82

A ⊆ N × N jest rekurencyjny, jeśli istnieje całkowita funkcja rekurencyjna f: N × N → N taka że:

f(a, b) = {
    1 : (a, b) ∈ A,
    0 : wpp
}

Jeśli nie możemy zdefiniować funkcji rekurencyjnych dla N × N → N, to możemy przekształcić pary N × N na liczby z N funkcją Cantora (po przekątnych), albo na 2^a3^b itp.

T: B = { n | istnieje m takie że (n, m) ∈ A } jest r.e.

Weźmy dowolny A ⊆ N × N rekurencyjny, niech ϕ_A – program rozstrzygający A. Konstruujemy program sprawdzający przynależność n do B:

- wczytaj n
- dla każdego m ∈ N:
    - jeśli ϕ_A(n, m) = 1, zwróć 1

Jeśli n ∈ B, to istnieje jakieś m takie że (n, m) ∈ A, program w końcu je znajdzie.
Jeśli nie, to nigdy się nie zakończy.

83

T: Dla danego B r.e. istnieje A ⊆ N × N rekurencyjny.

Weźmy dowolny B r.e., niech ϕ_B – program rozstrzygający B.
Niech A = { (n, m) | n ∈ B, ϕ_B(n) zatrzymuje się po m krokach }.
Program rozstrzygający A :

- wczytaj (n, m)
- dla _ od 0 do m:
    - wykonaj 1 krok ϕ_B(n)
    - jeśli otrzymano wartość, zwróć ją
- zwróć 0

85

T: A = { n | |Dom(ϕ_n)| ≥ 7 } jest r.e.

Idea: chcemy sprawdzić, czy istnieje przynajmniej 7 argumentów, dla których ϕ_n się zatrzymuje. Nie możemy jednocześnie sprawdzić wszystkich z A, ale możemy robić coś w stylu Cantora – iść przekątnymi po parach (x, m), gdzie x to argument, zaś m to liczba sprawdzonych kroków dla wszystkich argumentów nie większych niż x.

Pokażemy program rozstrzygający A:

- wczytaj n
- znajdź ϕ_n
- known_domain := ∅
- dla x ∈ N:
    - dla i od 0 do x, i ∉ known_domain:
        - wykonaj 1 krok ϕ_n(i)
        - jeśli otrzymano wartość:
            - known_domain ∪= {i}
    - jeśli |known_domain| >= 7:
        - zwróć 1

86

A, B, C, D – zbiory r.e. takie że każda liczba naturalna należy do dokładnie dwóch z nich.

T: A, B, C, D rekurencyjne.

BSO weźmy A, ϕ_A – program semirozstrzygający A. Pokażemy program rozstrzygający A:

- wczytaj n
- n_in_sets := ∅
- dopóki |n_in_sets| < 2:
    - wykonaj 1 krok ϕ_A
    - jeśli otrzymano wartość, zwróć ją
    - dla X ∈ {B, C, D}, x ∉ n_in_sets:
        - wykonaj 1 krok ϕ_X
        - jeśli otrzymano 1:
            - n_in_sets ∪= {X}
- zwróć 0

87

ϕ – niemalejąca, całkowita funkcja rekurencyjna

T: Zbiór wartości ϕ jest rekurencyjny.

Program rozstrzygający ϕ[N]:
- wczytaj n
- dla i $\in$ N:
    - x := ϕ(i)
    - jeśli x = n, zwróć 1
    - jeśli x > n, zwróć 0

Czy pozostaje to prawdą bez założenia o całkowitości?
Nie. Kontrprzykład: f(n) = ϕ_n(n), wtedy f[N] = K.

88

T: Każdy niepusty zbiór r.e. jest zbiorem wartości pewnej całkowitej funkcji rekurencyjnej.

Weźmy dowolny niepusty A r.e. i jego ϕ_A. Skonstruujemy f taką że f[N] = A.

Idea: dla danego n będziemy przeszukiwać jego otoczenie, z każdą iteracją rozszerzając je i wykonując jeden krok ϕ_A dla każdego elementu w tym otoczeniu, aż któryś ϕ_A zwróci 1 i znajdziemy element z A, który możemy zwrócić. Zawsze znajdziemy, bo A niepusty.

