Toruń 2003

Barbara Klunder

Podstawy Teorii Obliczalności

Materiały do wykładu

W wykładze zastanawiać będziemy się nad dwoma kluczowymi pytaniami:

Czym jest efektywna procedura? Hardwarem czy softwarem? Urządzeniem, które realizuje określone obliczenia (pierwsze komputery), czy ciągiem instrukcji (programem) w naszym ulubionym języku programowania?

Czym jest problem rozwiązywalny przez efektywną procedurę? Czy istnieją problemy nierozwiązywalne?

Intuicyjnie czujemy, że będziemy wykonywać **obliczenia**.

Zasadniczo prowadzić będziemy obliczenia na (skończonych ciągach liczb) liczbach naturalnych.

Przypuśćmy, że chcemy prowadzić obliczenia na zbiorze wejść i wyjść X różnym od N. Możemy to zrobić przyporządkowując każdemu elementowi z X liczbę naturalną zwaną jej kodem tak, aby różne elementy X miały różne kody.
Mając wejścia z X należy zastąpić je ich kodami i wykonać obliczenia, które zwracają numeryczne wyjście. Jeśli to wyjście jest kodem elementu z X, to ten element traktujemy jako końcowy wynik obliczeń.
Chcemy, by pierwszy i ostatni krok tej procedury był realizowany w sposób algorytmiczny. Wtedy mówimy, że kodowanie jest efektywne.
U nas X będzie: zbiorem wszystkich skończonych ciągów liczb naturalnych lub słów nad ustalonym (skończonym) alfabetem, zbiorem instrukcji lub programów pewnego języka itp.

1 Kodowania i numeracje

Przypomnijmy dwie definicje zbioru przeliczalnego.

Definicja 1.1 *Kodowaniem zbioru X nazywamy dowolną funkcję różnowartościową $f : X \longrightarrow N$.*

Zbiór jest przeliczalny, jeśli jest równoliczny z pewnym podzbiorem N.

Definicja 1.2 *Numeracją (lub wyliczeniem) zbioru X nazywamy dowolną funkcję $g : N \longrightarrow X$, która jest "na".*

Zbiór jest przeliczalny, jeśli jego elementy można ustawić w ciąg.

Przykłady

1. Funkcja $\pi : N \times N \longrightarrow N$ określona wzorem $\pi(n.m) = 2^n(2m+1) - 1$ jest bijektywnym kodowaniem par liczb naturalnych.
   Funkcja odwrotna zadana jest przez funkcje $\pi_1, \pi_2 : N \longrightarrow N$ zdefiniowane następująco:
   $\pi_1(x) = n$ wtw, gdy 2 pojawia się w rozkładzie x+1 na czynniki pierwsze z wykładnikiem n;
   $\pi_2(x) = \frac{1}{2}(\frac{x+1}{2^{\pi_1(x)}} - 1)$

2. Wtedy funkcja $\beta : N \times N \times N \longrightarrow N, \beta(n,m,p) = \pi(\pi(n,m),p)$ jest bijektywnym kodowaniem trójek liczb naturalnych. Jak wygląda funkcja odwrotna?

3. Rozważmy funkcję $\tau : \bigcup_{k>0} N^k \longrightarrow N$ taką, że

   $$\tau(a_0, a_1, \ldots, a_{k-1}) = 2^{a_0} + 2^{a_0+a_1+1} + \ldots + 2^{a_0+a_1+\ldots a_{k-1}+k-1} - 1.$$

   Jest to bijektywne kodowanie wszystkich skończonych ciągów liczb naturalnych.

Pokażemy później, że funkcje π i β są (RAM-)obliczalne, a numeracja τ^{-1} jest efektywna. Będą to kluczowe rezultaty techniczne.

2 Maszyny RAM

2.1 Maszyny z nieograniczoną pamięcią RAM (Sheperdson & Sturgis)

Budowa maszyny realizującej programy w języku zbliżonym do popularnych imperatywnych języków programowania, który będziemy rozważać, przypomina budowę rzeczywistych komputerów: maszyna składa się z licznika rozkazów oraz pamięci, która jest tablicą rozmiaru $\aleph_0$. Komórki pamięci są numerowane kolejnymi liczbami naturalnymi i mogą zawierać, tak jak licznik rozkazów, dowolną liczbę naturalną. W czasie obliczeń prawie wszystkie komórki pamięci zawierają liczbę 0.

Oznaczenie z[x] oznacza zawartość komórki o numerze x.

Lista instrukcji RAM

Kod instrukcji	Kod adresów	Oznaczenie	Semantyka	Uwagi
0	n	Z(n)	z[n]:=0	Licznik rozkazów zwiększ o 1
1	n	S(n)	z[n]:=z[n]+1	Licznik rozkazów zwiększ o 1
2	$\pi(m,n)$	T(m,n)	z[n]:=z[m]	Licznik rozkazów zwiększ o 1
3	$\beta(m,n,q)$	I(m,n,q)	if z[m]=z[n] then goto q	Licznik rozkazów zwiększ o 1, gdy z[m] $\neq$ z[n], w przeciwnym przypadku umieść w nim q

Instrukcje typu 0, 1, 2 nazywamy arytmetycznymi , a typu 3 warunkowymi.

Definicja 2.1 Programem *(na RAM) nazywamy dowolny niepusty ciąg RAM-instrukcji.*

Semantyka instrukcji typu 3 wymusza numerowanie instrukcji w programie.

Kultura matematyczna wymaga od nas sformalizowania pojęć stanu, instrukcji, semantyki programu itp. Wrócimy do tego w rozdziale 4. Na razie wykorzystamy naszą intuicję.

Przykładowy program

0 I(1,2,5)
1 S(2)
2 S(3)
3 I(1,2,5)
4 I(1,1,1)
5 T(3,0)

Co liczy ten program?

Oczywiście interesują nas komórki pamięci, które są aktywne w czasie obliczeń. Program zatrzymuje się, gdy z[1]=z[2]. Jeśli więc na początku z[2]$\leqslant$z[1], to wykonuje się zwiększanie z[2], aż równość stanie się prawdziwa. Zwiększa się z[3] o ilość dodawań, czyli z[1]-z[2]. Jeśli więc z[i] oznacza początkową zawartość komórki, to po zakończeniu obliczeń w komórce o numerze 0 znajduje się liczba z[3]+z[1]-z[2]. Zauważmy, że program nie zatrzyma się, gdy na początku z[2]>z[1]!

Z charakteru RAM wynika, że będziemy zajmować się programami, które na wejściu biorą ciągi (ustalonej długości) liczb naturalnych i zwracają jedną

liczbę naturalną.

Umowa : Program zwraca zawartość komórki o numerze 0.

Oczywiście nie zawsze program musi się zatrzymać na każdych danych.

Naturalne jest więc pytanie jakie funkcje są obliczane przez RAM-programy. Będzie to pierwszy problem, którym się zajmiemy. Okaże się, że funkcje arytmetyczne są RAM-obliczalne.

Definicja 2.2 *Niech $k > 0$. Program P oblicza funkcję częściową $f : N^k \longrightarrow N$, jeśli po umieszczeniu ciągu danych $\underline{x}$ w komórkach z[1], ... , z[k], wyzerowaniu pozostałych i uruchomieniu P, P zatrzyma się zwracając wartość f na danych $\underline{x}$, dokadnie gdy $\underline{x} \in Dom\ f$.*

Każdy program P oblicza pewną funkcję k-argumentową, dla $k > 0$, którą oznaczymy symbolem $\phi_P^{(k)}$.
Symbolem $\mathcal{C}_k$ oznaczymy zbiór wszystkich funkcji RAM-obliczalnych i k-argumentowych.

Przyklad 1 Na gruncie aksjomatyki Peano liczb naturalnych dodawanie definiuje się rekurencyjnie:

$$\text{x+0=x}$$
$$\text{x+(y+1)=(x+y)+1}$$

Nieformalnie: należy do x dodać 1 y-razy. Oto program RAM obliczający z[1]+z[2], przy założeniu, że z[3]=0.

0 I(2,3,5)
1 S(1)
2 S(3)
4 I(1,1,0)
5 T(1,0)

Przykład 2 Niech $x \div 1 = \begin{cases} x-1 & \text{gdy } x \geqslant 1 \\ 0 & \text{gdy } x = 0 \end{cases}$. Funkcja ta jest obliczana przez następujący program.

0 I(1,4,9)
1 S(3)
2 I(1,3,6)
3 S(2)
4 S(3)
5 I(1,1,2)
6 T(2,0)

Uzasadnienie: W czasie obliczeń (od instrukcji 2) różnica z[3]-z[2] stale równa jest 1, a obliczenia zatrzymują się, gdy z[3] osiągnie początkową wartość z[1].
Pytanie: Zgodnie z definicją, możemy zastanawiać się, jaką dwuargumentową funkcję liczy ten program?
Odpowiedź: $f(x,y) = y + (x \div 1)$.

Zadanie 0 Jaką funkcję trójargumentową oblicza powyższy program?
Zadanie 1 Pokazać, że następujące funkcje są RAM-obliczalne. Symbol ∞ oznaczać będzie, że wartość funkcji pozostaje nieokreślona.

1. $f(x) = \begin{cases} \frac{x}{2} & \text{gdy } x \text{ jest parzysta} \\ \infty & \text{gdy } x \text{ jest nieparzysta} \end{cases}$
2. $f(x) = \begin{cases} 1 & \text{gdy } x > 0 \\ 0 & \text{gdy } x = 0 \end{cases}$
3. $f(x) = 5$
4. $f(x,y) = \begin{cases} 1 & \text{gdy } x \neq y \\ 0 & \text{gdy } x = y \end{cases}$
5. $f(x,y) = \begin{cases} 1 & \text{gdy } x > y \\ 0 & \text{gdy } x \leqslant y \end{cases}$
6. $f(x) = \begin{cases} \frac{x}{3} & \text{gdy } x \text{ dzieli się przez 3} \\ \infty & \text{w przeciwnym przypadku} \end{cases}$

Umowa: GOTO(n)=I(x,x,n)

2.2 Inne maszyny typu RAM tej samej mocy obliczeniowej

1. Bazowa Shoenfielda wykonuje instrukcje
   Increase r[i] czyli S(i)
   GOTO n
   Decrease r[i] , n której wykonanie oznacza zmniejszenie r[i] o 1 i przejście do instrukcji o numerze n, gdy r[i]>0

2. Uogólniona RAM wykonuje instrukcje
   z[c]:=z[a]+z[b]
   z[c]:=z[a]+b
   z[c]:=z[a]*z[b]
   z[c]:=z[a]*b
   if z[a]=z[b] then goto c
   if z[a]<z[b] then goto c

Zadanie 2 Pokazać, że każdy program w jednym z języków można przetłumaczyć na pozostałe języki z zachowaniem semantyki.

3 Struktura zbioru funkcji RAM-obliczalnych

W DALSZYCH ROZWAŻANIACH PROGRAMY SĄ W POSTACI STANDARDOWEJ, CO OZNACZA, ŻE W INSTRUKCJACH WARUNKOWYCH I(n,m,q) DANEGO PROGRAMU P, $q \leqslant k+1$, GDZIE k JEST NUMEREM OSTATNIEJ INSTRUKCJI P.

Zadanie Wykaż, że dla dowolnego RAM-programu P istnieje RAM-program P' w postaci standardowej, który oblicza te same funkcje co P.

Idea rozwiązania Każdą instrukcję warunkową I(n,m,q) programu P taką, że q>k+1, zastąp instrukcją I(n,m,k+1).

Niech P i S będą programami w postaci standardowej. Łatwo jest poprawnie wprowadzić **składanie** programów PS, które ma odpowiadać przejściu od ostatniej instrukcji P do pierwszej instrukcji S. Należy każdą instrukcję I(n,m,q) programu S zastąpić instrukcją I(n,m,q+k+1), gdzie k dalej jest numerem ostatniej instrukcji P.

OZNACZENIE Dla programu P symbolem ρ(P) oznaczamy najmniejszą liczbę naturalną m taką, że P nie używa w czasie obliczeń komórki o numerze m, ani o numerach większych od m.

Aby uprościć pisanie RAM-programów wprowadzimy procedury. Niech P $= i_0, \ldots, i_p$ będzie programem w postaci standardowej, a $l_1, \ldots, l_n \notin$

$\{1, \ldots, n\}$. Wtedy $\mathrm{P}[l_1, \ldots, l_n \to l]$ oznacza program

$$\mathrm{T}(l_1, 1) \ldots \mathrm{T}(l_n, n)\mathrm{Z}(n+1) \ldots \mathrm{Z}(\rho(\mathrm{P}))\mathrm{PT}(0, l).$$

Zauważmy, że procedura zaczyna się od przygotowania pola dziaania dla programu P, czyli rejestrów 0,1, ... , ρ(P). Procedura jest więc wykonywana "w miejscu", a nie "na boku", co ułatwi nam analizowanie programów proceduralnych, ale jest sprzeczne z ogólnie przyjęta praktyką programistyczną.

Obecnie zajmiemy się własnościami zbioru $\mathcal{C}$ wszyskich funkcji RAM-obliczalnych. Omówimy różne konstruktory funkcji obliczalnych takie, jak rekursja czy minimalizacja.

3.1 Podstawianie

Twierdzenie 3.1 (O podstawianiu) *Niech $f \in \mathcal{C}_k$ a $g_1, \ldots, g_k \in \mathcal{C}_n$. Wtedy funkcja $h(\underline{x}) \simeq f(g_1(\underline{x}), ..., g_k(\underline{x})) \in \mathcal{C}_n$.*

Dowód Niech $f \in \mathcal{C}_k$ będzie obliczana przez program F, a $g_1, \ldots, g_k \in \mathcal{C}_n$ będą obliczane przez programy $\mathrm{G}_1, \ldots, \mathrm{G}_k$ odpowiednio. Wtedy funkcja $h(\underline{x})$ jest obliczana przez podany program H.
Niech $m = max\{n, k, \rho(F), \rho(G_1), \ldots, \rho(G_k)\}$.

$$\begin{array}{l}
\mathrm{T}(1, m+1) \\
\ldots \\
\ldots \\
\ldots \\
\mathrm{T}(n, m+n) \\
G_1[m+1, \ldots, m+n \to m+n+1] \\
\ldots \\
\ldots \\
\ldots \\
G_k[m+1, \ldots, m+n \to m+n+k] \\
F[m+n+1, ..., m+n+k \to 0]
\end{array}$$

Podstawianie pozwala utożsamiać i wprowadzać nieistotne argumenty. Niech $f(x, y)$ będzie RAM obliczalna. Wtedy f=cje $f(x, x),\ f(y, x),\ g(x, y, z) \simeq f(x, z)$ są obliczalne.

3.2 Rekursja

Rekursja to sposób zadania kolejnej wartości funkcji poprzez wartości wcześniej obliczone.

Twierdzenie 3.2 (O operatorze rekursji) *Niech $f(\underline{x}) \in \mathcal{C}_n$ i $g(\underline{x}, y, z) \in \mathcal{C}_{n+2}$. Wtedy funkcja $h(\underline{x}, y) : N^{n+1} \longrightarrow N$ określona układem*

$$(Rek) \begin{array}{l} h(\underline{x}, 0) \simeq f(\underline{x}) \\ h(\underline{x}, y+1) \simeq g(\underline{x}, y, h(\underline{x}, y)) \end{array}$$

jest obliczalna ($h \in \mathcal{C}_{n+1}$).

Dowód Niech $f(\underline{x}) \in \mathcal{C}_n$ i $g(\underline{x}, y, z) \in \mathcal{C}_{n+2}$ będą obliczane przez programy F i G odpowiednio. Wtedy funkcja $h(\underline{x}, y) : N^{n+1} \longrightarrow N$ jest obliczana przez podany program H.
Niech $m = max(n+2, \rho(F), \rho(G))$.
Używać będziemy rejestrów $1, 2, ..., m+n+1, m+n+2, m+n+3$ i oznaczenia $t = m + n$.

