



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY

Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

# TEORIA OBliczalności

## wykład

Kamila Barylska

Łukasz Mikulski

Marcin Piątkowski

UMK Toruń 2013

Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Informacje wstępne

### Program wykładu

1. Maszyny licznikowe
2. Funkcje częściowo rekurencyjne
3. Maszyny Turinga
4. Sieci liczące
5. Rozstrzygalność problemów obliczeniowych
6. Częściowa rozstrzygalność problemów obliczeniowych
7. Czasowa złożoność obliczeniowa
8. Pamięciowa złożoność obliczeniowa
9. Problemy trudne i zupełne w wybranych klasach złożoności

### Literatura

- Michael Sipser *Wprowadzenie do teorii obliczeń*
- Antonii Kościelski *Teoria obliczeń*
- Christos H. Papadimitriou *Złożoność obliczeniowa*
- David Harel *Rzecz o istocie informatyki – Algorytmika*
- John E. Hopcroft, Jeffrey D. Ullman *Wprowadzenie do teorii automatów, języków i obliczeń*
- Samuel Eilenberg *Automata, Languages, and Machines*
- S. Barry Cooper *Computability theory*
- John E. Savage *Models of Computation. Exploring the Power of Computing*



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Wykład 1

Podstawowym obiektem badań teorii obliczeń są funkcje obliczalne. Stanowią one odpowiednik intuicyjnego pojęcia algorytmu. Teoria obliczeń składa się z dwóch głównych części: teoria obliczalności poświęconej rozwiązywalności problemów obliczeniowych za pomocą komputera, oraz teoria złożoności obliczeniowej badającej złożoność problemów obliczeniowych przez określenie długości czasu oraz rozmiaru pamięci potrzebnych ich rozwiązania.

Jednym z podstawowych faktów określających możliwości obliczeniowe komputerów i innych modeli obliczalności jest Teza Churcha-Turinga.

### Twierdzenie 1 (Teza Churcha-Turinga)

*Każdy problem obliczeniowy, dla którego intuicyjnie istnieje algorytm jest obliczalny.*

Powyzsza teza nie jest twierdzeniem w ścisłe matematycznym sensie i nie jest możliwe jej formalne udowodnienie. W teorii obliczalności przyjmowana jest raczej jako aksjomat. Zgodnie z tezą Churcha-Turinga każdy problem, który może być rozwiązany na dowolnym komputerze (nawet takim, który nie został jeszcze skonstruowany), może być rozwiązany przy pomocy urządzenia maszynowego dysponującego nieograniczoną (ale skończoną) czasem oraz nieograniczoną (ale skończoną) zasobami pamięci. Istnieją również problemy, których nie da się obliczyć przy pomocy wspomnianych urządzeń maszynowych. Są to jednak problemy, które nie mieścią się nawet teoretycznie w granicach możliwości obliczeniowych komputera.

W teorii obliczalności zazwyczaj rozważamy funkcje *częściowe*  $f : \mathbb{N}^k \rightarrow \mathbb{N}$ , które nie muszą być określone dla wszystkich argumentów. Funkcję określoną dla wszystkich argumentów nazywamy *totalną*.

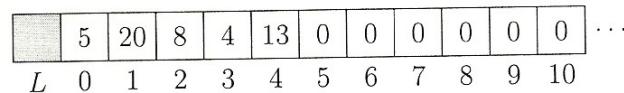
Precyjne badanie możliwości komputerów wymaga formalnego zdefiniowania modelu obliczeń. Najprostszymi modelami obliczeń są automaty skończone z pamięcią dostępną wyłącznie w skończonej liczbie stanów oraz automaty ze stosem z dodatkową pamięcią dostępną na zasadzie stosu. Są one jednak zbyt ograniczone, aby służyć jako ogólny model komputera. Musimy zatem rozważać bardziej rozbudowane modele obliczeń z nieograniczoną pamięcią o dostępie swobodnym, takie jak maszyny licznikowe czy maszyny Turinga.

## Maszyny licznikowe

Maszyna licznikowa jest teoretycznym modelem komputera wykonującym programy tworzone w języku przypominającym prosty assembler. Zbudowana jest ona z licznika rozkazu (zawierającego numer kolejnej instrukcji do



wykonania) oraz nieograniczonej pamięci składającej się z rejestrów. Każdy rejestr może zawierać dowolnie dużą liczbę naturalną. W czasie obliczeń maszyna licznikowa może wykorzystać dowolnie wiele rejestrów. Wszystkie rejestr, poza skońzoną liczbą używanych przez program, zawierają wartość 0. Zawartość rejestru o numerze  $n$  oznaczamy przez  $Z[n]$ .



Rysunek 1: Schemat pamięci maszyny licznikowej

Maszyna licznikowa może wykonywać następujące instrukcje:

Kod	Instrukcja	Semantyka	Opis
0	$Z(n)$	$Z[n] := 0$	zerowanie zawartości rejestru
1	$S(n)$	$Z[n] := Z[n] + 1$	inkrementacja rejestru
2	$T(m, n)$	$Z[n] := Z[m]$	kopiowanie wartości rejestru
3	$I(m, n, q)$	if $Z[m] = Z[n]$ then GOTO $q$	skok warunkowy

Tabela 1: Instrukcje maszyny licznikowej

Pierwsze trzy instrukcje nazywamy instrukcjami arytmetycznymi, ostatnią natomiast – instrukcją warunkową. Licznik rozkazów zawiera numer kolejnej instrukcji do wykonania. Jeśli numer instrukcji do wykonania jest większy niż liczba instrukcji w programie – program kończy działanie.

### Definicja 1

*Programem na maszynę licznikową (ML-programem) nazywamy dowolny nie-pusty ciąg ML-instrukcji.*

Instrukcje programu na maszynę licznikową wykonywane są sekwencyjnie. Po wykonaniu dowolnej instrukcji arytmetycznej wartość licznika rozkazów jest zwiększana o 1. Jeśli klauzula instrukcji warunkowej jest prawdziwa, wartość licznika rozkazów jest numer instrukcji podanej jako parametr skoku, w przeciwnym wypadku wartość licznika rozkazów zwiększana jest o 1.

Według przyjętej konwencji, argumenty programu znajdują się w rejestrach  $1, \dots, k$ , natomiast wartość zwracana jest do rejestru 0, nazywanego rejestrem wynikowym.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁCZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

### Uwaga

Wszystkie instrukcje oraz rejesty maszyny licznikowej muszą być adresowane bezpośrednio.

### Przykład 1

Program  $P$  na maszynę licznikową liczący funkcję:  $f(x_1, x_2) = x_1 + x_2$ .

W konfiguracji początkowej maszyny licznikowej rejesty nr 1 oraz 2 zawierają wartości argumentów programu  $x_1$  oraz  $x_2$ , natomiast pozostałe rejesty zawierają wartość 0.

	0	$x_1$	$x_2$	0	0	0	0	...
$L$	0	1	2	3	4	5	6	

Działanie programu  $P$  będzie polegać na wykonaniu inkrementacji rejestrów nr 1 i 2 razy. W celu kontroli liczby inkrementacji wykorzystany zostanie rejestr nr 3. Porównanie zawartości rejestrów nr 2 oraz nr 3 będzie stanowiło warunek zakończenia głównej pętli programu  $P$ .

Program  $P$ :

- 1)  $I(2, 3, 5)$  // warunek końca pętli
- 2)  $S(1)$  // inkrementacja pierwszego argumentu
- 3)  $S(3)$  // inkrementacja rejestrów kontrolnych
- 4)  $I(1, 1, 1)$  // skok na początek pętli
- 5)  $T(1, 0)$  // skopiowanie sumy do rejestrów wynikowych

Nie jest trudno zauważyć, że po zakończeniu działania programu  $P$ , rejestr wynikowy (nr 0) zawiera poprawną wartość  $x_1 + x_2$ .

Jesteśmy teraz gotowi do zdefiniowania pojęcia obliczalności na maszynie licznikowej ( $ML$ -obliczalności).