Program:
- wczytaj n
- lower, upper := n, n
- powtarzaj:
    - dla i od lower do upper:
        - wykonaj 1 krok ϕ_A(i)
        - jeśli otrzymano 1, zwróć i
    - lower -= 1
    - upper += 1

91

Rel:
https://math.stackexchange.com/questions/1682675/example-of-a-recursive-set-s-and-a-total-recursive-function-f-such-that-fs
https://www.cs.toronto.edu/~sacook/csc438h/notes/page71.pdf
f – funkcja rekurencyjna całkowita

1

T: Jeśli A jest rekurencyjny, to f[A] też.

No raczej nie, wtedy jeśli A byłby rekurencyjny i $A \leq_{rek} B$, to B też.
Kontrprzykład, korzystający z 82 / 83 – weźmy A = N × N i g: N × N → N tak że g(n, m) zwraca numer ϕ najbliższego n, które zatrzymuje się po m krokach.

Wtedy f[A] = K, K nie jest rekurencyjny.

2

T: Jeśli A jest rekurencyjny, to $f^{-1}[A]$ też.

Tak, odpowiada to temu, że jeśli $A \leq_{rek} B$ i B rekurencyjny, to A też.
Niech g – program rozstrzygający $f^{-1}[A]$, g(n) = ϕ_A(f(n)).

3

T: Jeśli A jest r.e., to f[A] też.

Tak. Weźmy g: N → N, tak że g(x) będzie sprawdzać kolejne a ∈ A, szukając takiego, dla którego f(a) = x. (Oczywiście musimy ograniczać ilość kroków). Jeżeli nie wiemy, które a są w A, to sprawdzamy wszystkie liczby naturalne i nasz warunek to ϕ_A(a) = 1 oraz f(a) = x.

4

T: Jeśli A jest r.e., to $f^{-1}[A]$ też.
Tak, jak 2.

Co zmieni się, jeśli założymy, że f jest funkcją częściową? 2 nie będzie prawdziwe, reszta bez zmian (?).

95

Chcemy w prosty sposób odwzorować wszystkie elementy z A i B.
Niech $C = \{ (a, 0) \mid a \in A \} \cup \{ (b, 1) \mid b \in B \}$.

1

T: $A \leq_{rek} C$ i $B \leq_{rek} C$

r_AC, r_BC: $N \to N$
r_AC(n) = (n, 0)
r_BC(n) = (n, 1)

2

T: Jeśli D jest taki że $A \leq_{rek} D$ i $B \leq_{rek} D$, to $C \leq_{rek} D$.

Weźmy r_AD i r_BD: $N \to N$ – redukcje z założenia. Wtedy:

```
r_CD(n, m) = {
        r_AD(n) : m = 0,
        r_BD(n) : m = 1
}
```

96

1

T: $K \leq_{rek} K'$.

Nie. Załóżmy nie wprost, że tak, niech r – redukcja.
Wiemy, że:

1. $x \in K \Rightarrow r(x) \in K'$,
2. $x \notin K \Rightarrow r(x) \notin K'$.

Można to zapisać równoważnie, korzystając z faktu, że $x \notin K \Leftrightarrow x \in K'$:

1. $x \notin K' \Rightarrow r(x) \notin K$,
2. $x \in K' \Rightarrow r(x) \in K$.

Czyli r działa też w drugą stronę, jako redukcja z K' do K.
Więc ten podpunkt jest równoważny drugiemu.

2

T: K' $\leq_{rek}$ K.

Nie, bo wtedy K byłby rekurencyjny.

Ogólnie w tym zadaniu w nie ma znaczenia, czym jest K i tak samo działa dla dowolnych A, A'.

94

Udowodnij, że zbiór numerów tych programów, które zatrzymują się dla wszystkich argumentów oprócz co najwyżej skończonej liczby, nie jest rekurencyjnie przeliczalny.

Czyli **nie zatrzymują się** dla skończonej liczby argumentów.

Niech A – ten zbiór, załóżmy nie wprost, że A r.e., ϕ_A – program semirozstrzygający przynależność do A.

