SpecVer & LEM'12 2018 1–18
Dombrova Research

On Collatz's theorem
DRAFT - please do not distribute

Grażyna Mirkowska
Dombrova Research
Partyzantów 19
05-092 Łomianki, POLAND
G.Mirkowska@uksw.edu.pl

Andrzej Salwicki
Dombrova Research
salwicki@mimuw.edu.pl

Abstract. We are showing two theorems:

T1) For every natural number $n > 0$, computation of Collatz's algorithm C, executed in standard model of arithmetics of natural numbers is finite.

T2) In a non-standard model of arithmetics of addition one can observe computations that can be arbitrarily prolonged (i.e. they are infinite). This observation can be strengthened: if in a data structure for addition there is an infinite computation of Collatz's algorithm for some n, then the structure is a non-standard model of natural numbers.

September 12, 2018

1. Introduction

In 1937, Lothar Collatz observed that, for any natural number $n > 0$ chosen by him, a sequence defined by following equations

$$a_1 = n$$

$$a_{j+1} = \begin{cases} a_j/2 & \text{if } a_j \text{ is even} \\ 3a_j + 1 & \text{if } a_j \text{ is odd} \end{cases}$$

always terminates on number 1. Collatz formulated the conjecture, *for any natural number* $n > 0$ *the sequence determined by the above equations reaches 1* i.e. for every natural number $n > 0$ there exists natural number k, such that $a_k = 1$.
Several mathematicians and computer scientists were involved in research of the conjecture, cf. [3].

Many papers were published. In the search of an eventual counter-example computers are working constantly. Till today the Collatz's conjecture is verified by all numbers up to $80 \cdot 2^{60}$, see [1].

In a programming language one can write the following program C

```
C:  while n ≠ 1 do
      if n ∈Even then n ← n/2 else n← 3n+1 fi
    od
```

The remark made by Collatz can be formulated as follows

Conjecture 1. For any natural number $n > 0$, computation of program C is finite.

For an execution of Collatz's algorithm C produces all numbers of the sequence $\{a_j\}$.
We shall prove Collatz's conjecture showing that a formula that expresses the halting property of Collatz's program C is a theorem of algorithmic theory of natural numbers.
We shall also show that Collatz's conjecture can not be proved in elementary theory of natural numbers. In this way we confirm the opinion of Paul Erdős [1, 3]. Our arguments are similar to those found in [6].

The structure of the paper is as follows: in next section we study the halting formula of Collatz's program, we define the notion of layer that is related to the halting formula and formulate a couple of lemmas. In section 3 we are giving an indirect proof of Collatz's theorem. The subsequent section is devoted to the question under which conditions the Collatz's program may iterate endlessly? A couple of appendices follow, added for the convenience of the reader.

2. Halting formula, modified algorithm, definition of a layer

We shall analyze algorithm C and its variations. The halting property of the algorithm C can be expressed by the following formula,

$$\left\{\begin{array}{l}\textbf{while } n \neq 1 \textbf{ do}\\ \quad \textbf{if } n \in \text{Even } \textbf{then } n \leftarrow n/2 \textbf{ else } n \leftarrow 3n+1 \textbf{fi}\\ \textbf{od}\end{array}\right\}(n = 1) \tag{lc1}$$

or by another, equivalent formula

$$\bigcup\{\textbf{if } n \neq 1 \textbf{ then if } odd(n) \textbf{ then } n \leftarrow 3n+1 \textbf{ else } n \leftarrow n/2 \textbf{ fi fi}\}(n = 1) \tag{lc2}$$

The *halting formula* of a program M is the weakest precondition that guarantees that the computation of program M is finite. We recall that algorithmic theory of natural numbers has one, up to isomorphisms, model, cf. [4] p. 55. It is the standard structure of natural numbers with addition and multiplication. Hence the following

Remark. The conjecture of Collatz is is true if and only if the halting formula of Collatz's algorithm is a theorem of algorithmic theory of numbers.

