20 czerwca 2008 *drak*

Kilka rozwiązań[1] łatwych zadań z języków formalnych i złożoności obliczeniowej

„Mało Zadeptany Śnieg"

Zadanie 43. Rozszerz definicję zbioru rekurencyjnego tak aby można było rozważać rekurencyjne zbiory par liczb naturalnych i udowodnij, że jeśli zbiór $A \subset \mathcal{N}^2$ jest rekurencyjny, to zbiór $\{n : \exists m\ [n,m] \in A\}$, czyli rzut A na pierwszą oś, jest zbiorem rekurencyjnie przeliczalnym.

Rozwiązanie. Skoro A jest zbiorem rekurencyjnym, to istnieje φ_A – całkowita funkcja rekurencyjna rozstrzygająca przynależność do A i $\varphi_A \subset \mathcal{N} \times \mathcal{N} \to \{0,1\}$.

Zbiór $B = \{n : \exists m\ [n,m] \in A\}$ jest rekurencyjnie przeliczalny.

Dowód. Zbudujemy[2] częściową funkcję φ_B, taką że $\varphi_B(n) = 1 \Leftrightarrow n \in B$:

- **read** n
- **for** $m \leftarrow 1$ **to** ∞ **do**:
  - **if** $\varphi_A(n,m) = 1$: **return** 1

□

Zadanie 44. Pokaż, że każdy zbiór rekurencyjnie przeliczalny jest rzutem pewnego zbioru rekurencyjnego, to znaczy jeśli B jest r.e. to istnieje taki rekurencyjny $A \subset \mathcal{N}^2$ rekurencyjny, że $B = \{n : \exists m\ [n,m] \in A\}$.

Rozwiązanie. Skoro $B = \{n : \exists m\ [n,m] \in A\}$ jest r.e., to istnieje φ_B, że $\varphi_B(n) = 1 \Leftrightarrow n \in B$. Niech $A = \{[n,m] : \varphi_B(n) = 1$ i $\varphi_B(n)$ zatrzymał się po m krokach$\}$.

Zbiór A jest rekurencyjny.

Dowód. Zbudujemy[3] [4] całkowitą, rekurencyjną funkcję $\varphi_A \subset \mathcal{N} \times \mathcal{N} \to \{0,1\}$ taką, że $\forall n \in \mathcal{N}\ \varphi_A(n) = 1 \Rightarrow n \in A$ i $\varphi_A(n) = 0 \Rightarrow n \notin A$:

- **wczytaj** n `i` m
- **uruchom** $\varphi_B(n)$ `na` m `kroków`
- **jeśli** $\varphi_B(n)$ `zatrzymał się po` m `krokach`:
  - **zwróć** 1
- `w przeciwnym przypadku` **zwróć** 0

□

[1] wcale nie twierdzę, że w 100% poprawnych i bardzo formalnie napisanych...
[2] oczywiście podamy program napisany w „M.U.J.P.^{TM}" obliczający tę funkcję...
[3] j.w.
[4] „M.U.J.P.^{TM}" jest mieszanką słów polskich i angielskich, oraz zawsze prawidłowo odczytuje intencje programisty

Zadanie 45. Powtórz podany na wykładzie dowód nierozstrzygalności problemu stopu, to znaczy faktu, że zbiór $K = \{n : \phi_n(n) \in \mathcal{N}\}$ nie jest rekurencyjny.

Rozwiązanie[5]**.**

Zbiór $K = \{n : \phi_n(n) \in \mathcal{N}\}$ nie jest rekurencyjny.

Dowód. (nie-wprost) Załóżmy, że zbiór K jest rekurencyjny. Istnieje wtedy całkowita funkcja rekurencyjna $\varphi_K \subset \mathcal{N} \to \{0, 1\}$ rozstrzygająca przynależność do K, t.j. taka, że $\forall n \in \mathcal{N}\ \varphi_K(n) = 1 \Rightarrow n \in K$ i $\varphi_K(n) = 0 \Rightarrow n \notin K$. Rozważmy zatem taki oto program ψ (o numerze p):

```
– wczytaj n
– jeśli φ_K(n) = 1: zapętl się
– w przeciwnym przypadku zwróć 1
```

Zachodzą następujące możliwości:

- $\psi(p) = 1 \Rightarrow p \notin K \Rightarrow \psi(p) \notin \mathcal{N}$ – $\lightning$,
- $\psi(p) = \perp \Rightarrow p \in K \Rightarrow \psi(p) = 1$ – $\lightning$,

□

Zadanie 46. Pokaż, że $\{n : |Dom(\phi_n)| \geq 7\}$ jest rekurencyjnie przeliczalny.

Rozwiązanie.

Zbiór $A = \{n : |Dom(\phi_n)| \geq 7\}$ jest rekurencyjnie przeliczalny.

Dowód. Zdefiniujemy[6] funkcję $\varphi_A \subset \mathcal{N} \to \{1\}$, że $\varphi_A(n) = 1 \Leftrightarrow n \in B$:

```
– read n
– arg_set ← φ, steps ← 1, m ← 0
– loop:
    – for arg ← 0 to m:
        – if arg ∉ arg_set:
            – uruchom φ_n(arg) na steps kroków
            – if φ_n(arg) zakończył się po steps krokach:
                – arg_set ← arg_set ∪ {arg}
    – if |arg_set| ≥ 7:
        – return 1
    – else:
        – steps++, m++
```

Idea programiku jest taka: przy każdym „obrocie" pętli (**loop**) powiększamy przedział argumentów ($\langle 0, m \rangle$) oraz wydłużamy czas (*steps*) jaki będziemy dawali funkcji ϕ_n na obliczenie wyniku (zatrzymanie się). □

[5] może nie jest idealnie takie jak na wykładzie, ale dowodzi tego co trzeba...

[6] „M.U.J.P.^{TM}" (tak jak np. Python) jest językiem wcięciowym!

Zadanie 47. Udowodnij, że jeśli ϕ jest niemalejącą, całkowitą funkcją rekurencyjną, to zbiór $\phi(\mathcal{N})$ jej wartości jest rekurencyjny. Czy pozostaje to prawdą bez założenia o całkowitości ϕ?

Rozwiązanie.

ϕ – niemalejąca, całkowita funkcja rekurencyjna,
$\phi(\mathcal{N})$ jest zbiorem rekurencyjnym.

Dowód.

1. Jeśli ϕ jest od pewnego miejsca stała, to zbiór $\phi(\mathcal{N})$ jest skończony, a zatem obliczalny.
2. W przeciwnym przypadku zdefiniujmy program ψ rozstrzygający przynależność do zbioru $\phi(\mathcal{N})$:

- **read** n
- $m \leftarrow 0$
- **loop**:
  - **if** $\phi(m) = n$: **return** 1
  - **if** $\phi(m) > n$: **return** 0
  - m++

Funkcja ϕ jest niemalejąca, co gwarantuje, że program ψ zawsze się zatrzyma.

□

Jeżeli ϕ nie jest całkowitą funkcją rekurencyjną, to $B = \phi(\mathcal{N})$ nie jest zbiorem rekurencyjnym.

Dowód. Wystarczy wziąć funkcję spełniającą równanie $\phi(n) = \begin{cases} n, & \text{jeśli } \varphi_n(n) \in \mathcal{N} \\ \perp, & \text{w p.p.} \end{cases}$ (gdzie $n \in \mathcal{N}$), której przeciwdziedzinę stanowi zbiór K[7]. □

Zadanie 48. Udowodnij, że każdy niepusty zbiór rekurencyjnie przeliczalny jest postaci $\phi(\mathcal{N})$ dla pewnej całkowitej funkcji rekurencyjnej ϕ.

Rozwiązanie.

Jeśli A jest niepustym zbiorem r.e., to $A = \phi(\mathcal{N})$, gdzie ϕ – całkowita funkcja rekurencyjna.

Dowód. Skoro A jest zbiorem rekurencyjnie przeliczalnym, to istnieje φ_A – program semi-rozstrzygający przynależność do A[8]. Niech $\psi \subset \mathcal{N} \to \mathcal{N} \times \mathcal{N}$ będzie bijekcją ze zbioru liczb naturalnych w zbiór par liczb naturalnych. Zdefiniujmy program (niech nazywa się γ) obliczający funkcję ϕ:

- **wczytaj** n
- *licznik* $\leftarrow 0$
- **for** $k \leftarrow 1$ **to** ∞ **do**:
  - $[i, j] \leftarrow \psi(k)$
  - **uruchom** $\varphi_A(i)$ `na` j `kroków`
  - **jeśli** $\varphi_A(i)$ `zwraca` 1 `po co najwyżej` j `krokach`:
    - *licznik*++
  - **jeśli** *licznik* $= n$:
    - **zwróć** i

[7] nie będący zbiorem rekurencyjnym...
[8] czyli taki, że $\varphi_A(n) = 1 \Leftrightarrow n \in A$

- ϕ jest rekurencyjna, bo istnieje program (γ) ją implementujacy,
- ϕ jest całkowita – skoro zbiór A jest niepusty, to ϕ zatrzymuje się dla dowolnego $n \in \mathcal{N}$,
- $\phi(\mathcal{N}) = A$, ponieważ

dla każdego $n \in \mathcal{N}$ istnieje takie $i \in \mathcal{N}$, że $\phi(n) = i$
wtedy i tylko wtedy, gdy
istnieje $j \in \mathcal{N}$, że $\varphi_A(i)$ zatrzymuje się po co najwyżej j krokach
(wtw $i \in A$)

□

Zadanie 49. Udowodnij, że każdy nieskończony zbiór rekurencyjnie przeliczalny jest postaci $\phi(\mathcal{N})$ dla pewnej całkowitej, różnowartościowej funkcji rekurencyjnej ϕ.

Rozwiązanie[9]**.**

Jeśli A jest nieskończonym zbiorem r.e., to $A = \phi(\mathcal{N})$,
gdzie ϕ – całkowita, różnowartościowa funkcja rekurencyjna.