Wtedy możemy skonstruować f semirozstrzygającą przynależność do K':
(Przypomnienie: K' = N \ K = { n $\in$ N | ϕ_n(n) nie zatrzymuje się nigdy }).

- wczytaj n
- t := numer tego programu:
    - wczytaj k
    - wykonaj k kroków ϕ_n(n)
    - jeśli otrzymano wynik, zapętl się
    - zwróć 1
- zwróć ϕ_A(t)

Czy ten program semirozstrzyga przynależność do N \ K?

- Weźmy dowolne n $\in$ K'
    - ϕ_n(n) nie zatrzyma się nigdy
    - dla dowolnego k ϕ_t(k) zwróci 1
    - ϕ_A(t) zwróci 1
- Weźmy dowolne n $\notin$ K' (n $\in$ K)
    - istnieje k' takie że ϕ_n(n) zwraca wynik po k' krokach
    - dla każdego k $\geq$ k' ϕ_t(m) się zapętli
    - ϕ_A(t) nie zwróci 1

Wynika z tego, że K' jest r.e., sprzeczność.

97

T = { (n, m) ∈ N × N | ϕ_n i ϕ_m to ta sama funkcja częściowa }

i

Analogicznie jak w 94, załóżmy nie wprost, że T jest r.e., niech ϕ_T: N × N → N – program semirozstrzygający przynależność do T, pokażemy, że wtedy da się skonstruować program semirozstrzygający przynależność do K'.

Intuicja: niech nasz program dla danego n użyje ϕ_T, żeby sprawdzić, czy "programom" p(_) = ϕ_n(n) i u(_) = ⊥ odpowiada ta sama funkcja częściowa.

- wczytaj n
- p := numer tego programu:
    - wczytaj _
    - zwróć ϕ_n(n)
- zwróć ϕ_T(p, u) // u jest taki sam dla każdego n

Dowód, że ten program semirozstrzyga przynależność do K':

- Dla n ∈ K':
    - p dla każdego wejścia się zapętla
    - ϕ_T(p, u) zwraca 1
- Dla n ∉ K' (n ∈ K)
    - p dla każdego wejścia zwraca tą samą liczbę
    - ϕ_T(p, u) nie zwraca 1

Wynika z tego, że K' jest r.e., sprzeczność.

ii

T' = (N × N) \ T = { (n, m) | ϕ_n i ϕ_m to różne funkcje częściowe }

Intuicja: jak w i., ale porównujemy p' = 1 ("program" stały) z p:

- wczytaj _
- wykonaj ϕ_n(n)
- zwróć 1

Dowód, że to semirozstrzyga przynależność do K':

- Dla n ∈ K':
  - p dla każdego wejścia się zapętla
  - ϕ_T(p, p') zwraca 1
- Dla n ∈ K:
  - p dla każdego wejścia zwraca 1
  - ϕ_T(p, p') nie zwraca 1

Wynika z tego, że K' jest r.e., sprzeczność.

98

Rel:
https://math.stackexchange.com/questions/1949380/is-the-set-of-indices-of-partial-computable-functions-with-finite-domains-r-e
https://www.wikiwand.com/en/Tarski%E2%80%93Kuratowski_algorithm
f: N × N → N
L = { n | f_n – funkcja rekurencyjna niepusta i Dom(f_n) skończona }
Jakie jest najmniejsze i, dla którego zachodzi L ∈ ∑i?

1

T: L ∈ ∑2
Pokażemy, że istnieje B co-r.e. taki że
A = { n ∈ N | istnieje m takie że f(n, m) ∈ B }

Niech B = { f(n, m) | f_n niepusta i zdefiniowana tylko dla i < m }
B jest co-r.e. Dowód:

1. $B \leq_{rek} K'$

2. B nie jest rekurencyjny

2

T: L ∉ ∑1
Czyli L nie jest r.e.

Dowód: załóżmy nie wprost, że jest, niech ϕ_L – program semirozstrzygajacy L, wtedy możemy skonstruować program semirozstrzygający przynależność do N \ K.