Note that the following program checks whether number m is a power of two

$$P2: \quad \begin{array}{l} ispow2 := true; \\ \textbf{while } m \neq 1 \textbf{ do if } odd(m) \textbf{ then } ispow2 := false; \textbf{ exit else } m := m/2 \textbf{ fi od}; \end{array}$$

It is evident that

$$\{P2\}\,(ispow2 \Leftrightarrow \exists_k\, m = 2^k).$$

Therefore we can replace program C by an equivalent program CS:

program CS;

> **Boolean function** $d(n) \overset{df}{\Leftrightarrow}$
> $m := n;\ ispow2 := true;$
> **while** $m \neq 1$ **do if** $odd(m)$ **then** $ispow2 := false;$ **exit else** $m := m/2$ **fi od**;
> $result := ispow2$

while $\neg\, d(n)$ **do**
  if $odd(n)$ **then** $n \leftarrow 3n+1$ **else** $n \leftarrow n/2$ **fi**
od

Program C has a finite computation if and only if the program CS halts. It is easy to observe that the computations of program CS are burdened with overhead on the length of computation. We shall neglect this overhead.
We are going to study the following halting formula of program CS

$$\bigcup \left\{ \begin{array}{l} \textbf{if} \neg(d(n)) \textbf{ then} \\ \quad \mathsf{K}: \left[\begin{array}{l} \textbf{if } odd(n) \\ \quad \textbf{then } n \leftarrow 3n+1 \\ \quad \textbf{else } n \leftarrow n \div 2 \end{array} \right] \\ \textbf{fi} \end{array} \right\} \big(d(n)\big) \qquad \textbf{(LC)}$$

The halting formula **LC** uses an existential iteration quantifier $\bigcup$. To give you an intuition of this iteration quantifier we shall present some formulas equivalent to the formula **LC**.
First we apply the axiom Ax_{21} and obtain

$$\big(d(n)\big) \vee \bigcup \left\{ \textbf{ if} \neg d(n) \textbf{ then } K \textbf{ fi } \right\} \left(\left\{ \textbf{ if} \neg d(n) \textbf{ then } K \textbf{ fi } \right\} \big(d(n)\big) \right) \qquad (\text{LC}_0)$$

which in turn is equivalent to

$$\big(d(n)\big) \vee \left(\left\{ \begin{array}{l} \textbf{if} \neg d(n) \\ \textbf{then } K \textbf{ fi} \end{array} \right\} \big(d(n)\big) \right) \vee \bigcup \left\{ \begin{array}{l} \textbf{if} \neg d(n) \\ \textbf{then } K \textbf{fi} \end{array} \right\} \left(\left\{ \begin{array}{l} \textbf{if} \neg d(n) \\ \textbf{then } K \textbf{ fi} \end{array} \right\}^2 \big(d(n)\big) \right) \qquad (\text{LC}_1)$$

next formula equivalent to the formula **LC** is given below

$$d(n) \vee \left\{ \begin{array}{l} \textbf{if} \neg d(n) \\ \textbf{then } K \\ \textbf{fi} \end{array} \right\} d(n) \vee \left\{ \begin{array}{l} \textbf{if} \neg d(n) \\ \textbf{then } K \\ \textbf{fi} \end{array} \right\}^2 d(n) \vee \bigcup \left\{ \begin{array}{l} \textbf{if} \neg d(n) \\ \textbf{then } K \\ \textbf{fi} \end{array} \right\} \left(\left\{ \begin{array}{l} \textbf{if} \neg d(n) \\ \textbf{then } K \\ \textbf{fi} \end{array} \right\}^3 d(n) \right) \tag{LC2}$$

The i-th formula equivalent to the halting formula is of the following form

$$\begin{array}{l} d(n) \vee \left(\left\{ \begin{array}{l} \textbf{if } \neg d(n) \\ \quad \textbf{then } K \\ \textbf{fi} \end{array} \right\} d(n) \right) \vee \\ \left(\left\{ \begin{array}{l} \textbf{if } \neg d(n) \\ \quad \textbf{then } K \\ \textbf{fi} \end{array} \right\}^2 d(n) \right) \vee \quad ... \left(\left\{ \begin{array}{l} \textbf{if } \neg d(n) \\ \quad \textbf{then } K \\ \textbf{fi} \end{array} \right\}^i d(n) \right) \\ \vee \bigcup \left\{ \begin{array}{l} \textbf{if } \neg d(n) \\ \quad \textbf{then } K \\ \textbf{fi} \end{array} \right\} \left(\left\{ \begin{array}{l} \textbf{if } \neg d(n) \\ \quad \textbf{then } K \\ \textbf{fi} \end{array} \right\}^{i+1} d(n) \right) \end{array} \tag{LCi}$$