### Definicja 2

Program  $P$  na maszynę licznikową oblicza funkcję częściową  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  ( $k > 0$ ), jeśli po umieszczeniu argumentów funkcji  $x_1, \dots, x_k$  w rejestrach 1, ...,  $k$  oraz wyzerowaniu pozostałych,  $P$  zatrzyma się zwieracając w rejestrze 0 wartość  $f(x_1, x_2, \dots, x_k)$  dokładnie wtedy, gdy  $(x_1, x_2, \dots, x_k)$  należy do dziedziny funkcji  $f$ .

### Definicja 3

Funkcję częściową  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  ( $k > 0$ ) nazywamy  $ML$ -obliczalną (lub prostu obliczalną), jeśli jest obliczana przez pewien program  $P$  na maszynę licznikową.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych” realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Każdy program  $P$  na maszynę licznikową oblicza pewną  $k$ -argumentową ( $k > 0$ ) funkcję częściową  $\phi_P^{(k)} : \mathbb{N}^k \rightarrow \mathbb{N}$ . Ponadto, dla uproszczenia zapisu, wprowadzimy następujące oznaczenia:

- $\mathbb{C}_k$  – zbiór wszystkich  $k$ -argumentowych funkcji obliczalnych
- $\mathbb{C} = \bigcup_{k \in \mathbb{N}} \mathbb{C}_k$  – zbiór wszystkich funkcji obliczalnych
- $\bar{x} = (x_1, \dots, x_k)$
- $P(\bar{x}) \downarrow$  – program  $P$  zatrzymuje się na danych  $\bar{x}$
- $P(\bar{x}) \downarrow y$  – program  $P$  zatrzymuje się na danych  $\bar{x}$  zwracając wartość  $y$
- $P(\bar{x}) \uparrow$  – program  $P$  nie zatrzymuje się na danych  $\bar{x}$

#### Definicja 4

Program  $P$  na maszynę licznikową nazywamy programem w postaci standardowej, jeśli dla każdej instrukcji  $I(m, n, q)$  zachodzi warunek  $q \leq k+1$ , gdzie  $k$  jest liczbą instrukcji programu  $P$ .

#### Ćwiczenie 1

Udowodnij, że dla dowolnego  $ML$ -programu  $P$  istnieje program  $P'$  w postaci standardowej obliczający tę samą funkcję.

#### Uwaga

Od tej pory będziemy rozważać wyłącznie programy w postaci standardowej.

#### Ćwiczenie 2

Uzasadnij, że dla dowolnego programu  $P$  na maszynę licznikową istnieje taka wartość  $k > 0$ , że  $\phi_P^{(k)} = \phi_P^{(k+1)}$ .

#### Pytanie

Jaka jest moc zbioru funkcji obliczalnych przez maszyny licznikowe?

Liczba funkcji  $ML$ -obliczalnych jest nieskończona. Naszym celem jest udowodnienie przeliczalności zbioru funkcji obliczalnych.

## Kodowanie i numeracja

Zbiór  $X$  nazywamy przeliczalnym jeśli jest równoliczny ze zbiorem liczb naturalnych (elementy  $X$  można ustawić w ciąg). W celu udowodnienia przeliczalności zbioru wszystkich funkcji obliczalnych będziemy potrzebowali definicji kodowania i numeracji.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

### Definicja 5

Kodowaniem zbioru  $X$  nazywamy dowolną funkcję  $f : X \rightarrow \mathbb{N}$ , która jest różnowartościowa (injekcja).

### Definicja 6

Numeracją (wyliczeniem) zbioru  $X$  nazywamy dowolną funkcję  $f : \mathbb{N} \rightarrow X$ , która jest „na” (surjekcja).

W dalszych rozważaniach będziemy zainteresowani głównie bijektywnymi (odwracalnymi) kodowaniami i numeracjami. Poniżej prezentujemy kilka przykładów bijektywnych kodowań par i ciągów liczb naturalnych.

1. Bijektywne kodowanie par liczb naturalnych:  $\pi : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

$$\pi(m, n) = 2^m \cdot (2n + 1) - 1$$

Funkcja odwrotna (numeracja)  $\pi^{-1} : \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$  określona jest wzorem:

$$\pi^{-1}(x) = (\pi_1(x), \pi_2(x)),$$

gdzie

$$\begin{cases} \pi_1(x) = \max \{k \in \mathbb{N} : 2^k | (x + 1)\} \\ \pi_2(x) = \frac{1}{2} \left( \frac{x + 1}{2^{\pi_1(x)}} - 1 \right) \end{cases}$$

2. Kodowanie Eulera  $\mathcal{E} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

$$\mathcal{E}(x, y) = \frac{(x + y)^2 + 3x + y}{2}$$

3. Bijektywne kodowanie trójkę liczb naturalnych  $\beta : \mathbb{N}^3 \rightarrow \mathbb{N}$

$$\beta(m, n, p) = \pi(\pi(m, n), p)$$

4. Bijektywne kodowanie ciągów liczb naturalnych:  $\tau : \bigcup_{k \geq 0} \mathbb{N}^k \rightarrow \mathbb{N}$  (skończonych)

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

### Ćwiczenie 3

Uzupełnij powyższe kodowania bijektywne o brakujące funkcje do nich odwrotne (numeracje).

### Ćwiczenie 4

Udowodnij, że zdefiniowane powyżej kodowania (a także funkcje do nich odwrotne) są ML-obliczalne.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Efektywna numeracja ML-programów

Aby zakodować *ML*-program jako liczbę naturalną potrzebujemy efektywnej metody zakodowania pojedynczych instrukcji  $K_I$  oraz efektywnej metody zakodowania ciągu kodów instrukcji  $K_P$ . Wykorzystamy w tym celu numerację instrukcji przedstawioną w Tabeli 1.

Rozważmy *ML*-program  $P = \{I_1, I_2, \dots, I_k\}$  składający się z  $k$  instrukcji. Każdą instrukcję  $P$  zakodujemy za pomocą wzoru

$$K_I(I_j) = 4 \cdot [\text{kod argumentów}] + [\text{nr instrukcji}]$$

W przypadku instrukcji  $Z(n)$  oraz  $S(n)$  kodem argumentów jest adres rejestru, w przypadku instrukcji  $T(m, n)$  używamy bijektywnego kodowania par  $\pi$ , natomiast w przypadku instrukcji  $I(m, n, q)$  używamy bijektywnego kodowania trójkę  $\beta$ . Ponieważ instrukcje maszyny licznikowej są numerowane za pomocą liczb  $\{0, 1, 2, 3\}$  zdefiniowane powyżej kodowanie instrukcji jest bijekcją.

Ciąg kodów poszczególnych instrukcji zakodujemy za pomocą bijektywnego kodowania skończonych ciągów liczb naturalnych  $\tau$ :

$$K_P(P) = \tau(K_I(I_1), K_I(I_2), \dots, K_I(I_k)).$$

Zdefiniowane powyżej kodowanie dowodzi, że każdemu *ML*-programowi możemy przyporządkować liczbę naturalną. Ponieważ wszystkie użyte funkcje kodujące są bijekcjami, również każdej liczbie naturalnej odpowiada pewien *ML*-program liczący pewną funkcję obliczalną. Zatem zbiór wszystkich *ML*-programów jest równoliczny ze zbiorem liczb naturalnych.

Mozemy teraz ustalić numerację wszystkich funkcji obliczalnych przyporządkowując funkcji  $\phi \in \mathbb{C}$  numer dowolnego *ML*-programu, który ją oblicza. Symbolem  $\phi_n$  oznaczać będziemy funkcję obliczaną przez *ML*-program o numerze  $n$ , symbolem  $W_n$  – dziedzinę funkcji obliczanej przez program o numerze  $n$ , zaś symbolem  $E_n$  – przeciwdziedzinę funkcji obliczanej przez program o numerze  $n$ , czyli  $w_n = D_{\phi_n}$  oraz  $E_n = \phi_n(W_n)$ .

### Przykład 2

Wyznaczmy teraz numer programu  $P$  obliczającego funkcję  $f(x) = x + 1$ .