99

$A, B \subseteq N$
$f: A \rightarrow B$ – redukcja z A w B
$f[N] = N$

T: $B \leq_{rek} A$

Wiemy, że $x \in A \Leftrightarrow f(x) \in B$
Chcemy skonstruować g – funkcję rekurencyjną całkowitą taką że $x \in B \Leftrightarrow g(x) \in A$.
f nie musi mieć funkcji odwrotnej, ale na pewno dla każdego y jesteśmy w stanie znaleźć x takie że y = f(x) w skończonym czasie.

g(x):
- znajdź x' takie że f(x') = x
- zwróć x'

Dowód, że to redukcja:
1. $x \in B$
   Wtedy znajdujemy x' takie że f(x') = x, więc $f(x') \in B$, więc $x' \in A$.
2. $x \notin B$
   Wtedy znajdujemy x' takie że f(x') = x, więc $f(x') \notin B$, więc $x' \notin A$.

100

a. Nie, bo jego dopełnienie nie jest r.e.

Dopełnienie B – N \ B – zbiór takich programów, których dziedzina NIE jest odcinkiem początkowym N.

Pokażemy, że $N \setminus K \leq_{rek} N \setminus B$

Niech r – redukcja, która dla n zwraca numer takiego programu.
- wczytaj m
- wykonaj m kroków ϕ_n(n)
- jeśli otrzymano wynik, zapętl się
- zwróć 1

Dowód, że to redukcja ($n \in K \Leftrightarrow r(n) \in B$):

- Dla $n \in K$:
  - ϕ_n(n) zwróci wynik w m krokach lub więcej
  - r(n) zwraca wynik dla argumentów [1, m)
  - $r(n) \in B$
- Dla $n \notin K$:
  - ϕ_n(n) nie zatrzyma się nigdy
  - r(n) zawsze zwraca wynik
  - dziedziną r(n) jest N, czyli nie jest to odcinek początkowy
  - $r(n) \notin B$

b. Tak

Niech A = { (n, m, k) | dziedziną ϕ_n jest [1, m] i dla każdego argumentu z tej dziedziny ϕ_n zatrzymuje się w k krokach }.

Pokażemy, że A jest co-r.e. Równoważnie N \ A jest r.e.
Program n należy do N \ A, jeśli nie zwraca wartości dla któregoś z [1, m] w k krokach lub zwraca wartość dla jakiegoś m' > m.

Program semirozstrzygajacy przynależność do N \ A:

- wczytaj n, m, k
- dla i od 1 do m:
  - wykonaj k kroków ϕ_n(i)
- jeśli nie otrzymano wyniku dla któregokolwiek i, zwróć 1
- dla i, j w porządku Cantora dla [m + 1, \inf) × [0, \inf):
  - wykonaj j kroków ϕ_n(i)
  - jeśli otrzymano wynik, zwróć 1

101

Zbiór na pewno jest rekurencyjny, jeśli jest skończony albo jego dopełnienie jest skończone.

102

Rozwiązanie jak w 99

1. tak, idziemy od 0 w górę i w końcu coś znajdziemy
2. tak, idziemy po przekątnych (argument i ilość kroków) i w końcu coś znajdziemy

103

$D \subseteq P(N)$
Wskazówka: mając daną redukcję f, $A = f^{-1}[B]$ (przeciwobraz zbioru B przez f) jest tylko jeden.

$R \Longrightarrow L$

Jeśli istnieje $B \subseteq N$ taki że dla każdego $A \in D$ zachodzi $A \leq_{rek} B$, to D jest przeliczalny.
Ze wskazówki zbiorów A jest tyle, co redukcji w B.
Tych jest przeliczalnie wiele (bo odpowiadają im programy), więc D jest przeliczalny.

$L \Longrightarrow R$

D jest przeliczalny $\Longrightarrow$ Istnieje bijekcja $b: N \to D$.
Niech Cantor – bijekcja $N \times N \to N$.

Wtedy niech $B = \{ Cantor^{-1}(b^{-1}(A), a) \mid a \in A, A \in D \}$.

Dla danego A redukcją jest $r(a) = Cantor^{-1}(b^{-1}(A), a)$.