We can apply the axiom of conditional instruction Ax_{20} and obtain

$$\begin{array}{l} \underbrace{d(n)}_{0\times} \vee \underbrace{\neg d(n) \wedge K\, d(n)}_{1\times} \vee \underbrace{\neg d(n) \wedge K \neg d(n) \wedge K^2\, d(n)}_{2\times} \vee \\ \qquad \underbrace{\neg d(n) \wedge K \neg d(n) \wedge K^2 \neg d(n) \wedge K^3\, d(n)}_{3\times} \vee \\ \qquad\qquad \ldots \\ \qquad \underbrace{\neg d(n) \wedge K \neg d(n) ... \wedge K^{i-1} \neg d(n) \wedge K^i\, d(n)}_{i\times} \vee \\ \qquad \bigcup \left\{ \begin{array}{l} \textbf{if } \neg d(n) \\ \quad \textbf{then } K \\ \textbf{fi} \end{array} \right\} \left(\left\{ \begin{array}{l} \textbf{if } \neg d(n) \\ \quad \textbf{then } K \\ \textbf{fi} \end{array} \right\}^{i+1} d(n) \right) \end{array}$$

One can observe that the halting formula **LC** of the program CS is, informally speaking, an infinite disjunction of ever-longer conjunctions. Each conjunction expresses the property: *the program CS will execute $i\times$ iterations and will stop*. The precise meaning of iteration quantifier is given by definition (IQE) on page 16. The structure of these conjunctions is even more complicated as will be seen later. For the program K is a conditional instruction. Applying the axiom Ax_{20} one obtains a disjunction.
For example, the conjunction that describes the property: program CS will execute exactly one iteration

of instruction K and will stop is equivalent to the following alternative.

$$\left(\left\{ \begin{array}{l} \textbf{if } \neg d(n) \textbf{ then} \\ \textbf{if } odd(n) \\ \quad \textbf{then } n \leftarrow 3n+1 \\ \quad \textbf{else } n \leftarrow n \div 2 \\ \textbf{fi fi} \end{array} \right\} d(n) \right) \Leftrightarrow \left(\begin{array}{l} \neg d(n) \wedge odd(n) \wedge \{n \leftarrow 3n+1\} d(n) \vee \\ \neg d(n) \wedge \neg odd(n) \wedge \{n \leftarrow n \div 2\} d(n) \end{array} \right). \qquad (1\times)$$

Luckily for us, we could discard the second part of the alternative for the conjunction $\neg d(n) \wedge d(n/2)$ is false for every n.

Inspired by these observations we accept the following definition of *layers*, i.e. the subsets of the set N.

Definition 2.1. (of layers)

$$\begin{array}{rcl} S_0 & \stackrel{df}{=} & \{n \in N : \exists k \ n = 2^k\} \\ S_1 & \stackrel{df}{=} & \{n \in N : \ n > 1 \wedge 3n+1 \in S_0\} \\ S_2 & \stackrel{df}{=} & \{n \in N : \ n/2 \in S_1\} \\ & \cdots & \\ S_{i+1} & \stackrel{df}{=} & \{n \in N : \ n \text{ is odd } \wedge \ 3n+1 \in S_i \ \vee n \text{ is even } \wedge \ n/2 \in S_i\} \end{array}$$

Collatz's conjecture is equivalent to the following one

Conjecture 2. Layers S_i make a covering of the set N.

$$N = \bigcup_{i=0}^{\infty} S_i$$

A quick examination of the properties of the layers brings the following observations

Fact 2.1.

∞) Each layer is an infinite set.