Dowód. Skoro A jest zbiorem rekurencyjnie przeliczalnym, to istnieje φ_A – program semi-rozstrzygający przynależność do A[10]. Niech $\psi \subset \mathcal{N} \to \mathcal{N} \times \mathcal{N}$ będzie bijekcją ze zbioru liczb naturalnych w zbiór par liczb naturalnych. Zdefiniujmy program (niech nazywa się γ) obliczający funkcję ϕ:

- **wczytaj** n
- *licznik* $\leftarrow 0$
- **for** $k \leftarrow 1$ **to** ∞ **do**:
  - $[i, j] \leftarrow \psi(k)$
  - **uruchom** $\varphi_A(i)$ `na` j `kroków`
  - **jeśli** $\varphi_A(i)$ `zwraca` 1 `po dokładnie` j `krokach`:
    - *licznik*++
  - **jeśli** *licznik* $= n$:
    - **zwróć** i

- ϕ jest rekurencyjna, bo istnieje program (γ) ją implementujacy,
- ϕ jest całkowita – skoro zbiór A jest nieskończony, to ϕ zatrzymuje się dla dowolnego $n \in \mathcal{N}$,
- $\phi(\mathcal{N}) = A$, ponieważ

dla każdego $n \in \mathcal{N}$ istnieje takie $i \in \mathcal{N}$, że $\phi(n) = i$
wtedy i tylko wtedy, gdy
istnieje $j \in \mathcal{N}$, że $\varphi_A(i)$ zatrzymuje się po dokładnie j krokach
(wtw $i \in A$)

- ϕ jest różnowartościowa. Gdyby tak nie było, to dla pewnego i istnieją takie j_1 oraz j_2, że $\varphi_A(i)$ zatrzymuje się po dokładnie j_1 oraz dokładnie j_2 krokach. ↯

□

[9] tak, niemalże identyczne do rozwiązania zadania 48.
[10] $\varphi_A(n) = 1 \Leftrightarrow n \in A$

Zadanie 50. Udowodnij, że zbiór $\{n : Dom(\phi_n) = \mathcal{N}\}$ nie jest rekurencyjnie przeliczalny.

Rozwiązanie.

Zbiór $A = \{n : Dom(\phi_n) = \mathcal{N}\}$ nie jest rekurencyjnie przeliczalny.

Dowód. (nie-wprost) Załóżmy, że A jest r.e. Istnieje zatem program φ_A semi-rozstrzygający go[11]. Rozważmy sobie taki program ψ:

- **wczytaj** n
- **znajdź** `numer takiego programu (nie uruchamiaj go!)`:
  - **wczytaj** m
  - **jeśli** $\varphi_n(n)$ `zwraca wynik po` m `krokach`:
    - `zapętl się`
  - `w przeciwnym przypadku` **zwróć** 1
- $t \leftarrow$ `numer znalezionego powyżej programu`
- **zwróć** $\varphi_A(t)$

Zatem:

- jeśli $n \in \widehat{K}$[12], to $\varphi_n(n) \notin \mathcal{N}$, więc nie istnieje takie m, żeby $\varphi_n(n)$ zakończył się po m krokach. Zatem dla takiego n program $\varphi_t(m) = 1$ dla każdego $m \in \mathcal{N}$. Skoro dziedziną programu φ_t jest zbiór liczb naturalnych, to $t \in A$. Zatem $\varphi_A(t) \in \mathcal{N}$, więc program ψ zatrzyma się.
- jeśli $n \notin \widehat{K}$, to $\varphi_n(n) \in \mathcal{N}$, więc istnieje takie m, że program $\varphi_n(n)$ zakończy działanie po m krokach. Zatem dla takiego n program $\varphi_t(m)$ dla pewnego m zapętli się. $Dom(\varphi_t) \neq \mathcal{N}$, zatem $t \notin A$. Z tego wynika, że program ψ się nie zatrzyma, bo $\varphi_A(t)$ się nie zatrzyma.

Program ψ semi-rozstrzyga przynależność do $\widehat{K}$, a ten nie jest rekurencyjnie przeliczalny[13]. ↯ □

Zadanie 53. Udowodnij, że zbiór $B = \{n : Dom(\phi_n)$ i $\mathcal{N} - Dom(\phi_n)$ są nieskończone$\}$ z poprzedniego zadania nie jest nawet rekurencyjnie przeliczalny.

Rozwiązanie.

Zbiór $B = \{n : Dom(\phi_n)$ i $\mathcal{N} - Dom(\phi_n)$ są nieskończone$\}$ nie jest r.e.

Dowód. (nie-wprost) Załóżmy, że B jest zbiorem rekurencyjnie przeliczalnym. Istnieje zatem φ_B – program semi-rozstrzygający go[14]. Rozważmy taki program γ:

- **wczytaj** n
- **znajdź** `numer takiego programu (nie uruchamiaj go!)`:
  - **wczytaj** m
  - **jeśli** m **mod** $2 \neq 0$: `zapętl się`
  - **uruchom** $\varphi_n(n)$ `na co najwyżej` m `kroków`
  - **jeśli** $\varphi_n(n)$ `zatrzymał się po` m `lub mniej krokach: zapętl się`
  - **zwróć** 1
- $t \leftarrow$ `numer znalezionego powyżej programu`
- **zwróć** $\varphi_B(t)$

[11] a JMa na wykładzie mówił, że nie lubi tego określenia...
[12] $\widehat{K}$ – dopełnienie zbioru K
[13] bo gdyby był, to K byłby rekurencyjny
[14] taki, że $\varphi_B(n) = 1 \Leftrightarrow n \in B$

Powyższy program świadczyłby o tym, że zbiór $\widehat{K}$[15] jest r.e. ↯

- dla $n \in \widehat{K}$ nie istnieje takie m, że $\varphi_n(n)$ zatrzyma się, zatem φ_t dla każdego parzystego m zwróci 1, więc $\varphi_B(t) \in \mathcal{N}$,
- dla $n \notin \widehat{K}$ istnieje takie m, że $\varphi_n(n)$ zatrzyma się po m krokach. Zatem istnieje i, że dla każdego $m > i$ $\varphi_t(m) = \perp$, a to oznacza, że $Dom(\varphi_t)$ jest skończona, więc $\varphi_B(t) \notin \mathcal{N}$.

□

Zadanie 54. Niech A, B, C, D będą zbiorami rekurencyjnie przeliczalnymi, takimi że każda liczba naturalna należy do dokładnie dwóch z nich. Udowodnij, że w takim razie wszystkie te cztery zbiory są rekurencyjne.

Rozwiązanie.

Dowód. Skoro A, B, C i D są zbiorami r.e., to niech φ_A, φ_B, φ_C oraz (odpowiednio) φ_D będą programami semi-rozstrzygającymi przynależność do tych zbiorów. Rozważmy program ψ_A (analogicznie ψ_B, ψ_C oraz ψ_D):

```
1 – wczytaj n
2 – Ω ← {φ_A, φ_B, φ_C, φ_D}
3 – for k ← 1 to ∞ do:
4     – uruchom wszystkie programy ze zbioru Ω z argumentem n na k kroków
5     – jeśli po k krokach φ_A(n) = 1: zwróć 1
6     – jeśli po k krokach φ_x(n) = 1 ∧ φ_x ≠ φ_A:
7         – Ω ← Ω\{φ_x}
8         – jeśli |Ω| = 2: zwróć 0
```

Program ψ_A zatrzymuje się zawsze. Zwraca wartość 1 jeśli podany argument należy do zbioru A i 0 w przeciwnym przypadku. Aby otrzymać program ψ_B (oraz odpowienio ψ_C i ψ_D) należy w linii 5 powyższego programu zamienić „$\varphi_A(n) = 1$” na „$\varphi_B(n) = 1$” („$\varphi_C(n) = 1$”, „$\varphi_D(n) = 1$”) i w linii 6 „$\varphi_x \neq \varphi_A$” na „$\varphi_x \neq \varphi_B$” („$\varphi_x \neq \varphi_C$”, „$\varphi_x \neq \varphi_D$”). □

Zadanie 55. Udowodnij, że zbiór numerów tych programów które zatrzymują się dla wszystkich argumentów oprócz co najwyżej skończonej ilości, nie jest rekurencyjnie przeliczalny.

Rozwiązanie.

B – zbiór opisany w treści zadania – nie jest r.e.

Dowód. (nie-wprost) Załóżmy, że B jest r.e., a φ_B to funkcja semi-rozstrzygająca go. Rozważmy taki oto program γ:

```
– wczytaj n
– znajdź numer takiego programu (nie uruchamiaj go!):
    – wczytaj m
    – jeśli m < 27: zapętl się
    – jeśli φ_n(n) zatrzyma się po co najwyżej m krokach: zapętl się
    – zwróć 1
– t ← numer znalezionego powyżej programu
– zwróć φ_B(t)
```

[15] $\widehat{K}$ – dopełnienie zbioru K

Analiza powyższego programu:

- jeśli $n \in \widehat{K}$ to nie istnieje takie m, że $\varphi_n(n)$ zatrzyma się po m krokach, więc dla wszystkich $m \geq 27$ (czyli wszystkich poza skończoną ilością) program $\varphi_t(m)$ zatrzyma się, więc $\varphi_B(t) \in \mathcal{N}$, a co za tym idzie $\gamma(n) \in \mathcal{N}$,
- jeśli $n \notin \widehat{K}$ to istnieje takie i, że dla każdego każdego $m > i$ program $\varphi_n(n)$ zatrzyma się po co najwyżej m krokach, zatem dla takich m program $\varphi_t(m)$ nie zatrzyma się, więc $t \notin B$, więc $\varphi_B(t) \notin \mathcal{N}$, zatem $\gamma(n) = \perp$.

Z tego wynika, że $\widehat{K}$ jest r.e. $\lightning$ □

Zadanie 57. Poniższe zbiory nie są rozstrzygalne. Nierozstrzygalność których z nich daje się udowodnić wprost z twierdzenia Rice'a?

