Na twierdzenie Collatza
BRULION - nie rozpowszechniaj tej wersji!

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

Andrzej Salwicki
Dombrova Research
salwicki@mimuw.edu.pl

Streszczenie. Wykazujemy, że:

F1) Dla każdej liczby naturalnej $n > 0$, obliczenie algorytmu Collatza C, wykonywanne w standardowym modelu arytmetyki liczb naturalnych jest skończone.

F2) W niestandardowym modelu arytmetyki dodawania można zaobserwować obliczenia dowolnie przedłużane, tj obliczenia nieskończone. Wynika stąd, że własność stopu algorytmu Collatza nie jest twierdzeniem elementarnej teorii liczb naturalnych, (i nie jest wyrażalna żadną formułą pierwszego rzędu).

27 sierpnia 2018

1. Wprowadzenie

W roku 1937 Lothar Collatz zaobserwował, że dla dowolnie wybranej przez niego liczby naturalnej $n > 0$, ciąg określony dwoma wzorami

$$a_1 = n$$

$$a_{j+1} = \begin{cases} a_j/2 & \text{gdy } a_j \text{ parzyste} \\ 3a_j + 1 & \text{gdy } a_j \text{ nieparzyste} \end{cases}$$

zawsze kończy się na liczbie 1. Collatz sformułował hipotezę, że dla dowolnej liczby $n > 0$, naturalnej, ciąg określony powyższymi wzorami osiąga liczbę 1, tzn. dla każdej liczby naturalnej n, istnieje liczba naturalna k, taka że element a_k ciągu określonego powyższymi wzorami jest równy 1.

Poszukiwaniem dowodu tej hipotezy, lub kontrprzykładu, zajmowały się tysiące matematyków i informatyków od przeszło 80 lat, zob. [3].
Opublikowano wiele artykułów. Do poszukiwania ewentualnego kontrprzykładu zatrudniono komputery i w ten sposób sprawdzono, że hipoteza Collatza jest spełniona przez wszystkie liczby aż do $80 \cdot 2^{60}$, zob. [1].
W latach 30-tych XX wieku nie istniały jeszcze komputery. Dzisiaj w języku programowania można napisać następujący program C

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

Obserwacja poczyniona przez Collatza może być sformułowana w ten sposób

Hipoteza 1. Dla każdej liczby naturalnej n program C ma obliczenie skończone.

Zauważmy, że wykonywanie programu CP (z podrozdziału 7.1), w którym dodano polecenie print(n), wewnątrz pętli while, przed instrukcją if, spowoduje drukowanie ciągu liczb $\{a_j\}$.
Udowodnimy prawdziwość hipotezy Collatza wykazując, że formuła wyrażająca własność stopu algorytmu Collatza jest twierdzeniem algorytmicznej teorii liczb naturalnych.
Wykażemy także, że hipoteza Collatza nie może być udowodniona w elementarnej teorii dodawania liczb naturalnych.
Struktura pracy jest następująca: w następnym rozdziale wprowadzimy definicję warstw w zbiorze liczb naturalnych i udowodnimy dwa lematy. W kolejnym rozdziale wykażemy prawdziwość hipotezy Collatza. Omówimy wnioski wyprowadzone z tego faktu i sformułujemy nowe pytania. W kolejnym rozdziale odpowiadamy na pytanie czy dowód hipotezy Collatza może przebiegać w elementarnej teorii dodawania i mnożenia liczb naturalnych.

2. Formuła stopu, modyfikacja algorytmu i definicja warstwy

Będziemy rozpatrywać program C i jego odmiany [1]. Własność stopu algorytmu C może być wyrażona na przykład formułą o takim schemacie

$$\{\mathsf{C}\}(n = 1)$$

a właściwie formułą następującą,

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

lub tak

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

[1] Zauważ, że program CS (zob. podrozdział **??**) jest równoważny programowi C. Można to formalnie udowodnić.