Rozważany program składa się z dwóch instrukcji:

$$\begin{aligned} I_1: & \quad S(1) \\ I_2: & \quad T(1, 0) \end{aligned}$$



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Kodami kolejnych instrukcji programu  $P$  są:

$$\begin{aligned} K_I(I_1) &= 4 \cdot 1 + 1 = 5 \\ K_I(I_2) &= 4 \cdot \pi(1, 0) + 2 = 4 \cdot 1 + 2 = 6 \end{aligned}$$

Kodem całego programu jest:

$$K_P(P) = \tau(5, 6) = 2^5 + 2^{5+6+1} - 1 = 4127$$

Zatem  $\phi_p = \phi_{4127}$ .

### Uwagi

1. Znając numer  $ML$ -programu możemy odkodować listę jego instrukcji. Nie jesteśmy jednak w stanie określić ile argumentów ma funkcja, którą on oblicza.
2. Każda funkcja obliczalna może być obliczania przez nieskończenie wiele programów – tym samym może mieć nieskończonie wiele numerów.

### Przykład negatywny

Powyżej wykazaliśmy, że zbiór funkcji obliczalnych jest przeliczalny. Uzasadnimy teraz istnienie funkcji, które nie są obliczalne.

### Twierdzenie 2

Istnieje funkcja totalna  $f : \mathbb{N} \rightarrow \mathbb{N}$ , która nie jest obliczalna.

### Dowód

Niech  $\{\phi_n\}_{n \in \mathbb{N}}$  będzie ciągiem wszystkich jednoargumentowych funkcji obliczalnych. Rozważmy funkcję:

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

i przyjmijmy, że jest ona obliczalna. Niech  $e$  będzie numerem dowolnego  $ML$ -programu, który ją oblicza. Wówczas  $f = \phi_e$  oraz  $f(e) = \phi_e(e)$ . Ponieważ funkcja  $f$  jest totalna, więc  $e \in W_e$  oraz z określenia  $f(e) = \phi_e(e) + 1$ . Otrzymana sprzeczność dowodzi, że funkcja  $f$  nie może być obliczalna.  $\square$



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych” realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Wykład 2

# Struktura zbioru funkcji ML-obliczalnych

W czasie poprzedniego wykładu zdefiniowany został model maszyny licznikowej oraz pojęcie *ML*-obliczalności. Bieżący wykład poświęcony będzie głębszemu zbadaniu zbioru funkcji obliczalnych przez maszyny licznikowe. Omówione zostaną sposoby konstrukcji funkcji *ML*-obliczalnych takie jak rekursja czy minimalizacja.

### Oznaczenie:

Dla *ML*-programu  $P$  przez  $\rho(P)$  oznaczamy funkcję zwracającą najmniejszy adres rejestru nieużywanego przez program  $P$ .

## Składanie programów

Rozpoczniemy od zdefiniowania składania programów polegającego na bezpośrednim przejściu od ostatniej instrukcji jednego programu do pierwszej instrukcji drugiego programu.

Niech  $P = \{I_1, I_2, \dots, I_n\}$  oraz  $R = \{J_1, J_2, \dots, J_m\}$  będą *ML*-programami w postaci standardowej. Złożeniem  $P$  i  $R$  nazywamy program

$$PR = \{I_1, \dots, I_n, J'_1, \dots, J'_m\},$$

gdzie  $J'_i = J_i$  jeśli  $J_i$  jest instrukcją postaci  $Z(k)$ ,  $S(k)$  lub  $T(m, n)$  oraz  $J'_i = I(m, n, q + n + 1)$  jeśli  $J_i = I(m, n, q)$ .

## Podprogramy

Dowolny program na maszynę licznikową może być wywołany jako podprogram wewnętrz innego *ML*-programu. Wywołanie podprogramu wymaga odpowiedniego przygotowania jego argumentów oraz obsłużenia zwróconej przez niego wartości.

Niech  $P = \{I_1, I_2, \dots, I_n\}$  będzie *ML*-programem w postaci standardowej liczącym funkcję  $\phi_P^{(k)}$ , oraz  $l_1, \dots, l_k \notin \{1, 2, \dots, k\}$ . Zapis  $P[l_1, \dots, l_k \rightarrow l]$  oznacza wywołanie programu  $P$  na argumentach znajdujących się w rejestrach  $l_1, \dots, l_k$  oraz umieszczenie zwróconej przez niego wartości w rejestrze  $l$ .

Przypomnijmy, że argumenty *ML*-programu powinny znajdować się w rejestrach  $1, \dots, k$ , natomiast zwracana przez niego wartość zapisywana jest



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI

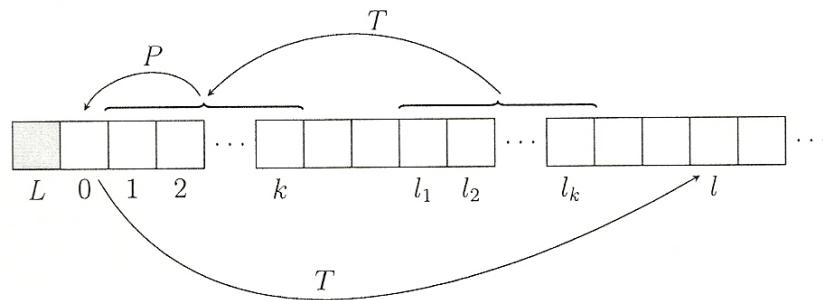


UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

w rejestrze 0. Przed uruchomieniem programu  $P$ , należy więc skopiować jego argumenty z rejestrów  $l_1, \dots, l_k$  do rejestrów  $1, \dots, k$  oraz wyzerować wszystkie pozostałe używane przez niego rejesty. Po zakończeniu działania programu  $P$  należy skopiować zwróconą przez niego wartość z rejestru 0 do rejestrzu  $l$ .



Rysunek 2: Schemat wywołania podprogramu

Przykładowy fragment kodu wywołującego program  $P$  jako podprogram jest przedstawiony poniżej:

```

 $T(l_1, 1)$       // kopiowanie argumentów
 $T(l_2, 2)$ 
 $\vdots$ 
 $T(l_k, k)$ 
 $Z(0)$           // przygotowanie miejsca
 $Z(k+1)$ 
 $Z(k+2)$ 
 $\vdots$ 
 $Z(\rho(P))$ 
 $P$               // uruchomienie programu  $P$ 
 $T(0, l)$         // kopiowanie zwróconego wyniku

```

#### Uwaga

W schemacie opisany powyżej zakładamy, że program  $P_G$  nie używa rejestrów o adresach  $0, \dots, \rho(P_P)$  (poza wykorzystaniem rejestrów 0 do zwrócenia wyniku). Możemy opuścić to założenie wymagając wykonania kopii pamięci programu  $P_G$  w rejestrach o adresach wyższych niż  $\max(\rho(P_G), \rho(P_P))$  przed



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych” realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

wykonaniem podprogramu i przywrócenia jej po jego wykonaniu (pomijając oczywiście rejestr zawierający wynik działania podprogramu).

### Ćwiczenie 5

Napisz program  $P$  na maszynę licznikową liczący funkcję  $f(x_1, x_2) = x_1 \cdot x_2$ .

1. Użyj programu liczącego funkcję  $f(x_1, x_2) = x_1 + x_2$  jako podprogramu.
2. Napisz program  $P$  bez wywoływania podprogramu.

### Podstawianie

Operacja podstawiania jest uogólnieniem operacji złożenia funkcji. Na przykład, jeśli zdefiniowane są funkcje  $f : \mathbb{N}^3 \rightarrow \mathbb{N}$  oraz  $g : \mathbb{N}^2 \rightarrow \mathbb{N}$ , to za pomocą operatora podstawiania można zdefiniować funkcję  $h : \mathbb{N}^3 \rightarrow \mathbb{N}$  taką, że  $h(x, y, z) = f(x, g(x, y), z)$ .

Udowodnimy teraz, że składanie funkcji obliczalnych daje w wyniku również funkcję obliczalną.