T$(1, m+1)$	
T$(2, m+2)$	
$\ldots$	
$\ldots$	
$\ldots$	
T$(n+1, m+n+1)$	
F$[1, 2, \ldots, n \rightarrow t+3]$	oblicz f(x) i zapisz w rejestrze t+3
q I$(t+2, t+1, p)$	jeśli z[t+2] równe jest y, to zakończ
G$(m+1, \ldots, m+n, t+2, t+3 \rightarrow t+3]$	jeśli nie posłuż się G
S$(t+2)$	zwiększ$z[t+2]$
GOTO q	
p T$(t+3, 0)$	

Zadania

1. Przypomnijmy definicję ciągu Fibonacciego: $f(0) = f(1) = 1$
   $f(n+2) = f(n) + f(n+1)$.
   Udowodnij, że funkcja f jest RAM-obliczalna.
   WSKAZÓWKA: Rozważ funkcję $g(n) = 2^{f(n)}3^{f(n+1)}$.

2. Uogólnij poprzednie zadanie. Niech $k > 0$. Niech $f_i(\underline{x}) \in \mathcal{C}_n$, gdzie $i \in \{0, 1, \ldots, k-1\}$ i $g(\underline{x}, y, \underline{z}) \in \mathcal{C}_{n+1+k}$ będą obliczlane. Wtedy funkcja $h(\underline{x}, y) : N^{n+1} \longrightarrow N$ określona układem

$$
\begin{array}{rl}
 & h(\underline{x}, 0) \simeq f_0(\underline{x}) \\
 & h(\underline{x}, 1) \simeq f_1(\underline{x}) \\
(Rek) & \\
 & h(\underline{x}, k-1) \simeq f_{k-1}(\underline{x}) \\
 & h(\underline{x}, y+k) \simeq g(\underline{x}, y, h(\underline{x}, y), \ldots, h(\underline{x}, y+k-1))
\end{array}
$$

jest RAM-obliczalna.

3.3 Ważne przykłady funkcji RAM-obliczalnych

1. Dodawanie $x + y$;

2. Mnożenie $x \cdot y$ można zdefiniować przez rekursję wykorzystując dodawanie: $x \cdot 0 = 0$; $x \cdot (y+1) = x \cdot y + x$. Uwaga: do schematu rekursji dobieramy za f funkcję stałą zero, za $g(x, y, z) = z + x$.

3. Potęgowanie podobnie definiuje się przez rekursję wykorzystujac mnożenie: $x^0 = 1$; $x^{y+1} = x^y \cdot x$.

4. Odejmowanie $x \div 1 = \begin{cases} x-1 & \text{gdy } x \geqslant 1 \\ 0 & \text{gdy } x = 0 \end{cases}$ i ogólnie
   $x \div y = \begin{cases} x-y & \text{gdy } x \geqslant y \\ 0 & \text{gdy } x < y \end{cases}$ również łatwo zdefiniować przez rekursję:
   (a) $0 \div 1 = 0$ $(x+1) \div 1 = x$;
   (b) $x \div 0 = x$ $x \div (y+1) = (x \div y) \div 1$.

Pozostałe przykłady są zebrane w poniższej tabeli.

Funkcja	Schemat
$sg(x) = \begin{cases} 0 & \text{gdy } x = 0 \\ 1 & \text{gdy } x > 0 \end{cases}$	$sg(0) = 0$ $sg(x+1) = 1$
$\overline{sg}(x) = \begin{cases} 1 & \text{gdy } x = 0 \\ 0 & \text{gdy } x > 0 \end{cases}$	$\overline{sg}(x) = 1 \div sg(x)$
$\lvert x - y \rvert$	$\lvert x - y \rvert = (x \div y) + (y \div x)$
$x!$	ćwiczenie
$min(x, y)$	$min(x, y) = x \div (x \div y)$
$max(x, y)$	$max(x, y) = x + (y \div x)$
$rm(x, 0) = 0$ $rm(x, y+1) = \begin{cases} 0 & rm(x,y)+1 = x \\ rm(x,y)+1 & rm(x,y)+1 \neq x \end{cases}$	$rm(x, 0) = 0$ $rm(x, y+1) =$ $(rm(x, y)+1)sg(\lvert x - (rm(x, y)+1)\rvert)$ $g(x, y, z) = (z+1)sg(\lvert x - z + 1)\rvert)$
$qt(x, 0) = 0$ $qt(x, y+1) = \begin{cases} qt(x,y)+1 & rm(x,y)+1 = x \\ qt(x,y) & rm(x,y)+1 \neq x \end{cases}$	$g(x, y, z) = z + \overline{sg}(\lvert x - rm(x, y) + 1)\rvert)$
$div(x, y) = \begin{cases} 1 & x \mid y \\ 0 & x \not\mid y \end{cases}$	$div(x, y) = \overline{sg}(rm(x, y))$

3.4 Inne konstrukcje

Niech $f(x, y)$ będzie dowolną funkcją (dwuargumentową, RAM-obliczalną).

- **Suma uogólniona** Schemat rekursji

  $$h(x, 0) = 0$$
  $$h(x, y+1) \simeq f(x, y) + h(x, y) \quad g(x, y, z) \simeq f(x, y) + z$$

  definiuje funkcję $h(x, y) \simeq \sum_{z<y} f(x, y)$.

- **Iloczyn uogólniony** Schemat rekursji

  $$t(x, 0) = 1$$
  $$t(x, y+1) \simeq f(x, y) \cdot t(x, y) \quad g(x, y, z) \simeq f(x, y) \cdot z$$

  definiuje funkcję $h(x, y) \simeq \prod_{z<y} f(x, y)$.

- Ograniczona minimalizacja

  Załóżmy, że $f(x, y)$ jest totalna. Wzór

  $$\mu\ z\ <\ y\ (f(x, z) = 0) = \sum_{v<y} \prod_{u \leqslant v} sg(f(x, u))$$

definiuje operator minimalizacji ograniczonej, zwany też operatorem wyszukiwania dla y najmniejszej liczby $z < y$, dla której $f(x, z) = 0$, gdy taka liczba nie istnieje zwracana jest liczba y.

Z twierdzenia o podstawianiu i rekursji wynika, że jeśli f jest RAM-obliczalna zdefiniowane funkcje również są RAM-obliczalne. Oczywiście założenie, że f jest 2-argumentowa jest nieistotne.

3.5 Minimalizacja

Dla $f(\underline{x}, y) : N^{n+1} \longrightarrow N$ połóżmy: $g(\underline{x}) \simeq \mu\ y\ (f(\underline{x}, y) = 0) \simeq$ najmniejsze y takie, że $f(\underline{x}, z)$ jest określona dla wszystkich $z \leqslant y$ i $f(\underline{x}, y) = 0$; gdy talie y nie istnieje wartość funkcji pozostaje nieokreślona.

Twierdzenie 3.3 (O operatorze minimalizacji) *Jeśli $f \in \mathcal{C}_{n+1}$, to $g \in \mathcal{C}_n$.*

Dowód Załóżmy, że f jest obliczana przez program F. Wtedy podany program G oblicza funkcję g. Niech $m = max(n + 1, \rho(F))$.

$$
\begin{array}{l}
\mathrm{T}(1, m+1) \\
\ldots \\
\ldots \\
\ldots \\
\mathrm{T}(n, m+n) \\
p\ \mathrm{F}[m+1, ..., m+n, m+n+1 \rightarrow 1] \\
\mathrm{I}(1, m+n+2, q) \\
\mathrm{S}(m+n+1) \\
\mathrm{I}(1, 1, p) \\
q\ \mathrm{T}(m+n+1, 0)
\end{array}
$$

Uwaga Niech $k > 0$, predykat $M(\underline{x}) \subseteq N^k$ jest **rozstrzygalny**, jeśli jego funkcja charakterystyczna c_M jest obliczalna.
Zadanie Wykorzystując, jeśli to konieczne, operator minimalizacji wykaż, że:

1. zbiór liczb pierwszych jest rekurencyjny;

2. funkcja $p : N \mapsto N$ opisująca rosnący ciąg wszystkich liczb pierwszych jest RAM-obliczalna (często będziemy pisać p_x zamiast $p(x)$);

3. funkcja $(x)_y$ zwracająca dla $x \neq 0$ wykładnik z jakim p_y pojawia się w rozkładzie na czynniki pierwsze x i 0 dla $x = 0$ jest RAM-obliczalna;

4. jeśli predykat $M(\underline{x}) \subseteq N^k$ jest rozstrzygalny i $f, g \in \mathcal{C}_k$, to funkcja $h(\underline{x}) \simeq \begin{cases} f(\underline{x}) & M(\underline{x}) \\ g(\underline{x}) & \neg M(\underline{x}) \end{cases}$ jest RAM-obliczalna.

4 Funkcje częściowo rekurencyjne

W rozdziale tym zajmiemy się zbiorem funkcji częściowo rekurencyjnych zdefiniowanych przez Kleene w 1952. roku. Jest to czysto **matematyczna** formalizacja pojęcia funkcji obliczalnej.

Lemat 4.1 *Funkcje zerowa i następnika:* $0, s : N \longrightarrow N$, $0(x) = 0, s(x) = x + 1$ *oraz funkcje rzutowania* $p_i^n : N^n \longrightarrow N$, $p_i^n(x_0, \ldots, x_{n-1}) = x_i$, *dla dowolnych* $n > 0, i < n$, *są RAM-obliczalne.*

Definicja 4.2 *Zbiór R funkcji częściowo rekurencyjnych definiujemy jako najmniejszy zbiór funkcji, który zawiera funkcje bazowe: zerową, następnika i rzuty oraz jest domknięty na operatory podstawiania, rekursji i minimalizacji.*

Definicja 4.3 *Podobnie PR oznacza zbiór funkcji prymitywnie rekurencyjnych, które powstają z funkcji bazowych przy użyciu operatorów podstawiania i rekursji.*

Odnotujmy, że z lematu 4.1 i twierdzeń poprzedniego rozdziału wynika bezpośredni wniosek.

Wniosek 4.4 *Każda funkcja częściowo rekurencyjna jest RAM-obliczalna.*

Nasuwa się pytanie, czy jest też odwrotnie.

Twierdzenie 4.5 *Każda funkcja RAM-obliczalna jest częściowo rekurencyjna.*

Zanim zajmiemy się dowodem powyższego twierdzenia musimy wprowadzić matematyczne definicje zbioru S stanów maszyny RAM i zbioru I jej instrukcji:

$$S = \{f : N \cup \{L\} \longrightarrow N : \exists_m \forall_{n>m} f(n) = 0\}$$

$$I = \{0, 1, 2, 3\} \times N$$

Wtedy możemy formalnie określić semantykę każdej instrukcji. Funkcję $\delta : I \times S \longrightarrow S$ zmiany stanu definiujemy następująco: dla stanu $f \in S$ oraz instrukcji $i =< k, a > \in I$ kładziemy ($g = \delta(i, f)$)

$$g(L) = \begin{cases} q & \text{gdy } k = 3 \wedge f(m) = f(n) \\ & \text{gdzie } a = \beta(m, n, q) \\ f(L) + 1 & \text{w pozostałych przypadkach} \end{cases}$$

k=0 $\quad g(c) = \begin{cases} 0 & c = a \\ f(c) & c \neq a \end{cases}$

k=1 $\quad g(c) = \begin{cases} f(c) + 1 & c = a \\ f(c) & c \neq a \end{cases}$

k=2 $\wedge a = \pi(m, n)$ $\quad g(c) = \begin{cases} f(m) & c = n \\ f(c) & c \neq n \end{cases}$

k=3 $\quad \forall_{c \in N}\ g(c) = f(c)$

Teraz możemy podać **formalną definicję funkcji** $\phi_P^{(m)}$ m-argumentowej obliczanej przez program P $= i_0, i_1 \ldots i_s$. Ciąg $\underline{x} \in N^m$ należy do dziedziny definiowanej funkcji wtw, gdy istnieje ciąg stanów $f_0, f_1, \ldots, f_t$ taki, że

1. f_0 jest konfiguracją początkową P z $\underline{x}$ na wejściu;
2. $\forall_{k=0\ldots t-1}\ f_k(L) \leqslant s$;
3. $\forall_{k=0\ldots t-1}\ f_{k+1} = \delta(f_k, i_{f_k(L)})$;
4. $f_t(L) \geqslant s + 1$

Wtedy $\phi_P^{(m)}(\underline{x}) = f_t(0)$.

Rozważmy funkcję $cd : S \longrightarrow N$ kodującą wszystkie konfiguracje RAM:

$$cd(f) = 2^{f(L)} \prod_{n \in N} p_{n+1}^{f(n)}.$$

Przypomnijmy, że p_n oznacza n-tą liczbę pierwszą.

Poszukujemy funkcji (prymitywnie) rekurencyjnej $\overline{\delta} : I \times N_+ \longrightarrow N$ takiej, że $cd \circ \delta = \overline{\delta} \circ (id_I \times cd)$, co matematycznie oznacza, że funkcja δ opisująca semantykę instrukcji jest (prymitywnie) rekurencyjna.

Podsumujmy wcześniejsze rezultaty.

1. Funkcje kodujące pary liczb naturalnych $\pi(n.m) = 2^n(2m+1) - 1$ i trójki $\beta(n,m,p) = \pi(\pi(n,m),p)$ są prymitywnie rekurencyjne oraz bijektywne.

2. Istnieją prymitywnie rekurencyjne funkcje $\pi_1, \pi_2, \beta_1, \beta_2, \beta_3 : N \longrightarrow N$ takie, że $< \pi_1, \pi_2 > i < \beta_1, \beta_2, \beta_3 >$ są odwrotnymi do π i β: $\pi_1(x) = (x+1)_0,\ \pi_2(x) = qt(2, qt(2^{\pi_1(x)}, x+1)),\ \beta_1(x) = \pi_1(\pi_1(x)),\ \beta_2(x) = \pi_2(\pi_1(x)),\ \beta_3(x) = \pi_2(x)$.

Teraz

$$\overline{\delta}(< k, a >, s) = \begin{cases} 2 \cdot qt(p_{a+1}^{(s)_{a+1}}, s) & k = 0 \\ 2 \cdot p_{a+1} \cdot s & k = 1 \\ 2 \cdot qt((p_{\pi_2(a)+1})^{(s)_{\pi_2(a)+1}}, s) \cdot (p_{\pi_2(a)+1})^{(s)_{\pi_1(a)+1}} & k = 2 \\ 2 \cdot s & k = 3 \wedge (s)_{\beta_1(a)} \neq (s)_{\beta_2(a)} \\ 2^{\beta_3(a)} \cdot qt(2^{(s)_0}, s) & k = 3 \wedge (s)_{\beta_1(a)} = (s)_{\beta_2(a)} \\ 0 & k \geqslant 4 \end{cases}$$

Funkcja ta jest prymitywnie rekurencyjna z określenia.

