Fundamenta Informaticae XX (2017) 1–27
IOS Press

A new proof of Euclid's algorithm

Andrzej Salwicki
salwicki@mimuw.edu.pl
Dąbrowa Leśna

revised manuscript: July 29, 2017

Abstract. All the proofs of correctness of Euclid's algorithm that we know (in mathematical as well as in computer science texts, cf. [Grz71, Sie50, BK82, Knu77]) are of semantical nature. They are conducted in an *intuitive* number theory. For 1° they study sequences of numbers telling these are computations of the algorithm, 2° they do not analyze the structure of algorithm, 3° all these proofs assume that the computations are done in the standard model of natural numbers (programmers say(!): unsigned integers). For these and other reasons the proofs go beyond the elementary Peano's theory.
We present a new proof of correctness of Euclid's algorithm E. Our proof has the following features: 1° We are constructing and later proving the halting formula H of the algorithm, we do the same for correctness formula, 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 algorithmic logic (i.e. *calculus of programs*), 4° only formulas accepted without proof are the axioms of the program calculus and axioms of algorithmic theory of natural numbers.

1. Introduction

History of Euclid's algorithm is over 2300 years long. The proof of the theorem that the algorithm computes the greatest common divisor of two natural numbers is commonly accepted.
Still there are some open questions. Many mathematical theories are *intuitive* – this means that neither the language of theory nor its axioms are precisely described. Since Euclid's *Elements*, the majority of mathematical theories is *axiomatized*. The *formalized* theories are studied when one wishes to avoid paradoxes cf.[RS63]. Till today, every one of known proofs of correctness of Euclid's algorithm is conducted in an intuitive number theory. Among others, it is assumed that the arguments are standard natural numbers. No axioms excluding non-standard elements accompany the proof. (Yes, we are aware that it is impossible in the frame of any first order theory. But are we limited to first-order logic?).

The computations of the algorithm are studied, without mentioning that the notion of algorithm does not belong to the theory. The correspondence between the text of the algorithm and its computations is assumed in an intuitive way. Nothing disturb us, we believe in the correctness of Euclid's algorithm. The proof is accepted by the vast majority of humans. Is it an intersubjective proof? Will it be accepted by computer? It is worthwhile to think of creating such a proof that will be accepted by a proof-checker.

A bit of history

We shall recapitulate the results achieved in the metamathematics and see how they are related to our problem.
In XIXth century Peano presented the set of axioms describing the structure of natural numbers (cf. 4). It helped in making many precise proofs. However, the proof of the correctnes of Euclid's algorithm was still led by a circuitous route, through analysis of its computations. The algorithm itself remained foreign to the theory. No one analyzed its structure, like was in the case of proofs of (first-order) formulas. The relation between the algorithm and its computation was intuitive one. In the beggining of XXth century it turned out (the results of Lövenheim and Skolem) that Peano theory has non-standard models (cf. [Grz69] p. 288). Twenty years later K. Goedel obtained a result on undecidability of Peano's arithmetic. From this C. Ryll-Nardzewski deduced that there is no finite set S of formulas of first -order logic, such that the only model of S would be the standard structure of natural numbers ([RN52]). In this way we learned there is no elementary theory in which one can construct a proof of correctness of Euclid's algorithm.
Summing up, we assert that

---

the formalized first-order theory $\mathcal{T}h_0$ based on the Peano's axioms APe,

$$\mathcal{T}h_0 = \langle \underbrace{L}_{language}, \underbrace{\mathcal{L}}_{logic}, \underbrace{APe}_{axioms} \rangle$$

does not contain a theorem on correctness of Euclid's algorithm.

There are three reasons of that: 1° The language L of the theory has no algorithms nor the formulas expressing the halting property of programs. 2° Calculus of predicates, i.e. first-order logic $\mathcal{L}$ has no tools helping in analysis of algorithms. 3° Peano's axioms APe have both the standard as well as non-standard model of natural numbers. In a non-standard model, Euclid's algorithm may have infinite computations cf. Appendix A. We shall see later, that it is necessary to replace all three components $\langle L, \mathcal{L}, APe\rangle$ of the formalized theory $\mathcal{T}h_0$ in order to obtain a theory that contains the theorem on correctness of Euclid's algorithm.