Rozwiązanie.

1. zbiór numerów tych maszyn Turinga które obliczają funkcje o dziedzinie różnej od $\mathcal{N}$;

   Nierozstrzygalność tego zbioru **<u>daje się</u>** udowodnić wprost z twierdzenia Rice'a.

   Dowód. Niech A będzie opisanym wyżej zbiorem.

   - A jest nietrywialny, ponieważ $A \neq \phi$ i $A \neq \mathcal{N}$,
   - A jest ekstensjonalny – weźmy dowolne dwie maszyny Turinga obliczające tę samą funkcję φ. Jeśli $Dom(\varphi) \neq \mathcal{N}$, to numery obu maszyn należą do zbioru A, a gdy $Dom(\varphi) = \mathcal{N}$, to numery obu maszyn nie należą do opisywanego zbioru.

   □

2. zbiór numerów tych maszyn Turinga które obliczają funkcję całkowitą i których czas działania jest rosnący względem rozmiaru danych;

   Nierozstrzygalność tego zbioru **<u>nie daje się</u>** udowodnić wprost z twierdzenia Rice'a.

   Dowód. Niech B będzie opisanym wyżej zbiorem. Ten zbiór nie jest ekstensjonalny. Weźmy dwie maszyny Turinga M_α i M_β. Maszyna M_α działa tak, że na taśmie wypisuje symbol 1 i kończy działanie. Maszyna M_β najpierw przepisuje swoje wejście[16] a następnie wypisuje symbol 1 i kończy działanie. Obie maszyny obliczają tę samą funkcję całkowitą ($\forall n \in \mathcal{N}\ f(n) = 1$), lecz $\alpha \notin B$, natomiast $\beta \in B$. □

3. zbiór numerów tych maszyn Turinga których czas działania dla żadnych danych nie jest liczbą pierwszą;

   Nierozstrzygalność tego zbioru **<u>nie daje się</u>** udowodnić wprost z twierdzenia Rice'a.

   Dowód. Niech C będzie opisanym wyżej zbiorem. Ten zbiór nie jest ekstensjonalny. Wystarczy wziąć dwie maszyny Turinga wypisujące na taśmie symbol 1, z których jedna przed wypisaniem symbolu wykona np. 16 ruchów głowicą[17]. Obie maszyny obliczają tę samą funkcję, a numer tylko jednej z nich należy do zbioru C. □

[16] potem w jakiś umowny sposób wymazuje to co napisała

[17] 16 ruchów + 1 po wypisaniu symbolu, czyli liczba kroków to 17

4. zbiór numerów tych maszyn Turinga które obliczają funkcje częściowe, których wartościami są wyłącznie liczby pierwsze;

   Nierozstrzygalność tego zbioru **daje się** udowodnić wprost z twierdzenia Rice'a.

   Dowód. Niech D będzie opisanym wyżej zbiorem.

   - D jest nietrywialny, ponieważ $D \neq \phi$ i $D \neq \mathcal{N}$,
   - A jest ekstensjonalny – weźmy dowolne dwie maszyny Turinga obliczające tę samą funkcję f. Jeśli wartościami tej funkcji są wyłącznie liczby pierwsze, to numery obu tych maszyn należą do zbioru D, w przeciwnym przypadku numery obu tych maszyn nie należą do D.

   □

Zadanie 58. Udowodnij nierozstrzygalność zbioru z punktu 2. poprzedniego zadania.

Rozwiązanie.

Zbiór $A = \{n \in \mathcal{N} : M_n$ oblicza funkcję całkowitą i $\forall m \in \mathcal{N}\ T(M_n(m)) < T(M_n(m+1))\}$ – nie jest rekurencyjny.

Dowód. (nie-wprost) Załóżmy, że A jest rekurencyjny. Istnieje zatem a φ_A – program rozstrzygający przynależność do niego[18]. Rozważmy taki oto program γ[19]:

```
1 – wczytaj n
2 – znajdź numer takiego programu (nie uruchamiaj go!):
3     – wczytaj m
4     – for i ← 1 to m do:
5         – skip
6     – uruchom φ_n(n)
7     – zwróć 1
8 – t ← numer znalezionego powyżej programu
9 – zwróć φ_A(t)
```

Powyższy program świadczyłby o tym, że zbiór K jest rekurencyjny. ↯

- jeśli $n \in K$ to $\varphi_n(n) \in \mathcal{N}$, a zatem dla każdego $m \in \mathcal{N}$ $\varphi_t(m) = 1$. Skoro φ_t oblicza funkcję całkowitą, a pętla w liniach 4 i 5 gwarantuje, że czas działania φ_t jest zależny od rozmiaru danych, to $\varphi_A(t) = 1$, a zatem $\gamma(n) = 1$,
- jeśli $n \notin K$ to $\varphi_n(n) \notin \mathcal{N}$, a zatem dla każdego $m \in \mathcal{N}$ $\varphi_t(m) = \perp$. Skoro φ_t nie oblicza funkcji całkowitej, to $\varphi_A(t) = 0$, a zatem $\gamma(n) = 0$.

□

[18] czyli $\varphi_A(n) = 0 \Rightarrow n \notin A \wedge \varphi_A(n) = 1 \Rightarrow n \in A$

[19] implicite korzystamy tu z Tezy Churcha i pokazujemy algorytm w „M.U.J.P.^{TM}" wiedząc, że takie obliczenie da się przeprowadzić na maszynie Turinga

Zadanie 59. Niech A, $B \subset \mathcal{N}$. Mówimy, że $A \leq_{rek} B$ jeśli istnieje całkowita funkcja rekurencyjna f (zwana redukcją) taka że $f(x) \in B$ wtedy i tylko wtedy gdy $x \in A$. Pokaż, że dla każdych dwóch zbiorów A, $B \subset \mathcal{N}$ istnieje ich najmniejsze ograniczenie górne w sensie $\leq_{rek}$, to znaczy taki zbiór C, że:

(i) $A \leq_{rek} C$ i $B \leq_{rek} C$,

(ii) jeśli D jest taki, że $A \leq_{rek} D$ i $B \leq_{rek} D$ to $C \leq_{rek} D$.

Rozwiązanie.

Dowód. Niech $C = \{n : n = 2m \wedge m \in A\} \cup \{n : n = 2m + 1 \wedge m \in B\}$. Pokażemy, że zbiór C jest najmniejszym ograniczeniem górnym (w sensie $\leq_{rek}$) zbiorów A i B.

(i) Aby pokazać, że C jest ograniczeniem górnym (w sensie $\leq_{rek}$) zbiorów A i B należy skonstruować odpowiednie redukcje. Są nimi całkowite funkcje rekurencyjne $f_A(n) = 2n$ i $f_B(n) = 2n + 1$ spełniające poniższe równoważności:

$$n \in A \Leftrightarrow f_A(n) \in C \quad \text{i} \quad n \in B \Leftrightarrow f_B(n) \in C$$

(ii) Weźmy dowolny zbiór D taki, że $A \leq_{rek} D$ i $B \leq_{rek} D$ oraz odpowiednie redukcje f_α i f_β (oczywiście takie, że $n \in A \Leftrightarrow f_\alpha(n) \in D$ oraz $n \in B \Leftrightarrow f_\beta(n) \in D$). Pokażemy, że $C \leq_{rek} D$, tzn. zdefiniujemy odpowiednią redukcję:

$$f(n) = \begin{cases} f_\alpha(\frac{n}{2}), & \text{dla } n \text{ parzystych} \\ f_\beta(\frac{n-1}{2}), & \text{dla } n \text{ nieparzystych} \end{cases}$$

Należy teraz pokazać, że ta redukcja spełnia równoważność $n \in C \Leftrightarrow f(n) \in D$:

1. ($\Rightarrow$)
   - n jest parzyste, wtedy $\frac{n}{2} \in A$ i $f(n) = f_\alpha(\frac{n}{2}) \in D$,
   - n jest nieparzyste, wtedy $\frac{n-1}{2} \in B$ oraz $f(n) = f_\beta(\frac{n-1}{2}) \in D$.
2. ($\Leftarrow$)
   - Gdy n jest parzyste to $f(n) = f_\alpha(\frac{n}{2})$, więc $\frac{n}{2} \in A$. Ale z definicji redukcji f_A wiemy, że $n \in A \Leftrightarrow 2n \in C$, zatem $2 \cdot \frac{n}{2} = n \in C$,
   - Gdy n jest nieparzyste to $f(n) = f_\beta(\frac{n-1}{2})$, więc $\frac{n-1}{2} \in B$. Ale z definicji redukcji f_B wiemy, że $n \in A \Leftrightarrow 2n + 1 \in C$, zatem $2 \cdot \frac{n-1}{2} + 1 = n \in C$.

□

Zadanie 60. (za 2 punkty) Czy $K \leq_{rek} \widehat{K}$? Czy $\widehat{K} \leq_{rek} K$?

Rozwiązanie.

1. $\widehat{K} \not\leq_{rek} K$

Dowód. (nie-wprost) Załóżmy, że $\widehat{K} \leq_{rek} K$. Istnieje zatem redukcja f, że $\forall n \in \mathcal{N}\ n \in \widehat{K} \Leftrightarrow f(n) \in K$. Wiemy, że K jest r.e., zatem istnieje program φ_K semi-rozstrzygający przynależność do niego. Rozważmy zatem taki oto program γ:

```
– wczytaj n
– uruchom φ_K(f(n))
– zwróć 1
```

Program γ zwraca 1 wtedy i tylko wtedy gdy podany mu argument n należy do zbioru $\widehat{K}$. A przecież $\widehat{K}$ nie jest rekurencyjnie przeliczalny. ↯ □

2. $K \not\leq_{rek} \widehat{K}$

Dowód. (nie-wprost) Załóżmy, że $K \leq_{rek} \widehat{K}$. Istnieje zatem redukcja f, że $\forall n \in \mathcal{N}\ n \in K \Leftrightarrow f(n) \in \widehat{K}$. Weźmy dowolny $x \in \widehat{K}$. Oczywiście $x \notin K$. Zatem $f(x) \notin \widehat{K}$[20], czyli $f(x) \in K$. Otrzymaliśmy, że $x \in \widehat{K} \Leftrightarrow f(x) \in K$, co implikuje, że $\widehat{K} \leq_{rek} K$, ale udowodniliśmy w punkcie 1., że nie jest to prawdą. ↯ □

Zadanie 62. Niech T będzie zbiorem tych par liczb $\langle n, m\rangle$ dla których ϕ_m i ϕ_n to ta sama funkcja częściowa.

a. Pokaż, że T nie jest rekurencyjnie przeliczalny.

b. Czy dopełnienie zbioru T jest rekurencyjnie przeliczalne?

