O tym, czego nie można policzyć i co z tego (nie) wynika. Jak nie interpretować twierdzenia Gödla.

Marek Czarnecki

10 lipca 2010

1 Słowo wstępne

W niniejszym artykule pokrótce przedstawiam motywacje i przemyślenia, które doprowadziły Alana Turinga do sformułowania najpopularniejszej dziś eksplikacji pojęcia obliczalności oraz omawiam kilka klasycznych wyników teorii obliczeń. Następnie przedstawiam związki obliczalności z twierdzeniami Gödla, by w ostatniej części zaprezentować przykłady błędnych rozumowań motywowanych przez twierdzenia Gödla.

2 Na czym polega obliczanie

W programie większości współczesnych kursów akademickich teorii obliczeń, jako model obliczeń o uniwersalnej mocy, przedstawiane są maszyny Turinga. Z punktu widzenia informatyki teoretycznej, taki stan rzeczy nie dziwi, ponieważ jest to bardzo prosty model, o którym łatwo dowodzić istotnych twierdzeń dotyczących pojęcia obliczalności. Z drugiej jednak strony *zaprogramowanie* maszyny Turinga tak, by obliczała nawet proste funkcje, czy rozstrzygała niezbyt złożone, a intuicyjnie – w sposób oczywisty – obliczalne problemy, kosztuje wiele pracy, a owocuje powstaniem bardzo skomplikowanej funkcji przejścia, z której niesłychanie trudno odczytać, do czego tak naprawdę ona służy.

Niemniej Turing w 1936 [5] wyabstrahował pojęcie maszyny[1] właśnie z rozważań nad tym, jak przebiega obliczanie wykonywane przez człowieka. Obszerny komentarz na ten temat znajduje się w rozdziale 9 *The extent of the computable numbers*, gdzie Turing argumentuje za tym, że wprowadzone

[1] Nazywał je *automatic machines*.

przez niego maszyny potrafią liczyć dokładnie to, co jest obliczalne w intuicyjnym sensie. Na początku właśnie tego rozdziału postawione zostaje pytanie o wszelkie możliwe procedury, jakie mogłyby zostać użyte podczas obliczania pewnej liczby. Pytanie to jest w istocie równoważne tezie Churcha, zwanej również często tezą Churcha–Turinga.

Tak więc model obliczeń, jakim jest maszyna Turinga, powstał przez następującą abstrakcję:

Człowiek:	Maszyna:	Funkcja:
Kartka (w kratkę)	Taśma	Pamięć
Stany umysłu	Stany maszyny	Wykonywana procedura
Ołówek (i gumka)	Głowica	Dostęp do pamięci
Symbole	Alfabet	Zawartość pamięci
Algorytm	Funkcja przejścia	Sposób obliczania

W zasadzie wszystkie najważniejsze aspekty obliczania, kryją się w funkcji przejścia – algorytmie, przy pomocy którego problem jest rozwiązywany. Zanim jednak powiemy o niej coś więcej przyjrzyjmy się pozostałym wierszom tabeli. Jeżeli chodzi o alfabet, to ma się on składać z pewnej skończonej liczby symboli, z których każdy zajmować będzie jedną kratkę na kartce. Symboli tych powinno być skończenie wiele, aby można je było bez trudu odróżniać od siebie (żeby nie były zbyt podobne). W czasie obliczania będziemy pokrywać kartkę symbolami, wymazywać je i znów pokrywać kartkę kolejnymi symbolami. Mają one za zadanie reprezentować informacje, które przechowujemy, i z których będziemy mogli korzystać w kolejnych etapach obliczania. Jeżeli chodzi o pamięć, czyli o samą kartkę w kratkę, należy zwrócić uwagę na dwie jej cechy. Początkowo zawiera ona problem, który mamy rozwiązywać, zapisany przy użyciu skończonego ciągu symboli. Nietrudno jednak wyobrazić sobie dowolnie *duże* problemy, wymagające dowolnie wiele miejsca na kartce (nie mówiąc nawet o ich rozwiązaniu, które może być nieporównywalnie dłuższe). Już sam ten fakt sprawia, że do obliczania będzie potrzebna potencjalnie nieskończona ilość papieru. Z drugiej jednak strony, ponieważ każdy problem ma być zapisany w postaci skończonego ciągu symboli, a podczas każdego kroku obliczania (o czym zaraz poniżej) będziemy dopisywać co najwyżej pewną skończoną ilość symboli, zatem w każdej chwili na kartce znajdzie się ich jedynie skończenie wiele. Stanów umysłu, podobnie, jak symboli w alfabecie (z tych samych powodów), powinno być ich skończenie wiele. Mają one odzwierciedlać, co w danej chwili robimy – jaki podproblem rozwiązujemy. Ołówek i gumka pozwalają nam dokonywać zapisków na kartce, jednak pełnią jeszcze jedna rolę. W swojej pracy Turing