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.
  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 5 gives a flavour of such 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 calculus of programs that contains the predicate calculus.

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 be treated as "*first class citizens*" of the theory, like the formulas are, 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 counterexemple.
We expect that the proofs will be intersubjective ones. It means, that everyone reading the 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.

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.) For example, in some programming languages we find a class[2] Int. The details of implementation can be hidden (even covered by patents). One may doubt, whether such a class is a proper model, and of which theory. In the appendix A we show such a class *Cn* that satisfies the axioms of the theory of addition. We are also showing that for some arguments the computations of Euclid's algorithm need not to be finite.

Which calculus to choose?

It seems important that a theory in which we shall conduct the correctness proof of Euclid's algorithm will not have non-standard models. Therefore, we are seeking for a categorical axiomatisation of natural numbers. We can choose among: weak second order logic, logic of infinite disjunctions $\mathcal{L}_{\omega_1\omega}$ [Eng67, Kar64] and algorithmic logic $\mathcal{L}_{\mathcal{A}}$. We prefer algorithmic logic for it has a system of axioms and inference rules. Moreover in the language of algorithmic logic we dispose the formulas that express the semantical properties of programs.
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. Consequently 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 we present in the table 1.

[1] proof-checkers of algorihmic proofs do not exists yet
[2] class is a kind of program module

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 precondtion α and postcondition β onto two subgoals: 1) to prove that if some result exists then it satisfies the postcondtion 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 $n - m$. The second subgoal is harder. 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 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 claim is made that the scheme of induction is equivalent to the *principle of minimum*, 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*) The claim on finiteness of descending sequences is true conditionally. The so called principle of minimum is valid only in the standard model. One has to assume that the algorithm works in the standard model of natural numbers. However, no elementary theory can garantee that every of its models is isomorphic to the standard one. Moreover, the assumption on standard model is not written at all.

(*ii*) The proof contains a phrase "for any natural numbers n and m ... This is perfectly ok as long as the arguments of the algorithm are standard natural numbers. What happens if they come from another model of Presburger or Peano axioms? If Euclid's algorithm is executed in the non-standard model of Presburger arithmetic its computations may be infinite ones. See the Appendix A.

(*iii*) The proof analyzes some sequences of numbers saying this is an execution sequence of the algorithm. It means that the proof goes around. It would be acceptable if the proof itself led correctly to the conclusion.

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

Theory	**Language & Logic**	**Axioms**	**Is there a proof?**
$\mathcal{T}h_0$	$\mathcal{L}$ – 1-st order	Peano	*No* - the language does not accept algorithms, nor algorithmic formulas. There is nothing to prove.
$\mathcal{T}h_1$	$\mathcal{L}_A$ – algorithmic	Pressburger	*No* - the halting formula is independent from the axioms of this theory. A programmable counterexample is presented.
$\mathcal{T}h_2$	$\mathcal{L}_A$ – algorithmic	Peano	*No* - the halting formula is independent from the axioms of this theory.
$\mathcal{T}h_3$	$\mathcal{L}_A$ – algorithmic	Algorithmic Arithmetic	*Yes* - there exists a proof.

2. A few words on calculus of programs

The reader familiar with the algorithmic logic [MS87] can safely skip 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 and it is a superset of deterministic while programs, 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. Moreover, 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 $\mathcal{F}_{FOL}$.