Formuła stopu jest najsłabszym warunkiem wstępnym (tj. poczatkowym obliczenia), który gwarantuje, że obliczenie programu będzie skończone. Przypomnijmy, że algorytmiczna teoria liczb naturalnych ma tylko jeden, z dokładnością do izomorfizmu, model. Jest nim standardowa struktura liczb naturalnych z dodawaniem i mnożeniem. Wynika stąd

Spostrzeżenie. Hipoteza Collatza jest prawdziwa wtedy i tylko wtedy, gdy formuła stopu algorytmu Collatza jest twierdzeniem algorytmicznej teorii liczb naturalnych.

Przyjmijmy następującą definicję $d(n) \stackrel{df}{\equiv} (\exists_k\ n = 2^k)$, a więc boolowska funkcja $d(n)$ odpowiada na pytanie czy liczba n jest potęgą dwójki.

Nasz program C możemy zastąpić równoważnym programem CS:

program CS;

> **Boolean function** $d(n) \stackrel{df}{\Leftrightarrow}$
> $m := n;\ inS0 := true;$
> **while** $m \neq 1$ **do if** $odd(m)$ **then** $inS0 := false;$ **exit else** $m := m/2$ **fi od**;
> $result := inS0$

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

Program C kończy obliczenie wtedy i tylko wtedy, gdy obliczenie kończy program CS. Łatwo zauważyć, że obliczenia programu CS obarczone są narzutem (na długość obliczenia) w porównaniu do programu C. Narzut ten nie jest w tej chwili dla nas najważniejszy. Zauważ, że badanie czy n jest potęgą liczby 2 można wykonać w $\log n$ krokach. W programie CS posługujemy się algorytmem definiującym funkcję boolowską $d(n)$.

Należy więc wykazać prawdziwość poniższej formuły

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

Formuła stopu **LC** jest równoważna następującej formule

$$\big(d(n)\big) \vee \bigcup \left\{ \textbf{ if} \neg d(n) \textbf{ then } K \textbf{ fi } \right\} \left(\left\{ \textbf{ if} \neg d(n) \textbf{ then } K \textbf{ fi } \right\} \big(d(n)\big) \right) \qquad (\mathrm{LC}_1)$$

co jest równoważne kolejnej formule

$$\big(d(n)\big) \vee \left(\left\{ \begin{array}{l} \textbf{if} \neg d(n) \\ \textbf{then } K \textbf{ fi} \end{array} \right\} \big(d(n)\big) \right) \vee \bigcup \left\{ \begin{array}{l} \textbf{if} \neg d(n) \\ \textbf{then } K \textbf{fi} \end{array} \right\} \left(\left\{ \begin{array}{l} \textbf{if} \neg d(n) \\ \textbf{then } K \textbf{ fi} \end{array} \right\}^2 \big(d(n)\big) \right) \qquad (\mathrm{LC}_2)$$

kolejna formuła równoważna formule **LC** wygląda tak

$$d(n) \vee \left\{ \begin{array}{l} \textbf{if} \neg d(n) \\ \textbf{then } K \\ \textbf{fi} \end{array} \right\} d(n) \vee \left\{ \begin{array}{l} \textbf{if} \neg d(n) \\ \textbf{then } K \\ \textbf{fi} \end{array} \right\}^2 d(n) \vee \bigcup \left\{ \begin{array}{l} \textbf{if} \neg d(n) \\ \textbf{then } K \\ \textbf{fi} \end{array} \right\} \left(\left\{ \begin{array}{l} \textbf{if} \neg d(n) \\ \textbf{then } K \\ \textbf{fi} \end{array} \right\}^3 d(n) \right) \tag{LC3}$$

Zobaczmy i-tą kolejną formułę równoważną formule stopu **LC**, ma ona następującą postać

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

Formuła ta jest równoważna następującej, zastosowaliśmy aksjomat instrukcji warunkowej Ax_{20}

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

Składniki tej alternatywy to coraz dłuższe koniunkcje. Struktura tych składników jest jednak bardziej skomplikowana. Program oznaczony symbolem K jest instrukcją warunkową. Zastosowanie aksjomatu instrukcji warunkowej skłania nas do rozpatrywania formuł jeszcze bardziej złożonych. Np. składnik opisujący warunek zakończenia obliczeń po jednokrotnym wykonaniu instrukcji K jest alternatywą

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

Drugi człon tej alternatywy można odrzucić, ponieważ koniunkcja $\neg d(n) \wedge d(n/2)$ ma wartość fałsz, dla każdego n.