zwraca uwagę na to, że posiadamy skończone pole widzenia. Przytacza przykład dwóch liczb: 999999999999999999 i 9999999999999999999, o których nie możemy *z biegu* powiedzieć, czy są równe, czy też nie. Rozstrzygnięcie ich równości wymaga osobnego obliczenia. Zatem nasz dostęp do pamięci jest ograniczony. Wreszcie algorytm ma w pełni opisywać, w jaki sposób mamy przeprowadzać obliczenia. W zależności od tego, jakie symbole są w naszym polu widzenia, i w jakim jesteśmy stanie umysłu mamy zmienić stan umysłu, dokonać pewnych modyfikacji symboli na kartce (ale w obrębie pola widzenia) oraz przenieść *wzrok* w pobliskie miejsce na kartce.

Należy zwrócić uwagę, że rozważamy obliczanie czysto mechaniczne, takie, które można opisać przy pomocy *skończonych środków*. Algorytm jednoznacznie wyznacza nasze zachowanie podczas obliczania i nie ulega zmianie w jego toku. To znaczy, że jeżeli przypadkiem znajdziemy się w tym samym stanie umysłu, patrząc na te same symbole więcej niż raz, to zawsze wykonamy tą samą czynność. W swoim artykule z 1936 roku Turing co prawda o liczbach obliczalnych, mając na myśli problem liczenia kolejnych cyfr w dziesiętnym, czy dwójkowym, rozwinięciu obliczanej liczby. Istnieje jednak naturalna odpowiedniość pomiędzy problemem obliczania kolejnych cyfr rozwinięcia dwójkowego liczby rzeczywistej, a problemem należenia do zbioru liczb naturalnych. Przy pomocy liczby naturalnej i zbioru liczb naturalnych można zakodować dowolną nieujemną liczbę rzeczywistą: liczba wskazuje po której cyfrze należy postawić przecinek, natomiast zbiór przechowuje informację o pozycjach, na których znajdują się jedynki (na pozostałych miejscach są zera).

3 Czego nie można policzyć?

Posiadając gotowy model obliczeń można zadać pytanie o jego siłę, czyli o to, jakie problemy można przy jego pomocy rozwiązywać. Zauważmy, że przedstawiony wyżej model obliczeń dopuszcza sytuacje, w których obliczenie nigdy się nie zakończy – wtedy nie można mówić o jego wyniku. Może się też zdarzyć, że wiele różnych algorytmów będzie rozwiązywać ten sam problem. Te dwie cechy modelu okazują się nie do obejścia, jeżeli chcemy zachować siłę obliczeniową. Jeżeli istnieje maszyna, która zatrzymuje się dokładnie wtedy, gdy na wejściu otrzyma liczbę z danego zbioru, to zbiór ten nazwiemy częściowo obliczalnym. Jeżeli zaś maszyna zatrzymuje się na każdym wejściu, a ponadto odpowiada TAK, gdy liczba dana na wejściu należy do zbioru oraz NIE w przeciwnym przypadku, to zbiór ten nazywamy obliczalnym. Czy zatem wszystko można policzyć? Czy każda liczba rze-

czywista, każdy podzbiór liczb naturalnych jest obliczalny? Oczywiście nie! Najprostszy dowód istnienia liczb nieobliczalnych odwołuje się do argumentu ilościowego – każdy algorytm można przedstawić w postaci skończonej listy instrukcji, zatem wszystkich algorytmów jest co najwyżej przeliczalnie wiele, podczas gdy wszystkich liczb rzeczywistych jest nieprzeliczalnie wiele.