104

a. Tak,

Niech r będzie redukcją, która dla n zwraca numer takiego programu:
- wczytaj _
- znajdź (przekątniowo) jakiś argument, dla którego M_n zwróci wynik
- zwróć 1

Jeżeli $n \in Nemp$, to r(n) zwróci wynik dla każdego $n \in N$ (ten sam).
Jeżeli nie, to M_n nie zatrzyma się dla żadnego argumentu, więc r(n) się zapętli.

b. Nie, bo Nemp jest r.e., a Tot nie.

Program semirozstrzygający Nemp – uruchamiamy przekątniowo M_n i zwracamy 1 gdy dostaniemy jakiś wynik.

Dowód, że Tot nie jest r.e. – pokażemy, że $N \setminus K \leq_{rek} Tot$.

Niech r będzie redukcją, która dla danego n zwraca numer następującego programu:
- wczytaj m
- uruchom m kroków ϕ_n(n)

- jeśli otrzymano wynik, zapętl się
- zwróć 1

105

Nie. Załóżmy nie wprost, że tak.
Niech f = { (n, k) | ϕ_n(n) zwraca wynik w k krokach }

Niech f' – całkowita funkcja rekurencyjna zawierająca f.

Korzystając z f' możemy napisać program rozstrzygający K:
- wczytaj n
- k := f(n)
- wykonaj k kroków ϕ_n(n)
- jeśli otrzymano wynik, zwróć 1, wpp zwróć 0

K nie jest rekurencyjny, co daje sprzeczność.

https://www-m5.ma.tum.de/foswiki/pub/M5/Allgemeines/MA5116_2012S/lecture.pdf

112

https://www.wikiwand.com/en/Counter_machine

117

Ogólnie proste – robimy jak dla normalnego problemu słów, nie interesuje nas w jakim stanie się zatrzyma, ale czy w ogóle się zatrzyma. Sam fakt skończonej ilości konfiguracji nie wystarczy – możemy mieć ich skończoną ilość, ale cyklicznie przechodzić między nimi w nieskończoność. Dlatego dodajemy licznik ze wskazówki. Problem jest z tym, gdzie go dodać – jeśli na poziomie produkcji Thuego, to to jest dosyć trudne, bo potrzebujemy stanu, symbolu z prawej i symbolu z lewej, więc nie ma chyba gdzie go "wcisnąć". Jeśli na poziomie samej maszyny (jak to by wynikało ze wskazówki), to spoko. Ale tutaj trzeba pokazać, że dla dowolnej maszyny Turinga można "gdzieś na taśmie" dodać licznik kroków, co też jest trudne, bo co jeśli maszyna potrzebuje nieskończonej taśmy w dwie strony, a liczby kroków też nie możemy z góry ograniczyć? Machanie rękami…

118

To jest jak 110, tylko bez stanu. To jest niby deterministyczne…

128

Niech P(∑, Π, w, v) – problem z zadania.

Jakby dodać warunek, że v niepuste, to tak. (Wystarczyłoby przeglądać wszystkie możliwe transformacje BFSem wg rosnącej długości).

Nie mamy tego warunku, ale chyba bez niego problem też jest rozstrzygalny. Wiemy, że problem, czy z danej CFG można wygenerować słowo w, jest rozstrzygalny. Pokażemy redukcję z P do tego problemu.

Konstruujemy CFG odpowiadającą P.

1. Niech w = w_1w_2w_3… Zaczynamy od produkcji:
   S → W_1W_2W_3…
2. Dla każdego a_i ∈ ∑ dodajemy produkcję:
   A_i → a_i (terminal)
3. Dla każdego (w, v) ∈ Π, gdzie v = v_1v_2v_3… dodajemy produkcję:
   W → V_1V_2V_3…

Nasza CFG – krotka z rozszerzoną ∑, S, nowymi produkcjami i czymś jeszcze.
Wtedy P jest równoważny zapytaniu, czy z tej CFG da się wygenerować v.
Konwertujemy CFG do postaci normalnej Chomsky'ego i sprawdzamy wszystkie produkcje do 2|v| - 1 kroków.

129

Nie. Niech problem w zadaniu – P. Pokażemy, że semiThue $\leq_{rek}$ P.
Weźmy dowolny problem semiThue(∑, Π, w, v).
Skonstruujemy z Π nowe produkcje dla Π_P, być może rozszerzając ∑ o nowe symbole.