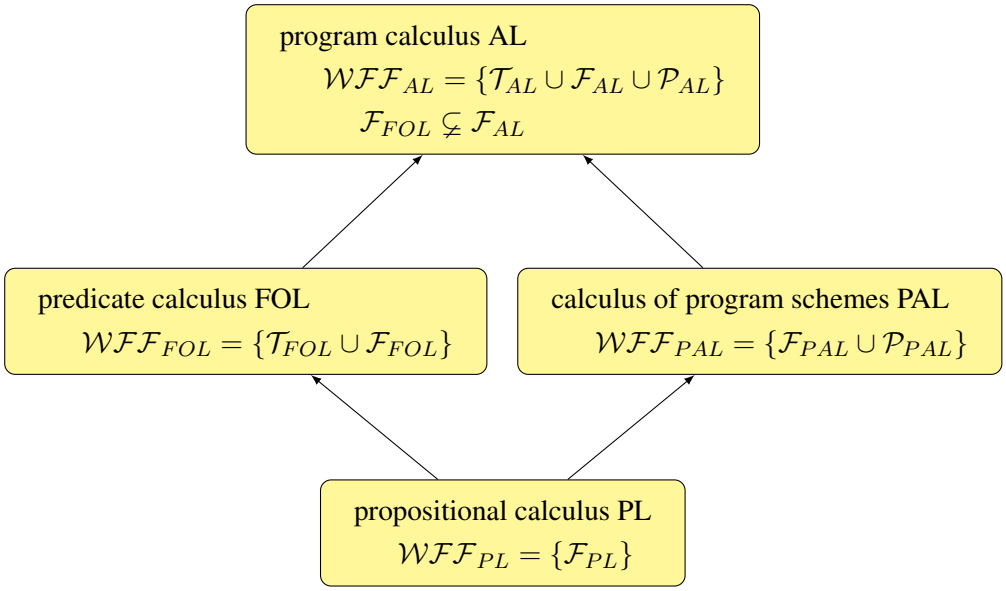

Figure 1. Comparison of logical calculi w.r.t. their $\mathcal{WFF}$ sets

2.1. Three algorithmic theories

From the earlier discussion it follows that the elementary theory $\mathcal{T}h_0$ is not suitable for proving the halting property of Euclid's algorithm. 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 determined by the axioms and inference rules of program calculus, see appendix B. The theories have different sets of specific axioms. The following expression is a program (in each theory).

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

And the following formula expresses the stop property of the program E.

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

All three algorithmic theories $\mathcal{T}h_1, \mathcal{T}h_2, \mathcal{T}h_3$ have 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.

The set $\mathcal{WFF}$ 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 expresions 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 expresions τ and σ are terms, then the expressions $(\tau = \sigma),\ (\tau < \sigma)$ are open formulas,
- if expresions 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 expresion 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 expresions K and M are programs, then the expression $K;\ M$ is a program,
- if expresion 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

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

  It explains why the formula H expresses the halting property of the program E.
- The calculus of programs i.e. algorithmic logic enjoys the property of completeness. For the theorem on completeness consult [MS87].

3. 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 3.1. The set of *specific axioms* of the theory $\mathcal{T}h_1$ consists of the following formulas:

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

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].

Fact 3.1. The formula H is not a theorem of the theory $\mathcal{T}h_1$.

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

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 3.2. The formula H is independent of axioms 1 - 10.

Remark that there exist a programmable model of this theory.

4. Algorithmic theory of addition and multiplication $\mathcal{T}h_2$

The set of axioms of the next theory $\mathcal{T}h_2$ consists of formulas 1 - 9 and two formulas defining 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 4.1. 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.

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.

5. Algorithmic theory of numbers

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

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

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

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

. . . . . .

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

$$x < y \stackrel{df}{\Leftrightarrow} \{w := 0; \textbf{while } w \neq y \wedge w \neq x \textbf{ do } w := s(w) \textbf{ od}\}(w = x \wedge w \neq y) \tag{L}$$

$$P(x) \stackrel{df}{=} \{\ w := 0;\ \textbf{if } x \neq 0 \textbf{ then } \textbf{while } s(w) \neq x \textbf{ do } w := s(w) \textbf{ od fi}\ \}(w) \tag{P}$$

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

The third of axioms [S] is an algorithmic, (not a first-order), formula.
In addition to these three formulas [I, M, S] we assume four more axioms [A, L, P, O] that are defining operations: *addition, predecessor, subtraction* $+$, P, $\dot{-}$ and ordering relation $<$.
This theory $\mathcal{T}h_3$ allows to prove the halting formula H of the Euclids 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 Euclids 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, the formula H is a theorem of the theory $\mathcal{T}h_3$.
Below we are presenting a detailed (and formalizable) proof of the correctness of Euclid's algorithm in algorithmic theory of numbers.