Argument ilościowy nie pokazuje nam jednak przykładu problemu nieobliczalnego. Najprostszym problemem nieobliczalnym jest problem stopu, który prezentujemy poniżej. Intuicyjnie: skoro każdy algorytm posiada skończony opis, to można go zapisać na kartce i przedstawić jako wejście dla obliczeń. Ściślej – dowolna maszyna posiada pewną skończoną specyfikację, którą można zakodować w postaci liczb i zapisać na taśmie innej maszyny. Przez M_n będziemy oznaczać maszynę, której kodem jest liczba n – można wprowadzić taką efektywną numerację maszyn, by z kodu można było odczytać specyfikację maszyny. Podobnie możemy kodować konfiguracje i obliczenia maszyn. Aby wiedzieć dokładnie jaki będzie kolejny ruch maszyny należy znać algorytm, którym się posługuje, zawartość taśmy, pozycję oraz stan głowicy. Wtedy stosując znaną funkcję przejścia dowiemy się jaka będzie następna konfiguracja. Z kolei poprawne (skończone) obliczenie to ciąg (skończony!) konfiguracji – zatem również obliczenia możemy kodować.

Skoro nieskończone działanie maszyny jest zjawiskiem niepożądanym, to chcielibyśmy wiedzieć, czy dana maszyna na danym wejściu zatrzyma się i poda nam odpowiedź, czy też będziemy musieli czekać na nią w nieskończoność. Chcielibyśmy zbudować maszynę, która dostaje dwie dane: kod maszyny oraz wejście dla tej maszyny i odpowiada na postawione przez nas pytanie. Oczywiście nasza maszyna musiałaby się zawsze zatrzymywać – inaczej pytanie jej, czy jakaś inna maszyna się zatrzyma nie miało by sensu, skoro ona sama nie musiałaby się zatrzymywać. Problemem stopu nazwiemy problem przynależności pary liczb naturalnych do zbioru S, gdzie $S(n,m) \equiv M_n(m)\downarrow$, czyli gdy maszyna o kodzie n zatrzymuje się (przechodzi w stan końcowy) na wejściu o kodzie m. Możemy zdefiniować przekątną tego zbioru: $K(n) \equiv M_n(n)\downarrow$. Załóżmy teraz, że faktycznie istnieje maszyna, która zawsze rozstrzygnie, czy dana maszyna zatrzyma się na danym wejściu, czyli że problem stopu jest obliczalny. Wtedy również rozstrzygalny jest zbiór K oraz jego dopełnienie $\overline{K}(n) \equiv M_n(n)\uparrow$. To znaczy, że istnieje algorytm M_k taki, że dla dowolnego $n \in \omega$ $\overline{K}(n)$ dokładnie wtedy, gdy $M_k(n)\uparrow$, czyli że M_k rozstrzyga $\overline{K}$. Jednak wtedy w szczególności dla $n = k$ mamy $\overline{K}(k) \equiv M_k(k)\downarrow$ i z definicji $\overline{K}$ $\overline{K}(k) \equiv M_k(k)\uparrow$. Zatem maszyna M_k na wejściu k musiałaby zatrzymywać się dokładnie wtedy, gdy oblicza w nieskończoność, a to nie jest możliwe – taka maszyna nie może istnieć, a zatem zbiór K nie może być obliczalny.