Dowód twierdzenia Niech P $=< k_0, i_0 >< k_1, i_1 > \ldots < k_s, i_s >$ będzie programem w postaci standardowej obliczającym funkcję $h(\underline{x})$ arności $n > 0$. Niech $\sigma(\underline{x}, k)$ będzie kodem konfiguracji uzyskanej po k krokach obliczeń P na $\underline{x}$. Przyjmować będziemy, że konfiguracją następną po końcowej jest ona sama oraz $cod_n(\underline{x})$ jest konfiguracją początkową (czyli ciąg $\underline{x}$ umieszczony jest w rejestrach o numerach $1, \ldots, n$). Wtedy

$$\sigma(\underline{x}, 0) = cd(cod_n(\underline{x}))$$

$$\sigma(\underline{x}, k+1) = \begin{cases} \overline{\delta}(< k_l, a_l >, \sigma(\underline{x}, k)) & l = (\sigma(\underline{x}, k))_0 < s+1 \\ \sigma(\underline{x}, k) & l = (\sigma(\underline{x}, k))_0 \geqslant s+1 \end{cases}$$

Inaczej funkcja σ definiowana jest przez rekursję:

$$f(\underline{x}) = cd(cod_n(\underline{x}))$$

$$g(\underline{x}, y, z) = c_{(z)_0 < s+1} \cdot \overline{\delta}(i_{(z)_0}, z) + c_{(z)_0 \geqslant s+1} \cdot z$$

Wtedy funkcje $c(\underline{x}, k) = (\sigma(\underline{x}, k))_1$ $j(\underline{x}, k) = (\sigma(\underline{x}, k))_0$ są prymitywnie rekurencyjne i opisują zawartość rejestru 0 (c!) i licznik rozkazów (j) po k krokach obliczeń P. Wtedy

$$h(\underline{x}) \simeq c(\underline{x}, \mu\ k\ (j(\underline{x}, k) \geqslant s+1)).$$

co oznacza, że h jest częściowo rekurencyjna.

4.1 Funkcja Ackermanna

Zauważmy, że użycie podstawiania i rekursji do funkcji totalnych daje funkcję totalną, co nie jest prawdą dla minimalizacji (rozważ funkcję $g(x) = \mu y((x + y+1) = 0)$). Ta prosta obserwacja spowodowała pojawienie się wielu ważnych pytań. Podstawowe to: Czy każda funkcja totalna i częściowo rekurencyjna jest prymitywnie rekurencyjna?. Zauważmy, że jest to pytanie, czy ma sens używanie operatora minimalizacji do funkcji totalnych, gdy wynikiem jest funkcja totalna.

Okazuje się, że istnieją funkcje totalne, które są częściowo rekurencyjne, ale nie prymitywnie rekurencyjne. Rezultat ten zawdzięczamy Ackermannowi, który skonstruował taką funkcję, zresztą bardzo skomplikowaną. W literaturze pod hasłem funkcja Ackermanna występuje prostsza funkcja wprowadzona przez Hermesa:

$$\begin{aligned} &A(0, y) = y + 1 \\ &A(x + 1, 0) = A(x, 1) \\ &A(x + 1, y + 1) = A(x, A(x + 1, y)) \end{aligned}$$

Warto zapamiętać ten przykład i przetestować na nim, czym różni się definicja funkcji przez *rekursję* od definicji *rekurencyjnej*. Jest to kolejna ilustracja obserwacji, że pewne definicje *rekurencyjne* w językach programowania wysokiego poziomu można zasąpić *iteracją*, a funkcja Ackermanna nie poddaje się temu procesowi.

4.2 Teza Churcha

Teza Churcha mówi, że nieformalnie i intuicyjnie określona klasa funkcji obliczalnych pokrywa się z klasą funkcji RAM-obliczalnych. Teza ta nie jest matematycznym twierdzeniem, ale istnieją ważkie argumenty na jej rzecz. Jednym z istotniejszych jest powyższe twierdzenie, które razem z wnioskiem 4.4 stwierdza, że **klasa gunkcji RAM-obliczalnych pokrywa się z klasą funkcji częściowo rekurencyjnych**.

W wykładzie rozważać będziemy jeszcze formalizację Turinga, dla której można udowodnić podobne twierdzenie.

5 Maszyny Turinga

Mając pewną wiedze techniczną na temat budowy komputera trudno przyjąć model rozważany wcześniej. Należy też uświadomoć sobie, że prosty pomysł łatwiej zrealizować technicznie. Tę zaletę ma definicja Turinga zaproponowana w 1936 roku. Po pierwsze, w tamtych czasach znane były urządzenia mechanicznie realizujące żmudne algorytmy takie jak maszyny szyfrujące czy różnicowe. Po drugie, były one (w najlepszym wypadku) urządzeniami elektrycznymi a nie elektronicznymi. Model Turinga (jedna maszyna) rozwiązuje jeden problem wykonując bardzo proste operacje. Maszyna ta ma nieskończoną taśmę (pamięci) podzieloną na komórki, w których może znajdować się co najwyżej jeden symbol ze skończonego zbioru. Ma też głowicę, która obserwuje dokładnie jedną komórkę i skończony zbiór stanów. Na postawie tych dwóch informacji (stan, litera) podejmuje decyzję jaką literę umieścić w komórce, jaki ruch wykonać i w jaki stan wejść.

Definicja 5.1 *Jednotaśmową maszyną Turinga nazywamy trójkę* $< S, Q, P >$, *gdzie* Q *jest skończonym alfabetem symboli taśmowych,* S *skończonym zbiorem stanów rozłącznym z* Q, *a* P *skończonym zbiorem instrukcji postaci:*

$$stan_1 \ litera_{\ 1} \rightarrow \ litera_2 \ ruch \ stan_2.$$

Matematycznie

$$P \subseteq S \times (Q \cup \{B\}) \times (Q \cup \{B\}) \times \{L, R, \_\} \times S$$

gdzie B *jest nowym symbolem zwanym* **blank** *oznaczającym, że komórka jest pusta, a litery* $L, R, \_$ *odpowiadają ruchom w lewo, w prawo i w miejscu.*

Wejściem dla maszyny będzie dowolne słowo nad alfabetem Q, analizowane od pierwszej litery i początkowego stanu q_0. Kolejny etap tej analizy nazywać będziemy konfiguracją.

Definicja 5.2 *Niech* $M =< S, Q, P >$ *będzie ustaloną maszyną Turinga.*

1. *Konfiguracją* M *nazywamy każde słowo postaci* uqw, *gdzie* u, w *są słowami nad alfabetem* $Q \cup \{B\}$, *a* q *jest stanem:* $u, w \in (Q \cup \{B\})^\star, q \in S$.
   Uwaga *Będziemy ignorować symbole puste na początku* u *i końcu* w.

2. *Dla instrukcji* $qa \to bRq'$ *i konfiguracji* $vqaw$ *wykonanie jej zapisujemy* $vqaw \vdash_M vbq'w$.
   Uwaga *Dokończ definicję: dla instrukcji* $qa \to bLq'$ *i konfiguracji* $vqaw$*...*

3. *Powyższe wzory definiują relację jednokrokowej zmiany konfiguracji. Matematycznie zbiór* K *konfiguracji* M *równy jest* $(Q \cup \{B\})^\star \times S \times (Q \cup \{B\})^\star$ *i* $\vdash_M \subseteq K \times K$*. Wtedy relację* $\vdash_M^\star \subseteq K \times K$ *będącą zwrotnio-przechodnim domknięciem* $\vdash_M$ *nazywamy relacją zmiany konfiguracji maszyny* M*.*

4. *Niech* KF *będzie zbiorem konfiguracji końcowych t.j. takich w których* M *nie może wykonać żadnej instrukcji:*

$$KF = \{z \in K \ ; \ \neg \exists_{w \in K} z \vdash_M w\}$$

   a $q_0 \in S$ *wyróżnionym stanem. Język* $L_M^{q_0}$ *słów akceptowanych przez* M *przy stanie początkowym* q_0 *definiujemy jako zbiór tych słów* s *nad alfabetem* Q*, które po umieszczeniu na taśmie słowa, ustawieniu głowicy przy przy jego pierwszej literze* w *stanie* q_0*, powodują zatrzymanie maszyny:*

$$L_M^{q_0} = \{s \in Q^\star \ ; \ \exists_{w \in KF} \ q_0 s \vdash_M^\star w\}$$

5. *Do zdefiniowania zbioru konfiguracji końcowych można wykorzystać zbiór* $F \subseteq S$ *zwany zbiorem stanów końcowych. Wtedy*

$$KF_F = \{z \in K \ ; \ \exists_{w,v \in (Q \cup \{B\})^\star} \exists_{q \in F} z = vqw\}$$

   Język $L_M^{q_0,F}$ *słów akceptowanych przez* M *przy stanie początkowym* q_0 *i zbiorze stanów końcowych* F *definiujemy jako zbiór tych słów* s *nad alfabetem* Q*, które po umieszczeniu na taśmie słowa, ustawieniu głowicy przy jego pierwszej literze* w *stanie* q_0*, powodują przejście maszyny* w *stan ze zbioru* KF_F*:*

$$L_M^{q_0,F} = \{s \in Q^\star \ ; \ \exists_{w \in KF_F} \ q_0 s \vdash_M^\star w\}$$

5.1 Jednotaśmowa maszyna Turinga dla palindromów

Niech $X = \{a_1, \ldots, a_n\}$ będzie dowolnym skończonym i niepustym alfabetem. Rozważmy MT o instrukcjach

$q_0 a_i \longrightarrow a_i R q_0$	szukaj prawego końca słowa
$q_0 B \longrightarrow B L q_1$	znalazłeś, to czego szukałeś
$q_1 a_i \longrightarrow B L s_i$	zapamiętaj, co było na końcu (a_i)
$s_i a \longrightarrow a L s_i \;\; a \in X$	wróć na początek (lewy koniec słowa)
$s_i B \longrightarrow B R t_i$	znalazłeś, to czego szukałeś
$t_i a_i \longrightarrow B R q_0$	usuń pierwszą literę, jeśli równa jest a_i

Maszyna ta zawsze się zatrzyma! W stanie q_1 lub t_i, gdy słowo na wejściu jest palindromem; W stanie t_i, gdy tak nie jest. W pierwszym przypadku na taśmie pozostaną same symbole puste B.

Zależnie od wariantu definicji języka akceptowanego przez MT dobieramy instrukcje:

1. $q_1 B \longrightarrow B L f, t_i B \longrightarrow B L f$
2. $t_i a_j \longrightarrow a_j R z_i \;\; z_i a \longrightarrow a L t_i \; a \in X$ gdy $i \neq j$

W pierwszym przypadku analizowanie palindromu zakończy się w stanie końcowym f, w drugim wejście, które nie jest palindromem spowoduje zapętlenie maszyny.

Ponieważ do zdefiniowania języka akceptowanego przez maszynę Turinga potrzebny jest stan początkowy q_0 i zbiór konfiguracji końcowych zwykle do jej definicji dodaje się ten stan i zbiór stanów końcowych zależnie od wariantu definicji. Powyższy przykład ilustruje dowód następującego twierdzenia.

Twierdzenie 5.3

1. *Dla dowolnej maszyny Turinga* $M =< S, Q, P, q_0 >$ *istnieje maszyna Turinga* $M' =< S', Q, P', q_0', F >$ *taka, że* $L_M^{q_0} = L_{M'}^{q_0', F}$;
2. *Dla dowolnej maszyny Turinga* $M' =< S', Q, P', q_0', F >$ *istnieje maszyna Turinga* $M =< S, Q, P, q_0 >$ *taka, że* $L_{M'}^{q_0', F} = L_M^{q_0}$;

5.2 Maszyna Turinga obliczająca następnik słowa w porządku leksykograficznym

Załóżmy, że alfabet $X = \{a_1, \ldots, a_n\}$ jest zbiorem uporządkowanym liniowo: $a_i < a_{i+1}$.

Słowo s poprzedza t w porządku leksykograficznym, jeśli jest krótsze albo s i t są tej samej długości i s poprzedza t w porządku alfabetycznym.

Porządek leksykograficzny jest liniowy i poprawnie określone jest pojęcie następnika słowa:$nast(sa_i a_n^k) = sa_{i+1}a_1^k$ $nast(a_n^k) = a_1^{k+1}$.

Rozważmy MT o instrukcjach

$q_0 a_i \longrightarrow a_i R q_0$	szukaj prawego końca słowa
$q_0 B \longrightarrow B L q_1$	znalazłeś, to czego szukałeś
$q_1 a_n \longrightarrow a_1 L q_1$	zastąp ostatnią na końcu pierwszą
$q_1 a_i \longrightarrow a_{i+1} L q_2$	$i < n$, teraz wystarczy wrócić na początek (lewy)
$q_2 a_i \longrightarrow a_i L q_2$	
$q_2 B \longrightarrow B R q$	to koniec
$q_1 B \longrightarrow a_1 L q_2$	wszystkie litery były równe a_n

Maszyna zawsze zatrzymuje się w stanie q, a na taśmie na prawo od głowicy znajduje się słowo będące następnikiem wejścia.

Przykład ten sugeruje, że maszyny Turinga mogą posłużyć jako urządzenia obliczające funkcje, tak jak programy na maszynę RAM. Zauważmy, że rozważane maszyny są deterministyczne.

Definicja 5.4 *1. Maszyna Turinga $M =< S, Q, P, q_0, F >$ jest deterministyczna, jeśli dla każdej pary (stan, litera) (konfiguracji zawierającej takie podsłowo) istnieje co najwyżej jedna instrukcja, którą można do niej zastosować.*

2. Dla liczby naturalnej n symbolem $b(n)$ oznaczać będziemy zapis binarny tej liczby bez zbędnych zer.

3. Detrministyczna maszyna Turinga $M =< S, \{0,1\}, P, q_0, F >$ oblicza funkcję częściową $f : N^n \mapsto N$, jeśli dla dowolnych $x_1, \ldots, x_n, y \in N$ zachodzi równoważność:

$$f(x_1, \ldots, x_n) = y \Longleftrightarrow \exists_{q \in F} q_0 b(x_1) B \ldots B b(x_n) \vdash_M^\star q b(y)$$

5.3 Maszyna Turinga obliczająca resztę z dzielenia przez 3 liczby naturalnej n, operując zapisami liczb w systemie dwójkowym bez zbędnych zer

$q_0 0 \longrightarrow BRq_0$	wczytana liczba daje resztę 0
$q_0 1 \longrightarrow BRq_1$	wczytana liczba daje resztę 1
$q_1 0 \longrightarrow BRq_2$	mnóż przez 2 liczbę dającą resztę 1
$q_1 1 \longrightarrow BRq_0$	mnóż przez 2 i dodaj 1 do liczby dającej resztę 1
$q_2 0 \longrightarrow BRq_1$	mnóż przez 2 liczbę dającą resztę 2
$q_2 1 \longrightarrow BRq_2$	mnóż przez 2 i dodaj 1 do liczby dającej resztę 2

Zadanie Zdefiniuj pozostałe instrukcje MT tak, aby w konfiguracji $q_i B$ wypisywała kod liczby i.

Uwaga W definicji funkcji obliczanej przez maszynę Turinga nie musimy zakładać, że na lewo od głowicy czytającej taśma jest pusta.

W ostatnim rozdziale zajmiemy się elementami teorii złożoności obliczeniowej. Istnieją różne miary złożoności algorytmu. Najważniejsze to **czas** i **pamięć**. Złożoność czasową mierzy się ilością kroków obliczeń. W przypadku modeli maszynowych (RAM czy Turinga) jest to związane z wywoływaniem instrukcji. Złożoność pamięciową mierzy się ilością komórek pamięci wykorzystywanych w czasie obliczenia. Wtedy możemy przyjąć, że interesują nas tylko dodatkowe koszty nie związane z umieszczeniem danych i wyników w pamięci. Główne rozważnia dotyczyć będą czasu. W rozdziale tym rozważać będziemy tylko **maszyny Turinga**. Wybór modelu jest uzasadniony:

1. tradycją;
2. konstrukcją rzeczywistych komputerów: są one raczej maszynami Turinga a nie RAM;
3. istotą problemu: można wykazać, że model RAM z **ustalonym** skończonym rozmiarem pamięci jest wystarczający w teoretycznych rozważaniach;

Zacznijmy od przykładu. Rozważmy język $A = \{0^k 1^k; k \geqslant 0\}$. Jak dużo potrzebuje czasu MT, aby rozstrzygnąć, czy $w \in A$? Trzeba rozważyć konkretną maszynę M_1;
M_1 na łańcuchu wejściowym w:

1. sprawdza, czy w słowie w pojawia się 0 na prawo od 1; jeśli tak jest odrzuca to słowo;

2. W przeciwnym przypadku powtarza następujące czynności;

3. poszukuje od lewej do prawej pary 0 i 1 usuwając je;

4. jeśli zostały 0 po usunięciu wszystkich 1 lub 1 po usunięciu wszystkich 0, to odrzuca słowo, w przeciwnym przypadku je akceptuje.

