf{od}\}(x = 0) \end{array} \right).$$

The antecedent of this implication is the axiom (S) of natural numbers. We deduce (by the rule R_1)
$\mathcal{T}h_3 \vdash \forall_x \{\mathbf{while}\, x \neq 0\, \mathbf{do}\, x := P(x)\, \mathbf{od}\}(x = 0)$. □

Remark. This theorem states that Euclid's algorithm halts if one of its arguments is one. Below,we shall prove the halting property in general case.
Another remark. Every model $\mathfrak{M}$ of the theory $\mathcal{T}h_1$ such that Euclid's algorithm halts when one of arguments is equal 1, is isomorphic to the standard model $\mathfrak{N}_0$ of natural numbers.
End of remarks.

We need the following inference rule

Lemma 6.11.
Let τ be a term such that no variable of a program M occurs in it, $Var(\tau) \cap Var(M) = \emptyset$. If the formula $\big((x = \tau) \Rightarrow M\,(x = P(\tau))\big)$ is a theorem of the theory $\mathcal{T}h_3$, then the formula
$\{\mathbf{while}\ x \neq 0\ \mathbf{do}\ M\ \mathbf{od}\}(x = 0)$ is a theorem of the theory too. Hence, the following inference rule is sound

$$\mathcal{T}h_3 \vdash \frac{(x = \tau) \Rightarrow M\,(x = P(\tau))}{\{\mathbf{while}\ x \neq 0\ \mathbf{do}\ M\ \mathbf{od}\}(x = 0)}$$

in the theory $\mathcal{T}h_3$

Proof:
For every i the following formula is a theorem of AL

$$\{x := P(x)\}^i(x = 0) \Rightarrow \{M\}^i(x = 0).$$

We are using the premise $((x = k) \Rightarrow \{M\}(x = P(k))$. Hence, for every $i \in N$

$$\{x := P(x)\}^i(x = 0) \Rightarrow \{\mathbf{while}\ x \neq 0\ \mathbf{do}\ M\ \mathbf{od}\}(x = 0).$$

Now, we apply the rule R_3 and obtain

$$\{\mathbf{while} x \neq 0 \ \mathbf{do}\ x := P(x)\ \mathbf{od}\}(x = 0) \Rightarrow \{\mathbf{while}\ x \neq 0\ \mathbf{do}\ M\ \mathbf{od}\}(x = 0).$$

The antecedent of this implication has been proved earlier (Th 6.3), We apply the rule R_1 and finish the proof. □

The following lemma is useful in the proof of the main theorem.

Lemma 6.12. The following inference rule is sound in the theory $\mathcal{T}h_3$:

$$\mathcal{T}h_3 \vdash \frac{(x = k) \Rightarrow M\,(x < P(k))}{\{\mathbf{while}\ x \neq 0\ \mathbf{do}\ M\ \mathbf{od}\}(x = 0)}$$

Proof:
The proof is similar to the proof of preceding lemma. We leave it as an exercise. □

Corollary 6.1. Let x be an arbitrary number $x \in N$. Each descending sequence such that $a_1 = x$ and for every $i,\ a_{i+1} < a_i$, is finite and contains at most x elements.

6.5. Subtraction

The operation of subtraction is defined by the following axiom O.

Definition 6.4.

$$x \dot{-} y \stackrel{df}{=} \{w := x; t := 0; \mathbf{while}\ t \neq y\ \mathbf{do}\ t := s(t); w := P(w)\ \mathbf{od}\}(w) \tag{O}$$

Lemma 6.13.

$$\mathcal{T}h_3 \vdash \forall_x\, x \dot{-} 0 = x \tag{25}$$
$$\mathcal{T}h_3 \vdash \forall_x \forall_y\, x \dot{-} s(y) = P(x \dot{-} y) \tag{26}$$
$$\mathcal{T}h_3 \vdash \forall_x \forall_y\, (x > y > 0) \Rightarrow x \dot{-} y < x \tag{27}$$
$$\mathcal{T}h_3 \vdash \forall_x \forall_y\, (x < y) \Rightarrow x \dot{-} y = 0 \tag{28}$$

7. Proof of correctness of Euclid's algorithm.

The proof splits on two subgoals:

(*i*) to prove that for any natural numbers n and m, the computation of Euclid's algorithm is finite, i.e. we are to prove that the halting formula H is a theorem of the theory $\mathcal{T}h_3$,

(*ii*) to prove that the algorithm computes the greatest common divisor of numbers n and m.