Rozwiązanie.

a. Zbiór $T = \{\langle n, m\rangle : \phi_m \text{ i } \phi_n \text{ to ta sama funkcja częściowa}\}$ nie jest r.e.

Dowód. (nie-wprost) Załóżmy, że T jest zbiorem rekurencyjnie przeliczalnym. Istnieje zatem φ_T – program semi-rozstrzygający go[21]. Rozważmy taki program γ:

- **wczytaj** n
- **znajdź** `numer takiego programu (nie uruchamiaj go!)`:
  - **wczytaj** m
  - **jeśli** $m = 13$: `zapętl się`
  - **uruchom** $\varphi_n(n)$ `na co najwyżej` m `kroków`
  - **jeśli** $\varphi_n(n)$ `zatrzymał się po` m `lub mniej krokach: zapętl się`
  - **zwróć** 1
- $t \leftarrow$ `numer znalezionego powyżej programu`
- **znajdź** `numer takiego programu (nie uruchamiaj go!)`:
  - **wczytaj** m
  - **jeśli** $m = 13$: `zapętl się`
  - **zwróć** 1
- $u \leftarrow$ `numer znalezionego powyżej programu`
- **zwróć** $\varphi_T(\langle t, u\rangle)$

Powyższy program świadczyłby o tym, że zbiór $\widehat{K}$[22] jest r.e. ↯

- jeśli $n \in \widehat{K}$ to wtedy φ_t oblicza tę samą funkcję częściową co φ_u ($\varphi_n(n)$ nigdy się nie zatrzymuje, zatem nie istnieje m, by $\varphi_n(n)$ się zatrzymał po m krokach, zatem φ_t zapętla się tylko dla argumentu 13, tak jak φ_u), zatem para $\langle t, u\rangle \in T$, więc $\varphi_T(\langle t, u\rangle) = 1$, czyli $\gamma(n) = 1$,
- jeśli $n \notin \widehat{K}$ to φ_t i φ_u obliczają różne funkcje częściowe (istnieje takie $i \in \mathcal{N}$, że dla każdego $m > i$ $\varphi_t(m) = \perp$, natomiast φ_u zapętla się[23] tylko dla argumentu 13), zatem para $\langle t, u\rangle \notin T$, więc $\varphi_T(\langle t, u\rangle) \notin \mathcal{N}$, czyli $\gamma(n) \notin \mathcal{N}$.

□

[20] wiemy to z definicji redukcji f
[21] taki, że $\varphi_T(\langle n, m\rangle) = 1 \Leftrightarrow \langle n, m\rangle \in T$
[22] $\widehat{K}$ – dopełnienie zbioru K
[23] lub jest „niezdefiniowane”

b. Dopełnienie zbioru T, czyli zbiór $\widehat{T} = \{\langle n, m\rangle : \phi_m \text{ i } \phi_n \text{ to różne funkcje częściowe}\}$ nie jest r.e.

Dowód. (nie-wprost) Załóżmy, że $\widehat{T}$ jest zbiorem rekurencyjnie przeliczalnym. Istnieje zatem $\varphi_{\widehat{T}}$ – program semi-rozstrzygający go[24]. Rozważmy taki program γ:

- **wczytaj** n
- **znajdź** `numer takiego programu (nie uruchamiaj go!)`:
  - **wczytaj** m
  - **uruchom** $\varphi_n(n)$
  - **jeśli** $m = 13$: `zapętl się`
  - **zwróć** 1
- $t \leftarrow$ `numer znalezionego powyżej programu`
- **znajdź** `numer takiego programu (nie uruchamiaj go!)`:
  - **wczytaj** m
  - **jeśli** $m = 13$: `zapętl się`
  - **zwróć** 1
- $u \leftarrow$ `numer znalezionego powyżej programu`
- **zwróć** $\varphi_{\widehat{T}}(\langle t, u\rangle)$

Powyższy program świadczyłby o tym, że zbiór $\widehat{K}$ jest r.e. ↯

- jeśli $n \in \widehat{K}$ to φ_t i φ_u obliczają różne funkcje częściowe ($\varphi_t(m) = \perp$ dla każdego $m \in \mathcal{N}$, a φ_u zapętla się tylko dla argumentu 13), zatem para $\langle t, u\rangle \in \widehat{T}$, więc $\varphi_{\widehat{T}}(\langle t, u\rangle) \in \mathcal{N}$, czyli $\gamma(n) \in \mathcal{N}$,
- jeśli $n \notin \widehat{K}$ to wtedy φ_t oblicza tę samą funkcję częściową co φ_u ($\varphi_n(n)$ zatrzymuje się, zatem φ_t zapętla się tylko dla argumentu 13, tak jak φ_u), zatem para $\langle t, u\rangle \notin \widehat{T}$, więc $\varphi_{\widehat{T}}(\langle t, u\rangle) \notin \mathcal{N}$, czyli $\gamma(n) \notin \mathcal{N}$.

□

Zadanie 64. Skonstruuj maszynę Turinga rozpoznającą język $\{ww^R : w \in \{0,1\}^*\}$.

Rozwiązanie. Idea działania maszyny jest następująca:

(a) początkowo maszyna na taśmie ma wejściowe słowo poprzedzone symbolem α a zakończone symbolem ω,

(b) maszyna przesuwa głowicę na pierwszy symbol za symbolem α,

(c) zapamiętuje symbol który aktualnie „widzi" (przechodząc w odpowiedni stan), wypisuje symbol ω i przesuwa głowicę w prawo tak długo aż napotka symbol ω,

(d) po napotkaniu symbolu ω przesuwa głowicę w lewo i sprawdza, czy zapamiętany symbol zgadza się z tym który „widzi" aktualnie na taśmie,

(e) jeśli tak to wypisuje w to miejsce symbol ω i przesuwa głowicę w lewo aż do napotkania symbolu ω, a w przeciwnym przypadku przechodzi do stanu nieakceptującego i kończy działanie,

(f) przesuwa głowicę o jedno miejsce w prawo,

(g) jeśli „widzi" symbol różny od symbolu ω to przechodzi do pkt. (c), a w przeciwnym przypadku przechodzi do stanu akceptującego i kończy działanie.

[24] taki, że $\varphi_{\widehat{T}}(\langle n, m\rangle) = 1 \Leftrightarrow \langle n, m\rangle \in \widehat{T}$

Instrukcje tej maszyny pozwalające zrealizować powyższy opis:

- $\langle q_{start}, \alpha\rangle \to \langle q_{read}, \alpha, R\rangle$, realizacja (b),
- $\begin{cases} \langle q_{read}, 0\rangle \to \langle q_0, \omega, R\rangle \\ \langle q_{read}, 1\rangle \to \langle q_1, \omega, R\rangle \\ \langle q_{read}, \omega\rangle \to q_{accept} \end{cases}$, zapamiętanie widzianego symbolu i wypisanie symbolu ω, lub ew. akceptacja słowa,
- $\begin{cases} \langle q_0, 0\rangle \to \langle q_0, 0, R\rangle \\ \langle q_0, 1\rangle \to \langle q_0, 1, R\rangle \\ \langle q_1, 0\rangle \to \langle q_1, 0, R\rangle \\ \langle q_1, 1\rangle \to \langle q_1, 1, R\rangle \end{cases}$, przesuwanie głowicy w prawo,
- $\begin{cases} \langle q_0, \omega\rangle \to \langle q_{check0}, \omega, L\rangle \\ \langle q_1, \omega\rangle \to \langle q_{check1}, \omega, L\rangle \end{cases}$, przejście w stan sprawdzania zapamiętanego symbolu,
- $\begin{cases} \langle q_{check0}, 0\rangle \to \langle q_{back}, \omega, L\rangle \\ \langle q_{check0}, 1\rangle \to q_{failure} \\ \langle q_{check1}, 1\rangle \to \langle q_{back}, \omega, L\rangle \\ \langle q_{check1}, 0\rangle \to q_{failure} \end{cases}$, sprawdzenie symbolu i zastąpienie go (w razie powodzenia) symbolem ω,
- $\begin{cases} \langle q_{back}, 0\rangle \to \langle q_{back}, 0, L\rangle \\ \langle q_{back}, 1\rangle \to \langle q_{back}, 1, L\rangle \end{cases}$, przesuwanie głowicy w lewo,
- $\langle q_{back}, \omega\rangle \to \langle q_{read}, \omega, R\rangle$, realizacja (f),

Alfabetem opisanej maszyny jest oczywiście zbiór $A = \{0,\ 1,\ \alpha,\ \omega\}$[25],
zbiór stanów to $Q = \{q_{start},\ q_{read},\ q_0,\ q_1,\ q_{accept},\ q_{failure},\ q_{check0},\ q_{check1},\ q_{back}\}$, gdzie stanem akceptującym jest q_{accept}, a stanem nieakceptującym $q_{failure}$.