### Twierdzenie 3

*Niech  $f : \mathbb{N}^k \rightarrow \mathbb{N}$  oraz  $g_1, g_2, \dots, g_k : \mathbb{N}^n \rightarrow \mathbb{N}$  będą funkcjami obliczalnymi ( $f \in \mathbb{C}_k$ ,  $g_1, \dots, g_k \in \mathbb{C}_n$ ). Wówczas funkcja  $h : \mathbb{N}^n \rightarrow \mathbb{N}$  określona jako  $h(\bar{x}) = f(g_1(\bar{x}), \dots, g_k(\bar{x})) \in \mathbb{C}_n$  jest obliczalna ( $h \in \mathbb{C}_n$ ).*

### Dowód

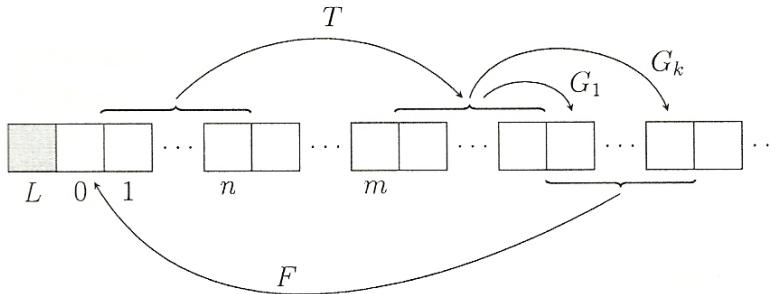
Niech  $f$  będzie obliczana przez program  $F$ , zaś  $g_1, \dots, g_k$  odpowiednio przez programy  $G_1, \dots, G_k$ . Ponadto, niech  $m = \max\{n, k, \rho(F), \rho(G_1), \dots, \rho(G_k)\}$  będzie najmniejszym numerem komórki nie używanej przez żaden z programów  $F, G_1, \dots, G_n$ .

Działanie programu  $H$  obliczającego funkcję  $h$  będzie polegało na skopiowaniu wartości argumentów (rejestry  $1, \dots, n$ ) w miejsce nieużywane przez żaden z rozważanych programów oraz kolejnym wywołaniu  $G_1, \dots, G_k$  jako podprogramów. Następnie wywołany zostanie (jako podprogram) program  $F$  na wynikach zwróconych przez programy  $G_1, \dots, G_k$ , a wynik jego działania zostanie umieszczony w rejestrze 0.

Kod programu  $H$  obliczającego funkcję  $h$  zdefiniowaną przez podstawienie jest przedstawiony poniżej:



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki



Rysunek 3: Schemat programu liczącego funkcję  $h$ .

```

 $T(1, m + 1)$  // kopiowanie argumentów
 $T(2, m + 2)$ 
⋮
 $T(n, m + n)$ 
 $G_1[m + 1, \dots, m + n \rightarrow m + n + 1]$  // przygotowanie argumentów
 $G_2[m + 1, \dots, m + n \rightarrow m + n + 2]$  // dla programu  $F$ 
⋮
 $G_k[m + 1, \dots, m + n \rightarrow m + n + k]$ 
 $F[m + n + 1, \dots, m + n + k \rightarrow 0]$  // wykonanie programu  $F$  □

```

### Wniosek

Operacja podstawienia pozwala utożsamiać i wprowadzać nieistotne argumenty. Przykładowo, jeśli funkcja  $f(x, y)$  jest obliczalna, funkcje  $f(x, x)$ ,  $f(y, x)$ ,  $g(x, y, z) = f(x, y)$ , itp., również są obliczalne.

### Przykład 3

Rozważmy funkcję  $f(x, y) = x + y$ . W oczywisty sposób  $f \in \mathbb{C}_2$  oraz  $f \notin \mathbb{C}_1$ . Ponadto, rozważaną funkcję  $f$  możemy określić jako  $f(x, y) = g(x, y, z)$  (ignorujemy trzeci argument). A zatem  $f \in \mathbb{C}_3$ .

### Ćwiczenie 6

Uzasadnij, że jeśli  $f \in \mathbb{C}_n$ , to również  $f \in \mathbb{C}_{n+1}$ .

### Rekursja

Rekursja jest sposobem określenia kolejnej wartości funkcji na podstawie wartości wcześniej obliczonych. Odpowiada iteracji spotykanej w językach programowania.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych” realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

#### Twierdzenie 4 (O operatorze rekursji)

Niech  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  oraz  $g : \mathbb{N}^{n+2} \rightarrow \mathbb{N}^{n+2}$  będą funkcjami obliczalnymi ( $f \in \mathbb{C}_n$ ,  $g \in \mathbb{C}_{n+2}$ ). Wówczas funkcja  $h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  określona rekurencyjnie

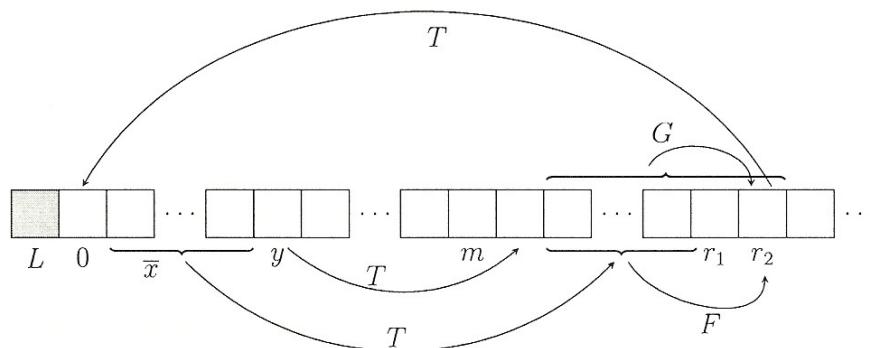
$$\begin{cases} h(\bar{x}, 0) = f(\bar{x}) \\ h(\bar{x}, y + 1) = g(\bar{x}, y, h(\bar{x}, y)) \end{cases}$$

również jest obliczalna ( $h \in \mathbb{C}^{n+1}$ ).

#### Dowód

Niech  $f \in \mathbb{C}_n$  oraz  $g \in \mathbb{C}_{n+2}$  będą obliczane odpowiednio przez programy  $F$  oraz  $G$ . Ponadto, niech  $m = \max\{n + 2, \rho(F), \rho(G)\}$  będzie najmniejszym adresem rejestru nie używanego przez żaden z programów  $F$  oraz  $G$ .

Działanie programu  $H$  liczącego funkcję  $h$  rozpoczyna się od skopiowania argumentów funkcji do nieużywanego obszaru pamięci i wyznaczenia wartości elementu zerowego za pomocą wywołania  $F$  jako podprogramu. Następnie, sekwencyjnie wywołujemy  $G$  jako podprogramy dla kolejnych wartości parametru  $y$  (rejestr  $r_1$ ), umieszczając wyniki w rejestrze  $r_2$ . Po zakończeniu działania programu, wynik kopowany jest do rejestru 0.



Rysunek 4: Schemat programu liczącego funkcję  $h$

Kod programu  $H$  obliczającego funkcję  $h$  zdefiniowaną za pomocą operatora rekursji jest przedstawiony poniżej:



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych” realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

$T(n+1, m+1)$	// kopiowanie argumentu $y$
$T(1, m+2)$	// kopiowanie argumentu $\bar{x}$
$T(2, m+3)$	
:	
$T(n, m+n+1)$	
$F[m+2, \dots, m+n+1 \rightarrow r_2]$	// wartość elementu zerowego
L: $I(m+1, r_1, K)$	// warunek końca pętli $Z[r_1] = y$
$G[m+2, \dots, r_1, r_2 \rightarrow r_2]$	// kolejna iteracja $G$
$S(r_1)$	// inkrementacja licznika ( $r_1$ )
$I(1, 1, L)$	// GOTO $L$
K: $T(r_2, 0)$	// kopiowanie wyniku

□

## Minimalizacja

Niech  $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  będzie funkcją  $ML$ -obliczalną. Minimalizacją  $f$  nazywamy funkcję  $g : \mathbb{N}^n \rightarrow \mathbb{N}$ , określoną jako najmniejsza wartość  $y \in \mathbb{N}$  taka, że dla każdego  $z < y$  funkcja  $f(\bar{x}, z)$  jest określona oraz  $f(\bar{x}, y) = 0$ . Jeśli taka wartość  $y \in \mathbb{N}$  nie istnieje, wartość funkcji  $g$  jest nieokreślona. Operator minimalizacji oznaczany jest przez  $g(\bar{x}) = \mu y (f(\bar{x}, y) = 0)$ .