S_0) Layer S_0 is an increasing sequence.

$$a_k = 2^k \qquad \text{for } \ k = 0, 1, ...$$

S_1) Numbers {5, 21,85, 341, ... } belong to the layer S_1.
The recurrent dependencies determine the membership to this layer $a_1 = 5, \ \ a_{j+1} = 4 * a_j + 1$
The j-th element of the layer S_1 is

$$a_j = (2^{2(j+1)} - 1)/3 \qquad \text{for } \ j = 1, 2, 3, ...$$

S_2) Layer S_2 is described by equations $b_1 = 10,\ \ b_{j+1} = 4 * b_j + 2$.
Or

$$b_j = 2 \cdot (2^{2(j+1)} - 1)/3 \qquad \text{dla } \ j = 1, 2, 3, ...$$

S_i) Membership of number n to the layer S_i is definable by an algorithmic formula, see Appendix A.

Lemma 2.1. Layers are pairwise disjoint sets, $S_l \cap S_p = \emptyset$ for $l \neq p$.

Proof:
Suppose that for some l, p intersection of layers is non-empty set $S_l \cap S_p \neq \emptyset$. Without loss of generality we can assume that $l < p$ and that l is the least natural number such that, for some p the layers S_l and S_p have common element $S_l \cap S_p \neq \emptyset$. Hence there is a number m such that $m \in S_l$ and $m \in S_p,\ l < p$.
If m is even number then $m/2 \in S_{l-1}$ and $m/2 \in S_{p-1}$. If m is odd number then $3m + 1 \in S_{l-1}$ and $3m + 1 \in S_{p-1}$.
Hence $S_{l-1} \cap S_{p-1} \neq \emptyset$. This contradicts our assumption that the number l is the least number such that the layer S_l has an element common with some other layer. □

Now our conjecture can be formulated as follows.

Conjecture 3. Set of layers S_i is a partition of the set N.

$$N = \bigcup_{i=0}^{\infty} S_i \qquad \text{and} \qquad S_l \cap S_p = \emptyset \ \text{ for } \ l \neq p$$

Let us make a couple of easy observations

Lemma 2.2. If number n belongs to the layer S_{k+1}, then after execution of instruction $\{\mathbf{if}\ odd(n)\ \mathbf{then}\ n \leftarrow 3n + 1\ \mathbf{else}\ n \leftarrow n \div 2\ \mathbf{fi}\}$ the new value of variable n belongs to the layer S_k.

$$(n \in S_{k+1}) \implies \{\mathbf{if}\ odd(n)\ \mathbf{then}\ n \leftarrow 3n + 1\ \mathbf{else}\ n \leftarrow n \div 2\ \mathbf{fi}\}(n \in S_k) \tag{1}$$

It is easy to remark that

Lemma 2.3. If a number n belongs to some layer S_i, then the computation of Collatz's algorithm that starts from n will stop after finite number of steps.

$$\forall_n\ n \in \bigcup_{i=0}^{\infty} S_i \implies \left\{ \begin{array}{l} \mathbf{while}\ \neg(n = 1)\ \mathbf{do} \\ \quad \mathbf{if}\ odd(n)\ \mathbf{then}\ n \leftarrow 3n + 1\ \mathbf{else}\ n \leftarrow n/2\ \mathbf{fi} \\ \mathbf{od} \end{array} \right\} (n = 1)$$

3. Collatz's algorithm halts

In this section we shall prove that Collatz's algorithm executed in standard structure of natural numbers halts. Namely, we shall show that the halting formula **LC** is a theorem of algorithmic theory of natural numbers $\mathcal{T}_A$.
There are two questions: 1° is there a cycle in computations of Collatz's algorithm? and 2° is there a natural number n such that the computation of Collatz's program is infinite?
First question has a negative answer. No cycles, since the program is deterministic one. See also the lemma 2.1.
In answering to the second question, we shall use three *algorithmic* theories $\mathcal{T}_A, \mathcal{T}_B, \mathcal{T}_W$. All three theories share the same language $\mathcal{L}$, based on the same alphabet. All three theories use the same operation of syntactical consequence $\mathcal{C}$, it means that they use the same set of logical axioms and inference rules of calculus of programs AL, see section 7.
The differences appear in the sets of specific, extra-logical axioms.