It is rather easy to prove the following fact

Fact 7.1.

$$\mathcal{T}h_3 \vdash \left(\begin{array}{l} n \neq m \wedge \\ max(n,m) = p \end{array} \right) \Rightarrow \left\{ \begin{array}{l} \textbf{if}\, n > m \\ \textbf{then} \\ \quad n := n \dot{-} m \\ \textbf{else} \\ \quad m := m - n \\ \textbf{fi} \end{array} \right\} (max(n,m) < p) \tag{29}$$

Proof:
In the proof we use the axiom Ax_{20} and lemma 27. □

Now, by lemma 6.12, we obtain the desired formula H. Hence the computations of Euclid's algorithm, in any structure that is a model of theory $\mathcal{T}h_3$, are finite.
It remains to be proved

Fact 7.2.

$$\mathcal{T}h_3 \vdash \Big((gcd(n,m) = p) \Rightarrow \left\{ \begin{array}{l} \textbf{if}\, n > m \\ \textbf{then} \\ \quad n := n \dot{-} m \\ \textbf{else} \\ \quad m := m \dot{-} n \\ \textbf{fi} \end{array} \right\} (gcd(n,m) = p) \Big). \tag{30}$$

In the proof we use a few useful facts

$$n > m \Rightarrow gcd(n,m) = gcd(n \dot{-} m, m)$$
$$m > n \Rightarrow gcd(n,m) = gcd(n, m \dot{-} n)$$
$$n = m \Rightarrow gcd(n,m) = n$$

All three implications are well known and we do not replicate their proofs here cf. [Grz71].
Combining these observations with formulas 29 and 30 we come to the conclusion that the following formula expressing the correctness of Euclid's algorithm is a theorem of theory $\mathcal{T}h_3$

Theorem 7.3.

$$\mathcal{T}h_3 \vdash \left(\begin{array}{l} n_0 > 0 \wedge \\ m_0 > 0 \end{array} \right) \Rightarrow \left\{ \begin{array}{l} n := n_0;\ m := m_0; \\ \textbf{while}\, n \neq m\, \textbf{do} \\ \quad \textbf{if}\, n > m \\ \quad \textbf{then} \\ \quad\quad n := n \dot{-} m \\ \quad \textbf{else} \\ \quad\quad m := m \dot{-} n \\ \quad \textbf{fi} \\ \textbf{od} \end{array} \right\} (n = gcd(n_0, m_0)) \tag{31}$$

This ends the proof.

7.1. Moreover

Making use of the lemma 6.12 we note another theorem of the theory $\mathcal{T}h_3$. It says that all computations of the following program are finite

Theorem 7.4.

$$\mathcal{T}h_3 \vdash \Big((n > m) \Rightarrow \left\{ \begin{array}{l} r := n; \\ \textbf{while}\, r \geq m\, \textbf{do} \\ \quad r := r \dot{-} m; \\ \textbf{od} \end{array} \right\} (0 \leq r < m)\Big)$$

This leads to another

Theorem 7.5.

$$\mathcal{T}h_3 \vdash \Big((n > m) \Rightarrow \left\{ \begin{array}{l} r := n; q := 0; \\ \textbf{while}\, r \geq m\, \textbf{do} \\ \quad r := r \dot{-} m; \\ \quad q := q + 1 \\ \textbf{od} \end{array} \right\} (0 \leq r < m \wedge n = q * m + r)\Big)$$

And the following

Theorem 7.6.

$$\mathcal{T}h_3 \vdash \Big((n_0 > m_0) \Rightarrow \left\{ \begin{array}{l} n := n_0;\ m := m_0;\ r := n; \\ \textbf{while}\, r \neq 0\, \textbf{do} \\ \quad r := n; \\ \quad \textbf{while}\, r \geq m\, \textbf{do} \\ \quad\quad r := r \dot{-} m \\ \quad \textbf{od}; \\ \quad n := m; \\ \quad m := r \\ \textbf{od} \end{array} \right\} (n = gcd(n_0, m_0))\Big)$$

8. Final remarks

So far, we succeeded in developing one small chapter of algorithmic theory of natural numbers. The whole theory contains much more theorems. Some are first-order formulas, some are algorithmic formulas. The theorem on correctness of Euclid's algorithm is deduced from a couple of earlier theorems. The algorithmic theory of numbers does not begin nor does it end by this theorem. We claim that the calculus

of programs (i.e. algorithmic logic) is a useful tool in building the algorithmic theory of numbers. Note the Fact 5.4. Think of its consequences.