1. Dla każdego w → v ∈ Π, jeśli |w| > 1 i |v| > 1, rozszerzamy ∑ o nowy symbol WV i dodajemy zasady do Π_P:
   w → WV
   WV → v
2. Dla każdego a → v ∈ Π_P, jeśli |v| > 2, niech v = v_1v_2… Rozszerzamy ∑ o nowy symbol V[i:] i dodajemy produkcje do Π_P:
   a → V[1:]
   V[1:] → v_1V[2:]
   V[2:] → v_2V[3:]
   …

3. Dla każdego v → a ∈ Π_P, jeśli |v| > 2, niech v = v_1v_2…v_k. Dodajemy produkcje:
   v_k-1v_k → V[k-1:]
   v_k-2v_k-1 → V[k-2:]
   …
   V[1:] → a
   (Być może tu jest kolizja z 2., wtedy “zwijamy” od początku używając V[:i]).

semiThue nie jest rozstrzygalny, więc P też nie.

130

Niech problem w zadaniu – P. Tutaj możemy pokazać, że K $\leq_{rek}$ P, jeżeli pokażemy, że da się tutaj zakodować działanie maszyny Turinga (przejścia między konfiguracjami). Mamy ∑ = {0, 1}, wychodzimy od 1111, więc możemy wpisywać lub kasować zera pomiędzy tymi jedynkami. Korzystając z zad. 112, moglibyśmy zakodować 2 liczniki i stan maszyny unarnie w 3 miejscach. Czemu nie 4 liczniki? Bo musimy wiedzieć, gdzie co się zaczyna i kończy, np. produkcja 001111 → 0111 zadziałałaby też dla 0001111, a tego nie chcemy.

Wtedy nasze produkcje są postaci:

1111 → 11q_011
1a1q1b1 → 1a’1q’1b’1

Dla q_1, a, b → q_2, dec(a), inc(x)(b): (x ∈ {2, 3, 5, 7})

01q_11 → 1q_210^x

I chcemy sprawdzić, czy da się przez takie produkcje wygenerować 11q_F11 (puste liczniki, chyba zawsze łatwo je wyzerować).

114

Niech P114 – problem z zadania, tj: “Dla CFG G, H czy L_G ∩ L_H ≠ ∅”?

T: PCP $\leq_{rek}$ P114

Dają nam ∑ oraz g, h – listy słów z ∑*. Chcemy rozstrzygnąć PCP przy użyciu P114.
Musimy skonstruować 2 CFG, które będą generowały odpowiednie konkatenacje słów z g, h.

G → g[0]G0 | g[1]G1 | … | g[n]Gn | #
H → h[0]H0 | h[1]H1 | … | h[n]Hn | #

Wtedy istnieje w generowane przez obie gramatyki, jeśli zgadza się rezultat (konkatenacja słów) oraz (odwrócony) ciąg indeksów, przy użyciu którego został uzyskany, czyli równoważne PCP.

Czy P114 jest r.e.? Tak, przeszukujemy BFS-em grafy produkcji G i H, w końcu znajdziemy jakieś wspólne słowo, jeśli takie istnieje.

115

Niech P115 – problem z zadania, tj. “Dla $\sum$ i CFG G, czy $L_G = \sum^*$”?

T: PCP $\leq_{rek}$ P114 $\leq_{rek}$ P115

Dają nam G i H. Chcemy rozstrzygnąć, czy $L_G \cap L_H = \varnothing$.
Tworzymy $L_K = L_G' \cup L_H' = (L_G \cap L_H)'$, sprawdzamy, czy $L_K = \sum^*$.
Tak się da, bo gramatyki z redukcji PCP $\leq_{rek}$ P114 są deterministyczne.
Dopełnienie deterministycznego CFL jest deterministycznym CFL.
Suma CFL jest CFL.

116

Niech P116 – problem z zadania, tj. “Dla CFG G, H czy $L_G = L_H$”?