1. We start by showing that all axioms of the theory $\mathcal{T}h_2$ are theorems of the theory $\mathcal{T}h_3$.
2. We shall prove also a couple of properties thet occur in the traditional proof of Euclids algorithm. We omit the proofs of these 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.

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

5.1. Scheme of induction

Lemma 5.1. 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.

$$\vdash \{y := 0; \textbf{while } y \neq x \textbf{ do } y := s(y) \textbf{ od}\}(x = y) \Leftrightarrow \{y := 0\} \bigcup \{\textbf{if } y \neq x \textbf{ then } y := s(y) \textbf{ fi}\}(x = y)$$

Another theorem of algorithmic logic is the following equivalence

$$\vdash \{y := 0\}\bigcup\{y := s(y)\}(x = y) \Leftrightarrow \{y := 0\}\bigcup\{\textbf{if } y \neq x \textbf{ then } y := s(y) \textbf{ fi}\}(x = y).$$

By propositional calculus we have

$$\vdash \{y := 0; \textbf{while } y \neq x \textbf{ do } y := s(y) \textbf{ od}\}(x = y) \Leftrightarrow \{y := 0\}\bigcup\{y := s(y)\}(x = y).$$

By modus ponens we obtain

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

□

Lemma 5.2. 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

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

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_{16} 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 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 5.2 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 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 teorem of algorithmic logic.

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

We apply the inference rule $R2$

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

and obtain another theorem of AL

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

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

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

We apply the axiom of assignment instruction

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

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 5.3. 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 5.3 are equivalent.

5.2. Addition

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

Definition 5.1.

$$x + y = z \stackrel{df}{=} \{t := 0; w := x; \textbf{while } t \neq y \textbf{ do } t := s(t); w := s(w) \textbf{ od}\}(z = w) \qquad \text{(A)}$$

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 5.4.

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

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).

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

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

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

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

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

□

Our next observation is

Lemma 5.5.

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

Proof:
From the definition we have

$$x + 0 = z \Leftrightarrow \{t := 0; w := x; \textbf{while}\ t \neq 0\ \textbf{do}\ t := s(t); w := s(w)\ \textbf{od}\}(z = w)$$

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

$$x + 0 = z \Leftrightarrow \{t := 0; w := x; \textbf{if}\ t = y\ \textbf{then}\ \ \textbf{else while}\ t \neq 0\ \textbf{do}\ t := s(t); w := s(w)\ \textbf{od}\}w$$

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

$$\mathcal{T}h_3 \vdash y = 0 \Rightarrow \{t := 0; w := x; \textbf{while}\ t \neq y\ \textbf{do}\ t := s(t); w := s(w)\}\alpha \Leftrightarrow \{t := 0; w := x;\}\alpha.$$

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

Our next goal is

Lemma 5.6.

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

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 5.3. □

5.3. Definition of relation $<$

Definition 5.2.

$$x < y \stackrel{df}{=} \{w := 0; \textbf{while } w \neq y \wedge w \neq x \textbf{ do } w := s(w) \textbf{ od}\}(w = x \wedge w \neq y).$$

We shall prove the useful property.

Lemma 5.7.

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

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

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

because, the formula $(w = x \wedge w \neq y \vee w \neq y \wedge w = y \vee x = y)$ is a theorem of AL and the implication $(w \neq y \wedge w \neq x) \Rightarrow w \neq x$ is a theorem of AL, too. From the axiom of algorithmic logic Ax_{15} we deduce

$$\begin{aligned} \mathcal{T}h_3 \vdash \forall_x \forall_y \big(& \{w := 0; \textbf{while } w \neq y \wedge w \neq x \textbf{ do } w := s(w) \textbf{ od}\}(w = x \wedge w \neq y) && (16) \\ & \vee \{w := 0; \textbf{while } w \neq y \wedge w \neq x \textbf{ do } w := s(w) \textbf{ od}\}(w \neq y \wedge w = y) && (17) \\ & \vee \{w := 0; \textbf{while } w \neq y \wedge w \neq x \textbf{ do } w := s(w) \textbf{ od}\}(x = y) \big). && (18) \end{aligned}$$

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 program does not change the variables x or y[3]. Finally we obtain $\mathcal{T}h_3 \vdash \forall_x \forall_y (x < y \vee x = y \vee y < x)$.