Zadanie 66. (za 2 punkty) Zauważ, że maszyna podobna do dwukierunkowego automatu ze stosem, lecz posiadająca dwa stosy, potrafi rozpoznać te same języki co maszyna Turinga. Udowodnij, że pozostaje to prawdą jeśli dysponujemy tylko jednym symbolem stosowym (oprócz symbolu dna stosu). Maszynę taką nazywamy *maszyną z dwoma licznikami. Wskazówka: najpierw udowodnij, że wystarczą cztery liczniki, potem spróbuj zakodować ich stan przy pomocy jednego licznika, drugiego będziesz mógł/mogła użyć do manipulowania pierwszym.*

Rozwiązanie[26]**.**

- Istotnie, łatwo zasymulować dwoma stosami taśmę maszyny Turinga. Wystarczy „przeciąć" ją w miejscu na które „patrzy" głowica. Wszystkie symbole z lewego kawałka „przeciętej" taśmy będą znajdowały się na „lewym stosie", a pozostałe symbole na „prawym stosie". Operacja przesunięcia głowicy w prawo (i odpowiednio w lewo) polega na przełożeniu symbolu z prawego stosu na lewy stos (i odpowiednio z lewego na prawy)[27].
- Pokażmy jak za pomocą dwóch liczników[28] symulować jeden stos. Musimy „zasymulować" operację umieszczania na stosie symboli 0 lub 1 oraz operację zdjęcia symboli 0 lub 1 ze szczytu stosu. Potraktujmy zawartość symulowanego stosu jako zapis pewnej liczby w systemie binarnym (ze szczytem stosu jako najmniej znaczącym bitem). Zatem:
  - Dostawienie zera jako najmniej znaczącego bitu do ciągu symboli reprezentującego pewną liczbę powoduje podwojenie wartości tej liczby, a dostawienie jedynki podwaja tą wartość plus jeden. Mając dwa liczniki (w tym jeden „pomocniczy") łatwo dokonać podwojenia zawartości

[25] a „B" – blank – nie był nigdzie potrzebny
[26] a raczej bardzo nieformalny jego szkic
[27] oczywiście należy jeszcze uwzględnić modyfikacje symboli przez głowicę
[28] czyli stosów z tylko jednym symbolem – oprócz dna stosu

jednego z nich: w każdym kroku zdejmując jeden symbol z licznika „głównego" wstawiamy dwa symbole na licznik „pomocniczy"[29]. Dostawienie jedynki jest oczywiste.

- Zdejmowanie symboli 0 lub 1 ze szczytu symulowanego stosu jest operacją dualną do opisanej w poprzednim podpunkcie.

Aby poradzić sobie z rozróżnianiem ilości zer na najbardziej znaczących pozycjach symulowanego stosu wystarczy zainicjować licznik jednym symbolem.

- Skoro mamy już sposób na symulację jednego stosu dwoma licznikami, to oczywiście potrafimy symulować dwa stosy czterema licznikami. Pokażemy teraz jak zakodować stan czterech liczników przy pomocy jednego. Drugi („pomocniczy") licznik będziemy wykorzystywać do manipulowania pierwszym. Zapis stanu czterech liczników (nazwijmy je A, B, C i D) będzie miał postać:

$$x = 2^a \cdot 3^b \cdot 5^c \cdot 7^d$$

gdzie a, b, c, d są odpowiednio ilościami symboli w licznikach A, B, C, D, a x jest ilością symboli w liczniku reprezentującą ten stan. Należy jeszcze opowiedzieć o tym jak inkrementować, bądź dekrementować stan któregoś z liczników A do D. Ach! Wystarczy pomnożyć (lub podzielić) przez odpowiednią liczbę (ze zbioru $\{2, 3, 5, 7\}$) ilość symboli w liczniku (używając licznika pomocniczego – zgodnie ze sposobem opisanym wcześniej).

Doprecyzowania wymaga (m.in.) zamiana maszyny Turinga na dwukierunkowy automat z dwoma stosami (alfabet taśmowy maszyny Turinga składa się z $\{\alpha, \omega, 0, 1, B, ...\}$ a w pokazanej konstrukcji operowaliśmy tylko symbolami 0 i 1).

Zadanie 67. Czy istnieje algorytm rozstrzygający dla danej gramatyki bezkontekstowej G, czy istnieje słowo w takie, że $ww^Rw \in L(G)$?

Rozwiązanie.

Nie istnieje algorytm rozstrzygający dla danej gramatyki bezkontekstowej G, czy istnieje słowo w, takie że $ww^Rw \in L(G)$.

Dowód. Niech Z będzie zbiorem gramatyk bezkontekstowych G, takich że istnieje[30] słowo w, że $ww^Rw \in L(G)$. Pokażemy, że $PCP \leq_{rek} Z$ i w tym celu skonstruujemy[31] odpowiednią redukcję f. Niechaj więc f będzie taka, że bierze instancę problemu PCP, czyli zbiór par $P = \{\langle l_1, r_1\rangle, ..., \langle l_s, r_s\rangle\}$ i zwraca gramatykę $G_P = \langle \Sigma_P, V_P, S_P, \Pi_P\rangle$[32] zdefiniowaną w następujący sposób:

- $\Sigma_P = A \cup \{z_1, ..., z_s\} \cup \{\#\}$, gdzie A jest alfabetem słów l_i, r_i dla $i \in \{1, ..., s\}$ instancji P problemu PCP, a $\{z_1, ..., z_s\}$ i $\#$ są „świeżymi" (nie zawierającymi się w A) symbolami terminalnymi,
- $V_P = \{L, R, S_P\}$,
- $\Pi_P = \left\{ \begin{array}{rcl} S_P & \rightarrow & \#\,L\,\#\,\#\,R\,\#\,\#\,L\,\#\,, \\ L & \rightarrow & l_i\,L\,z_i \mid l_i\,z_i\,, \\ R & \rightarrow & z_i\,R\,{r_i}^R \mid z_i\,{r_i}^R\,, \end{array} \right\}$, dla każdego $i \in \{1, ..., s\}$

Redukcja f spełnia następującą równoważność:

$$P \text{ ma rozwiązanie w postaci słowa } m \in \{1, ..., s\}^* \iff \text{istnieje } w \text{ takie, że } ww^Rw \in L(f(P)).$$

[29] wstawienie 2 symboli na licznik pomocniczy musi odbyć się przy użyciu jakiegoś stanu pomocniczego, ale łatwo to sobie wyobrazić

[30] oczywiście dla każdej gramatyki G może to być inne słowo

[31] podamy jej słowną specyfikację

[32] Σ_P – zbiór terminali, „alfabet", V_P – zbiór nieterminali, S_P – symbol startowy, Π_P – zbiór produkcji

($\Rightarrow$) Istnieje $m \in \{1, ..., s\}^*$, że $l_{m_1} l_{m_2} ... l_{m_k} = r_{m_1} r_{m_2} ... r_{m_k}$. Wtedy z gramatyki $f(P) = G_P$ da się wygenerować słowo:

$$(\clubsuit) \qquad \# l_{m_1} ... l_{m_k} z_{m_k} ... z_{m_1} \# \# z_{m_1} ... z_{m_k} {r_{m_k}}^R ... {r_{m_1}}^R \# \# l_{m_1} ... l_{m_k} z_{m_k} ... z_{m_1} \#,$$

które jest postaci ww^Rw,

($\Leftarrow$) Jeśli z gramatyki $G_P = f(P)$ można wygenerować słowo postaci ww^Rw, to może ono jedynie mieć postać ($\clubsuit$). Istnieje zatem rozwiązanie instancji P problemu PCP w postaci słowa $m \in \{1, ..., s\}^*$, które można łatwo odkodować (lub wręcz odczytać wprost) ze słowa $z_{m_1} ... z_{m_k}$.

□

Zadanie 68. Powtórz podany na wykładzie dowód nierozstrzygalności Problemu Odpowiedniości Posta.

Rozwiązanie.

- Instancją Problemu Odpowiedniości Posta jest skończony zbiór par słów $\Pi = \{\langle w_1, v_1\rangle, ..., \langle w_l, v_l\rangle\}$. Rozwiązaniem tej instancji problemu (o ile takie rozwiązanie istnieje) jest słowo $s \in \{1, ..., l\}$ takie, że $s \neq \epsilon$, $s = s_1 ... s_k$ i $w_{s_1} w_{s_2} ... w_{s_k} = v_{s_1} v_{s_2} ... v_{s_k}$.
- Instancją problemu słów dla semi-procesów Thuego jest trójka $\langle w, v, \Pi\rangle$. Rozwiązaniem jest odpowiedź na pytanie, czy $w \xrightarrow{\Pi*} v$ (to znaczy, czy używając produkcji ze zbioru Π można wyprodukować słowo v ze słowa w; $w \xrightarrow{\Pi} v$ zachodzi, gdy $\exists s_1, s_2\ \exists \langle w_i, v_i\rangle \in \Pi\ w = s_1 w_i s_2 \wedge v = s_1 v_i s_2$, natomiast $\xrightarrow{\Pi*}$ jest przechodnim domknięciem $\xrightarrow{\Pi}$). Zbiór SemiThue $= \{\langle w, v, \Pi\rangle : w \xrightarrow{\Pi*} v\}$ nie jest rozstrzygalny (co można udowodnić przez pokazanie, że $K \leq_{rek}$ SemiThue).

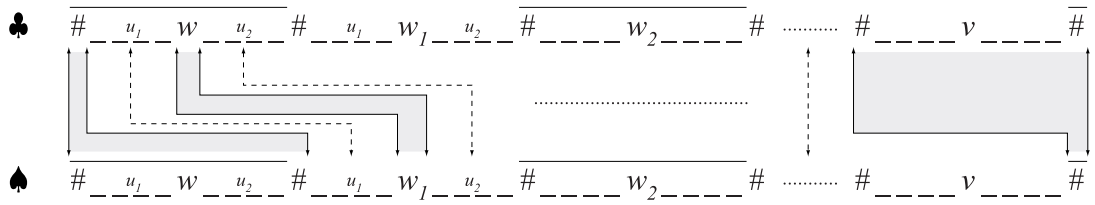
$$PCP = \{\{\langle w_1, v_1\rangle, ..., \langle w_l, v_l\rangle\} : \exists s \in \{1, ..., l\}\ s \neq \epsilon \wedge s = s_1 ... s_k \wedge w_{s_1} w_{s_2} ... w_{s_k} = v_{s_1} v_{s_2} ... v_{s_k}\}$$
nie jest rozstrzygalny.