Zbiór K jest natomiast częściowo obliczalny. Wspominaliśmy już, że maszyna może na wejściu otrzymać także kod innej maszyny (lub swój), i że z tego kodu można – algorytmicznie odczytać wszystkie informacje o zakodowanej maszynie. Zbiór K jest częściowo obliczalny przez uniwersalną maszynę, która na wejściu dostaje kod maszyny, następnie odkodowuje go, by zasymulować obliczenie tej maszyny na jej własnym kodzie. Jeżeli obliczenie to jest skończone, czyli $M_n(n)\downarrow$, to uniwersalna maszyna zatrzyma się. W przeciwnym wypadku będzie symulować obliczenie w nieskończoność.

A może nasz model jest za słaby, by rozstrzygać problem stopu? Może można stworzyć silniejsze maszyny? Od lat 30–tych powstało wiele modeli obliczeń, najbardziej znane to obok maszyn Turinga funkcje rekurencyjne Kleenego, rachunek λ, maszyny rejestrowe, while–programy (np. C++, PASCAL), czy Σ_1^0–formuły. Wszystkie okazały się równie silne, chociaż często powstawały na podstawie zupełnie innego typu abstrakcji. Nie jest to co prawda dowód, że nie możemy stworzyć silniejszych maszyn, jednak jest to pewien argument za tym, że moc obliczeniowa maszyn Turinga obejmuje dokładnie to, co można mechanicznie policzyć. Stwierdzenie, że intuicyjne pojęcie obliczalności pokrywa się z pojęciem obliczalności przez maszyny Turinga, nazywane jest tezą Churcha, bądź tezą Churcha–Turinga i istnieje obszerna literatura na jego temat. Rozważania te nie mieszczą się jednak w obrębie niniejszej pracy.

Przytoczę jeszcze jedno klasyczne twierdzenie teorii obliczeń – twierdzenie Posta.

Twierdzenie 1. *Zbiór jest obliczalny dokładnie wtedy, gdy jest częściowo obliczalny i jego dopełnienie jest częściowo obliczalne.*

Dowód jest bardzo prosty. Jeżeli zbiór A jest obliczalny, to maszyna częściowo obliczająca go wykonuje te same operacje, co ta, która go oblicza, tylko zamiast odpowiadać NIE dla elementów nie należących do niego wykonuje w nieskończoność jakąś bezsensowną operację (na przykład przesuwa głowicę w prawo w nieskończoność). Analogicznie w przypadku maszyny częściowo obliczającej dopełnienie A. Jeżeli natomiast mamy maszyny częściowo obliczające A oraz jego dopełnienie $\overline{A}$, to uruchamiamy je jednocześnie i sprawdzamy, która się zatrzyma (któraś musi). Jeżeli zatrzyma się maszyna częściowo obliczająca A, to odpowiadamy TAK, a w przeciwnym przypadku odpowiadamy NIE.

Następujące twierdzenie łączy teorię obliczeń z problemem rozstrzygalności teorii matematycznych.

Twierdzenie 2. *Jeżeli teoria T ma rekurencyjny zbiór aksjomatów i jest zupełna, to jest rozstrzygalna.*

Jest to oczywisty wniosek z twierdzenia Posta – aby sprawdzić, czy zdanie φ jest twierdzeniem teorii T szukamy jego dowodu (dowody, jako skończone obiekty mają kody, podobnie jak obliczenia) równolegle szukając dowodu jego negacji. Z zupełności teorii wynika, że w końcu znajdziemy jeden z nich, rekurencyjna aksjomatyzowalność jest konieczna, by móc kodować dowody i rozstrzygać czy dany kod jest dowodem jakiejkolwiek formuły. W powyższym twierdzeniu istotne są oba założenia. Mamy rekurencyjnie aksjomatyzowalne i zupełne, a więc rozstrzygalne teorie $Th(\mathbb{N},+), Th(\mathbb{N},\times), Th(\mathbb{R},+,\times,0,1)$[2], zupełne i nierozstrzygalne, a więc nie posiadającej rekurencyjnej aksjomatyzacji $Th(\mathbb{N},+,\times,0,1)$, niezupełne i rozstrzygalne $\{0=0\}$ i wreszcie rekurencyjnie aksjomatyzowalne, niezupełne i nierozstrzygalne np. PA, ZFC.