□

Lemma 5.8.

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

Proof:
We are recalling 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

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

Therefore

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

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}.

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

□

Lemma 5.9.

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

Lemma 5.10.

$$(x < y) \Leftrightarrow \{w := x; \textbf{while } w \neq y \textbf{ do } w := s(w) \textbf{ od}\}(w = y)$$

Lemma 5.11.

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

Proof:
Proof goes by induction with respect to the value of variable z. □

[3]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 $(x = k) \Rightarrow \{w := 0; \textbf{while } w \neq y \textbf{ do } w := s(w) \textbf{ od}\}(x = k)$.

5.4. Predecessor

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

Definition 5.3.

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

Lemma 5.12.

$$P(0) = 0$$

Lemma 5.13.

$$x \neq 0 \Rightarrow s(P(x)) = x$$

Lemma 5.14.

$$x \neq 0 \Rightarrow P(x) < x$$

Lemma 5.15.

$$(x < y) \Leftrightarrow \{w := y; \textbf{while } w \neq x \textbf{ do } w := P(w) \textbf{ od}\}(w = x)$$

Lemma 5.16.

$$P(s(x)) = x$$

Lemma 5.17. For every natural number i

$$P^i(s^i(x) = x$$

We shall prove the fundamental property of the predecessor operator.

Theorem 5.1.

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

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

$$\forall_x \{y := 0; (\textbf{if } y \neq x \textbf{ then } y := s(y) \textbf{ fi})^i\}(x = y) \Rightarrow \{y := 0; (\textbf{if } y \neq x \textbf{ then } y := s(y) \textbf{ fi})^i\}(x = y).$$

We use the scheme of mathematical induction and the lemma 5.17, to prove that for every $i \in N$, the following formula is a theorem of the theory $\mathcal{T}h_3$

$$\forall_x \{y := 0; (\textbf{if } y \neq x \textbf{ then } y := s(y) \textbf{ fi})^i\}(x = y) \Rightarrow \{(\textbf{if } x \neq 0 \textbf{ then } x := P(x) \textbf{ fi})^i\}(x = 0).$$

Remark, 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 $i \in N$

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

Now, we apply the inference rule R_6 to obtain

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

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. 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 5.18.
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
$\{$**while** $x \neq 0$ **do** M **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 5.1), We apply the rule R_1 and finish the proof. □

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

Lemma 5.19. The following inference rule is sound in the theory $\mathcal{T}h_3$:

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

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

Corollary 5.1. 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.

5.5. Subtraction

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

Definition 5.4.

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

Lemma 5.20.

$$\mathcal{T}h_3 \vdash \forall_x\, x \mathbin{\dot{-}} 0 = x$$

Lemma 5.21.

$$\mathcal{T}h_3 \vdash \forall_x \forall_y\, x \mathbin{\dot{-}} s(y) = P(x \mathbin{\dot{-}} y)$$

Lemma 5.22.

$$\mathcal{T}h_3 \vdash \forall_x \forall_y\, (x > y > 0) \Rightarrow x \mathbin{\dot{-}} y < x$$

Lemma 5.23.

$$\mathcal{T}h_3 \vdash \forall_x \forall_y\, (x < y) \Rightarrow x \mathbin{\dot{-}} y = 0$$

6. 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 6.1.

$$\mathcal{T}h_3 \vdash \Big((n \neq m \wedge (max(n,m) = p) \Rightarrow \left\{\begin{array}{l} \textbf{if } n > m \\ \textbf{then} \\ \quad n := n \dot{-} m \\ \textbf{else} \\ \quad m := m - n \\ \textbf{fi} \end{array}\right\} (max(n,m) < p)\Big) \tag{19}$$