Będziemy wyznaczać czas działania algorytmu jako funkcję rozmiaru wejścia. W przypadku MT jest to długość słowa. Wtedy będziemy liczyć przypadek najgorszy, czyli wymagający najwięcej obliczeń.

Definicja 5.5 *Niech M będzie deterministyczną MT* **zatrzymującą się na wszystkich wejściach**. *Czasem działania lub złożonością czasową M jest funkcja $f : N \mapsto N$, gdzie $f(n)$ jest maksymalną ilością kroków obliczeń M wykonywanych przez nią na każdym wejściu długości n. Mówimy też, że M działa w czasie $f(n)$.*

Widać, że raczej będziemu badać asymptotyczne zachowanie maszyn Turinga.

5.4 Notacja $\mathcal{O}$

Definicja 5.6 *Niech $f, g : N \mapsto R_+$ Mówimy, że f jest rzęd $\mathcal{O}(g(n))$ (zapis $f(n) = \mathcal{O}(g(n))$ jeśli istnieją liczby $c \in R_+, n_0 \in N$ takie, że dla każdego $n \geqslant n_0$ mamy: $f(n) \leqslant cg(n)$. Mówi się też, że $g(n)$ jest asymptotycznym ograniczeniem górnym $f(n)$.*

Jasne jest, że $f(n) = \mathcal{O}(5f(n))$ więc w notacji tej pomija się współczynniki.

Zadanie Niech $f(n) = 5n^3 + 2n^2 + 22n + 4$. Wykaż, że $f(n) = \mathcal{O}(n^3)$.

Wiemy z analizy, że

$$lim_n \frac{2n^2 + 22n + 4}{n^3} = 0$$

co pociąga, że $n^3 \geqslant 2n^2 + 22n + 4$ dla $n \geqslant n_0$ (wyznacz takie n_0). Wtedy $5n^3 + 2n^2 + 22n + 4 \leqslant 6n^3$ dla $n \geqslant n_0$.

Uwaga W tej notacji rozważając logarytmy o podstawie wiekszej od jeden podstawa jest nieważna, gdyż $log_a n = \frac{log_b n}{log_b a}$.

Teraz oszacujmy czsową złożoność obliczeniową podanej na wstępie maszyny M_1: dla danego na wejściu słowa długości n realizacja 1-go etapu wymaga $2n$ instrukcji (czytanie od lewej do prawej i powrót na początek słowa); 2-go i 3-go polega na dwukrotnym czytaniu $(2n)$ powtarzanym $\frac{n}{2}$ razy co daje $\mathcal{O}(n^2)$ kroków; na koniec 4-ty zrealizuje się w $\mathcal{O}(n)$ krokach: w sumie $\mathcal{O}(n^2)$.

Rozważać też będziemy niedeterministyczne MT.

Definicja 5.7 *Niech M będzie MT* **zatrzymującą się na wszystkich wejściach**. *Czasem działania lub złożonością czasową M jest funkcja $f : N \mapsto N$, gdzie $f(n)$ jest maksymalną ilością kroków obliczeń M wykonywanych przez nią na każdym wejściu długości n w czasie każdego przebiegu obliczeń. Mówimy też, że M działa w czasie $f(n)$.*

Funkcja $f(n)$ zapewnia, że każde obliczenia M na dowolnym wejściu długości n zakończą się w tym czasie.

Definicja 5.8 *Niech $t : N \mapsto N$ będzie totalna. Funkcja ta definiuje dwie klasy czasowej złożoności obliczeniowej:*
$TIME(t(n))$ tych języków L, które są rozstrzygane prez deterministyczną maszynę Turinga o czasowej złożoności $\mathcal{O}(t(n))$
$NTIME(t(n))$ tych języków L, które są rozstrzygane prez maszynę Turinga o czasowej złożoności $\mathcal{O}(t(n))$

Skupimy się na wyjaśnieniu użytych wyżej pojęć i omówieniu wagi tego problemu. Najpierw ważne twierdzenie.

Twierdzenie 5.9 *Dla niedeterministycznej maszyny Turinga M o czasowej złożoności obliczeniowej $f(n)$ takiej, że $n \leqslant f(n)$ dla wszystkich $n \in N$, istnieje deterministyczna trójtaśmowa maszyna Turinga M_1 rozpoznająca ten sam język, której czasowa złożoność obliczeniowa jest rzędu $\mathcal{O}(c^{f(n)})$, dla pewnej liczby $c > 1$.*

Idea konstrukcji maszyny M_1. Dla ustalenia uwagi załóżmy, że M w każdej konfiguracji może wykonać co najwyżej dwie instrukcje. Ponieważ jest niedeterministyczna w pewnej konfiguracji ma faktycznie wybór pomiędzy

dwiema instrukcjami. Dokonajmy ponumerowania liczbami 0 lub 1 tych instrukcji dla takich konfiguracji. Każdy ciąg obliczeń M na wejściu długości n można wtedy opisać ciągiem zer i jedynek długości co najwyżej $f(n)$, odpowiadającym wyborowi kolejnej instrukcji. Ciągów takich jest $2^{f(n)+1} - 1$. Maszynę deterministyczną M_1 konstruuje się tak, aby dla wejścia długości n umieszczanego na pierwszej taśmie wyznaczała na drugiej taśmie kolejny ciąg w porządku leksykograficznym, a następnie realizowała obliczenia M opisane przez wygenerowany ciąg na taśmie trzeciej. Każde z $\mathcal{O}(2^{f(n)})$ obliczeń wymaga

1. kopiowania wejścia z taśmy 1. na 3. (ponieważ $n \leqslant f(n)$ wymaga to $\mathcal{O}(f(n))$ kroków;
2. generowania następnego ciągu w $\mathcal{O}(f(n))$ krokach;
3. wykonania obliczeń na taśmie trzeciej w $\mathcal{O}(f(n))$ krokach.

co razem daje $\mathcal{O}(f(n)) * \mathcal{O}(2^{f(n)}) = \mathcal{O}(4^{f(n)})$ kroków.

Można wykazać, że konstruowana w dowodzie deterministyczna MT ma dodatkową własność: jest lewostronnie ograniczona, co oznacza, że można użyć dodatkowego znaku $\triangleright$ początku słowa będącego na taśmie; w sytuacji, gdy głowica obserwuje ten znak niemożliwy jest ruch **w lewo** i maszyna nie może zmazać znaku $\triangleright$. Wtedy konfiguracja początkowa jest postaci $\triangleright q_o w$.

Twierdzenie 5.10 *1. Dla maszyny Turinga M istnieje lewostronnie ograniczona (dwutaśmowa) maszyna Turinga M', która rozpoznaje ten sam język (i ma tę samą złożoność czasową w sensie notacji $\mathcal{O}$).*

2. Dla k-taśmowej maszyny Turinga M ($k \geqslant 2$) istnieje jednotaśmowa maszyna Turinga M', która rozpoznaje ten sam język. Jeśli M ma złożoność czasową $\mathcal{O}(f(n))$, to M' ma złożoność czasową $\mathcal{O}(f(n) * f(n))$.

Szkic dowodu(1) Dla M i każdej zabronionej instrukcji np. $q\triangleright \rightarrow aLq'$ należy zdefiniować procedurę (zestaw instrukcji M'), która całe słowo za $\triangleright$ przesuwa o jedną komórkę w prawo i wtedy w komórce za $\triangleright$ można umieścić a. Zauważmy, że do zrealizowania procedury przyda się znacznik $\triangleleft$ końca słowa. Taka maszyna będzie miała złożoność obliczeniową $\mathcal{O}(f(n) * f(n))$, gdyż symulacja każdej zabronionej instrukcji wymaga czasu proporcjonalnego do

długości słowa na taśmie, a ta jest rzędu $\mathcal{O}(f(n))$! Uzasadnienie jest bardzo proste: w czasie $f(n)$ maszyna Turinga odczytuje co najwyżej $f(n)$ komórek na taśmie. Aby skonstruować maszynę o podobnej czasowej złożoności obliczeniowej wystarczy użyć drugiej taśmy do przechowania w odwróconej kolejności liter słów powstających na lewo od znaku $\triangleright$.
(2) Dla k-taśmowej maszyny M jej konfiguracja opisana jest przez zawartość każdej z taśm; czyli ciąg słów $v_i q w_i, i = 1 \dots k$ zamieniamy na słowo $\#v_1\bar{q}w_1\#v_2\bar{q}w_2\# \dots v_k\bar{q}w_k\#$. Konstrukcja M' polega na zamianie każdej instrukcji M na ciąg instrukcji, które symulują jej realizację na opisanym słowie. Każde słowo powstające w czasie symulacji obliczeń M na słowie długości n jest długości $\mathcal{O}(f(n))$, a zmiana wymaga po prostu jego przeczytania.

Zadanie Skonstruuj dwutaśmową maszynę Turinga dla języka palindromów o liniowej czasowej złożoności obliczeniowej.

Teraz możemy porównać moc obliczeniową maszyny RAM i maszyn Turinga. Chyba nikogo nie zdziwi, że są one (moce) identyczne. Postaramy się raczej porównać złożoność obliczeniową programów na RAM i symulujących je maszyn Turinga.

Definicja 5.11 *Niech P będzie RAM programem.*

1. *Krokiem obliczeń P nazywamy wykonanie jednej instrukcji P; dla $\underline{x} \in N^n$ wprowadźmy oznaczenia:*

   $P(\underline{x}) \downarrow$ *oznacza, że obliczenia P na danej $\underline{x}$ zatrzymują się;*
   $P(\underline{x}) \downarrow k$ *oznacza, że obliczenia P na danej $\underline{x}$ zatrzymują się w co najwyżej k krokach;*
   $P(\underline{x}) \uparrow$ *oznacza, że obliczenia P na danej $\underline{x}$ nie zatrzymują się;*

2. *Zdefiniujmy funkcję*

$$t_P^{(n)}(\underline{x}) = \begin{cases} \text{liczba kroków obliczeń } P(\underline{x}) & P(\underline{x}) \downarrow \\ \infty & P(\underline{x}) \uparrow \end{cases}$$

3. *Niech P będzie takim programem, który zatrzymuje się na wszystkich danych $n \in N$. Mówimy, że P ma złożoność $f(n)$ przy jednorodnym kryterium kosztów jeśli $t_P^{(1)} \in \mathcal{O}(f(n))$.*

Z praktycznego punktu widzenia jest to nie najlepsza definicja. Rozważa się więc tzw. logarytmiczne kryterium kosztów.
Niech $l(n)$ będzie długością słowa $b(n)$ dla $n \in N$, czyli zapisu binarnego n bez zbędnych zer. Aby reprezentować konfigurację z maszyny RAM w czasie obliczeń RAM programu P na jednotaśmowej maszynie Turinga będziemy używać ciągu

$$b(z(L))\#\#b(0)\#b(z(0))\#\#b(1)\#b(z(1))\ldots\#\#b(\rho(P)-1)\#b(z(\rho(P)-1)) \quad (1)$$

zakładając, że P używa wszystkich komórek o numerach $0,\ldots,\rho(P)-1$
Wtedy logarytmiczny koszt realizacji każdej instrukcji RAM liczymy według wzorów:

$l_z(Z(n)) = l(n) + l(z(n))$
$l_z(S(n)) = l(n) + l(z(n))$
$l_z(T(n,m)) = l(n) + l(m) + l(z(n)) + l(z(m))$
$l_z(I(n,m,q)) = l(n) + l(m) + l(z(n)) + l(z(m)) + l(q)$

Wtedy wartość funkcji $t_P^n(\underline{x})$ definiujemy jako sumę logarytmicznych kosztów wszystkich instrukcji wykonywanych do zatrzymania programu.

Twierdzenie 5.12 *Dla lewostronnie ograniczonej deterministycznej maszyny Turinga $M =< S, Q, P >$ istnieje RAM program P symulujący jej obliczenia.*

Szkic dowodu Załóżmy, że $Q = \{0, 1, \ldots, n\}$ a $S = \{0, 1, \ldots, k\}$ (dokładniej ustalmy numeracje tych zbiorów); ponadto przyjmijmy, że do zakończenia obliczeń używamy pewnego zbioru stanów. Wtedy konfigurację M postaci uqw zapamiętujemy jako piątkę $(|u|, |w|, q, k(u), k(w))$, gdzie $|u|$ oznacza długość słowa, a $k(u)$ jego kod np. gödlowski: $k(a_1, \ldots, a_n) = \prod_{i=0}^{n} p_i^{a_i}$. Dla każdej instrukcji M można napisać procedurę na RAM realizującą jej wykonanie; n.p. instrukcja $qa \to a \_ q'$ oznacza tylko zmianę rejestru, w którym jest przechowywany stan. Trochę trudniejsze jest sprawdzenie, czy pierwszą literą w jest a: czyli czy można tę instrukcję zastosować. W sumie RAM program będzie będzie miał postać pętli **repeat**, w której umieszczone są wszyskie procedury symulujące instrukcje M.

Twierdzenie 5.13 *Dla RAM programu P obliczającego funkcję jednoargumentową h o logarytmicznym kryterium kosztów $f(n)$ istnieje trójtaśmowa maszyna Turinga M obliczająca tę funkcję o czasowej złożoności obliczeniowej $\mathcal{O}(f(n) * f(n))$.*

Szkic dowodu Zauważmy, że długość każdego słowa (1) opisującego konfiguracje maszyny w czasie działania P jest rzędu $\mathcal{O}(f(n))$. Maszyna Turinga M przechowuje takie słowo na pierwszej taśmie. Aby zrealizować n.p. instrukcję $Z(k)$ w konfiguracji z musi:

1. odszukać na pierwszej taśmie zapis binarny k;
2. przesunąć się (usuwając kolejne litery) za $b(z(k))$;
3. wszystko co jest dalej prznieść (usuwając z pierwszej) na taśmę drugą;
4. wróćić na koniec słowa na taśmie pierwszej, aby dopisać zawartość taśmy drugiej.

Realizacja takich obliczeń wymaga $\mathcal{O}(f(n))$ ruchów. Ponieważ maszyna RAM wykonuje co najwyżej $f(n)$ instrukcji w czasie obliczania $h(n)$ otrzymujemy żądaną tezę.

6 Efektywne numeracje programów

W rozdziale tym zgłębimy oczywistą obserwację, że RAM-programów jest przeliczalnie wiele. Istnieją więc numeracje zbioru Π wszystkich RAM-programów.

Definicja 6.1 *Niech $\beta : N \longrightarrow \Pi$ będzie dowolną numeracją zbioru RAM-programów (czyli funkcją "na"). Mówimy, że β jest efektywna, jeśli istnieją obliczalne funkcje $d : N \longrightarrow N, kod, A : N^2 \longrightarrow N$ takie, że dla wszelkich $x \in N$ i $\beta(x) =< i_0, \ldots, i_n >$ mamy:*

$$d(x) = n;$$

$$\forall_{0 \leqslant r \leqslant n} i_r =< kod(x, r), A(x, r) > .$$

F-cja d określa długość (ilość instrukcji) programu $\beta(x)$, natomiast kod, A określają kody kolejnych instrukcji oraz adresy komórek, którymi one manipulują.

Przypomnienie: $I = \{0, 1, 2, 3\} \times N \subset N \times N$.

Niech $x \in N$, $x = 4a + k$ i a będzie częścią całkowitą a k resztą z dzielenia x przez cztery.Odnotujmy oczywisty lemat.