### Twierdzenie 5 (O operatorze minimalizacji)

Niech  $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  będzie  $ML$ -obliczalna. Wówczas funkcja  $g : \mathbb{N}^n \rightarrow \mathbb{N}$  określona jako  $g(\bar{x}) = \mu y (f(\bar{x}, y) = 0)$  również jest  $ML$ -obliczalna.

#### Dowód

Niech funkcja  $f \in \mathbb{C}_{n+1}$  będzie obliczana przez program  $F$ . Ponadto, niech  $m = \max\{n+1, \rho(F)\}$  będzie najmniejszym adresem rejestru nie używanego przez program  $F$ .

Działanie programu  $G$  obliczającego funkcję  $g$  polega na obliczeniu wartości funkcji  $f$  dla kolejnych wartości parametru  $y$ . Program kończy działanie po otrzymaniu wartości 0. Zauważmy, że program  $G$  nie zatrzyma się jeśli funkcja  $f$  nigdy nie przyjmuje wartości 0 lub jeśli nie jest określona dla pewnych wartości  $y$ .

Kod programu  $H$  obliczającego minimalizację funkcji  $f$  jest przedstawiony poniżej:



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY

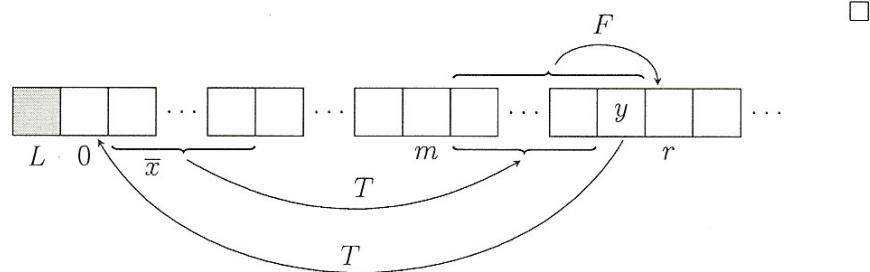


Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

```

 $T(1, m + 1)$  // kopiowanie argumentu  $\bar{x}$ 
 $T(2, m + 2)$ 
 $\vdots$ 
 $T(n, m + n)$ 
L:  $F[m + 1, \dots, m + n + 1 \rightarrow r]$  // wartość funkcji  $f(\bar{x}, y)$ 
 $I(r, m + n + 3, K)$  // warunek końca pętli  $Z[r] = 0$ 
 $S(m + n + 1)$  // inkrementacja wartości argumentu  $y$ 
 $I(1, 1, L)$  // GOTO L
K:  $T(m + n + 1, 0)$  // kopiowanie wynikowej wartości  $y$ 

```



Rysunek 5: Schemat programu liczącego operator minimalizacji

### Uwaga

Operator minimalizacji zastosowany do funkcji totalnej nie musi dać w wyniku funkcji totalnej.

### Ćwiczenie 7

Uzasadnij, że dla  $ML$ -programu  $P$  funkcja  $\rho(P)$  jest obliczalna.

### Ćwiczenie 8

Niech  $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  będzie  $ML$ -obliczalna. Udowodnij, że następujące modyfikacje operatora minimalizacji są obliczalne:

- $g_1(\bar{x}) = \mu y(f(\bar{x}, y) = k), k \in \mathbb{N}$
- $g_2(\bar{x}) = \mu y(f(\bar{x}, y) < k), k \in \mathbb{N}$
- $g_3(\bar{x}) = \mu y(f(\bar{x}, y) > k), k \in \mathbb{N}$
- $g_4(\bar{x}) = \mu y(f(\bar{x}, y) \leq k), k \in \mathbb{N}$
- $g_5(\bar{x}) = \mu y(f(\bar{x}, y) \geq k), k \in \mathbb{N}$



### Wykład 3

## Funkcje częściowo rekurencyjne

Pojęcie obliczalności może być zdefiniowane na wiele równoważnych sposobów. Bieżący wykład poświęcony jest funkcjom częściowo rekurencyjnym (Kleene 1952) będącym czysto matematyczną formalizacją pojęcia obliczalności. Pokażemy równość zbiorów funkcji częściowo rekurencyjnych oraz funkcji obliczalnych przez maszyny licznikowe.

### Definicja 7

*Funkcjami prostymi nazywamy funkcje:*

- zerową:  $Z : \mathbb{N}^n \longrightarrow \mathbb{N}$ , gdzie  $Z(x_1, \dots, x_n) = 0$
- następnika:  $S : \mathbb{N} \longrightarrow \mathbb{N}$ , gdzie  $S(x) = x + 1$
- rzutowania:  $p_i^n : \mathbb{N}^n \longrightarrow \mathbb{N}$ , gdzie  $p_i^n(x_1, \dots, x_n) = x_i$

### Definicja 8

Zbiór  $\mathcal{PR}$  funkcji pierwotnie rekurencyjnych definiujemy jako najmniejszy zbiór funkcji zawierający funkcje proste (zerową, następnika i rzutowania), który jest zamknięty ze względu na operacje podstawiania oraz rekursji.

### Przykład 4

Funkcja  $f(x) = n$  ( $n \in \mathbb{N}$ ) jest pierwotnie rekurencyjna, ponieważ jest złożeniem funkcji prostych

$$f(x) = \underbrace{S(S(\dots S(}_{n} Z(x)) \dots)).$$

Funkcja  $f(x) = x + n$  ( $n \in \mathbb{N}$ ) jest pierwotnie rekurencyjna, ponieważ jest złożeniem funkcji prostych – rzutowania  $P_1^1(x)$  oraz następnika  $S(x)$ :

$$f(x) = \underbrace{S(S(\dots S(}_{n} p_1^1(x)) \dots)).$$

Funkcja  $f(x, y) = x + y$  jest pierwotnie rekurencyjna, ponieważ może być uzyskana z funkcji prostych za pomocą operacji rekursji:

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



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

### Definicja 9

Zbiór  $\mathcal{R}$  funkcji częściowo rekurencyjnych definiujemy jako najmniejszy zbiór funkcji zawierający funkcje bazowe (zerową, następnika i rzutowania), który jest zamknięty ze względu na operacje podstawiania, rekursji oraz minimalizacji. Funkcje częściowo rekurencyjne, które są totalne, nazywamy funkcjami (ogólnie) rekurencyjnymi.

### Twierdzenie 6

Każda funkcja częściowo rekurencyjna jest  $ML$ -obliczalna.

### Dowód

Zauważmy, że funkcje proste są w oczywisty sposób  $ML$ -obliczalne. Ponadto, zastosowanie operacji podstawienia, rekursji oraz minimalizacji nie wyprowadza poza klasę funkcji  $ML$ -obliczalnych. Zatem każda funkcja częściowo rekurencyjna jest  $ML$ -obliczalna.  $\square$

### Twierdzenie 7

Każda funkcja  $ML$ -obliczalna jest częściowo rekurencyjna.