One has to take into consideration that the future development of algorithmic theory of numbers will demand to analyze more complicated algorithms – the intuitive way of describing computations may happen to be error prone or leading to paradoxes. On the other hand, it is very probable that, in programming, we shall encounter some erroneous (or fake) classes that pretend to implement the structure of unsigned integer (i.e. natural numbers)[5].

We hope, that programmers and computer scientists will note that *proving of programs* need not to start a new, with every program one wishes to analyze. In the process of proving some semantical property s_P of a certain program P, one can use lemmas and theorems on other semantical properties of programs, that have been proved earlier. We demonstrate this pattern within the correctness proof of Euclid's algorithm. In other words, we propose to develop the algorithmic theory of natural numbers. In fact, we did it in the book [MS87] p. 155. Such a theory may be of interest also to mathematicians. One can note the appearance of books on algorithmic theory of numbers [BS96], algorithmic theory of graphs[McH90], etc. We are offering calculus of programs i.e. algorithmic logic as a tool helpful in everyday work of informaticians and mathematicians.

Acknowledgments

Grażyna Mirkowska read the manuscript and made several helpful suggestions.

Appendix A - a class implementing nonstandard model of theory $\mathcal{T}h_1$

In this section we present a class *Cn* that implements a programmable and non-standard model $\mathfrak{M}$ of axioms of addition theory $\mathcal{T}h_1$ (cf. section. 4). We show that Euclid's algorithm, executed in this model has infinite computations.

It is well known that the set of axioms of the theory $\mathcal{T}h_1$ has non-standard models. We are reminding that the system

$$\mathfrak{M} = \langle M, zero, one, s, add, subtract; equal, less\rangle$$

where

- The set M is defined as follow

  $$\langle k, x\rangle \in M \Leftrightarrow \{k \in \mathcal{Z} \wedge x \in \mathcal{R} \wedge x \geq 0 \wedge (x = 0 \Rightarrow k \geq 0)\}$$

  here k is an integer, x is a non-negative rational number and when x is 0 then $k \geq 0$,
- the operation addition is defined component wise, as usual in a product,
- the successor operation is defined as follow $s(\langle k, x\rangle) = \langle k + 1, x\rangle$,

[5]In computer algebra and in computer geometry one may need Euclid's algorithm executing in a class that implements the algebra of polynomials or structure of segments in a three dimensional space. It is of importance to check that such a class is not a non-standard model similar to our class Cn of Appendix A.

- constant zero 0 is $\langle 0, 0\rangle$.
- relation *less* has a type $\omega + (\omega^* + \omega) \cdot \eta$

is a non-standard and recursive(i.e. computable) model of axioms of theory $\mathcal{T}h_1$.

Now, class *Cn* is written in Loglan programming language [SZ13]. This class defines and implements an algebraic structure $\mathfrak{C}$. The universe of the structure consists of all objects of the class NCN (this is an infinite set). Operations in the structure $\mathfrak{C}$ are defined by the methods of class Cn: *add, equal, zero* and *s*. All the axioms of the algorithmic theory $\mathcal{T}h_1$ are valid in the structure $\mathfrak{C}$, i.e. the structure is a model of the theory. We show that for some data the execution of Euclid's algorithm is infinite.

```
unit Cn: class;
  unit NSN : class(intpart, nomprt, denom : integer);
  begin
    if nomprt = 0 and intpart < 0 orif nomprt * denom < 0 orif denom = 0
    then raise Exception fi
  end NSN;
  unit add : function(n, m : NSN) : NSN;
  begin result := new NSN(n.intpart + m.intpart,
    n.nomprt * m.denom + n.denom * m.nomprt, n.denom * m.denom) end add;
  unit subtract : function(n, m : NSN) : NSN;
  begin if less(n, m) then result := zero
      else result := newNSN(n.intprt - m.intprt,
        n.nomprt * m.denom - n.denom * m.nomprt, n.denom * m.denom) fi
  end subtract;
  unit equal : function(n, m : NSN) : Boolean;
  begin   result := (n.intpart = m.intpart) and
    (n.nomprt * m.denom = n.denom * m.nomprt) end equal;
  unit zero : function : NSN;
  begin result := new NSN(0, 0, 1) end zero;
  unit s : function(n : NSN) : NSN;
  begin result := new NSN(n.intpart + 1, n.nomprt, n.denom) end s;
  unit less : function(n, m : NSN) : Boolean;
  begin if n.nomprt = 0 andif m.nomprt = 0 then result := n.intprt < m.intprt
    else if n.nomprt = 0 andif m.nomprt > 0 then result := true
     else if n.nomprt > 0 andif m.nomprt = 0 then result := false
       else if n.intprt = / = m.intprt then result := n.intprt < m.intprt
         else result := n.nomprt * m.denom < n.denom * m.nomprt fi fi fi fi end less
end Cn;
```