Istnieje zatem wzajemna odpowiedniość pomiędzy teoriami, a obliczeniami. Z jednej strony w odpowiednio silnych teoriach możemy reprezentować obliczenia, z drugiej możemy obliczać co jest twierdzeniem teorii szukając dowodów. Zanim przejdziemy do analizy wybranych nietrafnych użyć twierdzeń Gödla przypomnijmy je.

Twierdzenie 3. *(Gödel 1. w wersji Rosera [3].) Niech T będzie dostatecznie silną, niesprzeczną, rekurencyjnie aksjomatyzowalną teorią. Wtedy istnieje zdanie φ takie, że $T \nvdash \varphi$ oraz $T \nvdash \neg\varphi$.*

Twierdzenie 4. *(Gödel 2.) Niech T będzie dostateczne silną, niesprzeczną, rekurencyjnie aksjomatyzowalną teorią. Wtedy $T \nvdash Con_T$.*

Przed dostatecznie silną teorię rozumiemy taką, która jest prawdziwa o liczbach naturalnych oraz zawiera arytmetykę PA. W istocie zawieranie całej arytmetyki PA nie jest konieczne – wystarczy jej niewielki fragment. Zdanie Con_T jest zdaniem arytmetycznym wyrażającym niesprzeczność teorii T.

4 Co z tego (nie) wynika?

Jednym z najbardziej znanych argumentów motywowanych twierdzeniem Gödla jest argument Lucasa. Przez wiele lat ewoluował on pod naporem krytyki i trudno znaleźć jego ostateczną formę. W [1] Franzen przypisuje Lucasowi następujące zdanie: *Twierdzenie Gödla mówi, że każdy niesprzeczny system S wystarczająco silny, by uprawiać w nim prostą arytmetykę zawiera formuły, których nie może dowieść, a o których my wiemy, że są prawdziwe.* Podobna, błędna interpretacja twierdzenia Gödla znalazła się w pracy Rudy'ego Ruckera z 1995 roku [4]. Przedstawił on ją w siedmiu punktach,

[2]Przez $Th(M)$ oznaczamy zbiór zdań pierwszego rzędu prawdziwych w modelu M.

natomiast poniższa krytyka tego argumentu wraz z ósmym punktem argumentacji pochodzi od Franzena [1] (zamieniam wszędzie nazwisko Gödel na Rucker).

1. Ktoś daje Ruckerowi UTM (uniwersalną maszynę prawdy) mówiąc *oto uniwersalna maszyna prawdy* odpowiada poprawnie na każde pytanie.

2. Rucker prosi o specyfikację UTM: P(UTM).

3. Zadowolony z siebie Rucker buduje na bazie P(UTM) zdanie G: *maszyna zbudowana na bazie P(UTM) nigdy nie powie, że to zdanie jest prawdziwe.*

4. Rucker pyta UTM o prawdziwość G.

5. Gdyby UTM miała potwierdzić, że G jest prawdziwe, to G faktycznie musiałoby być prawdziwe, zatem UTM nie mogłaby nigdy powiedzieć, że G jest prawdziwe. Gdyby z kolei UTM miała zaprzeczyć, że G jest prawdziwe, to znaczyłoby, że G jest fałszywe, czyli że UTM powie zdanie fałszywe, czego jej nie wolno. Zatem UTM nie może odpowiedzieć na pytanie Ruckera.