### Dowód (szkic)

Zbiorem stanów (konfiguracji) maszyny licznikowej nazywamy zbiór funkcji

$$S = \{f : \mathbb{N} \cup \{L\} \rightarrow \mathbb{N}, \text{ t.z. } \exists_m \forall_{n > m} f(n) = 0\}.$$

Wartości funkcji  $f$  opisującej stan maszyny interpretujemy jako:

- $f(L)$  – zawartość licznika rozkazów
- $f(n)$  – zawartość rejestru o numerze  $n$

Funkcją zmiany stanu maszyny licznikowej nazywamy funkcję  $\delta : \mathcal{I} \times S \rightarrow S$ , gdzie  $\mathcal{I}$  jest jej zbiorem instrukcji. Wartość funkcji  $f'$  dla stanu  $\delta(i, f)$  określona jest następująco:

$$\begin{aligned} f'(L) &= \begin{cases} q & \text{jeśli } i \approx I(x, y, q) \wedge Z[x] = Z[y] \\ f(L) + 1 & \text{w przeciwnym przypadku} \end{cases} \\ i \approx Z(x) \implies f'(c) &= \begin{cases} 0 & \text{jeśli } c = x \\ f(c) & \text{w przeciwnym przypadku} \end{cases} \\ i \approx S(x) \implies f'(c) &= \begin{cases} f(c) + 1 & \text{jeśli } c = x \\ f(c) & \text{w przeciwnym przypadku} \end{cases} \end{aligned}$$



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych” realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

$$i \approx T(x, y) \implies f'(c) = \begin{cases} f(x) & \text{jeśli } c = y \\ f(c) & \text{w przeciwnym przypadku} \end{cases}$$

$$i \approx I(x, y, q) \implies \forall_{c \in \mathbb{N}} f'(c) = f(c)$$

Niech  $P = \{I_1, \dots, I_k\}$  będzie programem na maszynę licznikową, zaś  $\phi_P^{(m)}$  będzie  $m$ -argumentową funkcją obliczaną przez  $P$ . Element  $\bar{x}$  należy do dziedziny funkcji  $\phi_P^{(m)}$  wtedy i tylko wtedy, gdy istnieje ciąg stanów maszyny licznikowej  $f_0, f_1, \dots, f_l$  taki, że:

- $f_0$  jest konfiguracją początkową programu  $P$
- $\forall_{j=0, \dots, l-1} f_j(L) \leq k$
- $\forall_{j=0, \dots, l-1} f_{j+1} = \delta(I_{f_j(L)}, f_j)$
- $f_l(L) \geq s + 1$

Wówczas  $\phi_P^{(m)}(\bar{x}) = f_l(0)$ .

Kodowanie  $K_I$  instrukcji  $ML$ -programu zdefiniowaliśmy w czasie jednego z poprzednich wykładów (patrz str. 8). Kodowanie  $K_f$  stanu (konfiguracji) maszyny licznikowej definiujemy jako  $K_f = \tau(f(L), f(0), \dots, f(\rho(P)))$ .

Możemy teraz określić funkcję zmiany stanu  $\delta' : K_I \times K_f \rightarrow K_f$  na parze liczb naturalnych  $(k_i, k_s)$ , oznaczających odpowiednio kod instrukcji oraz kod stanu maszyny, wykorzystując kodowania bijektywne  $\pi, \beta$  oraz  $\tau$ .

Niech  $\sigma : \mathbb{N}^{n+1} \rightarrow \mathbb{N}^n$  będzie funkcją zwracającą kod stanu maszyny licznikowej po  $j$  krokach obliczeń na  $\bar{x} \in \mathbb{N}^n$ . Wartość  $\sigma(\bar{x}, j)$  określamy:

$$\sigma(\bar{x}, 0) = K_f(f_0)$$

$$\sigma(\bar{x}, j+1) = \begin{cases} \delta'(K_I(I_j), \sigma(\bar{x}, j)) & \iff j < k+1 \\ \sigma(\bar{x}, j) & \iff j \geq k+1 \end{cases}$$

Określamy funkcje (częściowo rekurencyjne):

$$g_L(\sigma(\bar{x}, k)) = p_0(\tau^{-1}(\sigma(\bar{x}, j)))$$

$$g_W(\sigma(\bar{x}, k)) = p_1(\tau^{-1}(\sigma(\bar{x}, j)))$$

zwracające zawartość licznika rozkazów oraz rejestru  $Z[0]$  po  $j$  krokach obliczeń maszyny licznikowej.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Wówczas funkcja  $\phi_P^{(m)}$  określona jako:

$$\phi_P^{(m)}(\bar{x}) = g_W\left(\bar{x}, \mu j(g_L(\bar{x}, j) \geq k+1)\right)$$

jest częściowo rekurencyjna, co kończy dowód.  $\square$

### Ćwiczenie 9

Uzasadnij, że wszystkie wykorzystane w dowodzie twierdzenia 6 funkcje są częściowo rekurencyjne.

### Funkcja Ackermanna (1928)

Przypomnijmy, że stosując operator podstawienia i rekursji do funkcji totalnej otrzymujemy również funkcję totalną. Operator minimalizacji nie ma tej własności. W celu uzasadnienia, że operator minimalizacji jest istotny, zaś zbiory funkcji pierwotnie rekurencyjnych i częściowo rekurencyjnych nie są równe (dokładnie  $\mathcal{PR} \subsetneq \mathcal{R}$ ), pokażemy dwa przykłady funkcji częściowo rekurencyjnych (a więc obliczalnych), które nie są pierwotnie rekurencyjne.

### Definicja 10

Funkcja Ackermanna  $A : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  określona jest następująco:

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

### Przykład 5

Przykładowe wartości funkcji Ackermanna:

$x \setminus y$	0	1	2	3	4	...	n
0	1	2	3	4	5	...	$n + 1$
1	2	3	4	5	6	...	$n + 2$
2	3	5	7	9	11	...	$2n + 3$
3	5	13	29	61	125	...	$2^{n+3} - 3$
4	$2^{2^2} - 3$	$2^{2^{2^2}} - 3$	$2^{2^{2^{2^2}}} - 3$	$2^{2^{2^{2^{2^2}}}} - 3$	$2^{2^{2^{2^{2^{2^2}}}}} - 3$	...	$\underbrace{2^{2^{\dots^2}}}_{n+3} - 3$

### Uwaga

Złożoności obliczeniowej algorytmu liczącego wartość funkcji Ackermanna



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

nie da się wyrazić za pomocą funkcji elementarnych. Trudność obliczeniowa wynika ze złożoności wywołań rekurencyjnych. W celu obliczenia jej wartości wykorzystywane są wyłącznie proste operacje arytmetyczne (dodawanie i odejmowanie) jednak ogromna liczba wywołań rekurencyjnych może doprowadzić do przepełnienia stosu (*stack overflow*).

### Twierdzenie 8

*Funkcja Ackermanna jest częściowo rekurencyjna.*

#### Dowód

Dla każdego wywołania rekurencyjnego  $A(x, y)$  zachodzi jeden z warunków:

- wartość  $x$  zmniejsza się
- wartość  $x$  nie zmienia się, natomiast wartość  $y$  zmniejsza się

Za każdym razem kiedy  $y$  osiągnie 0 wartość  $x$  maleje. Liczba wywołań rekurencyjnych jest więc ograniczona. Zauważmy jednak, że kiedy wartość  $x$  się zmniejsza, wartość  $y$  znacznie rośnie. Liczba wywołań rekurencyjnych, choć ograniczona, będzie przez to ogromna.  $\square$

### Twierdzenie 9

*Funkcja Ackermanna nie jest pierwotnie rekurencyjna.*

#### Idea dowodu

Funkcja Ackermanna nie może być pierwotnie rekurencyjna, ponieważ jej wartości rosną znacznie szybciej niż wartości dowolnej funkcji pierwotnie rekurencyjnej.

#### Własności funkcji Ackermanna

- Dla ustalonej wartości  $y \in \mathbb{N}$  funkcja  $A_y(x) = A(x, y)$  jest pierwotnie rekurencyjna.
- Dla ustalonej wartości  $x \in \mathbb{N}$  funkcja  $A_x(y) = A(x, y)$  jest pierwotnie rekurencyjna.
- Dla dowolnej funkcji pierwotnie rekurencyjnej  $f$  istnieją takie stałe  $x_f, y_f \in \mathbb{N}$ , że

$$\forall_{x \geqslant x_f} \max \{f(x_1, \dots, x_n) | x_1, \dots, x_n \leqslant x\} < A_{y_f}(x),$$

czyli wartości funkcji Ackermanna rosną znacznie szybciej niż wartości dowolnej funkcji pierwotnie rekurencyjnej (wskazanie wartości stałych  $x_f, y_f$  dla funkcji prostych oraz operatorów podstawiania i rekursji)



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

- Funkcja Ackermanna jest rosnąca ze względu na pierwszą współrzędną:  
 $A(x+1, y) > A(x, y)$  (indukcja)
- Funkcja Ackermanna jest rosnąca ze względu na drugą współrzędną:  
 $A(x, y+1) > A(x, y)$  (indukcja)

### Ćwiczenie 10

Udowodnij powyższe własności funkcji Ackermanna.