Lemat 6.2 *Funkcja $\alpha : N \longrightarrow I$ taka, że $\alpha(x) =< k, a >$ jest bijektywną numeracją zbioru instrukcji.*

Posłużymy się nią przy zdefiniowaniu efektywnej numeracji programów. Rozważmy funkcję $\tau : \bigcup_{k>0} N^k \longrightarrow N$ taką, że

$$\tau(a_0, a_1, \ldots, a_{k-1}) = 2^{a_0} + 2^{a_0+a_1+1} + \ldots + 2^{a_0+a_1+\ldots+a_{k-1}+k-1} - 1$$

Jest to bijektywne kodowanie wszystkich skończonych ciągów liczb naturalnych. Omówimy dokładniej numerację odwrotną τ^{-1} , gdyż chcemy udowodnić lemat.

Lemat 6.3 *Funkcja*

$$\mu(x) =< \alpha(a_0), \ldots, \alpha(a_n) >, \ \textit{gdzie } \tau(a_0, \ldots, a_n) = x$$

jest efektywną numeracją programów.

Oczywiście funkcja τ^{-1} powiązana jest z przedstawianiem liczb naturalnych w systemie dwójkowym.

Każda liczba naturalna większa od zera jest sumą rosnących potęg dwójki, czyli dla $x \in N$:

$$x + 1 = 2^{b_0} + 2^{b_1} + \ldots 2^{b_m} \quad \text{gdzie } b_i < b_{i+1}$$

Wtedy wzory $a_0 = b_0$ i $a_{i+1} = b_{i+1} - b_i - 1$ dla $i < m$ opisują ciąg, którego kodem (poprzez τ) jest liczba x.

Wyznaczanie przedstawienia x opisują podane funkcje (prymitywnie) rekurencyjne.

1. $l(x) = \mu(y < x)(x < 2^{y+1})$

2. Pomocniczy ciąg

$$c(0, x) = qt(2, x) \ \ c(n + 1, x) = qt(2, c(n, x))$$

3. Wtedy

$$b(0, x) = rm(2, x) \ \ b(n + 1, x) = rm(2, c(n, x))$$

Zauważmy, że jest to klasyczny algorytm wyznaczania zapisu liczby naturalnej x w systemie binarnym opierający się na spostrzeżeniu, iż cyfra "jedności" jest resztą z dzielenia przez 2 ($b(0,x)$), a następną cyfrę (od prawej!) wyznacza się podobnie biorąc zamiast x część całkowitą z tego dzielenia.

Teraz można odszukać cyfry 1 w powyższym przedstawieniu.

$$l_1(x) = \Sigma_{y \leqslant l(x+1)} sg(b(y, x+1))$$

$$b_1(0,x) = \mu\, y \ \leqslant l(x+1)(b(y,x+1) = 1)$$

$$b_1(i+1,x) = \mu\, y \ \leqslant l(x+1)(b(y,x+1) = 1 \wedge b_1(i,x) < y)$$

$$a_1(0,x) = b_1(0,x)$$

$$a_1(i+1,x) = b_1(i+1,x) - b_1(i,x) - 1$$

$$x = \sum_{i=0}^{l_1(x)} 2^{a_1(i,x)} - 1$$

Dowód lematu 6.3 Wystarczy zauważyć, że $d(x) = l_1(x)$

$$kod(x,r) = rm(4, a_1(r,x))$$

$$A(x,r) = qt(4, a_1(r,x))$$

są poszukiwanymi funkcjami.

Oznaczenia Od tej pory używać będziemy tej numeracji i oznaczenia $\mu(e) = P_e$, czyli P_e jest programem, który w tej numeracji otrzymał numer e. Ustalmy $n \geqslant 1$.

1. Wtedy $\phi_e^{(n)}$ jest n-argumentową funkcją obliczaną przez ten program. Będziemy mówić, że funkcja obliczalna ma numer e.

2. Zbiór $W_e^{(n)}$ jest dziedziną tej funkcji:

$$W_e^{(n)} = \text{Dom } \phi_e^{(n)} = \{\underline{x} \in N^n; P_e(\underline{x}) \downarrow\}.$$

3. $E_e^{(n)}$ jest zbiorem wartości tej funkcji:

$$E_e^{(n)} = \text{Ran } \phi_e^{(n)} = \{y \in N; \exists_{\underline{x} \in N^n} P_e(\underline{x}) \downarrow y\}.$$

4. **Jeśli nie będzie pojawiał się indeks górny n należy przyjąć, że jest równy 1**.

Nasuwa się wiele pytań: Czy każdy program ma dokładnie jeden numer? A funkcja obliczalna? Czy dla danej funkcji obliczalnej f istnieje procedura rozsztrzygająca, które liczby naturalne są jej numerami? itp. Ponieważ zbiór funkcji obliczalnych ustalonej argumentowości jest przeliczalny jasne jest, że istnieją funkcje, które nie są obliczalne. Co ciekawsze "sensownie" definiowane funkcje są nieobliczalne.

6.1 Metoda diagonalizacji

Twierdzenie 6.4 *Istnieje funkcja totalna $f : N \mapsto N$, która nie jest obliczalna.*

Dowód Niech $\phi_n, n \in N$ będzie ciągiem wszystkich funkcji obliczalnych jednoargumentowych. Rozważmy funkcję:

$$f(n) = \begin{cases} \phi_n(n) + 1 & n \in W_n \\ 0 & n \notin W_n \end{cases}$$

Przypuśćmy, że jest ona obliczalna. Niech e będzie numerem programu ją obliczającego, czyli $f = \phi_e$. Wtedy $f(e) = \phi_e(e)$, ale $e \in W_e$, bo f jest totalna, więc z definicji f otrzymujemy $f(e) = \phi_e(e) + 1$, co daje sprzeczność.

Udwodnimy dwa twierdzenia: o parametryzacji i funkcji uniwersalnej, które dadzą matematyczne narzędzia do odpowiedzi na powyższe pytania. Należy zaznaczyć, że każda formalizacja pojęcia algorytmu powinna mieć te własności (efektywna numeracja algorytmów-programów, tw. o parametryzacji i funkcji uniwersalnej).

7 Twierdzenie o parametryzacji

Twierdzenie o parametryzacji pozwoli m.in. odpowiedzieć na pytanie: czy dla danej funkcji $f \in C_2$ o numerze e można efektywnie wyznaczyć dla dowolnego parametru $a \in N$ numer funkcji $f(a, \_)$?

Twierdzenie to jest nieco ogólniejsze.

Twierdzenie 7.1 (O parametryzacji, s-m-n) *Dla dowolnych liczb naturalnych $n, m \geqslant 1$ istnieje totalna i obliczalna funkcja $s_n^m : N^{1+m} \longrightarrow N$ taka, że*

$$\forall_{e\in N}\forall_{\underline{a}\in N^m}\ \ \phi_e^{(m+n)}(\underline{a}, \_) \simeq \phi^{(n)}_{s_n^m(e,\underline{a})}(\_)$$

Dowód, szkic Dla prostoty założymy, że $n = m = 1$ i będziemy swobodnie używać tych oznaczeń.
Niech $e, a \in N$. Przez $\Pi_{e,a}$ oznaczymy program powstały z P_e poprzez modyfikacje:

1. na początku umieszczamy ciąg instrukcji T(1,2), Z(1), S(1), ... , S(1); przy czy instrukcja S(1) występuje a-razy;

2. potem następuje tekst programu P_e ze zmodyfikowanymi instrukcjami warunkowymi.

Z konstrukcji wynika, że

$$\phi_e^{(2)}(a, y) \simeq \phi^{(1)}_{\Pi_{e,a}}(y)$$

Pozostaje zauważyć, że $\mu(m) = \Pi_{e,a} \iff s_1^1(e, a) = m$.
Zadanie Sformalizuj powyższy dowód.

Wróćmy do początkowych rozważań.

Wniosek 7.2 *Niech $n, m \geqslant 1$ i $\psi \in \mathcal{C}_{n+m}$ będzie dowolną funkcją obliczalną $n + m$-argumentową. Wtedy istnieje totalna funkcja obliczalna $f \in \mathcal{C}_m$ taka, że*

$$\forall_{\underline{a}\in N^m}\ \ \psi(\underline{a}, \_) \simeq \phi^{(n)}_{f(\underline{a})}(\_)$$

Dowód Niech e będzie jakimkolwiek numerem ψ. Za f wystarczy wziąć $s_n^m(e, \_)$./

8 Programy i funkcje uniwersalne

Niech $n > 0$ będzie ustaloną liczbą naturalną.

Definicja 8.1 *Funkcją uniwersalną dla n-argumentowych funkcji RAM-obliczalnych nazywamy funkcję $\Psi_U^{(n)} : N^{n+1} \longrightarrow N$ taką, że:*

$$\forall_{e\in N}\forall_{\underline{x}\in N^n}\Psi_U^{(n)}(e, \underline{x}) \simeq \phi_e^{(n)}(\underline{x})$$

Twierdzenie 8.2 (O funkcjach uniwersalnych) *Każda funkcja uniwersalna jest częściowo-rekurencyjna, a więc i RAM-obliczalna.*

Definicja 8.3 *Każdy program obliczający funkcję uniwersalną nazywamy* **programem uniwersalnym**.

Dowód Niech e będzie dowolną liczbą naturalną. Niech $\sigma_n(e, \underline{x}, k)$ będzie kodem konfiguracji uzyskanej po k krokach obliczeń P_e na $\underline{x}$. Przyjmować będziemy, że konfiguracją następną po końcowej jest ona sama. Wtedy $(\sigma_n(e, \underline{x}, k))_0$ jest zawartością licznika rozkazów, natomiast $(\sigma_n(e, \underline{x}, k))_1$ zawartocią rejestru zerowego po k krokach obliczeń P_e .

Inaczej, funkcja $\sigma_n(e, \underline{x}, k)$ definiowana jest przez rekursję.

$$\sigma_n(e, \underline{x}, 0) = cd(cod_n(\underline{x}))$$

$$\sigma_n(e, \underline{x}, k{+}1) = \begin{cases} \overline{\delta}(< kod(e,l), A(e,l) >, \sigma_n(e, \underline{x}, k)) & l = (\sigma_n(e, \underline{x}, k))_0 < d(e)+1 \\ \sigma_n(e, \underline{x}, k) & l = (\sigma_n(e, \underline{x}, k))_0 \geqslant d(e)+1 \end{cases}$$

Zadanie. Wskaż schemat rekursji użyty w definicji funkcji $\sigma_n(e, \underline{x}, k)$.

Wtedy funkcje $c_n(e, \underline{x}, k) = (\sigma_n(e, \underline{x}, k))_1$ $j_n(e, \underline{x}, k) = (\sigma_n(e, \underline{x}, k))_0$ są prymitywnie rekurencyjne i opisują zawartość rejestru 0 (c!) i licznik rozkazów (j) po k krokach obliczeń P_e. Wtedy

$$\Psi_U^{(n)}(e, \underline{x}) \simeq c_n(e, \underline{x}, \mu\, k\ (j_n(e, \underline{x}, k) \geqslant d(e)+1)).$$

co oznacza, że funkcja uniwersalna jest częściowo rekurencyjna.

Uwaga Czy ten dowód nie wydaje Ci się znajomy? A powinien!

8.1 Konsekwencje

Wydaje się, że tw. o parametryzacji i funkcji uniwersalnej są niezbyt istotne w rozważaniach dotyczących funkcji obliczalnych. Nic bardziej mylnego! Przykładowo pokażemy, że pewne ważne problemy są (nie)rozstrzygalne.

Wniosek 8.4 *Dla dowolnego $n \geqslant 1$ następujące predykaty są rozstrzygalne:*

1. *$S_n(e, \underline{x}, y, t) \equiv$ program o nr e na danej $\underline{x}$ zatrzymuje się w co najwyżej t krokach zwracając y;*

2. $H_n(e, \underline{x}, t) \equiv$ *program o nr e na danej* $\underline{x}$ *zatrzymuje się w co najwyżej t krokach;*

Dowód Nieformalne uzasadnienie jest bardzo proste: konkretny program (uniwersalny) można na danych $e, \underline{x}$ śledzić przez czas t aby ustalić, czy się zatrzymał i co obliczył. Formalnie:

$$S_n(e, \underline{x}, y, t) \equiv (\sigma_n(e, \underline{x}, t))_0 \geqslant d(e) + 1 \wedge (\sigma_n(e, \underline{x}, t))_1 = y$$

$$H_n(e, \underline{x}, t) \equiv (\sigma_n(e, \underline{x}, t))_0 \geqslant d(e) + 1$$

Inną konsekwencją tw. o funkcjach uniwersalnych jest

Wniosek 8.5 (Twierdzenie Kleene o postaci normalnej) *Istnieje totalna funkcja obliczalna* $U(x)$ *i dla każdego* $n \geqslant 1$ *predykaty* $T_n(e, \underline{x}, z)$ *takie, że:*

1. $\phi_e^{(n)}(\underline{x})$ *określone* $\iff \exists_z T_n(e, \underline{x}, z)$*;*
2. $\phi_e^{(n)}(\underline{x}) \simeq U(\mu\ z\ T_n(e, \underline{x}, z))$.

Dowód $T_n(e, \underline{x}, z) = S_n(e, \underline{x}, (z)_0, (z)_1)$ i $U(x) = (x)_0$

Na deser najważniejsze Twierdzenie

Twierdzenie 8.6 *Problem "*ϕ_x *jest totalna" jest nierozstrzygalny.*

Dowód(Nie wprost) Przypuśćmy, że funkcja

$$c(x) = \begin{cases} 1 & \phi_x \text{ totalna} \\ 0 & \phi_x \text{ nie jest totalna} \end{cases}$$

jest obliczalna. Wtedy obliczalna jest funkcja

$$t(x, y) = \begin{cases} \Psi_U(x, y) & c(x) = 1 \\ 0 & c(x) = 0 \end{cases}$$

Z określenia każda funkcja totalna $f \in C_1$ jest postaci $t(e, \_)$ dla pewnego e. Rozważmy fukcję $g(x) = t(x, x)+1$, która oczywiście jest totalna i obliczalna. Przypuśćmy, że $g(x) = t(i, x)$; wtedy $g(i) = t(i, i)$ (z definicji t) i $g(i) = t(i, i) + 1$ (z definicji g), co daje sprzeczność.

9 Zbiory rekurencyjne

9.1 Podstawowe własności

Przypomnijmy definicję

Definicja 9.1 *Niech $n \geqslant 1$ i $A \subseteq N^n$ będzie dowolnym zbiorem a $c_A : N^n \mapsto$ n jego funkcją charakterystyczną:*

$$c_A(\underline{x}) = \begin{cases} 1 & \underline{x} \in A \\ 0 & \underline{x} \notin A \end{cases}$$

Mówimy, że A jest rekurencyjny jeśli $c_A \in \mathcal{C}_n$. Wtedy problem (predykat) $\underline{x} \in A$ jest rozstrzygalny. Jeśli $c_A \notin \mathcal{C}_n$, to problem $\underline{x} \in A$ jest nierozstrzygalny.

Twierdzenie 9.2 *Dla ustalonego $n \geqslant 1$ zbiór wszystkich podzbiorów rekurencyjnych $A \subseteq N^n$ jest zamknięty na operacje teoriomnogościowe.*

Dowód Oczywiście $\emptyset, N^n$ są rekurencyjne. Niech $A, B \subseteq N^n$ będą rekurencyjne. Wtedy wobec wcześniejszych twierdzeń wzory

$$c_{N^n \setminus A}(\underline{x}) = 1 \dot{-} c_A(\underline{x}) \text{ , } c_{A \cup B}(\underline{x}) = max(c_A(\underline{x}), c_B(\underline{x})) \text{ i } c_{A \cap B}(\underline{x}) = c_A(\underline{x}) * c_B(\underline{x})$$

dowodzą obliczalności odpowiednich funkcji.

Zwykle przez zbiór rekurencyjny rozumie się podzbiór N. Rozdział ten poświęcimy zbiorom, które nie są rekurencyjne. Pokażemy że wiele ważnych problemów w Teorii Obliczalności jest nierozstrzygalnych. Często procedurę obliczającą funkcję c_A nazywa się **rozstrzygającą**.