Inspirując się powyższymi obserwacjami przyjmujemy następującą definicję indukcyjną ciągu *warstw* tj. podzbiorów zbioru N

Definicja 1.

$$\begin{aligned} S_0 &\stackrel{df}{=} \{n \in N : \exists k\ n = 2^k\} \\ S_1 &\stackrel{df}{=} \{n \in N : n > 1 \wedge 3n+1 \in S_0\} \\ S_2 &\stackrel{df}{=} \{n \in N : n/2 \in S_1\} \\ &\dots \\ S_{i+1} &\stackrel{df}{=} \{n \in N : n \text{ jest nieparzyste i } 3n+1 \in S_i \text{ lub n parzyste i } n/2 \in S_i\} \end{aligned}$$

Hipoteza Collatza jest równoważna następującemu zdaniu

Hipoteza 2. Warstwy S_i stanowią pokrycie zbioru N.

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

Pobieżne badanie właściwości warstw S_k przynosi następujące obserwacje:

($S_i \neq \emptyset$) Każda warstwa jest nieskończonym ciągiem rosnącym.

(S_0) Warstwa S_0 ma oczywiste własności.

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

Zbadanie czy liczba n należy do warstwy S_0 wymaga nie więcej niż $\log n$ kroków.

(S_1) Do kolejnej warstwy S_1 należą liczby {5, 21,85, 341, ... } . Elementy tej warstwy możemy wyznaczyć posługując się następującymi zależnościami $a_1 = 5,\ \ a_{j+1} = 4 * a_j + 1$
Element warstwy S_1 o numerze j ma wartość

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

(S_2) Warstwa S_2 opisana jest wzorem $b_1 = 10,\ \ b_{j+1} = 2 * (4 * b_j + 1)$.
Czyli

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

($n \in S_i$) Należenie liczby n do warstwy S_i wyraża się prostym algorytmem, zob. Dodatek A.

Lemat 1. Warstwy są parami rozłączne.Dla $l \neq p$ zachodzi $S_l \cap S_p = \emptyset$,

Proof:
Załóżmy przeciwnie, że $S_l \cap S_p \neq \emptyset$. Bez zmniejszenia ogólności rozważań można ptzyjąć, że $l < p$ i że l jest najmniejszą liczbą taką, że zachodzi $S_l \cap S_p \neq \emptyset$, dla jakiegokolwiek p. A więc istnieje liczba m taka, że $m \in S_l$ oraz $m \in S_p$, $l < p$. Jeśli m jest liczbą parzystą to $m/2 \in S_{l-1}$ oraz $m/2 \in S_{p-1}$. Jeśli m jest liczbą nieparzystą to $3m+1 \in S_{l-1}$ oraz $3m+1 \in S_{p-1}$.
Otrzymaliśmy sprzeczność z założeniem, że liczba l jest najmniejszą liczbą o założonej własności. □

Naszą hipotezę można więc sformułować w następujący sposób.

Hipoteza 3. Zbiór warstw S_i stanowi rozbicie zbioru N.

Dołączmy jeszcze jedną łatwą obserwację

Lemat 2. Jeśli liczba n należy do warstwy S_{k+1}, to po wykonaniu instrukcji $\{\mathbf{if}\ odd(n)\ \mathbf{then}\ n \leftarrow 3n+1\ \mathbf{else}\ n \leftarrow n \div 2\ \mathbf{fi}\}$ nowa wartość zmiennej n należy do warstwy S_k.