#### Dowód twierdzenia 9 (przez sprzeczność)

Założymy, że funkcja Ackermanna  $A(x, y)$  jest pierwotnie rekurencyjna. Zatem istnieją stałe  $x_A, y_A \in \mathbb{N}$ , takie że

$$\forall_{x \geq x_A} \max \{A(x_1, x_2) | x_1, x_2 \leq x\} < A_{y_A}(x) (= A(x, y_A)).$$

Przyjmijmy  $x \geq \max\{x_A, y_A\}$  oraz  $x_1 = x_2 = x$ . Wówczas otrzymujemy nierówność  $A(x, x) < A(x, y_A)$ , która przeczy monotoniczności funkcji Ackermanna ze względu na drugą współrzędną. Zatem funkcja Ackermanna nie jest pierwotnie rekurencyjna.  $\square$

### Funkcja Sudana (1927)

Funkcja Sudana jest kolejnym przykładem funkcji częściowo rekurencyjnej, która nie jest pierwotnie rekurencyjna.

#### Definicja 11

*Funkcją Sudana nazywamy funkcję  $F_i : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  określoną następująco:*

$$F_0(x, y) = x + y,$$

$$F_n(x, 0) = x, \text{ dla } n \geq 0,$$

$$F_{n+1}(x, y+1) = F_n(F_{n+1}(x, y), F_{n+1}(x, y) + y + 1), \text{ dla } n \geq 0.$$

### Przykład 6

Przykładowe wartości funkcji Sudana  $F_1(x, y)$ :

x \ y	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	3	5	7	9	11
2	4	8	12	16	20	24
3	11	19	27	35	43	51
4	26	42	58	74	90	106
5	57	89	121	153	185	217
6	120	184	248	312	376	440



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Przykładowe wartości funkcji Sudana  $F_2(x, y)$ :

$x \setminus y$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	8	27	74	185	440
2	19	10228	$\approx 1.55 \cdot 10^{10}$	$\approx 5.74 \cdot 10^{24}$	$\approx 3.67 \cdot 10^{58}$	$\approx 5.02 \cdot 10^{135}$

### Twierdzenie 10

Funckja Sudana jest funkcją częściowo rekurencyjną, ale nie jest pierwotnie rekurencyjna.

### Ćwiczenie 11

Uzasadnij prawdziwość tezy twierdzenia 10.

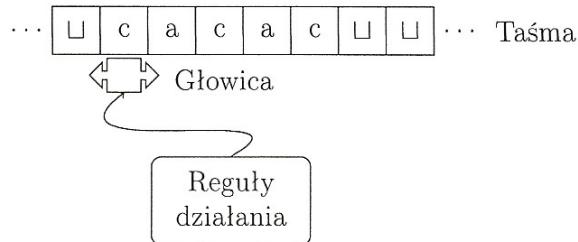


## Wykład 4

Poznaliśmy do tej pory dwa uniwersalne modele obliczeń: maszyny licznikowe oraz funkcje częściowo rekurencyjne. Wykazaliśmy również ich równoważność w sensie obliczeniowym. Bieżący wykład poświęcony będzie kolejnemu modelowi – maszynie Turinga. Dowód równoważności maszyn licznikowych oraz maszyn Turinga zostanie zaprezentowany w czasie kolejnego wykładu.

### Maszyna Turinga

Maszyna Turinga jest modelem obliczeń zaproponowanym przez Alana Turингa w 1936 roku. Model ten podobny jest do automatu skończonego, jednak w odróżnieniu od niego dysponuje nieograniczoną pamięcią o swobodnym dostępie. Może być więc modelem komputera służącego do ogólnych zastosowań. Co ciekawe, na mocy tezy Churcha-Turinga, maszyna ta może obliczyć wszystko, co jest w stanie obliczyć prawdziwy komputer.



Rysunek 6: Schemat maszyny Turinga

Maszyna Turinga składa się taśmy stanowiącej pamięć, głowicy czytającej/piszącej oraz zbioru reguł określających jej działanie. Taśma jest nieskończona (nieograniczona w lewo oraz w prawo) i składa się z komórek. Każda komórka taśmy może przechowywać pojedynczą literę. Wszystkie komórki, poza skończoną liczbą, są puste i zawierają symbol pusty □. Głowica czytająco/pisząca może poruszać się nad taśmą w obu kierunkach, czytać z niej oraz zapisywać na niej. W ustalonym momencie głowica jest w stanie przeczytać lub zapisać wyłącznie pojedynczą komórkę taśmy. Przed rozpoczęciem działania maszyny taśma zawiera wyłącznie dane wejściowe, zaś głowica czytająco/pisząca znajduje się nad pierwszą literą wejścia. Przejście maszyny w stan końcowy (zarówno akceptujący jak i odrzucający) powoduje natychmiastowe zakończenie jej działania.



### Przykład 7

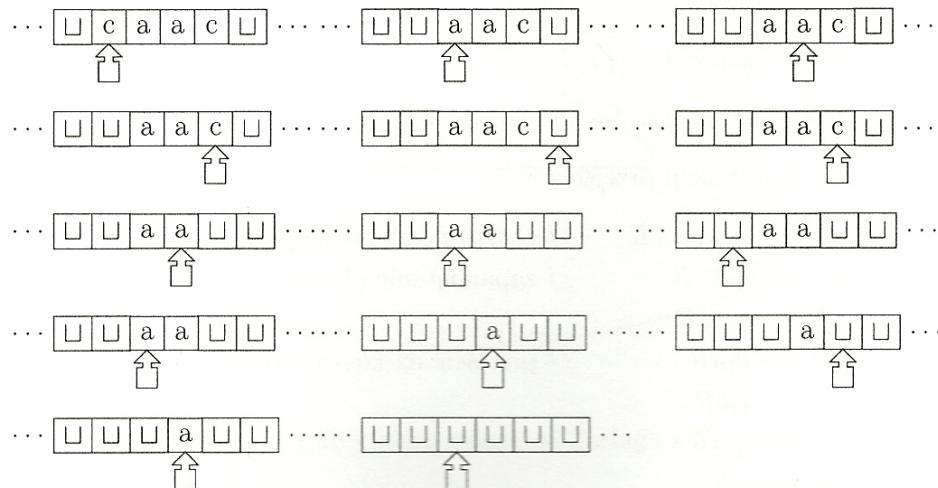
Maszyna Turinga akceptująca język palindromów.

Maszyna czyta słowo wejściowe zapisane na taśmie. Akceptuje je, jeśli jest ono palindromem, oraz odrzuca w przeciwnym przypadku.

Opis działania:

- Jeśli taśma jest pusta zakończ w stanie akceptującym.
- Usuń i zapamiętaj literę z lewego końca słowa (usuniętej literze odpowiada stan maszyny).
- Znajdź prawy koniec słowa.
- Jeśli ostatnia litera słowa jest taka sama jak pierwsza, usuń ją i wróć na początek słowa.
- Jeśli ostatnia litera słowa jest inna niż pierwsza, zakończ w stanie nie-akceptującym.
- Jeśli pierwsza litera była jedyną literą na taśmie, zakończ w stanie akceptującym.

Przykład działania:



Powyższy opis maszyny jest wyłącznie zarysem sposobu jej działania. Aby uzyskać dokładny opis trzeba przedstawić jej formalną definicję.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY

Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych” realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

### Definicja 12

*Maszyną Turinga nazywamy siódemkę uporządkowaną*

$$MT = (Q, \Sigma, \Gamma, \delta, q_0, q_{ACC}, q_{REJ}),$$

gdzie:

- $Q$  jest skończonym zbiorem stanów
- $\Sigma$  jest alfabetem wejściowym (nie zawierającym symbolu pustego  $\sqcup$ )
- $\Gamma$  jest alfabetem taśmy ( $\sqcup \in \Gamma$  oraz  $\Sigma \subset \Gamma$ )
- $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$  jest funkcją przejścia
- $q_0 \in Q$  jest wyróżnionym stanem początkowym
- $q_{ACC}$  jest wyróżnionym stanem akceptującym
- $q_{REJ}$  jest wyróżnionym stanem odrzucającym ( $q_{ACC} \neq q_{REJ}$ )

### Przykład 8

Opis formalny maszyny Turinga M akceptującej język palindromów.