9.2 Problemy nierozstrzygalne w Teorii Obliczalności

Twierdzenie 9.3 *Problem "$x \in W_x$" jest nierozstrzygalny czyli zbiór $K = \{x \in N : x \in W_x\}$ nie jest rekurencyjny.*

Dowód Przypuśćmy, że c_K jest obliczalna. Wtedy wzór

$$g(x) = \begin{cases} 1 & x \notin W_x \\ \infty & x \in W_x \end{cases}$$

definiuje obliczalną funkcję. Niech m będzie numerem jakiegokolwiek programu obliczającego g, czyli $\phi_m = g$. Wtedy

$$m \in W_m \longleftrightarrow m \in Domg \longleftrightarrow m \notin W_m$$

i mamy sprzeczność.
Powyższe twierdzenie nie mówi, że dla konkretnego a nie umiemy rozstrzygnąć, czy $a \in Dom\ \phi_a$. Czasem jest to bardzo proste zadanie. Twierdzenie ma ogólny charakter: nie istnieje algorytm rozstrzygający dla każdego $x \in N$, czy $x \in Dom\ \phi_x$.

Wniosek 9.4 *Istnieje obliczalna funkcja jednoargumentowa h, dla której problemy "$x \in Dom\ h$", "$x \in Ran\ h$" są nierozstrzygalne.*

Dowód Rozważmy funkcję $h(x) = x * 1(\Psi_U(x, x))$, która jest obliczalna. Wtedy;

$$x \in Dom\ h \longleftrightarrow x \in W_x \longleftrightarrow x \in Ran\ h$$

Twierdzenie 9.5 (Problem Stopu jest nierozstrzygalny) *Problem "$\phi_x(y)$ jest określone", równoważnie "$P_x(y)\downarrow$" czy "$y \in W_x$", jest nierozstrzygalny.*

Dowód nieformalny jest bardzo prosty: gdyby problem "$y \in W_x$" był rozstrzygalny, to problem "$x \in W_x$" również, co przeczy twierdzeniu 9.3. Dokładniej funkcja

$$g(x, y) = \begin{cases} 1 & y \in W_x \\ 0 & y \notin W_x \end{cases}$$

jest funkcją charakterystyczną tego problemu. Stosując proste podstawienie otrzymujemy: $c_K(x) = g(x, x)$, czyli funkcję charakterystyczną zbioru K. Gdyby więc $g \in \mathcal{C}_2$, to $c_K \in \mathcal{C}_1$.

Powyższe twierdzenie jest ważne: wiele problemów da się sprowadzić podobną techniką do problemu stopu. Zauważmy, że funkcja $1(\Psi_U(x, y))$ jest obliczalna i jest częściową funkcją charakterystyczną problemu stopu; problem ten jest więc **częściowo rozstrzygalny**. Zauważmy, że wniosek 9.4 mówi, że istnieją programy, dla których problem stopu jest nierozstrzygalny, więc tym bardziej musi zachodzić twierdzenie 9.5.

Zanim sformułujemy ogólne twierdzenie dotyczące wszystkich problemów w Teorii Obliczalności jeszcze przykład

Twierdzenie 9.6 (Problem poprawności jest nierozstrzygalny) *Problem "ϕ_x jest funkcją tożsamościowo równą zero" jest nierozstrzygalny.*

Dowód Rozważmy funkcję

$$g(x,y) = \begin{cases} 0 & x \in W_x \\ \infty & x \notin W_x \end{cases}$$

która jest obliczalna, gdyż $g(x,y) = 0(\Psi_U(x,x))$. Niech $\{g_x, x \in N\}$ będzie rodziną funkcji takich, że $g_x(y) = g(x,y)$ dla wszelkich $x, y \in N$. Wtedy:

$$\forall_{x \in N}\ g_x = 0 \text{ wtw } x \in W_x.$$

Zastosujmy do funkcji g (wniosek z) twierdzenie o parametryzacji; niech $k \in \mathcal{C}_1$ będzie totalna i taka, że

$$\forall_{x,y \in N}\ g(x,y) = \phi_{k(x)}(y) \text{ tj. } \phi_{k(x)} = g_x.$$

Wtedy

$$\forall_{x \in N}\ x \in W_x \longleftrightarrow \phi_{k(x)} = 0.$$

Oznacza to, że **problem "$x \in W_x$" jest uproszczeniem problemu "$\phi_x = 0$" i funkcja k jest efektywną procedurą sprowadzającą problem "$x \in W_x$" do "$\phi_x = 0$"**. Gdyby funkcja f charakterystyczna problemu "$\phi_x = 0$" była obliczalna, to wobec zależności $c_K(x) = f(k(x))$, problem "$x \in W_x$" byłby rozstrzygalny, a to przeczy twierdzeniu 9.3.
Zadanie Wykaż, że problemy:

1. "$\phi_x = \phi_y$";
2. "$\phi_x = f$", gdzie $f \in \mathcal{C}_1$ jest dowolną funkcją;

nie są rozstrzygalne.

9.3 Twierdzenie Rice'a

Podsumujmy ostatnie rozważania.

Definicja 9.7 *Niech A, B będą podzbiorami N. Niech $f \in \mathcal{C}_1$ będzie totalna i taka, że*

$$\forall_{x \in N} x \in A \longleftrightarrow f(x) \in B.$$

Wtedy f jest sprowadzeniem problemu "$x \in A$" do problemu "$x \in B$". Wtedy problem "$x \in A$" jest sprowadzalny do "$x \in B$".

Najważniejsza w dowodzie ostatniego twierdzenia jest obserwacja:

Niech f będzie sprowadzeniem "$x \in A$" do "$x \in B$" Jeśli funkcja c_B jest obliczalna, to wobec równości $c_A(x) = c_B(f(x))$ obliczalna jest funkcja c_A. Wtedy z nierozstrzygalności problemu "$x \in A$" wynika nierozstrzygalność problemu "$x \in B$".

Twierdzenie Rice'a mówi, że wszystkie nietrywialne problemy w teorii obliczalnych funkcji jednoargumentowych są nierozstrzygalne.

Twierdzenie 9.8 (Rice) *Niech $\mathcal{B}$ będzie właściwym i niepustym podzbiorem $\mathcal{C}_1$. Wtedy problem "$\phi_x \in \mathcal{B}$" jest nierozstrzygalny; bardziej formalnie zbiór $B = \{x \in N; \phi_x \in \mathcal{B}\}$ nie jest rekurencyjny.*

Ponieważ zbiory N i $\emptyset$ są rekurencyjne otrzymujemy inne sformułowanie twierdzenia: Problem "$\phi_x \in \mathcal{B}$" jest rozstrzygalny wtw, gdy $\mathcal{B} = \emptyset$ lub $\mathcal{B} = \mathcal{C}_1$.
Dowód Niech $f_\emptyset$ będzie funkcją o pustej dziedzinie. Funkcja ta jest obliczalna. Załóżmy, że funkcja ta nie należy do $\mathcal{B}$. Jeśli tak jest, to prowadzimy dowód dla zbioru $\mathcal{C}_1 \backslash \mathcal{B}$ i korzystamy z banalnej obserwacji, że problem "$\phi_x \in \mathcal{B}$" jest rozstrzygalny wtw, gdy problem "$\phi_x \notin \mathcal{B}$" jest rozstrzygalny. Niech $f \in \mathcal{B}$ będzie ustaloną funkcją.
Rozważmy funkcję

$$g(x,y) = \begin{cases} f(y) & x \in W_x \\ \infty & x \notin W_x \end{cases} .$$

Ponieważ $g(x,y) = f(p_2^2(\Psi_U(x,x),y))$ funkcja g jest obliczalna i można do niej stosować wniosek z tw. o parametryzacji. Niech $k \in \mathcal{C}_1$ będzie totalna i taka, że:

$$\forall_{x,y \in N} g(x,y) = \phi_{k(x)}(y)$$

Wtedy $x \in W_x$ pociąga, że $\phi_{k(x)} = f \in \mathcal{B}$; a $x \notin W_x$ pociąga, że $\phi_{k(x)} = f_\emptyset \notin \mathcal{B}$, czyli

$$\forall_{x \in N} x \in W_x \longleftrightarrow \phi_{k(x)} \in \mathcal{B}.$$

Problem "$\phi_x \in \mathcal{B}$" nie może więc być rozstrzygalny.

Wykorzystamy tw. Rice'a aby udowonić następujące twierdzenie:

Twierdzenie 9.9 *Niech $c \in N$ będzie ustaloną liczbą naturalną. Następujące problemy są nierozstrzygalne:*

1. *Problem wejścia: "$c \in W_x$";*

2. *Problem wyjścia: "$c \in E_x$";*

Dowód (1) Weźmy zbiór $\mathcal{B} = \{f \in \mathcal{C}_1; c \in Dom\ f\}$. Oczywiście $B = \{x \in N; \phi_x \in \mathcal{B}\} = \{x \in N; c \in W_x\}$. Wystarczy więc wykazać, że $\mathcal{B}$ jest niepustym i właściwym podzbiorem $\mathcal{C}_1$. Faktycznie, funkcja $c(x)$ tożsamościowo równa c należy do zbioru $\mathcal{B}$, więc jest on niepusty; funkcja pusta $f_\emptyset$ do niego nie należy, więc jest on właściwym podzbiorem.
(2) Podobnie, niech $\mathcal{B} = \{f \in \mathcal{C}_1; c \in Ran\ f\}$. Wystarczy zauważyć, że $c(x) \in \mathcal{B}$ oraz $f_\emptyset \notin \mathcal{B}$.

10 Zbiory rekurencyjnie przeliczalne

10.1 Podstawowe własności

Definicja 10.1 *Niech $n \geqslant 1$ i $A \subseteq N^n$ będzie dowolnym zbiorem a $cc_A : N^n \mapsto N$ jego częściową funkcją charakterystyczną:*

$$cc_A(\underline{x}) = \begin{cases} 1 & \underline{x} \in A \\ \infty & \underline{x} \notin A \end{cases}$$

Mówimy, że A jest rekurencyjnie przeliczalny (r.e.)jeśli $cc_A \in \mathcal{C}_n$. Wtedy problem (predykat) $\underline{x} \in A$ jest częściowo rozstrzygalny.

Wiemy już, że problem "$x \in W_x$" jest nierozstrzygalny, ale jest częściowo rozstrzygalny. Każdy algorytm obliczający cc_A nazywamy częściową procedurą rozstrzygającą lub sprawdzającą dla A.

Przykłady

1. Problem stopu: zbiór $\{(x, y) \in N^2; y \in W_x\}$ jest r.e. bo funkcja $1(\Psi_U(x, y))$ jest częściową funkcją charakterystyczną tego zbioru.

2. Każdy zbiór rekurencyjny jest r.e. z definicji (potraktuj ją jako definicję przez przypadki).

3. Podobnie dla dowolnej funkcji $g \in \mathcal{C}_n$ ($n \geqslant 1$) problem stopu: "$\underline{x} \in Dom\ g$" jest częściowo rozstrzygalny, bo $1(g(\underline{x}))$ jest częściową funkcją charakterystyczną tego problemu.

4. Problem "$x \notin W_x$" nie jest częściowo rozstrzygalny. Faktycznie, niech $\overline{K} = \{x \in N; x \notin W_x\}$, wtedy $x \in Dom\ cc_{\overline{K}} \longleftrightarrow x \notin W_x$, czyli $cc_{\overline{K}}$ różni się od ϕ_x w punkcie x, więc $cc_{\overline{K}}$ nie jest obliczalna.

Bezpośarednio z definicji i przykładu 3. otrzymujemy:

Twierdzenie 10.2 *Zbiór $A \subseteq N^n$ jest r.e. wtw, gdy istnieje obliczalna funkcja n-argumentowa g taka, że $A = Dom\ g$.*

Przykład Rozważmy zbiór $L(x,y) = \{(x,y) \in N^2; x^2 = y\}$. Oczywiście jest on rekurencyjny. Niech $M(y) \longleftrightarrow \exists_{x \in N} L(x,y) \longleftrightarrow \exists_{x \in N} x^2 = y$. Zbiór ten jest r.e., gdyż $cc_M(y) = 1(\mu x(c_L(x,y) = 1))$, jest też rekurencyjny $c_M(y) = 1(\mu x \leqslant y(c_L(x,y) = 1))$. Pierwsze rozumowanie jest ogóle (następne tw.), drugie wykorzystuje matematyczną obserwację: pierwiastek z liczby naturalnej nie jest od niej większy.

Twierdzenie 10.3 *Zbiór $M \subseteq N^n$ jest r.e. wtw, gdy istnieje rekurencyjny zbiór $L \subseteq N^{n+1}$ taki, że $M(\underline{x}) \longleftrightarrow \exists_{y \in N} L(\underline{x}, y)$.*

Dowód Jeśli zbiór $L \subseteq N^{n+1}$ jest rekurencyjny, to f-cja $g(\underline{x}) = \mu y(c_L(\underline{x}, y) = 1)$ jest obliczalna, a wtedy $\underline{x} \in M \longleftrightarrow \underline{x} \in Dom\ g$.
Odwrotnie, niech P będzie programem obliczającym cc_M o numerze e. Rozważmy zbiór L taki, że

$$(\underline{x}, y) \in L \longleftrightarrow P(\underline{x}) \downarrow \text{ w co najwyżej y krokach } \longleftrightarrow H_n(e, \underline{x}, y).$$

Z twierdzenia o funkcji uniwersalnej wiemy, że zbiór ten jest rekurencyjny. Ponieważ $\underline{x} \in M \longleftrightarrow \exists_{y \in N}(\underline{x}, y) \in L$ otrzymujemy tezę.

Z powyższego twierdzenia wynika ważny

Wniosek 10.4 *Jeśli zbiór $L \subseteq N^{n+1}$ jest r.e., to zbiór $M \subseteq N^n$ taki, że $\underline{x} \in M \longleftrightarrow \exists_{y \in N}(\underline{x}, y) \in L$ jest r.e.*

Dowód Niech $L_1 \subseteq N^{n+2}$ będzie rekurencyny i taki, że $(\underline{x}, y) \in L \longleftrightarrow \exists_{z \in N}(\underline{x}, y, z) \in L_1$. Wtedy $\underline{x} \in M \longleftrightarrow \exists_{y,z \in N}(\underline{x}, y, z) \in L_1$. Wtedy $g(\underline{x}) = \mu v(c_{L_1}(\underline{x}, (v)_0, (v)_1) = 1))$ jest funkcją obliczalną o dziedzinie M.

Wniosek 10.5 *Jeśli zbiór $L \subseteq N^2$ jest rekurencyjny, to zbiór $M \subseteq N^2$ taki, że $(x,y) \in M \longleftrightarrow \exists_{z \leqslant y}(x,z) \in L$ jest rekurencyjny.*

Chociaż z twierdzenia 10.3 i istnienia zbiorów r.e., które nie są rekurencyjne wynika, że istnieją rozstrzygalne predykaty $L(\underline{x}, y)$, dla których predykat $\exists_{y \in N}(\underline{x}, y) \in L$ nie jest rozstrzygalny, wydaje się, że trudno je znaleźć w świecie arytmetyki liczb naturalnych i predykatów diofantycznych. Przekonanie to jest mylne, gdyż istota dowodu Matijasevica pokazującego, że X problem Hilberta jest nierosztrzygalny polega na wykazaniu, że każdy predykat częściowo rozstrzygalny jest diofantyczny.

Na zakończenie tego podrozdziału zajmiemy się własnościami zbiorów rekurencyjnych.