6. Zatem UTM nie powie nigdy, że G jest prawdziwe, czyli na mocy sformułowania G – istotnie – G jest prawdziwe.

7. Rucker obwieszcza – wiem coś czego nie wie UTM: otóż G.

8. (Franzen) Ruckerowi opada szczęka, gdy UTM nagle ogłasza, że G jest prawdziwe. Zadziwiony Rucker wyjękuje *ale przecież miałaś zawsze mówić prawdę!*, na co UTM odpowiada *zdaje się, że jednak nie.*

Na pierwszy rzut oka wprowadzony prze Franzena ósmy punkt argumentacji wydaje się absurdalny – przecież takie zachowanie przeczy naturze UTM. Jednakże jedyną rzeczą, którą Rucker wie (punkt 6.), to: *jeżeli UTM jest niesprzeczna, to UTM nie powie G.* Tłumacząc to zdanie na język arytmetyki (to pierwotne podłoże argumentów z twierdzeń Gödla) Rudy Rucker wie jedynie, że $Con_{PA} \longrightarrow \neg Pr_{PA}(Con_{PA})$. Nie jest to jednak nic odkrywczego, ponieważ UTM też musi to wiedzieć, jako że $PA \vdash Con_{PA} \longrightarrow \neg Pr_{PA}(Con_{PA})$. Zatem aby wiedzieć coś więcej niż UTM Rucker powinien zbadać niesprzeczność UTM.

To, że Rucker nie może tego wiedzieć *zachowując się racjonalnie* pokazuje Stanisław Krajewski w [2]. Pokazuje on, że nawet przy bardzo łagodnej

interpretacji słów Lucasa, żaden tego typu argument nie może być trafny. Poniżej przedstawiam rozumowanie Krajewskiego.

Załóżmy, że istnieje funkcja F, która dla dowolnej maszyny zwraca kod zdania Gödla dla tej maszyny. Czyli $F(n) = G_{M_n}$. Jeżeli przez $S(M_n)$ oznaczymy zbiór twierdzeń dowodzonych przez M_n, to $S(M_n) \nvdash F(n)$ o ile $S(M_n)$ jest teorią niesprzeczną, a w przeciwnym wypadku niech $F(n)$ będzie dowolnym zdaniem. Nie zakładamy o zadaniach $F(n)$ nic więcej – nie interesuje nas ich prawdziwość, złożoność, ani czy je rozumiemy. Funkcja F musi stosować się co najmniej do wszystkich niesprzecznych maszyn. Jednak zbiór $C = \{n : S(M_n) \text{ niesprzeczna}\}$ nie jest rekurencyjny. Jeżeli chcielibyśmy zatem, by argument Lucasa był sensowny funkcja F nie może być określona dokładnie na C, ponieważ w takim wypadku od razu dalibyśmy Lucasowi narzędzie, które wykracza poza możliwości maszyn i argument z twierdzenia Gödla nie byłby potrzebny, ponieważ posiadanie funkcji F dawałoby wiedzę nieznaną żadnej maszynie. Zatem F musi być funkcją częściową. Z drugiej strony, aby móc *wygödlować* formułę dla dowolnej niesprzecznej teorii musi zachodzić $C \subsetneq dom(F)$. Poniżej raz jeszcze podaję wszystkie poczynione założenia.

1. F jest częściową funcką rekurencyjną.
2. $C \subsetneq dom(F)$.
3. $\forall n \in C\, S(M_n) \nvdash F(n)$.

Twierdzimy, że zbiór $A = \{F(n) : n \in dom(F)\}$ wartości F jest sprzeczny. Załóżmy nie wprost, że A jest niesprzeczny. Ponieważ A jest częściowo rekurencyjny, więc istnieje maszyna M_k taka, że $A = S(M_k)$, a skoro A niesprzeczny, więc $k \in C$, a zatem $F(k)$ jest określone. Na mocy 3. $S(M_k) \nvdash F(k)$, a zatem $F(k) \notin S(M_n)$. Czyli $F(k) \notin A$, co przeczy definicji A. Zatem gdyby antymechanicysta, jak Lucas, posiadał metodę, która pozwalałaby dla dowolnej niesprzecznej maszyny znaleźć zdanie od niej niezależne (wystarczy nawet przez nią niedowodliwe) i twierdziłby, że faktycznie wie, że te zdania są prawdziwe, to zbiór twierdzeń, które głosi byłby sprzeczny. Trudno w takiej sytuacji uznać argument Lucasa i twierdzić, że naprawdę wie o prawdziwości tych zdań.