T: P115 $\leq_{rek}$ P116

Dają nam $\sum$ i G. Chcemy rozstrzygnąć, czy $L_G = \sum^*$.
Tworzymy H t. że $L_H = \sum^*$, sprawdzamy, czy $L_G = L_H$.

120

T: K $\leq_{rek}$ P120

Dają nam TM, chcemy rozstrzygnąć, czy się zatrzyma.

Niech czerwony – kolor specjalny, biały – blank, reszta kolorów – zbiór $\sum \cup Q \times \sum$.
Zamiast nieestetycznych chyba lepiej definiować estetyczne ułożenia, tj. dozwolone.

Estetyczne ułożenia kafelków:

Dla każdych q, a jeśli ∂(q, a) = (q', a', L):

x	(q, a)
(q', x)	a'

Dla każdych q, a jeśli ∂(q, a) = (q', a', R):

(q, a)	x
a'	(q', x)

Dodatkowo, żeby móc zacząć od czerwonego kafelka i "wrócić" z qF do czerwonego kafelka:

(q0, B)	

oraz

(qF, B)	

(Zakładamy, że mamy TM, która potrafi skończyć w qF z pustą taśmą).

Być może przyda się też coś takiego, żeby kwadrat mógł być kwadratowy, ale to raczej detal:

Jeżeli istnieje kwadrat o skończonym boku, który da się tak pokryć, to TM się zatrzymuje.

121

T1: kcomp może być dowolnie duże.
(Dla każdego k istnieje n takie że kcomp(n) > k).

T2: kcomp nie jest rekurencyjna.

Korzystamy z T1. Załóżmy nie wprost, że kcomp jest rekurencyjna. Niech LI(f) – liczba znaków programu (obliczającego) f.

Napiszemy program ϕ:

- LTOT := LI(kcomp) + LI(ϕ)
- dla każdego n ∈ N:
    - jeśli kcomp(n) > LTOT:
        - wypisz n

ϕ() zwróci n, do którego wypisania potrzeba programu o większej niż LTOT liczbie znaków, jednocześnie samemu mając dokładnie LTOT znaków, co daje sprzeczność.

136

Dają nam instancję PCP(∑, w, v), gdzie w, v to listy słów.
Chcemy skonstruować Żuczka Kleksiorka, który akceptuje słowo będące odwzorowaniem rozwiązania PCP (lista indeksów + być może dodatkowe informacje).

Słowo do zaakceptowania:
efekt końcowy w # indeksy # efekt końcowy v

Idea:
Czytamy indeks i, wchodzimy do stanu, w którym wiemy, jakie jest w[i], przechodzimy do efektu końcowego w, sprawdzamy, czy początek niezakleksowanej części się zgadza, jak tak, to zakleksowujemy dalsze w[i] (jak coś pójdzie nie tak, to fail), przechodzimy do stanu, w którym znamy v[i] oraz efektu końcowego v, postępujemy analogicznie, jeśli wszystko wyszło ok to wracamy i zakleksowujemy obecny indeks. Jeśli rozwiązanie PCP jest OK, to cała taśma zostanie zakleksowana i możemy przejść do qF.

122

a. H10prim $\leq_{rek}$ H10
b. H10 $\leq_{rek}$ H10prim

Dają nam instancję H10prim. Każdą liczbę całkowitą da się przedstawić jako różnicę dwóch liczb naturalnych, więc:

a1x1^e1x2^e2…xk^ek + … = 0
a1(x1’ - x1’’)^e1(x2’ - x2’’)^e2…(xk’ - xk’’)^ek + … = 0

Dają nam instancję H10. Każdą liczbę naturalną da się przedstawić jako sumę kwadratów 4 liczb całkowitych (Lagrange), więc:

a1x1^e1x2^e2…xk^ek + … = 0
a1(x1a^2 + x1b^2 + x1c^2 + x1d^2)^e1… + … = 0

123

T: H10bis jest rozstrzygalny.

Mamy dwie maszyny Turinga M1 i M2, niech M1 sprawdza po kolei rozwiązania, a M2 próbuje pokazać, że nie ma rozwiązania – któraś się w końcu zatrzyma.
https://math.stackexchange.com/questions/181380/second-degree-diophantine-equations
https://www.wikiwand.com/en/Hasse_principle