B) Theory $\mathcal{T}_B$ could be skipped. We present it for dydactical reasons. The reader may want to see the similarities and differences of two other theories, that are used in the proof of Collatz's theorem. Theory $\mathcal{T}_B = \langle \mathcal{L}, \mathcal{C}, \mathcal{B} \rangle$ has the following set of axioms $\mathcal{B}$ (in addition to the axioms of the calculus of programs *AL*, these are listed in section 7)

$$\mathcal{B}: \left\{ \begin{array}{ll} N_0) & \forall_x\, 0 \neq s(x) \\ N_1) & \forall_{x,y}\, s(x) = s(y) \implies x = y \\ D_0) & \forall_x\, 0 + x = x \\ D_1) & \forall_{x,y}\, (y+1) + x = (x+y) + 1 \\ P_0) & P(0) = 0 \\ P_1) & \forall_x\, P(s(x)) = x \\ A_1) & odd(n) \Leftrightarrow \exists k (n = k + k + 1) \\ A_2) & w/2 = k \Leftrightarrow (k + k = w \vee k + k + 1 = w) \\ A_3) & 3 * n = n + n + n \end{array} \right\}$$

Axioms $\mathcal{B}$ of the theory $\mathcal{T}_B$ are first-order formulas, while the language $\mathcal{L}$ contains programs and algorithmic formulas.

A) Theory $\mathcal{T}_A = \langle \mathcal{L}, \mathcal{C}, \mathcal{A} \rangle$ has the same language and the same operation of syntactical consequence as theory $\mathcal{T}_B$. The set of axioms of this theory $\mathcal{A} = \mathcal{B} \cup \{P3\}$ is the set $\mathcal{B}$ filled with the following algorithmic formula

$$P_3) \quad \forall_x\, \{\mathbf{while}\ x \neq 0\ \mathbf{do}\ \ x \leftarrow P(x)\ \mathbf{od}\}(x = 0)$$

It is known that every model of this theory is isomorphic to the standard structure of natural numbers with addition, see [4] thm. 4.2 on page 55.
Theory $\mathcal{T}_A = \langle \mathcal{L}, \mathcal{C}, \mathcal{A} \rangle$ is therefore complete. [1]. Remember this.

[1] **Definition**. A theory T is *complete* iff for every sentence φ, either this sentence φ is theorem of theory T or the sentence contradicts the axioms of theory T.

W) We shall consider also the following algorithmic theory $\mathcal{T}_W = \langle \mathcal{L}, \mathcal{C}, \mathcal{W} \rangle$. Axioms of this theory are the axioms from the set $\mathcal{B}$ and the following sentence

$$\forall_n \left\{ \begin{array}{l} \textbf{while } \neg d(n) \textbf{ do} \\ \quad \textbf{if } odd(n) \textbf{ then } n \leftarrow 3n+1 \textbf{ else } n \leftarrow n/2 \textbf{ fi} \\ \textbf{od} \end{array} \right\} (d(n)) \qquad (\mathbf{LC}')$$

Yes, it is not an error.(!) $\mathcal{W} = \mathcal{B} \cup \{\mathbf{LC}'\}$. The formula **LC'** is the halting formula of Collatz algorithm.
Theory $\mathcal{T}_W$ is consistent. For not every formula of the language $\mathcal{L}$ is a theorem of the theory.
We shall show that the formula P_3 is a theorem of theory $\mathcal{T}_W$.

Lemma 3.1.

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

A proof of the lemma is in Appendix A, section 5.

3.1. Proof of Collatz theorem

An indirect, meta-logical, proof of the theorem goes in four steps.

1. Formula P3, an axiom of theory $\mathcal{T}_A$, is a theorem of the theory $\mathcal{T}_W$, by lemma 3.1. All the remaining axioms $\mathcal{B}$ are common to both theories.

2. Therefore the set of theorems of theory $\mathcal{T}_A$ is a subset of the set of theorems of theory $\mathcal{T}_W$.

$$\mathrm{Theorems}(\mathcal{T}_A) \subseteq \mathrm{Theorems}(\mathcal{T}_W)$$

3. Remind yourself, theory $\mathcal{T}_A$ is complete. It is also maximal. Hence, theory $\mathcal{T}_W$ is complete too.