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

Nietrudno udowodnić następujące spostrzeżenie

Lemat 3. Jeśli liczba n należy do którejkolwiek warstwy S_i, to algorytm Collatza rozpoczynając obliczenie od n osiąga 1 po skończonej liczbie kroków.

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

3. Algorytm Collatza nie zapętla się

W tym rozdziale wykażemy, że program Collatza wykonywany w standardowej strukturze liczb naturalnych ma własność stopu. Udowodnimy mianowicie, że formuła stopu **LC** jest twierdzeniem algorytmicznej teorii $\mathcal{T}_A$. W dowodzie wystąpią trzy teorie algorytmiczne $\mathcal{T}_A, \mathcal{T}_B, \mathcal{T}_W$. Wszystkie te teorie mają ten sam alfabet i język $\mathcal{L}$ tj. zbiór wyrażeń poprawnie zbudowanych, wszystkie trzy teorie mają tę samą operację syntaktycznej konsekwencji $\mathcal{C}$ tzn. wykorzystują ten sam zbiór aksjomatów i reguł wnioskowania rachunku programów AL, zob. rozdział 9, Dodatek D. Różnice występują w zbiorach aksjomatów specyficznych, pozalogicznych.

B) Teoria $\mathcal{T}_B$ mogłaby zostać pominięta. Prezentujemy ją z powodów dydaktycznych, czytelnik zechce dostrzec podobieństwa i różnice dwu innych teorii, jakie mają znaczenie dla dowodu

twierdzenia Collatza. Teoria $\mathcal{T}_B = \langle \mathcal{L}, \mathcal{C}, \mathcal{B} \rangle$ ma następujący zbiór aksjomatów $\mathcal{B}$ (oprócz aksjomatów rachunku programów AL, te są podane w rozdziale 9).

$$\mathcal{B}: \left\{ \begin{array}{ll} N_0) & \forall_x\, 0 \neq s(x) \\ N_1) & \forall_{x,y}\, s(x) = s(y) \implies x = y \\ D_0) & \forall_x\, 0 + x = x \\ D_1) & \forall_{x,y}\, (y+1) + x = (x+y) + 1 \\ P_0) & P(0) = 0 \\ P_1) & \forall_x\, P(s(x)) = x \end{array} \right\}$$

Aksjomaty teorii $\mathcal{T}_B$ są formułami z języka pierwszego rzędu. Natomiast język $\mathcal{L}$ zawiera programy i formuły algorytmiczne, rachunek programów AL jest systemem dedukcyjnym $\mathcal{C}$ tej teorii.

A) Teoria $\mathcal{T}_A = \langle \mathcal{L}, \mathcal{C}, \mathcal{A} \rangle$ ma ten sam język i tę samą operację syntaktycznej konsekwencji. Zbiór aksjomatów tej teorii $\mathcal{A} = \mathcal{B} \cup \{P3\}$ to zbiór aksjomatów teorii $\mathcal{T}_B$ uzupełniony następującą formuła algorytmiczną

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

Wiadomo, że każdy model tej teorii jest izomorficzny z modelem standardowym [4]p.123.
Teoria $\mathcal{T}_A = \langle \mathcal{L}, \mathcal{C}, \mathcal{A} \rangle$ jest więc zupełna[2]. Zapamiętajmy to.

W) Rozpatrzymy z kolei, teorię $\mathcal{T}_W = \langle \mathcal{L}, \mathcal{C}, \mathcal{W} \rangle$. Aksjomatami tej teorii są zdania ze zbioru $\mathcal{B}$ oraz następujące zdanie

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

Tak, to nie pomyłka.(!)
Zbiór $\mathcal{W} = \mathcal{B} \cup \{\mathbf{LC}'\}$. Formuła **LC**' jest formułą stopu algorytmu Collatza.
Wykażemy, że twierdzeniem teorii $\mathcal{T}_W$ jest formuła P_3.

Lemat 4.

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

Dowód lematu znajdziesz w dodatku A, rozdział 6

Dowód twierdzenia Collatza przebiega w następujący sposób. Zwróć uwagę, jest to dowód pośredni.