Theorem 8.1. The algebraic structure $\mathfrak{C}$ which consists of the set $|NSN|$ of all objects of class *NSN* together with the methods *add*, *s*, *equal* and constant *zero*, *equal, less*

$$\mathfrak{C} = \langle |NSN|, zero, s, add, subtract, equal, less\rangle$$

satisfies all axioms of natural numbers with addition operation, cf. section 4.

Proof:
This is a slight modification of the arguments found in Grzegorczyk's book [Grz71]p.239. □

Have a look at the following example and verify that Euclid's algorithm has infinite computations, i.e. does not halt, when interpreted in the data structure $\mathfrak{C}$.

Example 8.1. Suppose that the values of variables x, y, z are determined by the execution of three instructions

$x :=$ **new** $NSN(12,0,1);$ $y :=$ **new** $NSN(15,0,2);$ $z :=$ **new** $NSN(15,1,2);$

Now, the computation of the algorithm $E(m,n)$:

$$\left\{\begin{array}{l} \textbf{while not } equal(m,n) \textbf{ do} \\ \quad \textbf{if } less(m,n) \\ \quad \textbf{then } n := subtract(n,m) \\ \quad \textbf{else } n := subtract(n,m) \\ \quad \textbf{fi} \\ \textbf{od} \end{array}\right\}$$

for $m = x$ and $n = y$ is finite and results is **new** $NSN(3,0,1)$.

An attempt to compute $E(x,z)$ results in an infinite computation, or more precisely, in a computation that can be arbitrarily prolonged, as it is shown in the table below.

States of memory during a computation

n	**m**
new NSN(12,0,1)	**new** NSN(15, 1,2)
new NSN(12,0,1)	**new** NSN(3, 1,2)
new NSN(12,0,1)	**new** NSN(-9, 1,2)
new NSN(12,0,1)	**new** NSN(-21, 1,2)
new NSN(12,0,1)	**new** NSN(-33, 1,2)
...	...
new NSN(12,0,1)	**new** NSN(15-i*12, 1,2)
...	...

Class Cn which implements a non-standard programmable model of theory $\mathcal{T}h_1$ has more applications.

Example 8.2. One can easily extend class Cn adding two functions: $even$ and $\mathbf{div}2$. Class Cn extended in this way brings a counterexample to Collatz hypothesis c.f.[Lag10]. Consider the program

$$\begin{array}{l} \textbf{while } n \neq 1 \textbf{do} \\ \quad \textbf{if } even(n) \\ \quad \textbf{then } n := n\,\mathbf{div}2 \\ \quad \textbf{else } n := 3n+1 \\ \quad \textbf{fi} \\ \textbf{od} \end{array}$$

An attempt to execute the program for $n = z$ results in an infinite computation.
It is even simpler, when you consider $n =$ **new** $NSN(8, 1, 2)$. The computation never reaches $s(zero)$, i.e. **new** $NSN(1, 0, 2)$.

n	*explanation*
new NSN(8, 1, 2)	$\langle 8, \frac{1}{2}\rangle$ is even; divide by 2
new NSN(4, 1, 4)	$\langle 4, \frac{1}{4}\rangle$ is even; divide by 2
new NSN(2, 1, 8)	$\langle 2, \frac{1}{8}\rangle$ is even; divide by 2
new NSN(1, 1,16)	$\langle 1, \frac{1}{16}\rangle$ is odd; multiply by 3; add 1
new NSN(4, 3,16)	$\langle 4, \frac{3}{16}\rangle$ is even; divide by 2
new NSN(2, 3,32)	$\langle 2, \frac{3}{32}\rangle$ is even; divide by 2
new NSN(1, 3,64)	$\langle 1, \frac{3}{64}\rangle$ is odd; multiply by 3; add 1
new NSN(4, 9,64)	$\langle 4, \frac{9}{64}\rangle$ is even; divide by 2
...	
new NSN(1, 3^i, 2^{2i+2})	at step $3i + 1$ the value of n is $\langle 1, \frac{3^i}{2^{2i+2}}\rangle$
...	