Dowód. Pokażemy, że SemiThue $\leq_{rek} PCP$. Zbudujemy zatem całkowitą, obliczalną funkcję (redukcję) f, która jako argument weźmie krotkę $\langle w, v, \Pi\rangle$ i w wyniku zwróci instancję PCP Π'. Niech Σ oznacza alfabet Π. Zdefiniujemy $\overline{\Sigma} = \{\overline{a} : a \in \Sigma\}$, natomiast operacja „kreska" ma następujące właściwości:

$$\overline{\overline{a}} = a, \qquad \overline{\epsilon} = \epsilon, \qquad w \in (\Sigma \cup \overline{\Sigma})^*,\ a \in (\Sigma \cup \overline{\Sigma})\ \Rightarrow\ \overline{aw} = \overline{a}\,\overline{w}.$$

Dla danej trójki $\langle w, v, \Pi\rangle$ zbiór Π' będzie zbudowany nad alfabetem $\Sigma_{\Pi'} = \Sigma \cup \overline{\Sigma} \cup \{\#, \overline{\#}\}$ i będzie zawierał następujące produkcje (pary):

1. $\langle \overline{\#},\ \overline{\# w \#}\rangle$,
2. $\langle \# v \overline{\#},\ \overline{\#}, \rangle$,
3. $\langle a,\ \overline{a}\rangle$, gdzie $a \in \Sigma \cup \overline{\Sigma}$,
4. $\langle \#,\ \overline{\#}\rangle$,
5. $\langle \overline{\#},\ \#\rangle$,
6. $\{\langle s, \overline{t}\rangle,\ \langle \overline{s}, t\rangle : \langle s, t\rangle \in \Pi\}$,



Intuicja: ♣ – słowo powstałe z „lewych" słów Π', ♠ – słowo powstałe z „prawych" słów Π',

Powyższa redukcja f spełnia równoważność: $w \overset{\Pi*}{\to} v \iff f(\langle w, v, \Pi\rangle) \in PCP$.

($\Rightarrow$) Skoro istnieje wyprowadzenie $w \overset{\Pi*}{\to} v$ to można je przedstawić w postaci ♣ (przy użyciu „świeżego" symbolu # oraz używając operacji „kreska" na przemian – tak jak na rys. powyżej):

- na początku zostanie użyta para z pkt. 1. jako jedyna pasująca (jest to jedyna para, w której obu słowach pierwszy symbol jest taki sam – $\overline{\#}$),
- aby przeprowadzić słowa u_1 i u_2 na $\overline{u_1}$ i $\overline{u_2}$ należy użyć odpowiednich par z pkt. 3.,
- skoro $u_1\, w\, u_2 \overset{\Pi}{\to} u_1\, w_1\, u_2$, to w Π istniała odpowiednia produkcja $\langle w, w_1\rangle$, a zatem do Π' należą pary (zgodnie z pkt. 6.) $\langle w, \overline{w_1}\rangle$ i $\langle \overline{w}, w_1\rangle$,
- do przeprowadzania # na $\overline{\#}$ i vice-versa służą odpowiednie pary z pkt. 4. i 5.,
- aby zakończyć „układankę" i otrzymać dwa identyczne słowa ♣ i ♠ należy użyć pary z pkt. 2. (jako jedynej, w której obu słowach ostatni symbol jest taki sam – $\overline{\#}$).

($\Leftarrow$) Jeśli $f(\langle w, v, \Pi\rangle) = \Pi' \in PCP$ to znaczy, że istnieje słowo $s \neq \epsilon$, $s = s_1 \ldots s_k$ takie, że $w_{s_1} w_{s_2} \ldots w_{s_k} = v_{s_1} v_{s_2} \ldots v_{s_k}$[33]. Aby słowo $w_{s_1} w_{s_2} \ldots w_{s_k}$ było równe słowu $v_{s_1} v_{s_2} \ldots v_{s_k}$ para $\langle w_{s_1}, v_{s_1}\rangle \in \Pi'$ musi być postaci z pkt. 1., natomiast para $\langle w_{s_k}, v_{s_k}\rangle \in \Pi'$ musi być postaci z pkt. 2. Ten warunek powoduje, że koniecznym jest (jeśli słowa ♣ i ♠ są sobie równe) by wśród par $\langle w_{s_2}, v_{s_2}\rangle \ldots \langle w_{s_{(k-1)}}, v_{s_{(k-1)}}\rangle$ znajdowały się pary postaci z pkt. 6. Ciąg par tej postaci jest dowodem na to, że $w \overset{\Pi*}{\to} v$.

□

Zadanie 69. Dla gramatyki bezkontekstowej G niech L_G oznacza generowany przez nią język. Skorzystaj z nierozstrzygalności problemu odpowiedniości Posta aby pokazać, że zbiór tych gramatyk G, H dla których zachodzi $L_G \cap L_H \neq \phi$ nie jest rekurencyjny. Czy jest on rekurencyjnie przeliczalny?

Rozwiązanie.

Zbiór $PCFL = \{\langle G, H\rangle \in CFG \times CFG : L_G \cap L_H \neq \phi\}$ nie jest rekurencyjny.

Dowód. Pokażemy, że $PCP \leq_{rek} PCFL$. Zbudujemy zatem redukcję f, która jako argument weźmie instancję problemu odpowiedniości Posta $\Pi = \{\langle w_1, v_1\rangle, \ldots, \langle w_l, v_l\rangle\}$, a w wyniku zwróci dwie gramatyki G_Π i H_Π:

- Niech Σ oznacza alfabet Π. Zdefiniujemy alfabet gramatyk G_Π i H_Π jako $\Sigma_\Pi = \Sigma \cup \{i_1, \ldots, i_l\}$,
- $G_\Pi = \langle \Sigma_\Pi, V_G, S_G, P_G\rangle$, gdzie
  - $V_G = \{S_G\}$ – zbiór nieterminali, S_G – symbol startowy,
  - $P_G = S_G \to \epsilon \mid w_j S_G i_j$, dla $j = 1, \ldots, l$ – zbiór produkcji,
- $H_\Pi = \langle \Sigma_\Pi, V_H, S_H, P_H\rangle$, gdzie
  - $V_H = \{S_H\}$ – zbiór nieterminali, S_H – symbol startowy,
  - $P_H = S_H \to \epsilon \mid v_j S_H i_j$, dla $j = 1, \ldots, l$ – zbiór produkcji,

[33] nastąpiła tu pewna kolizja nazw – należy pamiętać, że w_{s_i} oraz v_{s_i} są innymi słowami niż w czy v branymi jako argumenty redukcji f

Podana redukcja f spełnia równoważność: $\Pi \in PCP \iff f(\Pi) = \langle G_\Pi, H_\Pi \rangle \in PCFL$.

($\Rightarrow$) Jeśli $\Pi \in PCP$ to znaczy, że istnieje słowo $s \neq \epsilon$, $s = s_1 ... s_k$ takie, że $w_{s_1} w_{s_2} ... w_{s_k} = v_{s_1} v_{s_2} ... v_{s_k}$, z czego z kolei wynika, że słowa $w_{s_1} w_{s_2} ... w_{s_k} i_{s_k} ... i_{s_2} i_{s_1} \in L_{G_\Pi}$ oraz $v_{s_1} v_{s_2} ... v_{s_k} i_{s_k} ... i_{s_2} i_{s_1} \in L_{H_\Pi}$ są sobie równe, a zatem $L_{G_\Pi} \cap L_{H_\Pi} \neq \phi$, czyli $\langle G_\Pi, H_\Pi \rangle \in PCFL$,

($\Leftarrow$) Skoro $\langle G_\Pi, H_\Pi \rangle \in PCFL$ to znaczy, że $L_{G_\Pi} \cap L_{H_\Pi} \neq \phi$ a zatem istnieje słowo $u \in L_{G_\Pi} \wedge u \in L_{H_\Pi}$. Ponieważ umówiliśmy się[34], że nie zwracamy uwagi na słowo puste, to słowo u musi mieć postać $w_{s_1} w_{s_2} ... w_{s_k} i_{s_k} ... i_{s_2} i_{s_1} \in L_{G_\Pi}$ oraz $v_{s_1} v_{s_2} ... v_{s_k} i_{s_k} ... i_{s_2} i_{s_1} \in L_{H_\Pi}$. Łatwo z niego odczytać słowo $s = s_1 ... s_k$ będące rozwiązaniem instancji Π problemu PCP.

□

Zbiór $PCFL = \{\langle G, H \rangle \in CFG \times CFG : L_G \cap L_H \neq \phi\}$ jest rekurencyjnie przeliczalny.

Dowód. Łatwo napisać algorytm γ sprawdzający (czy też „semi-rozstrzygający") dla danych gramatyk G i H, czy $L_G \cap L_H \neq \phi$ (to znaczy taki, że $\gamma(\langle G, H \rangle) = 1 \Leftrightarrow L_G \cap L_H \neq \phi$). Taki algorytm polegałby na przeszukiwaniu wszerz grafów wyprowadzeń słów z symboli startowych przy użyciu produkcji z – odpowiednio – gramatyk G i H. Oczywiście taki systematyczny sposób sprawdzania gwarantuje odnalezienie wspólnego dla obu języków L_G i L_H słowa u o ile takie słowo istnieje. □

Zadanie 72. Jak każdy pamięta, deterministyczny automat ze stosem, to urządzenie zadane przez skończony zbiór instrukcji w formacie: *jeśli widzisz na taśmie wejściowej a, jesteś w stanie q, a z czubka stosu zdjąłeś b, to przejdź do stanu q', a na czubek stosu włóż słowo w.* Taki automat czyta słowo wejściowe literka po literce, zmieniając przy tym stan jak zwykły automat skończony, a do tego jeszcze buduje sobie stos. Czy istnieje algorytm odpowiadający, dla danych dwóch deterministycznych automatów ze stosem, czy istnieje niepuste słowo wejściowe, po przeczytaniu którego oba te automaty będą miały na swoich stosach takie same ciągi symboli?