1. Formuła P3 , aksjomat teorii $\mathcal{T}_A$, jest twierdzeniem teorii $\mathcal{T}_W$. Pozostałe aksjomaty teorii $\mathcal{T}_A$ są aksjomatami teorii $\mathcal{T}_W$.

[2]**Definicja**. Teoria T jest *zupełna* wttw gdy dla każdego zdania φ, albo zdanie φ jest twierdzeniem teorii T albo zdanie to jest sprzeczne z aksjomatami teorii T.

2. Zbiór twierdzeń teorii $\mathcal{T}_A$ jest podzbiorem zbioru twierdzeń teorii $\mathcal{T}_W$.

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

3. Teoria $\mathcal{T}_A$ jest zupełna. A więc zupełna jest też teoria $\mathcal{T}_W$ z której wyprowadziliśmy aksjomaty teorii $\mathcal{T}_A$.

4. Zbiory twierdzeń tych teorii są więc równe $Theorems(\mathcal{T}_A) = Theorems(\mathcal{T}_W)$. A więc

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

.

Twierdzenie 5. (*Collatza*) Każde obliczenie algorytmu Collatza w standardowej strukturze liczb naturalnych z dodawaniem jest skończone.

q.e.d.

4. Uwagi i wnioski

- Powyżej przedstawiliśmy dowód pośredni. Wykazaliśmy istnienie dowodu. Warto i trzeba skonstruować dowód bezpośredni.
- Podać funkcję o(n) – ograniczenie liczby kroków algo Collatza.
- Czy funkcja o(n) jest pierwotnie rekurencyjna?
- Algorytm Collatza wyznacza numer warstwy... Funkcja przyporządkowująca liczbie n numer jej warstwy jest funkcją częściowo-rekurencyjną – to jest obserwacja Collatza. Nasz dowód przekonuje, że jest to funkcja ogólnie-rekurencyjna czyli obliczalna.

5. Algorytm Collatza może się zapętlić

W tym rozdziale rozpatrywać będziemy struktury dodawania i rozmaite teorie sformalizowane dotyczące takich struktur. Rozpatrujemy klasę $\mathcal{K}$ modeli elementarnej arytmetyki dodawania. Klasa ta zawiera model standardowy i modele niestandardowe.

Definicja 2. System algebraiczny (tj. strukturę danych) typu

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

będziemy nazywać *modelem elementarnej arytmetyki dodawania* wtedy i tylko wtedy gdy 1°) X jest zbiorem, symbol + oznacza dwuargumentową operację określoną dla każdej pary elementów $x, y \in X$,

0 i 1 sa stałymi i który 2°) spełnia następujące warunki

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

a także zapewnia prawdziwość każdej formuły o poniższym schemacie,

symbol Φ oznacza dowolną formułę pierwszego rzędu

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

□

Symbol $\mathcal{K}$ oznaczać będzie klasę struktur dodawania. Klasa $\mathcal{K}$ zawiera standardowy model liczb naturalnych i różne struktury nie izomorficzne z modelem standardowym.
W szczególności modelem niestandardowym jest struktura opisana w Dodatku C. Podsumujmy to

Fact 5.1. Algorytm Collatza ma w niestandardowym modelu arytmetyki dodawania obliczenie nieskończone dla n będącego niestandardowym tj. nieosiągalnym elementem tego modelu.

W połączeniu z twierdzeniem Collatza możemy sformułować następujące twierdzenie

Twierdzenie 6. Jeżeli w pewnej strukturze dodawania algorytm Collatza ma obliczenie nieskończone dla pewnego n to struktura ta jest niestandardowym modelem arytmetyki dodawania liczb naturalnych.

Wniosek 1. Własność stopu algorytmu Collatza nie jest wyrażalna żadną formułą elementarnej teorii dodawania.

6. Dodatek A – dowód lematu 4

Należy wykazać, że formuła $\forall_x\, \{\mathbf{while}\ x \neq 0\ \mathbf{do}\ \ x \leftarrow P(x)\ \mathbf{od}\}(x = 0)$ jest twierdzeniem teorii $\mathcal{T}_W$.