A) This means that halting formula of Collatz program is not a theorem of theory $\mathcal{T}h_1$.

B) Note, the program makes no use of multiplication operation. Hence, it seems unlikely that the halting formula is a theorem of theory $\mathcal{T}h_2$. By Tennenbaum's theorem[Ten59] it is impossible to construct a programmable (recursive) and non-standard model of Peano arithmetic. However it suffices to show that there is a non-standard model of Peano arithmetic (i.e. of theory $\mathcal{T}h_2$) such that the functions $even$ and $div2$ are recursive. This need not to contradict Tennenbaum theorem.

C) On the other hand, if Collatz hypothesis is true then it is provable in algorithmic theory $\mathcal{T}h_3$ for it is a categorical and hence complete theory.

The class *Cn* has more applications, e.g in the work on specification of stacks, cf. [MSST00]

Appendix B - axioms and inference rules of program calculus AL

For the convenience of reader we cite the axioms and inference rules of algorithmic logic.
Note. *Every axiom of algorihmic logic is a tautology.*
Every inference rule of AL is sound. [MS87]

Axioms

axioms of propositional calculus

Ax_1 $((\alpha \Rightarrow \beta) \Rightarrow ((\beta \Rightarrow \delta) \Rightarrow (\alpha \Rightarrow \delta)))$

Ax_2 $(\alpha \Rightarrow (\alpha \vee \beta))$

Ax_3 $(\beta \Rightarrow (\alpha \vee \beta))$

Ax_4 $((\alpha \Rightarrow \delta) \Rightarrow ((\beta \Rightarrow \delta) \Rightarrow ((\alpha \vee \beta) \Rightarrow \delta)))$

Ax_5 $((\alpha \wedge \beta) \Rightarrow \alpha)$

Ax_6 $((\alpha \wedge \beta) \Rightarrow \beta)$

Ax_7 $((\delta \Rightarrow \alpha) \Rightarrow ((\delta \Rightarrow \beta) \Rightarrow (\delta \Rightarrow (\alpha \wedge \beta))))$

Ax_8 $((\alpha \Rightarrow (\beta \Rightarrow \delta)) \Leftrightarrow ((\alpha \wedge \beta) \Rightarrow \delta))$

Ax_9 $((\alpha \wedge \neg\alpha) \Rightarrow \beta)$

Ax_{10} $((\alpha \Rightarrow (\alpha \wedge \neg\alpha)) \Rightarrow \neg\alpha)$

Ax_{11} $(\alpha \vee \neg\alpha)$

axioms of predicate calculus

Ax_{12} $((\forall x)\alpha(x) \Rightarrow \alpha(x/\tau)))$

where term τ is of the same type as the variable x

Ax_{13} $(\forall x)\alpha(x) \Leftrightarrow \neg(\exists x)\neg\alpha(x)$

axioms of calculus of programs

Ax_{14} $K((\exists x)\alpha(x)) \Leftrightarrow (\exists y)(K\alpha(x/y))$ for $y \notin V(K)$

Ax_{15} $K(\alpha \vee \beta) \Leftrightarrow ((K\alpha) \vee (K\beta))$

Ax_{16} $K(\alpha \wedge \beta) \Leftrightarrow ((K\alpha) \wedge (K\beta))$

Ax_{17} $K(\neg\alpha) \Rightarrow \neg(K\alpha)$

Ax_{18} $((x := \tau)\gamma \Leftrightarrow (\gamma(x/\tau) \wedge (x := \tau)true)) \wedge ((q := \gamma\prime)\gamma \Leftrightarrow \gamma(q/\gamma\prime))$

Ax_{19} **begin** $K; M$ **end** $\alpha \Leftrightarrow K(M\alpha)$

Ax_{20} **if** γ **then** K **else** M **fi** $\alpha \Leftrightarrow ((\neg\gamma \wedge M\alpha) \vee (\gamma \wedge K\alpha))$

Ax_{21} **while** γ **do** K **od** $\alpha \Leftrightarrow ((\neg\gamma \wedge \alpha) \vee (\gamma \wedge K(\textbf{while}\ \gamma\ \textbf{do}\ K\ \textbf{od}(\neg\gamma \wedge \alpha))))$