Rozwiązanie.

Nie istnieje algorytm taki, jak opisany powyżej, to znaczy zbiór
$EqStack = \{\langle D_1, D_2 \rangle :$ istnieje w że po jego przeczytaniu na stosach D_1 i D_2 są te same słowa$\}$
nie jest rekurencyjny[35].

Dowód. Pokażemy, że $PCP \leq_{rek} EqStack$. Niech redukcja f jako argument bierze instancję problemu PCP, czyli zbiór $\Pi = \{\langle l_1, r_1 \rangle, ..., \langle l_k, r_k \rangle\}$ a w wyniku zwraca dwa deterministyczne automaty ze stosem[36] D_1 i D_2. Załóżmy, że l_i, r_i są słowami nad pewnym alfabetem $\mathcal{A}$. Niech:

- $D_1 = \langle Q, \Sigma, \Gamma, q, Z, \delta_1, F \rangle \qquad D_2 = \langle Q, \Sigma, \Gamma, q, Z, \delta_2, F \rangle$,
- gdzie:
  - $Q = \{q\}$ – zbiór stanów, q – stan początkowy, $F = \{q\}$ – zbiór stanów akceptujących,
  - $\Sigma = \{s_1, ..., s_k\}$ – zbiór symboli taśmowych,
  - $\Gamma = \mathcal{A} \cup \{Z\}$ – zbiór symboli stosowych, Z – symbol dna stosu,
  - $\delta_1(q, s_i, b) = (q, l_{s_i}\, b)$, $\delta_2(q, s_i, b) = (q, r_{s_i}\, b)$ dla wszystkich $i \in \{1, ..., k\}$ oraz $b \in \Gamma$ – funkcje przejścia automatów D_1 i D_2,

[34] na wykładzie

[35] D_1 i D_2 to deterministyczne automaty ze stosem

[36] intuicja jest taka, że te automaty będą czytały słowo s będące potencjalnym rozwiązaniem instancji Π problemu PCP odkładając przy tym na stosie słowa skonstruowane z – odpowiednio – „lewych słów" i „prawych słów" par produkcji z Π

Powyższa redukcja f spełnia równoważność: $\Pi \in PCP \iff f(\Pi) = \langle D_1, D_2 \rangle \in EqStack$.

($\Rightarrow$) Jeśli $\Pi \in PCP$ to znaczy, że istnieje słowo $s \neq \epsilon$, $s = s_1 \dots s_k$ takie, że $l_{s_1} l_{s_2} \dots l_{s_k} = r_{s_1} r_{s_2} \dots r_{s_k}$. Ale wtedy $\widehat{\delta_1}(q, s, Z) = (q, l_{s_k} \dots l_{s_2} l_{s_1} Z)$ i $\widehat{\delta_2}(q, s, Z) = (q, r_{s_k} \dots r_{s_2} r_{s_1} Z)$ czyli istnieje słowo (s) po przeczytaniu którego zawartość stosów automatów D_1 i D_2 jest taka sama, a zatem $\langle D_1, D_2 \rangle \in EqStack$,

($\Leftarrow$) Skoro $f(\Pi) = \langle D_1, D_2 \rangle \in EqStack$ to znaczy, że istnieje słowo $(s = s_1 \dots s_k)$ po przeczytaniu którego zawartość stosów automatów D_1 i D_2 jest taka sama, a zatem $\widehat{\delta_1}(q, s, Z) = (q, l_{s_k} \dots l_{s_2} l_{s_1} Z)$ i $\widehat{\delta_2}(q, s, Z) = (q, r_{s_k} \dots r_{s_2} r_{s_1} Z)$ z czego oczywiście wynika równość $l_{s_1} l_{s_2} \dots l_{s_k} = r_{s_1} r_{s_2} \dots r_{s_k}$, a zatem s jest rozwiązaniem instancji Π problemu PCP, więc $\Pi \in PCP$.

□

Zadanie 74. Powiemy, że semiproces Thuego Π jest bezkontekstowy, jeśli dla każdej pary $[w, v] \in \Pi$ słowo w składa się z jednej litery a słowo v jest niepuste. Czy problem słów dla bezkontekstowych semiprocesów Thuego jest rozstrzygalny?

Rozwiązanie.

Problem słów dla bezkontekstowych semiprocesów Thuego **jest** rozstrzygalny.

Dowód. Podamy algorytm sprawdzający, czy dla danych słów w i v oraz bezkontekstowego semiprocesu Thuego Π zachodzi $w \xrightarrow{\Pi*} v$. Będziemy przeszukiwać[37] graf wszystkich możliwych wyprowadzeń słowa w w którym wierzchołkami będą pewne słowa u. Powiemy, że n jest sąsiadem u jeśli $u \xrightarrow{\Pi} n$. Ponieważ dla rozpatrywanych semiprocesów nie da się wyprowadzić z dowolnego słowa u słowa od niego krótszego, to algorytm nie będzie przeszukiwał wierzchołków o kluczach dłuższych od v.

```
1  – read w, v, Π
2  – Q ← [w]
3  – Visited ← ϕ
4  – while Q ≠ []:
5      – u ← Q.get()
6      – if |u| ≤ |v| ∧ u ∉ Visited:
7          – for each n in neighbours(u, Π):
8              – if n ∉ Q:
9                  – Q.put(n)
10                 – if n = v:
11                     – return YES
12     – Visited ← Visited ∪ {u}
13 – return NO
```

Zmienna Q wprowadzona w 2 linii jest kolejką ze zdefiniowanymi operacjami pobierania elementu z początku – `get()`[38], wstawiania na koniec – `put()` oraz sprawdzania przynależności – „$\in$” i „$\notin$”. Zmienna $Visited$ jest zbiorem, natomiast w wyniku działania funkcji `neighbours`(u, Π) otrzymujemy zbiór wszystkich sąsiadów (w sensie „$\xrightarrow{\Pi}$”) słowa u przy użyciu produkcji ze zbioru Π[39]. □

[37] wszerz

[38] oczywiście próba pobrania elementu z pustej kolejki kończy się jakąś niesamowitą katastrofą...

[39] a konstrukcja **for each** pozwala na przejrzenie wszystkich – w dowolnej kolejności – elementów zbioru

Zadanie 75. Powiemy, że semiproces Thuego Π jest prawie bezkontekstowy, jeśli dla każdej pary $[w, v] \in \Pi$ jedno ze słów w i v składa się tylko z jednej litery, drugie zaś z dwóch liter. Czy problem słów dla prawie bezkontekstowych semiprocesów Thuego jest rozstrzygalny? *Uwaga. Użyta w tym i poprzednim zadaniu nomenklatura wymyślona została tylko by wygodniej było sformułować te zadania i nie ma nic wspólnego z żadnym standardem.*

Rozwiązanie.

Problem słów dla prawie bezkontekstowych semiprocesów Thuego **nie jest** rozstrzygalny, tzn.
zbiór $\textsc{AlmostThue} = \{\langle w, v, \Pi\rangle : w \overset{\Pi*}{\to} v\}$ nie jest rekurencyjny.

Dowód. Pokażemy, że $K \leq_{rek} \textsc{AlmostThue}$, tzn. pokażemy[40] całkowitą, rekurencyjną funkcję[41] f, której argumentami będą liczby naturalne k a wartościami trójki $\langle w(k), v(k), \Pi(k)\rangle$ o takiej własności, że $w(k) \overset{\Pi(k)*}{\longrightarrow} v(k) \Leftrightarrow M_k(k)$ się zatrzyma. Bez straty ogólności możemy przyjąć, że każda maszyna Turinga ma jeden akceptujący stan końcowy q_f. Niech $\delta_k : (Q_k \times \Sigma_k) \to (Q_k \times \Sigma_k \times \{L, R\})$ będzie funkcją przejścia maszyny M_k, Q_k zbiorem jej stanów, a Σ_k jej alfabetem taśmowym. Dla danej liczby $k \in \mathcal{N}$ mamy:

- alfabetem konstruowanego prawie bezkontekstowego semiprocesu Thuego $\Pi(k)$ będzie $\Sigma_{\Pi(k)} = \Sigma_k \cup Q_k$. Dodatkowo w $\Sigma_{\Pi(k)}$:
  - dla każdej pary $\langle q_i, a_j\rangle \in Q_k \times \Sigma_k \wedge i, j \in \mathcal{N}$ są unikatowe symbole $S_{i,j}$,
  - dla każdej pary $\langle a_i, q_j\rangle \in \Sigma_k \times Q_k \wedge i, j \in \mathcal{N}$ są unikatowe symbole $T_{i,j}$,
  - dla każdej trójki $\langle a_h, q_i, a_j\rangle \in \Sigma_k \times Q_k \times \Sigma_k \wedge h, i, j \in \mathcal{N}$ są unikatowe symbole $U_{i,h,j}$,
  - dla każdej pary $\langle a_i, a_j\rangle \in \Sigma_k \times \Sigma_k \wedge i, j \in \mathcal{N}$ są unikatowe symbole $W_{i,j}$,
- słowo $w(k) = q_0\, \alpha\, \underline{\text{reprezentacja binarna } k} \omega\, B$[42] i konwencja jest taka, że słowo $q_x\, a$ oznacza, że maszyna znajduje się w stanie q_x a jej głowica znajduje się nad symbolem a,
- produkcje $\Pi(k)$ konstruuje się w następujący sposób:
  - jeśli w zbiorze instrukcji maszyny[43] znajduje się para $\langle\, \langle q_i, a_h\rangle\,,\ \langle q_j, a_l, R\rangle\, \rangle$ to do konstruowanego zbioru produkcji $\Pi(k)$ dodajemy produkcje:
    $$q_i\, a_h \to S_{i,h} \quad \text{oraz} \quad S_{i,h} \to a_l\, q_j$$
  - jeśli w zbiorze instrukcji maszyny znajduje się para $\langle\, \langle q_i, a_h\rangle\,,\ \langle q_j, a_l, L\rangle\, \rangle$ to dla każdego symbolu $a_g \in \Sigma_k$ dodajemy produkcje:
    $$a_g\, q_i \to T_{g,i} \qquad T_{g,i}\, a_h \to U_{j,g,l} \qquad U_{j,g,l} \to q_j\, W_{g,l} \qquad W_{g,l} \to a_g\, a_l$$
  - umieszczamy ponadto produkcję $B \to B\, B$, oraz dla każdego $a \in \Sigma_k$ produkcje $q_f\, a \to q_f$ oraz $a\, q_f \to q_f$.
- oczywiście słowo $v(k)$ składa się z jednego symbolu: $v(k) = q_f$.