W dowodzie symbol K oznacza program $\{\mathbf{if}\ odd(n)\ \mathbf{then}\ n \leftarrow 3n+1\ \mathbf{else}\ n \leftarrow n/2\ \mathbf{fi}\}$.

Relacja $n \in S_x$ jest obliczalna przez uproszczoną wersję algorytmu C. Zgodnie z definicją liczba n należy do warstwy S_x wtedy i tylko wtedy gdy algorytm Collatza CS zatrzymuje się po x krokach.

Definicja 3. Relacja $n \in S_k$ jest definiowana przez następującą deklarację funkcji $w(n,k)$

Boolean function $w(n,k) \stackrel{df}{\Leftrightarrow}$

```
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;
```

Zauważmy, że dla każdej liczby naturalnej x, istnieje liczba naturalna n, która należy do warstwy o numerze x. Zachodzi więc warunek $w(n,x)$.

1	$\forall x \exists n\ w(n,x)$	każda warstwa x jest niepusta
2	$\left\{\begin{array}{l}\textbf{while } \neg d(n) \textbf{ do}\\ \quad K\\ \textbf{od}\end{array}\right\} d(n)$	to jest aksjomat teorii $\mathcal{T}_W$
3	$\left\{\begin{array}{l}\textbf{while } \neg d(n) \textbf{ do}\\ \quad K\\ \quad x \leftarrow P(x)\\ \textbf{od}\end{array}\right\} d(n)$	to jest twierdzenie $\mathcal{T}_W$, zastosowano pomocniczą regułę wnioskowania Aux1, zob. [5] s.110
4	$\left\{\begin{array}{l}\textbf{while } \neg w(n,0) \textbf{ do}\\ \quad K\\ \quad x \leftarrow P(x)\\ \textbf{od}\end{array}\right\} w(n,0)$	warunki $d(n)$ i $w(n,0)$ są równoważne, stosujemy regułę Aux2
5	$\left\{\begin{array}{l}\textbf{while } \neg w(n,0) \wedge w(n,x) \textbf{ do}\\ \quad K\\ \quad x \leftarrow P(x)\\ \textbf{od}\end{array}\right\} \left(\begin{array}{l} w(n,0)\wedge\\ w(n,x)\end{array}\right)$	warunek $w(n,x)$ jest niezmiennikiem programu $\{K; x \leftarrow P(x)\}$, por. Lemat 2. Wykorzystujemy też pomocniczą regułę Aux3 ,zob. poniżej
6	$\left\{\begin{array}{l}\textbf{while } \neg x = 0 \wedge w(n,x) \textbf{ do}\\ \quad K\\ \quad x \leftarrow P(x)\\ \textbf{od}\end{array}\right\} \left(\begin{array}{l} x \neq 0\wedge\\ w(n,x)\end{array}\right)$	warunek $x \neq 0 \wedge w(n,x)$ wynika z $\neg w(n,0) \wedge w(n,x)$
7	$\left\{\begin{array}{l}\textbf{while } \neg x = 0 \textbf{ do}\\ \quad K\\ \quad x \leftarrow P(x)\\ \textbf{od}\end{array}\right\} x = 0$	możemy opuścić warunek $w(n,x)$ bo jest on stale spełniony, na początku do x dobrano n tak by zachodził ten warunek
8	$\left\{\begin{array}{l}\textbf{while } \neg x = 0 \textbf{ do}\\ \quad x \leftarrow P(x)\\ \textbf{od}\end{array}\right\} x = 0$	możemy opuścić program K bo nie występuje w nim zmienna x
		q.e.d.

W dowodzie posługujemy się pomocniczymi, wtórnymi regułami wnioskowania rachunku programów

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

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

W kroku 5 korzystamy z następującej pomocniczej reguły wnioskowania AL:

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

7. Dodatek B – programy

7.1. Program Collatza drukuje

Poniższy program pozwala przekonać się, że program C jest odpowiednikiem sformułowania indukcyjnego.