4. Therefore the sets of theorems of these theories are equal

$$\mathrm{Theorems}(\mathcal{T}_A) = \mathrm{Theorems}(\mathcal{T}_W)$$

Hence, the halting formula of Collatz algorithm is a theorem of theory $\mathcal{T}_A$.

Theorem 3.1. (*Collatz*) Each computation of Collatz algorithm is finite, when the algorithm is executed in standard structure of natural numbers with addition.

$$\mathcal{T}_A \vdash \mathbf{LC}'$$

q.e.d.

Comments and corollaries

- Collatz's algorithm calculates the number $l(n)$ of layer of a given number n. It is evidently a partial-recursive function. We proved that the function $l(n)$ is recursive i.e. computable function.
- There is a need to write a direct proof of Collatz's theorem. You must be aware that the proof will be using ω-rule R_6, somewhat similar to the proofs in [6].
- It is desirable to estimate the cost of Collatz's algorithm. Give a function $o(n)$ that majorize the number of steps.
- Is the function $o(n)$ primitive recursive?

4. Collatz's algorithm may loop

In this section we are studying the following question: *is it possible that Collatz's algorithm loops? what it would eventually mean?*.
We shall consider various addition structures and various formalized theories: elementary and algorithmic ones.

Definition 4.1. An algebraic system (i.e. a data structure) of type

$$\langle X, +, 0, 1\rangle$$

will be called an *addition structure* iff
1°) X is a set, symbol $+$ denotes two-argument operation on X, 0 and 1 are elements of X, and
2°) such that the structure satisfies the following conditions

$$\forall_x\, 0 \neq x + 1$$
$$\forall_{x,y}\, x + 1 = y + 1 \Rightarrow x = y$$
$$\forall_x\, 0 + x = x$$
$$\forall_{x,y}\, (y + 1) + x = (y + x) + 1$$

and also assures the validity of every formula of the induction scheme,

here symbol Φ denotes any first-order formula

$$\{\Phi(x/0) \wedge \forall_x(\Phi(x) \Rightarrow \Phi(x + 1))\} \Rightarrow \forall_x\, \Phi(x)$$

□

Symbol $\mathcal{K}$ will denote the class of addition structures. This class contains the standard structure of natural numbers with addition as well as various structures non-isomorphic with the standard structure.
In particular the structure described in Appendix C is a non-standard addition structure.

Having look at Appendix C, example 6.1 we formulate

Fact 4.1. Collatz algorithm if executed in a non-standard model $\mathfrak{M}$ of arithmetic of addition has infinite computation of element n which is a nonstandard element of the model.

Making use of the Collatz theorem 3.1 we can formulate the following

Theorem 4.1. If in an addition structure $\mathfrak{A}$ there exists an infinite computation of Collatz algorithm for certain element n, then the structure $\mathfrak{A}$ is a non-standard model of arithmetic of addition of natural numbers.

Corollary 4.1. Halting property of Collatz algorithm is not expressible in the first-order theory of addition.

5. Appendix A – proof of lemma 3.1

To the end of this section the symbol K denotes the following program
$\{\mathbf{if}\ odd(n)\ \mathbf{then}\ n \leftarrow 3n+1\ \mathbf{else}\ n \leftarrow n/2\ \mathbf{fi}\}$.

Remark that the relation $n \in S_x$ is computable by a simplified version of the algorithm C. From the definition of layer, a number n belongs to layer S_x if and only if the Collatz algorithm CS stops after exactly x steps.

Definition 5.1.
Boolean function $w(n,k) \overset{df}{\Leftrightarrow}$

```
m := n;
if k = 0 then result := d(m)
else
   i := 1;
   while i ≤ k ∧ ¬d(m) do                                   result
      if odd(m) then m := 3m + 1 else m := m/2 fi;
      i := i + 1
   od;
   result := (i = k) ∧ d(m)
fi;
```

Fact 5.1. $w(n,0) \Leftrightarrow d(n)$

Fact 5.2. The program in the above definition of predicate $w(n,k)$ does not loop.