Dowód, że powyższa algorytm spełnia równoważność „$w(k) \overset{\Pi(k)*}{\longrightarrow} v(k) \Leftrightarrow M_k(k)$ się zatrzyma" polega na głębokiej kontemplacji jego konstrukcji, zagłębienia się w detale i rozpatrzenia pewnych przypadków lub indukcyjnym pokazaniu[44] dwóch implikacji: „$w(k) \overset{\Pi(k)*}{\longrightarrow} v(k) \Leftarrow M_k(k)$ się zatrzyma" oraz „$w(k) \overset{\Pi(k)*}{\longrightarrow} v(k) \Rightarrow M_k(k)$ się zatrzyma". □

[40] w postaci opisu algorytmu
[41] „redukcję obliczalną"
[42] q_0 oznacza początkowy stan maszyny
[43] Na funkcję δ można patrzeć jak na zbiór par $\langle x, y\rangle$ o takiej własności, że jeśli do tego zbioru należą pary $\langle x, v\rangle$ oraz $\langle x, w\rangle$, to $v = w$. Taką parę – w przeprowadzanym rozumowaniu – nazywamy instrukcją maszyny.
[44] co jest nudne jak flaki z olejem

Zadanie 77. (za 2 punkty) Rozpatrzmy skończony zbiór par słów P i binarną relację $\to$ na słowach zdefiniowaną jak następuje: $w \to_P v$ wtedy i tylko wtedy gdy istnieje para $\langle a, b\rangle \in P$ taka że $w = ax$ i $v = xb$ gdzie x jest pewnym słowem. Niech $\to_P^*$ będzie przechodnim domknięciem $\to_P$ (to znaczy najmniejszą relacją przechodnią zawierającą $\to_P$). Czy problem: *dane P, x, y, czy* $x \to_P^* y$ *?* jest rozstrzygalny?

Rozwiązanie.

Problem opisany w zadaniu nie jest rozstrzygalny, to znaczy
zbiór ROBACZEK $= \{\langle P, x, y\rangle : x \to_P^* y\}$ nie jest rekurencyjny.

Dowód. Pokażemy, że $K \leq_{rek}$ ROBACZEK, to znaczy skonstruujemy redukcję f, która jako argument weźmie liczbę naturalną k a w wyniku zwróci instancję problemu ROBACZEK w postaci trójki $\langle P(k), w(k), v(k)\rangle$[45]:

- dla danej liczby k należy odnaleźć[46] maszynę Turinga M_k,
- niech $\delta_k : (Q_k \times \Sigma_k) \to (Q_k \times \Sigma_k \times \{L, R\})$ będzie funkcją przejścia maszyny M_k, Q_k zbiorem jej stanów, a Σ_k jej alfabetem taśmowym, oraz – dodatkowo – niech F_k będzie zbiorem stanów akceptujących M_k,
- alfabetem słów w zbiorze par $P(k)$ konstruowanej instancji problemu ROBACZEK będzie $\Sigma_{P(k)} = \Sigma_k \cup Q_k \cup \left\{\overline{B}, \text{✂}\right\}$,
- (♣) słowo $w(k) = q_0\, \alpha\, \underline{\textit{reprezentacja binarna } k} \omega\, \overline{B}$[47] i konwencja jest taka, że słowo $q_x\, a$ oznacza, że maszyna znajduje się w stanie q_x a jej głowica znajduje się nad symbolem a,
- produkcje $P(k)$ konstruuje się w następujący sposób:
  - (♡) dla każdego symbolu $a \in \Sigma_P(k)$ dodajemy produkcję $\langle a, a\rangle$[48],
  - jeśli w zbiorze instrukcji maszyny[49] M_k znajduje się para $\langle\, \langle q, a\rangle\,,\ \langle q', a', R\rangle\, \rangle$ to do konstruowanego zbioru produkcji $P(k)$ dodajemy produkcję $\langle q\, a,\ a'\, q'\rangle$,
  - jeśli w zbiorze instrukcji maszyny M_k znajduje się para $\langle\, \langle q, a\rangle\,,\ \langle q', a', L\rangle\, \rangle$ to dla każdego symbolu $x \in \Sigma_k$ dodajemy produkcję $\langle x\, q\, a,\ q'\, x\, a'\rangle$,
  - (♠) dla każdego symbolu $q \in Q_k \backslash F_k$ dodajemy produkcję $\left\langle q\, \overline{B},\ q\, B\, \overline{B}\right\rangle$,
  - (♢) dla każdego symbolu $q \in F_k$ dodajemy produkcję $\left\langle q, \text{✂}\right\rangle$,
  - (♢) dla każdego symbolu $a \in \Sigma_P(k)$ dodajemy produkcję $\left\langle a\text{✂}, \text{✂}\right\rangle$,
- oczywiście słowo $v(k)$ składa się z jednego symbolu: $v(k) =$ ✂.

[45] tak – ten dowód jest mocno inspirowany dowodem nierozstrzygalności zbioru *SemiThue*

[46] w zbiorze wszystkich maszyn

[47] $q_0 \in Q_k$ oznacza początkowy stan maszyny

[48] i nazwiemy ją sobie „przerzucacz"

[49] Na funkcję δ_k można patrzeć jak na zbiór par $\langle x, y\rangle$ o takiej własności, że jeśli do tego zbioru należą pary $\langle x, v\rangle$ oraz $\langle x, w\rangle$, to $v = w$. Taką parę – w przeprowadzanym rozumowaniu – nazywamy instrukcją maszyny.

Tak skonstruowana redukcja f spełnia równoważność: $k \in K \iff w(k) \rightarrow^*_{P(k)} v(k)$.

($\Rightarrow$) Jeśli $k \in K$ to $M_k(k) \in \mathcal{N}$, czyli istnieje taki ciąg konfiguracji maszyny Turinga, że $M_k(k)$ na końcu swojego działania znajdzie się w jakimś stanie akceptującym ($q \in F_k$). Taki ciąg konfiguracji można przedstawić w postaci ciągu słów skonstruowanych zgodnie z konwencją opisaną w (♣). Początkowa konfiguracja maszyny to $w(k)$. Oczywiście taśma maszyny Turinga jest potencjalnie nieskończona „w prawo", a zatem do zapewnienia tego warunku mamy produkcje opisane w (♠). Kiedy maszyna Turinga znajdzie się w stanie akceptującym, to produkcje opisane w (♢) pozwolą jej finalną konfigurację przeprowadzić na słowo $v(k)$. Oczywiście tak zapisany ciąg konfiguracji nie jest jeszcze prawidłowym ciągiem przeprowadzeń słów w sensie relacji $\rightarrow_P$ – jak wynika z definicji tej relacji (w treści zadania) użycie produkcji $\langle a, b\rangle \in P$ do przeprowadzenia słowa w na w' jest możliwe tylko wtedy, gdy $w = ax$, a po takim przeprowadzeniu słowo $w' = xb$. Aby zatem umożliwić „legalne" przeprowadzanie jednej konfiguracji maszyny Turinga w drugą (czyli, aby dowolną konfigurację zapisaną zgodnie z konwencją (♣) móc zapisać w postaci $w = ax$, gdzie a jest słowem które „nas interesuje"[50]) mamy produkcję opisaną w (♡), czyli „przerzucacza". Zatem z faktu, że $k \in K$ udało się wywnioskować (na podstawie analizy konstrukcji redukcji f), że trójka $f(k) = \langle P(k), w(k), v(k)\rangle \in$ Robaczek.

($\Leftarrow$) Skoro $w(k) \rightarrow^*_{P(k)} v(k)$ to istnieje „legalny" ciąg[51] słów $w(k) \rightarrow_{P(k)} w_2 \rightarrow_{P(k)} w_3 \rightarrow_{P(k)} \ldots \rightarrow_{P(k)} w_{m-1} \rightarrow_{P(k)} v(k)$. Słowo $w(k)$ – zgodnie z podaną konstrukcją redukcji f – jest początkową konfiguracją maszyny Turinga M_k zapisaną zgodnie z konwencją (♣). Pomijając słowa w_i które powstały przez zastosowanie produkcji opisanych w (♡) oraz (♢) (i – być może – w (♠), gdyż mogło się okazać, że ktoś „złośliwie" wygenerował wiele niepotrzebnych nikomu do niczego „Blank'ów") otrzymamy poprawny ciąg konfiguracji maszyny Turinga M_k, że $M_k(k)$ na końcu swojego działania znajduje się w jakimś stanie akceptującym ($q \in F_k$). Zatem z faktu, że $w(k) \rightarrow^*_{P(k)} v(k)$ udało się wywnioskować, że $k \in K$.

□

[50] czyli jest takiej postaci, by móc zastosować którąś z opisanych produkcji
[51] w sensie relacji $\rightarrow_P$