Twierdzenie 10.6 *Zbiór $M \subseteq N^n$ jest rekurencyjny wtw, gdy M i $N^n \backslash M$ są rekurencyjnie przeliczalne.*

Dowód Wiemy, że jeśli M jest rekurencyjny, to $N^n \backslash M$ również; a każdy zbiór rekurencyjny jest r.e.
Załóżmy więc, że oba te zbiory są r.e.; niech L_1, L_2 będą rekurencyjnymi podzbiorami N^{n+1} takimi, że $M(\underline{x}) \longleftrightarrow \exists_y L_1(\underline{x}, y)$ i $\neg M(\underline{x}) \longleftrightarrow \exists_y L_2(\underline{x}, y)$. Wtedy funkcja

$$f(\underline{x}) = \mu y(L_1(\underline{x}, y) \vee L_2(\underline{x}, y))$$

jest obliczalna i **totalna**, a $\underline{x} \in M \longleftrightarrow (\underline{x}, f(\underline{x})) \in L_1$, więc zbiór M jest rekurencyjny.

Wiele zagadnień rozważanych w ostatnim rozdziale będzie miało raczej charakter decyzyjny. Odnotujmy więc następne

Twierdzenie 10.7 *Niech $f : N^n \mapsto N$ będzie funkcją (niekoniecznie totalną). Funkcja f jest obliczalna wtw, gdy problem "$f(\underline{x}) = y$" jest częściowo rozstrzygalny.*

Dowód Jeśli $f = \phi_e^n$, czyli f jest obliczana przez program o numerze e, to $\forall_{\underline{x} \in N^n} \forall_{y \in N} f(\underline{x}) = y \longleftrightarrow \exists_{t \in N} S_n(e, \underline{x}, y, t)$, gdzie $S_n(e, \underline{x}, y, t)$ jest rozstrzygalnym predykatem takim, że $S_n(e, \underline{x}, y, t) \longleftrightarrow P_e(\underline{x}) \downarrow y$ w co najwyżej t krokach.
Niech $L(\underline{x}, y, t)$ będzie rozstrzygalnym predykatem (tw.10.3) takim, że $f(\underline{x}) = y \longleftrightarrow \exists_{t \in N} L(\underline{x}, y, t)$. Wtedy f można opisać wzorem:

$$f(\underline{x}) = \mu y(L(\underline{x}, y, \mu t(L(\underline{x}, y, t)))).$$

Następne twierdzenie wyjaśnia dlaczego zbiory liczb naturalnych, których częściowa funkcja charakterystyczna jest obliczalna nazywa się rekurencyjnie przeliczalnymi.

Twierdzenie 10.8 *Niech $A \subseteq N$. następujące warunki są równoważne:*

1. *A jest r.e.;*
2. *$A = \emptyset$ albo istnieje totalna i obliczalna funkcja $f \in \mathcal{C}_1$ taka, że A jest zbiorem jej wartości: $Ran\ f = A$;*
3. *A jest dziedziną pewnej funkcji obliczalnej jednoargumentowej :$A = W_x$ dla pewnego $x \in N$.*

Dowód Wykazaliśmy równoważność warunków (1) i (3).
(3)→(2) Załóżmy, że A jest niepusty i $A = W_e$. Rozważmy funkcję:

$$g(x,t) = \begin{cases} x & H_1(e,x,t) \\ a & \neg H_1(e,x,t) \end{cases}$$

gdzie $a \in A$ jest ustalonym elementem. Wtedy g jest obliczalna (df. typu if then else). Niech $h(z) = g((z)_0, (z)_1)$. Wtedy h też jest obliczalna i $Ran\ h = Ran\ g = A$.
(2)→(1) Niech $f \in \mathcal{C}_1$ będzie totalna i taka, że $Ran\ f = A$. Predykat $R(x,y) \leftrightarrow f(x) = y$ jest (częściowo) rozstrzygalny a $y \in A \leftrightarrow \exists_{x \in N} R(x,y)$, więc problem $y \in A$ jest częściowo rozstrzygalny.

Zadanie Wykaż, że jeśli $A, B \subseteq N$ są r.e., to $A \cap B$ i $A \cup B$ są r.e.
Rozwiązanie Jeśli $A = W_x$ i $B = W_y$, to zbiór $W_x \cap W_y$ jest dziedziną funkcji $\phi_x * \phi_y$;
Jeśli $A = E_x$ i $B = E_y$ oraz ϕ_x, ϕ_y są totalne, to wzór

$$h(t) = \begin{cases} \phi_x(z) & t = 2z \\ \phi_y(z) & t = 2z+1 \end{cases}$$

definiuje totalną i obliczalną funkcję taką, że $Ran\ h = A \cup B$.

Twierdzenie 10.9 *Nieskończony zbiór $A \subseteq N$ jest rekurencyjny wtw, gdy jest zbiorem wartości totalnej, rosnącej funkcji $f \in \mathcal{C}_1$.*

Dowód Załóżmy, że c_A jest obliczalna. Wtedy układ

$$f(0) = \mu x(c_A(x) = 1)$$

$$f(n+1) = \mu x(c_A(x) = 1 \wedge x > f(n))$$

definiuje przez rekursję funkcję, której zbiorem wartości jest A. Z określenia f jest totalna i rosnąca.
Załóżmy, że $A = Ran\ f$ dla totalnej i obliczalnej funkcji f. Ponieważ f jest rosnąca łatwo pokazać, że $n \leqslant f(n)$ dla każdego n. Jeśli więc $y \in N$ jest wartością funkcji f na pewnym n, to $n \leqslant y$. Stąd

$$y \in A \leftrightarrow y \in Ran\ f \leftrightarrow \exists_{n \leqslant y} f(n) = y$$

a ostatni predykat jest **rozstrzygalny**.

10.2 Twierdzenie Rice'a-Shapiro

Twierdzenie 10.10 (Rice'a-Shapiro) *Niech $\mathcal{A} \subseteq \mathcal{C}_1$ będzie takim zbiorem funkcji RAM-obliczalnych, że zbiór indeksów $A = \{x \in N : \phi_x \in \mathcal{A}\}$ jest rekurencyjnie przeliczalny. Wtedy dla dowolnej funkcji $f \in \mathcal{C}_1$ następujące warunki są równoważne:*

1. *$f \in \mathcal{A}$;*
2. *Istnieje skończona funkcja $\Theta \subseteq f$ taka, że $\Theta \in \mathcal{A}$.*

UWAGA Funkcję traktujemy tutaj jak (binarną) relację, czyli wiadomo, co to znaczy, że zbiór Θ jest skończony i ma sens inkluzja $\Theta \subseteq f$.

Dowód (1) pociąga (2): nie wprost. Niech $f \in \mathcal{A}$ będzie taką funkcją, że żadna skończona funkcja $\Theta \subseteq f$ nie należy do $\mathcal{A}$. Niech $P = P_e$ będzie programem obliczającym częściową funkcję charakterystyczną zbioru $K = \{x \in N; x \in W_x\}$. Rozważmy funkcję

$$g(z,t) = \begin{cases} f(t) & \neg H(e,z,t) \\ \infty & H(e,z,t) \end{cases}$$

Oczywiście g jest obliczalna (predykat $H(e,z,t)$ jest rozstrzygalny, f i funkcja pusta są obliczalne), więc możemy stosować tw. o paramatryzacji. Niech $s \in$

$\mathcal{C}_1$ będzie totalna i taka, że $g(z,t) = \phi_{s(z)}(t)$ dla wszystkich z, t. Z określenia $\phi_{s(z)} \subseteq f$, ale $z \in K$ pociąga, że $\phi_{s(z)}$ jest skończona, więc nie należy do $\mathcal{A}$. Gdy $z \notin K$, $\phi_{s(z)} = f \in \mathcal{A}$. Funkcja s pokazuje, że problem $z \in A$ jest trudniejszy, niż $z \notin K : \forall_z z \notin K \Leftrightarrow s(z) \in A$, więc ten pierwszy nie może być r.e., co daje sprzeczność z założeniem. (2) pociąga (1): nie wprost. Przypuśćmy, że istnieje obliczalna funkcja $f \notin \mathcal{A}$ dla której $\Theta \in \mathcal{A}$ dla pewnej skończonej funkcji $\Theta \subseteq F$. Postąpimy podobnie. Niech

$$g(z,t) = \begin{cases} f(t) & t \in Dom\ \Theta \vee z \in K \\ \infty & \text{w przeciwnym wypadku} \end{cases}$$

Najpierw wykażemy, że g jest obliczalna. Oczywiście zbiór $Dom\ \theta$ jest skończony, a więc i rekurencyjny. Wtedy zbiory $\{(t,z); t \in Dom\ \Theta\}, \{(t,z); z \in K\}$ są r.e. i zbiór $B = \{(t,z); t \in Dom\ \Theta \vee z \in K\}$ będący ich sumą też jest r.e.. Na konic zauważmy, że $g(z,t) = f(t) * cc_B(t,z)$. Niech z s-m-n twierdzenia s będzie totalna i obliczalna i taka, że $g(z,t) = \phi_{s(z)}(t)$ dla wszystkich z, t. Wtedy $z \in K$ pociąga, że $\phi_{s(z)}(t) = f \notin \mathcal{A}$; a $z \notin K$ pociąga $\phi_{s(z)}(t) = \Theta \in \mathcal{A}$. Otrzymujemy ponownie $\forall_z z \notin K \Leftrightarrow s(z) \in A$.

Wykorzystując twierdzenie Rice'a-Shapiro łatwo wykazać, że problem totalności :"ϕ_x jest totalna" nie jest nawet częściowo rozstrzygalny, bo do zbioru $\mathcal{A}$ wszystkich funkcji totalnych i obliczalnych nie należą funkcje skończone. Podobnie do zbioru $\mathcal{C}_1 \backslash \mathcal{A}$ należą wszystkie funkcje skończone (w szczególności funkcja pusta), a zbiór ten jest różny od $\mathcal{C}_1$, więc problem "ϕ_x nie jest totalna" również nie jest częściowo rozsztrzygalny.

11 Problemy niepodatne

W ostatnim rozdziale omówimy dwie podstawowe klasy czasowej złożoności obliczeniowej P oraz NP i sformułujemy jeden z siedmiu problemów milenijnych.

11.1 Sprowadzalność problemu ścieżki Hamiltona do problemu spełnialności

Dla digrafu G=(V,E) o n wierzchołkach ponumerowanych kolejnymi liczbami naturalnymi ze zbioru $\{1, \ldots, n\}$ zdefiniujemy wyrażenie logiczne (funkcję

zdaniową) f_G o n^2 zmiennych x_{ij}, którym nadamy wartość logiczną 1 wtw, gdy j-ty wierzchołek występuje na i-tej pozycji ścieżki Hamiltona.

Wtedy dla każdego digrafu G, G ma ścieżkę Hamiltona wtw, gdy wyrażenie f_G jest spełnialne.

f_G jest koniunkcją wyrażeń logicznych stwierdzających:

1. Każdy wierzchołek j musi się pojawić na ścieżce: $(x_{1j} \vee x_{2j} \vee \ldots \vee x_{nj})$, ale co najwyżej raz: $(\neg x_{kj} \vee \neg x_{ij})$ dla wszystkich $k \neq i$;

2. Jakiś wierzchołek musi być na i-tej pozycji: $(x_{i1} \vee x_{i2} \vee \ldots x_{in})$, ale nie dwa: $(\neg x_{ik} \vee \neg x_{ij})$ dla wszystkich $k \neq j$;

3. Kolejne wierzchołki muszą być połączone krawędzią: dla każdej pary $(j, i) \notin E$ oraz $k \in \{1, \ldots, n-1\}$: $(\neg x_{kj} \vee \neg x_{k+1i})$;

Procedura sprowadzania musi spełniać dodatkowe w-ki. Najogólniej, proces sprowadzania nie może być trudniejszy niż sam problem.

Zakładać będziemy, że języki są rekurencyjne; w związku z tym maszyny Turinga będą zawsze się zatrzymywać i do odróżnienia słów akceptowanych od odrzucanych wykorzystamy stany MT.

11.2 Redukcje

Funkcja $R : X^\star \longrightarrow X^\star$ jest obliczalna przez deterministyczną MT M w pamięci logarytmicznej o ile

1. M jest wielotaśmowa,

2. ma dwie wyróżnione taśmy: wejściową i wyjściową,

3. słowo pojawiające się na każdej z pozostałych taśm (zwanych roboczymi) w czasie obliczeń M na słowie x długości n jest długości $\mathcal{O}(log\ n)$.

Niech L,L' będą językami nad ustalonym alfabetem X.

Definicja 11.1 *Redukcją języka L do L' nazywamy funkcję $R : X^\star \longrightarrow X^\star$, która jest obliczalna przez deterministyczną MT w pamięci logarytmicznej i sprowadzającą L do L':*

$$\forall_{x \in X^\star}\ x \in L \longleftrightarrow\ R(x) \in L'$$

Niech $\mathcal{C}$ będzie klasą złożoności.

Definicja 11.2 *Język $L \in \mathcal{C}$ jest* **zupełny** *dla $\mathcal{C}$, jeśli dla każdego $L' \in \mathcal{C}$ istnieje redukcja L' do L.*

Obecnie zajmiemy się dwiema klasami złżoności: P jest klasą wszyskich języków rozpoznawanych przez deterministyczne maszyny Turinga w czasie wielomianowym; NP jest klasą wszyskich języków rozpoznawanych przez maszyny Turinga w czasie wielomianowym.

Twierdzenie 11.3 *Niech M będzie maszyną Turinga, która jest deterministyczna i operuje pamięcią $O(log\ n)$. Wtedy M ma wielomianową czasową złożoność obliczeniową.*

Twierdzenie 11.4 (COOK (1971)) *Język SAT wyrażeń logicznych, które są spełnialne jest zupełny w klasie NP.*

Ćwiczenie Wykaż, że SAT $\in$ NP.

Dowód polega na zbudowaniu w pamięci logarytmicznej dla dowolnej MT M, operującej wejściowym alfabetem X , o wielomianowej czasowej złożoności obliczeniowej, oraz dowolnego słowa $x \in X^\star$ wyrażenia logicznego R(x), które jest spełnialne wtw, gdy x jest akceptowane przez M.

Załóżmy, że maszyna M mając na wejściu słowo x wykonuje co najwyżej $|x|^k$ ruchów przed zaakceptowaniem lub odrzuceniem słowa.

Ponieważ M jest lewostronnie ograniczona, aby prześledzić działanie M na x wystarczy obserwować tablicę T $|x|^k$ na $|x|^k$, gdzie wiersze odpowiadać będą taśmie M w kolejnych krokach obliczeń: T(i,j) reprezentuje j-tą pozycję na taśmie w i-tym kroku obliczeń. Należy też uwzględnić stany M, więc elementy T nie będą tylko literami ale również trójkami (stan, litera, numer-nastepnej-instrukcji). Jeżli stan jest końcowym, będzie nas tylko interesować, czy jest akceptujący.

Formalnie:

- Γ oznacza zbiór symboli mogących pojawić się w tablicy.
- q oznacza maksymalną ilość instrukcji, które M może wykonać w każdej konfiguracji; zakładamy że definicja M obejmuje numerowanie kolejnymi liczbami naturalnymi wszystkich instrukcji dotyczących danej konfiguracji (stan, litera).

- Obliczenia zatrzymują się dokładnie w $|x|^k - 1$ krokach, czyli wiersz o numerze $|x|^k - 1$ odpowiada konfiguracji końcowej.

Jeżli w wierszu $|x|^k - 1$ pojawia się konfiguracja akceptujaca, to tablicę T nazywamy **akceptującą**.

Oczywiście, słowo x jest akceptowane przez M wtw, gdy istnieje akceptująca tablica obliczeń M na x.