In the proof, below, we are using the following auxiliary inference rules of program calculus AL

$$\frac{\alpha \Leftrightarrow K\,\alpha,\ \alpha \Leftrightarrow K;M\,\alpha}{\{\mathbf{while}\ \alpha\ \mathbf{do}\ M\ \mathbf{od}\}\neg\alpha \Leftrightarrow \{\mathbf{while}\ \alpha\ \mathbf{do}\ M;K\ \mathbf{od}\}(\neg\alpha)} \tag{Aux1}$$

$$\frac{\alpha \implies \beta}{\{\mathbf{while}\ \beta\ \mathbf{do}\ M\ \mathbf{od}\}\neg\beta \implies \{\mathbf{while}\ \alpha\ \mathbf{do}\ M\ \mathbf{od}\}(\neg\alpha)} \tag{Aux2}$$

$$\frac{\delta \implies M\,\delta}{\delta \wedge \{\mathbf{while}\ \gamma\ \mathbf{do}\ M\ \mathbf{od}\}\neg\gamma \implies \{\mathbf{while}\ \gamma\ \mathbf{do}\ M\ \mathbf{od}\}(\neg\gamma \wedge \delta)} \tag{Aux3}$$

In the table below formula in the row 2+i+1 follows from the row 2+i, i=0, ..., 5.

1	$\forall x \exists n\ w(n,x)$	for each layer S_x is a non-empty set
2	$\left\{ \begin{array}{l} \textbf{while } \neg d(n) \textbf{ do} \\ \quad K \\ \textbf{od} \end{array} \right\} d(n)$	this is an axiom of theory $\mathcal{T}_W$
3	$\left\{ \begin{array}{l} \textbf{while } \neg d(n) \textbf{ do} \\ \quad K\,; \\ \quad x \leftarrow P(x) \\ \textbf{od} \end{array} \right\} d(n)$	this is a theorem of $\mathcal{T}_W$, we applied the auxiliary inference rule Aux1, see. [5] s.110
4	$\left\{ \begin{array}{l} \textbf{while } \neg w(n,0) \textbf{ do} \\ \quad K\,; \\ \quad x \leftarrow P(x) \\ \textbf{od} \end{array} \right\} w(n,0)$	conditions $d(n)$ and $w(n,0)$ are equivalent, we apply the rule Aux2
5	$\left\{ \begin{array}{l} \textbf{while } \neg w(n,0) \wedge w(n,x) \textbf{ do} \\ \quad K\,; \\ \quad x \leftarrow P(x) \\ \textbf{od} \end{array} \right\} \left(\begin{array}{l} w(n,0) \wedge \\ w(n,x) \end{array} \right)$	condition $w(n,x)$ is an invariant of the program $\{K; x \leftarrow P(x)\}$, cf. Lemma 2.2. We apply the rule Aux3.
6	$\left\{ \begin{array}{l} \textbf{while } \neg x = 0 \wedge w(n,x) \textbf{ do} \\ \quad K\,; \\ \quad x \leftarrow P(x) \\ \textbf{od} \end{array} \right\} \left(\begin{array}{l} x = 0 \wedge \\ w(n,x) \end{array} \right)$	condition $x \neq 0 \wedge w(n,x)$ is equivalent to condition $\neg w(n,0) \wedge w(n,x)$
7	$\left\{ \begin{array}{l} \textbf{while } \neg x = 0 \textbf{ do} \\ \quad K\,; \\ \quad x \leftarrow P(x) \\ \textbf{od} \end{array} \right\} x = 0$	we can skip the subformula $w(n,x)$ since its value is true at every iteration, see row 5.
8	$\left\{ \begin{array}{l} \textbf{while } \neg x = 0 \textbf{ do} \\ \quad x \leftarrow P(x) \\ \textbf{od} \end{array} \right\} x = 0$	we can skip the program K since it does not contain the variable x

q.e.d.

6. Appendix C – an example of infinite computation

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 Collatz'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

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

where

- The set M is defined as follow

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

  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 $s(\langle k, x \rangle) = \langle k + 1, x \rangle$,
- constant zero 0 is $\langle 0, 0 \rangle$.