Kolejne niewłaściwe wykorzystanie twierdzeń Gödla stosowane jest do szerzenia sceptycyzmu, w kontekście naszych możliwości poznawczych, a w szczególności poznania naukowego. Często spotykane są poglądy typu (przytaczam za Franzenem [1]):

- Niczego nie można dowieść w matematyce (gdyż teorie są wątpliwej pewności).

- PA i ZFC są wątpliwe z uwagi na drugie twierdzenie Gödla.
- (Encyclopedia Britannica) Twierdzenie Gödla głosi, że w każdym sensownym systemie matematycznym są zdania, których nie można ani dowieść, ani obalić na bazie aksjomatów tego systemu, a zatem nie jest pewne, czy nawet proste aksjomaty arytmetyki nie pozwolą wyprowadzić sprzeczności.
- Na mocy twierdzenia Gödla każda teoria jest albo niezupełna, albo sprzeczna. Zatem nie jest możliwe *całkowite dowiedzenie* jakiegokolwiek twierdzenia.

Na powyższe sądy składają się w głównej mierze dwa czynniki. Przekonanie, że niesprzeczność pewnych bądź wszystkich teorii matematycznych jest wątpliwa, oraz że niesprzeczność nie może być dowiedziona w takim samym sensie, jak inne twierdzenia matematyczne. Jednak nasz sceptycyzm odnośnie, na przykład aksjomatów arytmetyki Peano nie zależy w ogóle od twierdzenia Gödla. Jeżeli nie wątpimy w aksjomaty, ani w reguły dowodzenia, to nie mamy podstaw, by wątpić w niesprzeczność PA. Dziwne wydaje się również stwierdzenie, że skoro jesteśmy pewni aksjomatów ZFC, to ZFC powinna dowodzić swojej niesprzeczności. Jest oczywiste, że nie możemy dowieść *wszystkiego* w matematyce, jednak nie potrzebujemy twierdzeń Gödla, by to stwierdzić. I wreszcie, co dałby nam (w sensie wątpliwości) ewentualny dowód niesprzeczności ZFC przeprowadzony wewnątrz ZFC? Gdybyśmy nie ufali aksjomatom ZFC, dlaczego mielibyśmy uznać oparty o nie dowód niesprzeczności za poprawny i mniej wątpliwy niż one same?

Jest jeszcze jedna luka w argumentach z twierdzeń Gödla wysuwanych przez sceptyków. Nie jest jasne, w jakim sensie niesprzeczność miałaby być miarą prawdziwości, czy pewności teorii? Jeżeli arytmetyka PA jest niesprzeczna, to teoria $PA + \neg Con_{PA}$ również jest niesprzeczna, tymczasem dowodzi ona fałszywego zdania: $\neg Con_{PA}$. Niesprzeczność nie zabrania teorii dowodzić zdań fałszywych. Więc o ile sprzeczność teorii gwarantuje, że jest ona fałszywa, o tyle wiedza, że pewna teoria jest niesprzeczna, nie dostarcza nam żadnej informacji o prawdziwości. Zatem wszelkie argumenty z twierdzeń Gödla w kontekście sceptycyzmu poznawczego są chybione.

Literatura

[1] T. Franzen, *Godel's Theorem: An Incomplete Guide to Its Use and Abuse*, A K Peters, 2005.

[2] S. Krajewski *Twierdzenie Godla i jego interpretacje filozoficzne*, Wydawnictwo IFiS PAN, 2003.

[3] B. Roser: *Extensions of some theorems of Gödel and Church*, Journal of Symbolic Logic, 1 (1936), N1, s. 87–91.

[4] R. Rucker, *Infinity and the Mind*, Princeton University Press, 1995.

[5] A. Turing, *On computable numbers, with an application to the Entscheidungsproblem*, Proc. London Math. Soc., Vol. 2, No. 42. (1936), pp. 230-265.