CP:

while n $\neq$ 1 **do**
  print(n);
  if n $\in$Parzyste **then** n $\leftarrow$ n/2 **else** n$\leftarrow$ 3n+1 **fi**
od

8. Dodatek C – przykład obliczenia nieskończonego

Tu napisać podobnie jak w dodatku A do pracy o algo Euklidesa.

8.1. 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. **??**). 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

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

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

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

Now, class *Cn* is written in Loglan programming language [6]. This class defines and implements an algebraic structure $\mathfrak{C}$. The universe of the structure consists of all objects of the class NCN (this is an infinite set). Operations in the structure $\mathfrak{C}$ are defined by the methods of class Cn: *add, equal, zero* and s. All the axioms of the algorithmic theory $\mathcal{T}h_1$ are valid in the structure $\mathfrak{C}$, i.e. the structure is a model of the theory. We show that for some data the execution of Euclid's algorithm is infinite.

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

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

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

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

Proof:
This is a slight modification of the arguments found in Grzegorczyk's book [2]p.239. □

Example 8.1. 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[7] 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.

9. Dodatek D – Krótka prezentacja rachunku programów

Tu będzie o AL (per genus proximus et differentia specifica) a także aksjomaty i reguły wnioskowania.

9.1. A short exposition of calculus of programs

The reader familiar with the algorithmic logic [4] can safely skip the rest of this section. For the convenience of other readers we offer a few words on the calculus of programs and in the 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 $\mathcal{F}_{FOL}$.

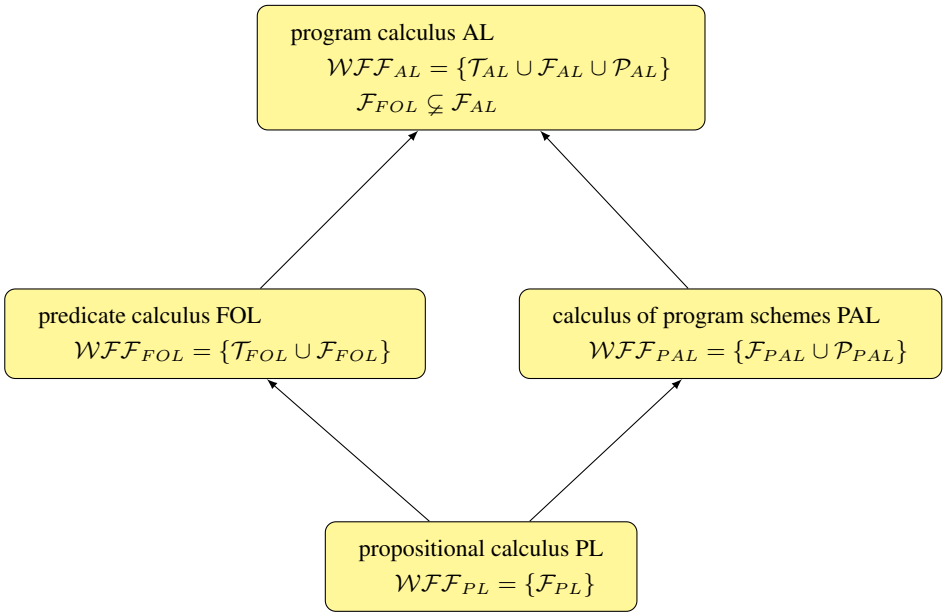

Figure 1. Comparison of logical calculi w.r.t. their $\mathcal{WFF}$ sets

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 9.1. 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 9.2. 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 9.3. 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 9.4. 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

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

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

$\bigcap K\,\alpha$ means *for each iteration of program K formula $K^i\alpha$ holds*.
The signs $\bigcup$ and $\bigcap$ are *iteration quantifiers.* The meaning of these formulas is defined as follow.

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

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

9.2. 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 algorihmic 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.

Twierdzenie o pełności rachunku programów
Twierdzenie o nieistotności definicji

Literatura

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

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