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 [7]. 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 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;
end Cn;
```

Theorem 6.1. 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*,

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

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 [2]p.239. □

Example 6.1. 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.[3]. An attempt to execute the program C for $n =$ **new** $NSN(8, 1, 2)$ results in an infinite computation.. 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 C 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[8] 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.

7. Appendix D – An outline of calculus of programs

The reader familiar with the algorithmic logic [4] can safely skip the rest of this section.

7.1. A short exposition of calculus of programs

For the convenience of other readers we offer a few words on the calculus of programs and in the following subsection 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 $\mathcal{F}_{FOL}$.

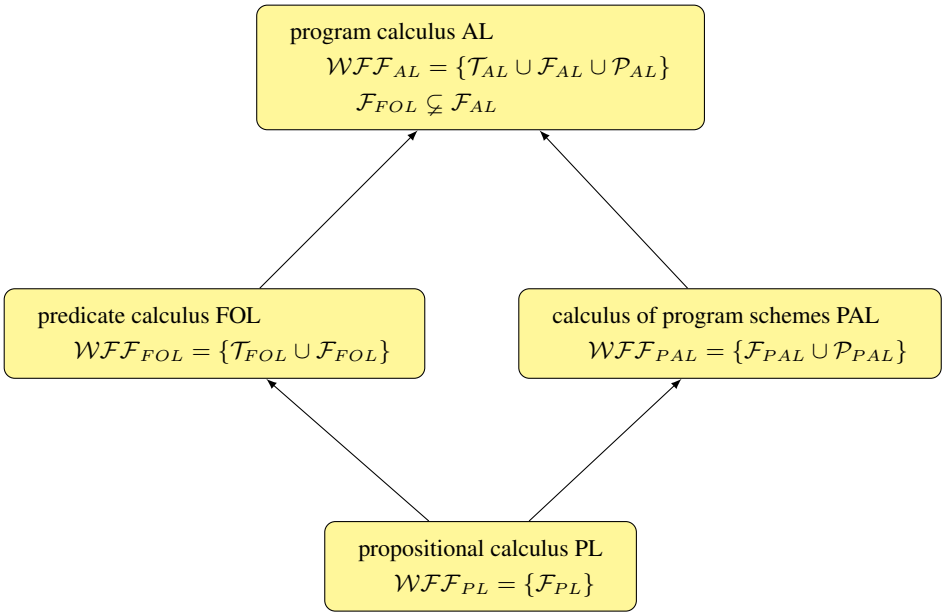

Figure 1. Comparison of logical calculi w.r.t. their $\mathcal{WFF}$ sets

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 7.1. 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 7.2. 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 $(\alpha \wedge \beta)\ (\alpha \vee \beta),\ (\alpha \implies \beta),\ \neg\alpha$ are open formulas. □

Definition 7.3. 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 7.4. 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 [4]. 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

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

This explains why the formula **LC** expresses the halting property of the program CS, on page 3.
Define $K^i\alpha$ by induction: $K^0\alpha \stackrel{df}{=} \alpha$ and $K^{i+1}\alpha \stackrel{df}{=} KK^i\alpha$.
We read the formula $\bigcup K\,\alpha$ as *there exists an iteration of program K such that* formula $K^i\alpha$ *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.

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

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

- The calculus of programs i.e. algorithmic logic, enjoys the property of completeness. For the **completeness theorem** consult [4].

7.2. 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 algorithmic logic is a tautology.*
Every inference rule of AL is sound. [4]

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} $\bigcup K\alpha \equiv (\alpha \vee (K \bigcup K\alpha))$

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

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

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

[1] Collatz conjecture. "https://en.wikipedia.org/wiki/Collatz_conjecture".

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

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

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

[5] Grażyna Mirkowska and Andrzej Salwicki. *Logika algorytmiczna dla programistów*. WNT, Warszawa, 1992.

[6] Andrzej Salwicki. A new proof of Euclid's algorithm. "http://lem12.uksw.edu.pl/images/4/49/On-Euclids-algorithm-2018.pdf", 2018. "[Online; accessed 3-September-2018]".

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

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