Proof:
In the proof we use the axiom of **if** instruction – Ax_{20} and lemma 5.22. □

Now, by lemma 5.19, 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 6.2.

$$\mathcal{T}h_3 \vdash \Big((gcd(n,m) = p) \Rightarrow \left\{\begin{array}{l} \textbf{if } n > m \\ \textbf{then} \\ \quad n := n \dot{-} m \\ \textbf{else} \\ \quad m := m \dot{-} n \\ \textbf{fi} \end{array}\right\} (gcd(n,m) = p)\Big). \tag{20}$$

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 19 and 20 we come to the conclusion that the formula expressing the correctness of Euclid's algorithm is a theorem of theory $\mathcal{T}h_3$

Theorem 6.1.

$$\mathcal{T}h_3 \vdash \left\{\begin{array}{l} \textbf{while } n \neq m \textbf{ do} \\ \quad \textbf{if } n > m \\ \quad \textbf{then} \\ \quad\quad n := n \dot{-} m \\ \quad \textbf{else} \\ \quad\quad m := m \dot{-} n \\ \quad \textbf{fi} \\ \textbf{od} \end{array}\right\} (n = gcd(n,m)) \tag{21}$$

This ends the proof.

Moreover

Making use of the lemma 5.19 we note another theorem of the theory $\mathcal{T}h_3$. It says that the following program has all computations finite

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

This leads to another observation

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

And another fact

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

7. 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.

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 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).
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, Lov87], algorithmic theory of graphs[McH90, Gib85], 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 suggestions. We also thank an anonymous reviewer for many helpful comments.

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. 3). 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 componentwise, as usual in a product,
- the successor operation is defined as follow $s(\langle k, x\rangle) = \langle k + 1, x\rangle$,
- constant zero 0 is $\langle 0, 0\rangle$.
- relation *less* has a type $\omega + (\omega^* + \omega) \cdot \eta$

is a no-standard model of axioms of theory $\mathcal{L}h_1$.

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 := new NSN(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 7.1. 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*

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

satisfies all axioms of natural numbers with addition operation, cf. section 3.

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 7.1. 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 $E(x, y)$ 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 7.2. 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

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

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.

Another application of class *Cn* leads to the conclusion that elementary theory of stacks is decidable and has non-standard model, 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

$$R_1 \qquad \frac{\alpha, (\alpha \Rightarrow \beta)}{\beta}$$

predicate calculus

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

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

calculus of programs AL

$$R_2 \qquad \frac{(\alpha \Rightarrow \beta)}{(K\alpha \Rightarrow K\beta)}$$

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

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

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

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 implicationis 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

[BK82] Lech Banachowski and Antoni Kreczmar. *Elementy analizy algorytmów*. Biblioteka Inżynierii Oprogramowania. WNT, Warszawa, 1982.

[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.

[Gib85] Alan Gibbons. *Algorithmic graph theory*. Cambridge University Press, 1985.

[Grz69] Andrzej Grzegorczyk. *Zarys logiki matematycznej*, volume 20 of *Biblioteka matematyczna*. PWN, Warszawa, drugie wydanie edition, 1969.

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

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

[Knu77] Donald Knuth. *The Art of Programming*. 1977.

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

[Lov87] Laszlo Lovasz. *An Algorithmic Theory of Numbers, Graphs and Convexity*. SIAM, 1987.

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

[MS87] Grażyna Mirkowska and Andrzej Salwicki. *Algorithmic Logic*. PWN and J.Reidel, Warszawa, 1987.

[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 . pages 92–101,395, 1929.

[RN52] Czesław Ryll-Nardzewski. The role of the axiom of induction in elementary Arithmetic. *Fundamenta mathematicae*, 39:239–263, 1952.

[RS63] Helena Rasiowa and Roman Sikorski. *Mathematics of metamathematics*. PWN, Warszawa, 1963.

[Sie50] Wacław Sierpiński. *Teoria Liczb*. Monografie Matematyczne. PWN, 1950.

[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.