Ax_{22} $\bigcap K\alpha \Leftrightarrow (\alpha \wedge (K \bigcap K\alpha))$

Ax_{23} $\bigcup K\alpha \Leftrightarrow (\alpha \vee (K \bigcup K\alpha))$

Inference rules

propositional calculus

R_1 $\dfrac{\alpha, (\alpha \Rightarrow \beta)}{\beta}$ (also known as modus ponens)

predicate calculus

R_6 $\dfrac{(\alpha(x) \Rightarrow \beta)}{((\exists x)\alpha(x) \Rightarrow \beta)}$

R_7 $\dfrac{(\beta \Rightarrow \alpha(x))}{(\beta \Rightarrow (\forall x)\alpha(x))}$

calculus of programs AL

R_2 $\dfrac{(\alpha \Rightarrow \beta)}{(K\alpha \Rightarrow K\beta)}$

R_3 $\dfrac{\{s(\textbf{if}\ \gamma\ \textbf{then}\ K\ \textbf{fi})^i(\neg\gamma \wedge \alpha) \Rightarrow \beta\}_{i \in N}}{(s(\textbf{while}\ \gamma\ \textbf{do}\ K\ \textbf{od}\ \alpha) \Rightarrow \beta)}$

$$R_4 \qquad \frac{\{(K^i\alpha \Rightarrow \beta)\}_{i\in N}}{(\bigcup K\alpha \Rightarrow \beta)}$$

$$R_5 \qquad \frac{\{(\alpha \Rightarrow K^i\beta)\}_{i\in N}}{(\alpha \Rightarrow \bigcap K\beta)}$$

In rules R_6 and R_7, it is assumed that x is a variable which is not free in β, i.e. $x \notin FV(\beta)$. The rules are known as the rule for introducing an existential quantifier into the antecedent of an implication and the rule for introducing a universal quantifier into the successor of an implication. The rules R_4 and R_5 are algorithmic counterparts of rules R_6 and R_7. They are of a different character, however, since their sets of premises are infinite. The rule R_3 for introducing a **while** into the antecedent of an implication of a similar nature. These three rules are called ω-rules. The rule R_1 is known as *modus ponens*, or the *cut*-rule.

In all the above schemes of axioms and inference rules, α, β, δ are arbitrary formulas, γ and γ' are arbitrary open formulas, τ is an arbitrary term, s is a finite sequence of assignment instructions, and K and M are arbitrary programs.

References

[BS96] Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory, Volume 1 Efficient Algorithms*. MIT Press, Cambridge MA, 1996.

[Eng67] Erwin Engeler. Algorithmic Properties of Structures. *Math. Systems Theory*, 1:183–195, 1967.

[Grz71] Andrzej Grzegorczyk. *Zarys Arytmetyki Teoretycznej*. PWN, Warszawa, 1971.

[Grz74] Andrzej Grzegorczyk. *An Outline of Mathematical Logic*. Springer, 1974.

[Kar64] Carol R. Karp. *Languages with expressions of infinite length*. North Holland, 1964.

[Lag10] Jeffrey C. Lagarias, editor. *The Ultimate Challenge: The 3x+1 Problem*. American Mathematical Society, Providence R.I., 2010.

[McH90] James A. McHugh. *Algorithmic graph theory*. Prentice Hall, 1990.

[MS87] Grażyna Mirkowska and Andrzej Salwicki. Algorithmic Logic. "`http://lem12.uksw.edu.pl/wiki/Algorithmic_Logic`", 1987. "[Online; accessed 7-August-2017]".

[MSST00] G. Mirkowska, A. Salwicki, M. Srebrny, and A. Tarlecki. First-Order Specifications of Programmable Data Types. *SIAM Journal on Computing*, 30:2084–2096, 2000.

[Pre29] Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen , in welchem die Addition als einzige Operation hervortritt . *Comptes Rendus du I congrès de Mathématiciens des Pays Slaves*, Comptes Rendus du I congrès de Mathématiciens des Pays Slaves:92–101,395, 1929.

[Sal70] Andrzej Salwicki. Formalized Algorithmic Languages. *Bulletin of Polish Academy of Sciences, Ser. Math. Astr. Phys.*, 18:227–232, 1970.

[Sta84] Ryan Stansifer. Presburger's Article on Integer Arithmetic: Remarks and Translation. Technical Report TR84-639, 1984.

[SZ13] Andrzej Salwicki and Andrzej Zadrożny. Loglan'82 - website. "http://lem12.uksw.edu.pl/wiki/Loglan'82_project", 2013. "[Online; accessed 27-July-2017]".

[Ten59] Stanley Tennenbaum. Non-archimedian models for arithmetic . *Notices of the American Mathematical Society*, (6):270, 1959.