Podobnie jak dla problemu ścieżki Hamiltona, skonstruujemy wyrażenie logiczne, kodujące wszystkie tablice obliczeń M na x, które jest spełnialne dokładnie, gdy jedna z nich jest akceptująca. Wprowadzamy zmienne T_{ijY}, gdzie $i, j \in \{0, 1, \ldots, |x|^k - 1\}, Y \in \Gamma$. R(x) wyraża:

1. dla każdych i,j prawdziwa jest dokładnie jedna zmienna T_{ijY};

$$\bigwedge_{i,j} \bigvee_{Y} (T_{ijY} \wedge \bigwedge_{Z \neq Y} (\neg T_{ijY} \vee \neg T_{ijZ}))$$

2. zbiór $\{T_{0jY}\}$ opisuje konfigurację początkową M z x na wejściu; Niech $x = a_1 a_2 ... a_n$, czyli $n = |x|$

   (a) $T_{00\triangleright}$

   (b) $T_{01Y_1} \vee \ldots \vee T_{01Y_q}$, gdzie Y_i odpowiada za wybór kolejnej instrukcji $i \leqslant q$

   (c) $\bigwedge_{2 \leqslant i \leqslant n} T_{0ia_i}$

   (d) $\bigwedge_{n < i < n^k - 1} T_{0iB}$

3. zbiór $\{T_{ijY}\}$ dla $i = |x|^k - 1$ opisuje konfigurację końcową;

$$\bigvee_{0 < j \leqslant n^k - 1} (\bigvee_{Z \in F} T_{n^k - 1 jZ})$$

   gdzie F jest zbiorem stanów końcowy akceptujących.

4. zbiór $\{T_{i+1jY}\}$ powstaje z $\{T_{ijY}\}$ przez wykonanie instrukcji M; element (i+1,j) w tablicy obliczeń jest jednoznacznie określony przez elementy (i,j-1), (i,j), (i,j+1). Można zdefiniować predykat f(V,X,Y,Z), który mówi, że Z jest na pozycji j, o ile w poprzednim kroku obliczeń V, X, Y były na pozycjach j-1, j, j+1.

$$\bigwedge_{0 < i < n^k - 1} \bigwedge_{0 < j < n^k - 1} (\bigvee_{f(V,X,Y,Z)} T_{ij-1V} \wedge T_{ijX} \wedge T_{ij+1Y} \wedge T_{i+1jZ})$$

Udowodnij, że wygenerowanie wyrażenia R(x) można przeprowadzić w pamięci logarytmicznej.

Definicja 11.5 *Wyrażenie logiczne jest w* **postaci normalnej koniunkcyjnej PNK**, *jeżli jest koniunkcją alternatyw zmiennych lub ich zaprzeczeń.*

Symbolem **PNK** oznaczymy zbiór wszystkich takich wyrażeń, **SAT-PNK** oznacza zbiór wszystkich wyrażeń w PNK, które są spełnialne.

UWAGA Nie wszystkie wyrażenia logiczne rozważane wcześniej były koniunkcjami alternatyw zmiennych zdaniowych lub ich zaprzeczeń, chociaż łatwo je zastąpić równoważnymi w tej postaci.

W istocie można udowodnić

Twierdzenie 11.6 *Język SAT-PNK jest NP-zupełny.*

Na zakończenie przedstawimy trzy NP-zupełne problemy grafowe.

W tym celu rozważymy język SAT-3PNK spełnialnych wyrażeń w PNK, w której każda alternatywa ma co najwyżej trzy elementy.

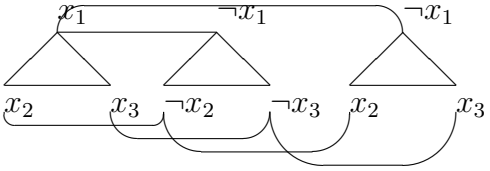
Zadanie Rozważmy wyrażenie logiczne $f(\underline{x}) = x_1 \vee x_2 \vee \ldots \vee x_m$, gdzie $m > 3$ oraz $g(\underline{x}, \underline{y}) = (x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee x_3 \vee y_2) \wedge \ldots \wedge (\neg y_{m-3} \vee x_{m-1} \vee x_m)$. Wykaż, że Każde wartościowania zmiennych $\underline{x}$, dla których f staje się zdaniem prawdziwym, wyznacza wartościowanie zmiennych $\underline{x}, \underline{y}$, dla których g staje się zdaniem prawdziwym oraz każde wartościowanie $\underline{x}, \underline{y}$, przy którym g staje się prawdziwe, jest też wartościowaniem przy którym f staje się prawdziwe.

Twierdzenie 11.7 *Język SAT-3PNK jest NP-zupełny.*

Dowód twierdzenia opiera się o powyższe zadanie, które daje redukcję problemu SAT-PNK do SAT-3PNK i następujący

Lemat 11.8 *Jeżeli R jest redukcją języka L_1 do L_2, a R' redukcją L_2 do L_3, to $R' \circ R$ jest redukcją L_1 do L_3.*

Dowód Ponieważ R, R' są redukcjami otrzymujemy od razu, że $x \in L_1$ wtw, gdy $R'(R(x)) \in L_3$. Istotne jest, czy obliczanie $R'(R(x))$ można przeprowadzić w pamięci logarytmicznej. Niech MR, MR' będą maszynami Turinga, które obliczają R i R' odpowiednio, w pamięci logarytmicznej. M konstruujemy następująco: składamy obie maszyny w jedną trakując taśmę wejściową



Rysunek 1: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

MR jako wejściową M, a taśmę wyjściową MR' jako wyjściową M. Wtedy taśma wyjściowa MR nie może być taśmą roboczą M, bo na niej mogą się pojawiać zbyt długie słowa. Należy zamiast niej użyć taśmy, na której generowane będą numery liter wyjścia MR potrzebne do symulowania instrukcji MR'. Należy zbudować M, która:

1. mając na wejściu x i na tej taśmie i symuluje obliczenia MR aby wypisać i-tą literę $R(x)$,

2. potem wykonuje jedną instrukcję MR'.

Wniosek 11.9 *Niech język $L \in NP$ będzie NP-zupełny. Jeśli istnieje redukcja języka L do języka $L' \in NP$, to L' jest NP- zupełny.*

11.3 Problem zbioru niezależnego

Niech G=(V,E) będzie grafem. Zbiór wierzchołków I nazywamy **niezależnym**, jeśli żadne dwa wierzchołki z I nie są połączone krawędzią. Oczywiście każdy graf ma zbiór niezależny!

Problem ZN Dla grafu G=(V,E) i $k \in N$ rozstrzygnąć, czy G ma zbiór niezależny mocy k.

Twierdzenie 11.10 *Problem zbioru niezależnego jest NP-zupełny.*

Dowód Sprowadzenie języka SAT-3PNK do problemu ZN.
Idea: wyrażeniu $\alpha = \bigwedge_{1 \leqslant i \leqslant m}(\alpha_{i_1} \vee \alpha_{i_2} \vee \alpha_{i_3})$ przyporządkujemy graf o 3m wierzchołkach, który posiada zbiór niezależny mocy m wtw, gdy wyrażenie α jest spełnialne. Wprowadzamy wierzchołki α_{ij}, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant 3$, krawędziami łączymy te, które należą do jednego elementu koniunkcji oraz te, które odpowiadają zmiennej i jej zaprzeczeniu (patrz rysunek!).

Jeśli wyrażenie α jest spełnialne, to przy pewnym wartościowaniu zmiennych w nim wystepujących, każda z koniunkcji ma wartość 1, a pewien składnik każdej z alternatyw również ma wartość 1. Do zbioru niezależnego wybieramy po jednym takim składniku. Odwrotnie, każdy zbiór niezależny wyznacza takie wartościowanie, że dokładnie jeden składnik każdej z alternatyw ma wartość 1.

11.4 Przykład dwóch pokrewnych problemów NP-zupełnych

Problem kliki: Dla grafu G i $k \in N$ rozstrzygnąć, czy istnieje pełny podgraf G o k wierzchołkach.

Problem pokrycia wierzchołkowego: Dla grafu G i $k \in N$ rozstrzygnąć, czy istnieje k-elementowy zbiór B wierzchołków taki, że każda krawędź w G jest koincydentna przynajmniej z jednym wierzchołkiem z B.

Łatwo sprowadzić problem zbioru niezależnego, do każdego z tych problemów.

Zbiór $K \subseteq V$ grafu G=(V,E) jest kliką wtw, gdy jest zbiorem niezależnym grafu G=(V,E'), gdzie $\{i,j\} \in E' \longleftrightarrow \{i,j\} \notin E$.

Zbiór $K \subseteq V$ grafu G=(V,E) jest pokryciem wierzchołkowym wtw, gdy zbiór V-K jest niezależny.

11.5 Problem P=NP

Problem P=NP jest jednym z siedmiu wielkich problemów milenijnych. Problemy te zostały sformułowane w czasie sesji Clay Mathematical Institute w College de France w Paryżu w maju 2000 r przez znanych uczonych J.Tate i M.Atyiaha. Za rozwiązanie każdego z tych problemów jest wyznaczona nagroda w wysokości 1 miliona dolarów. Choć dominuje przeświadczenie, że nie jest prawdą, iż P=NP (prównaj Hopcroft, Ullman str. 419-423) nie brak też prób udowodnienia tej równości. Standardowo poszukuje się algorytmów deterministycznych o wielomianowej złożoności czasowej dla problemów NP-zupełnych.

A więc do dzieła!

Mam nadzieję, że nagroda miliona dolarów jest wystarczającą zachętą, choć i względy praktyczne są równie ważne, gdyż lista problemów NP-zupełnych jest bardzo długa i zawiera mnóstwo praktycznych (dla informatyków) problemów. Wystarczy tu wspomnieć o problemie całkowitoliczbowego programowania liniowego czy problemie komiwojażera.

12 Zadania kontrolne

1. Wykazać, że istnieje totalna funkcja RAM-obliczalna $f : N \mapsto N$ taka, że $\forall_{x \in N}\ E_{f(x)} = \bigcup_{y \in W_x}\ W_y$.

2. Wykazać, że istnieje totalna funkcja RAM-obliczalna $f : N \mapsto N$ taka, że $\forall_{x \in N}\ W_{f(x)} = \{x\}$.

3. Wykazać, że istnieje totalna funkcja RAM-obliczalna $f : N \mapsto N$ taka, że $\forall_{x \in N}\ |E_{f(x)}| = \begin{cases} 0 & \text{x pierwsza} \\ \aleph_0 & \text{x złożona} \end{cases}$.

4. Wykazać, że istnieje totalna funkcja RAM-obliczalna $f : N \mapsto N$ taka, że $\forall_{x \in N}\ |W_{f(x)}| = \begin{cases} 0 & x \notin W_x \\ \aleph_0 & x \in W_x \end{cases}$.

5. Wykazać, że istnieje totalna funkcja RAM-obliczalna $g : N \times N \mapsto N$ taka, że $\forall_{x,y \in N}\ W_{g(x,y)} = \phi_x(W_y)$.

6. Wykazać, że istnieje totalna funkcja RAM-obliczalna $g : N \times N \mapsto N$ taka, że $\forall_{x,y \in N}\ W_{g(x,y)} = W_x \oplus W_y$, gdzie $A \oplus B = \{2k\ :\ k \in a\} \cup \{2k+1\ :\ k \in B\}$.

7. Wykazać, że istnieje totalna funkcja RAM-obliczalna $g : N \times N \mapsto N$ taka, że $\forall_{x,y \in N}\ W^{(2)}_{g(x,y)} = E_x \times E_y$.

8. Wykazać, że istnieje totalna funkcja RAM-obliczalna $g : N \times N \mapsto N$ taka, że $\forall_{x,y \in N}\ E_{g(x,y)} = E_x \cup W_y$.

9. Wykazać, że następujące zbiory nie są rekurencyjne.

   (a) $A = \{x \in N\ :\ |W_x| < \aleph_0\}$.

   (b) $A = \{x \in N\ :\ W_x = E_x\}$.

(c) $A = \{x \in N \ : \ E_x$ jest zbiorem liczb parzystych$\}$.

(d) $A = \{x \in N \ : \ 2001 \in W_x\}$.

(e) $A = \{(x, y) \in N^2 \ : \ x \in E_y\}$.

(f) $A = \{(x, y) \in N^2 \ : \ E_x = W_y\}$.

(g) $A = \{(x, y) \in N^2 \ : \ |W_x| < |E_y|\}$.

(h) $A = \{(x, y) \in N^2 \ : \ \phi_y \circ \phi_x = id_N\}$.

(i) $A = \{(x, y, z) \in N^3 \ : \ W_x \subseteq E_y \subseteq W_z\}$.

(j) $A = \{x \in N \ : \ E_x$ nie jest rekurencyjny$\}$.

10. Wykazać, że następujące relacje są częściowo rozstrzygalne.

(a) $H(x, y) \Leftrightarrow y \in W_x \wedge x \in E_y$.

(b) $H(x, y) \Leftrightarrow W_x \cap E_y \neq \emptyset$.

(c) $H(x, y) \Leftrightarrow \phi_x \circ \phi_y \neq \emptyset$.

(d) $H(x, y, n) \Leftrightarrow \phi_x(y) = n^2$.

13 Program wykładu

kod: PTO

Przedmiot: Podstawy teorii obliczalności
Rok: II rok
Specjalność: informatyka
Semestr: letni
Wykładowca: dr Barbara Klunder
Wymiar godzin: *30 godz. wykładu i 30 godz. ćwiczeń*
Egzamin: po semestrze letnim

Program wykładu

1. **Metainformatyka i toria informatyki**

(a) Metainformatyka: jej cele, charakter stawianych pytań i rodzaj stosowanych metod.

(b) Relacje z torią informatyki.

2. **Funkcje obliczalne**

   (a) Algorytmy lub procedury obliczeniowe.

   (b) Funkcje obliczalne i relacje rozstrzygalne w sensie ogólnym.

   (c) Kodowanie obiektów skończonych liczbami naturalnymi.

3. **Funkcje RAM-obliczalne**

   (a) Maszyna z nieograniczoną pamięcią.

   (b) Predykaty i problemy obliczalne.

   (c) Generowanie funkcji obliczalnych: podstawianie, rekursja i minimalizacja.

4. **Inne podejścia do obliczalności: Teza Churcha**

   (a) Funkcje częściowo rekurencyjne (Gödel, Kleene).

   (b) Obliczalność w sensie Turinga.

   (c) Podstawowe dwa style programowania: imperatywny i deklaratywno-funkcyjny.

   (d) Teza Chucha i argumenty na jej rzecz.

5. **Efektywne numeracje programów**

   (a) Numeracje programów i funkcji obliczalnych.

   (b) Metoda diagonalizacji- przykład funkcji, która nie jest obliczalna.

6. **Twierdzenie o parametryzacji i programy uniwersalne**

7. **Zbiory rekurencyjne**

   (a) Podstawowe własności.

   (b) Twierdzenie Rice'a.

   (c) Problemy nierozstrzygalne: problem stopu i pochodne.

8. **Zbiory rekurencyjnie przeliczalne**

(a) Powiązania ze zbiorami rekurencyjnymi.

(b) m-sprowadzalność i sprowadzalność w sensie Turinga.

(c) Twierdzenie Rice'a-Shapiro.

9. **Złożoność obliczeniowa, przykłady miar złożoności.**

10. **Złożoność obliczeniowa maszyn Turinga: Klasy P i NP, problemy NP-zupełne.**

Literatura

- J.M. Brady *Informatyka teoretyczna w ujęciu programistycznym* WNT 1983
- N. Cutland *Computability. An Introduction to Recursive Function Theory* Cambridge Univeristy Press 1980
- D. Harel *Komputery - spółka z o.o., czego komputery naprawdę nie umieją robić* WNT 2002
- J.E. Hopcroft J.D. Ullman *Wprowadzenie do teorii automatów, języków i obliczeń* PWN 1994
- A. Kościelski *Teoria obliczeń. Wykłady z matematycznych podstaw informatyki* Wydawnictwo Uniwersytetu Wrocławskiego 1997
- Ch. Papadimitriou *Computability and Complexity* Addisson and Weseley 1995, WNT 2002 (wydanie polskie)
- J.R. Shoenfield *Recursion Theory* Springer Verlag 1993