- alfabet wejściowy:  $\Sigma = \{a, b\}$
- alfabet taśmy:  $\Gamma = \{a, b, \sqcup\}$
- zbiór stanów:  $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_{ACC}, q_{REJ}\}$
- definicja funkcji przejścia  $\delta$ :

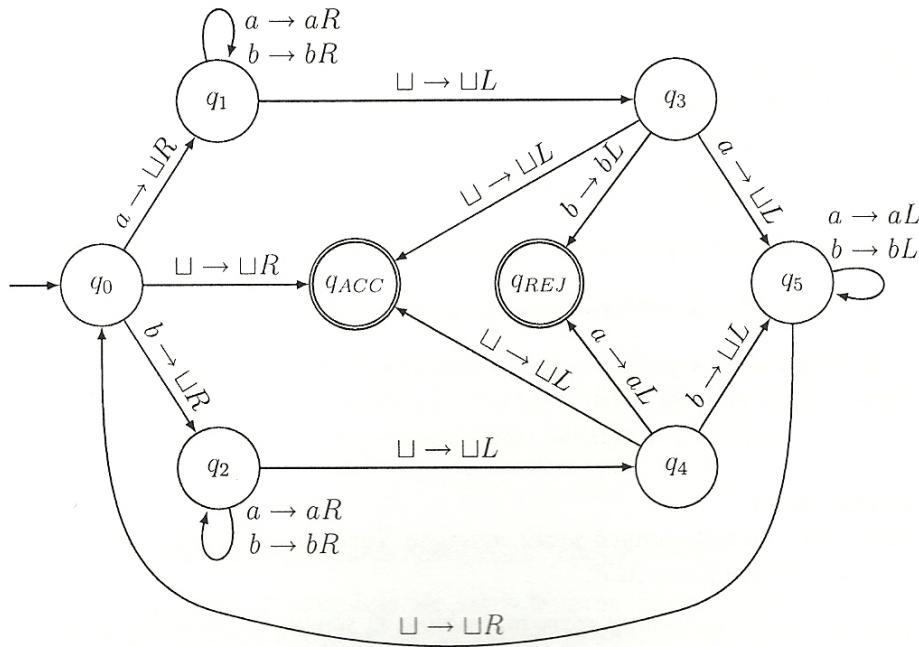
$$\begin{aligned}
 q_0 \sqcup &\longrightarrow q_{ACC} \sqcup R && // \text{ akceptujemy słowo puste} \\
 q_0 a &\longrightarrow q_1 \sqcup R && // \text{ zapamiętanie pierwszej litery} \\
 q_0 b &\longrightarrow q_2 \sqcup R \\
 q_1 a &\longrightarrow q_1 a R && // \text{ przejście na koniec słowa} \\
 q_1 b &\longrightarrow q_1 b R \\
 q_2 a &\longrightarrow q_2 a R \\
 q_2 b &\longrightarrow q_2 b R \\
 q_1 \sqcup &\longrightarrow q_3 \sqcup L && // \text{ sprawdzenie ostatniej litery} \\
 q_2 \sqcup &\longrightarrow q_4 \sqcup L \\
 q_3 b &\longrightarrow q_{REJ} b L && // \text{ odrzucenie w przypadku niezgodności} \\
 q_4 a &\longrightarrow q_{REJ} a L
 \end{aligned}$$



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

$$\begin{aligned}
 q_3a &\longrightarrow q_5 \sqcup L && // \text{ powrót na początek słowa} \\
 q_4b &\longrightarrow q_5 \sqcup L \\
 q_3 \sqcup &\longrightarrow q_{ACC} \sqcup L && // \text{ zaakceptowanie pojedynczej litery} \\
 q_4 \sqcup &\longrightarrow q_{ACC} \sqcup L \\
 q_5a &\longrightarrow q_5aL && // \text{ powrót na początek słowa} \\
 q_5b &\longrightarrow q_5bL \\
 q_5 \sqcup &\longrightarrow q_0 \sqcup R && // \text{ rozpoczęcie kolejnej iteracji}
 \end{aligned}$$

Zbiór stanów maszyny Turinga oraz definicja funkcji przejścia mogą być również przedstawione za pomocą *diagramu stanów* opisującego jej działanie w sposób graficzny.



Rysunek 7: Diagram stanów maszyny Turinga z Przykładu 8.

### Konfiguracja maszyny Turinga

Niech  $a, b, c \in \Gamma$  oraz  $u, v \in \Gamma^*$ . Konfiguracją maszyny Turinga  $MT$  nazywamy trójkę (stan, pozycja głowicy, zawartość taśmy). Zapis  $uq_iv$  oznacza, że



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych” realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

maszyna jest w stanie  $q_i$ , na taśmie znajduje się słowo  $uv$ , zaś głowica czytająco/pisząca znajduje się nad pierwszą literą słowa  $v$ . Konfiguracją początkową maszyny  $MT$  dla wejścia  $w$  nazywamy konfigurację postaci  $q_0w$ . Konfiguracją akceptującą (odpowiednio odrzucającą) maszyny  $MT$  nazywamy dowolną konfigurację, dla której maszyna  $MT$  znajduje się w stanie akceptującym (odpowiednio odrzucającym). Konfiguracją końcową maszyny  $MT$  nazywamy dowolną konfigurację akceptującą lub odrzucającą.

Powiemy, że maszyna Turinga przechodzi z konfiguracji  $C_1$  do konfiguracji  $C_2$  (ozn.  $C_1 \rightarrow C_2$ ), jeśli prawdziwy jest jeden z warunków:

- $C_1 = uaq_i bv, C_2 = uq_j acv$  oraz  $\delta(q_i, b) = (q_j, c, L)$ .
- $C_1 = uaq_i bv, C_2 = uacq_j v$  oraz  $\delta(q_i, b) = (q_j, c, R)$ .

### Definicja 13

Powiemy, że maszyna Turinga  $MT$  akceptuje (odrzuca) słowo  $w$ , jeśli istnieje taki ciąg jej konfiguracji  $C_1, C_2, \dots, C_n$ , że:

- $C_1$  jest konfiguracją początkową  $MT$  dla słowa  $w$
- $\forall_{0 < i < n}$  z konfiguracji  $C_i$  maszyna  $MT$  przechodzi do konfiguracji  $C_{i+1}$
- $C_n$  jest konfiguracją akceptującą (odrzucającą)

Ciąg konfiguracji  $C_1, C_2, \dots, C_n$  nazywamy akceptującą (odrzucającą) historią obliczeń maszyny  $MT$ .

### Definicja 14

Zbiór słów akceptowanych przez maszynę Turinga  $MT$  nazywamy językiem rozpoznawanym przez  $MT$ .

Język  $L \subseteq \Sigma^*$  nazywamy rozpoznawalnym w sensie Turinga, jeżeli jest rozpoznawany przez pewną maszynę Turinga.

Zauważmy, że dla ustalonego słowa wejściowego  $w \in \Sigma^*$  obliczenia maszyny Turinga  $MT$  mogą przebiegać na trzy wykluczające się sposoby:

1.  $MT$  kończy obliczenia w stanie akceptującym.
2.  $MT$  kończy obliczenia w stanie odrzucającym.
3.  $MT$  nigdy nie kończy obliczeń.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Chcielibyśmy wyróżnić maszyny zatrzymujące się dla każdego wejścia.

### Definicja 15

Powiemy, że maszyna Turinga  $MT$  rozstrzyga język  $L \subseteq \Sigma^*$ , jeżeli dla każdego słowa  $w \in L$  zatrzymuje się w stanie akceptującym, natomiast dla każdego słowa  $w \notin L$  zatrzymuje się w stanie odrzucającym.

Język  $L \subseteq \Sigma^*$  nazywamy rozstrzygalnym w sensie Turinga, jeżeli jest rozstrzygany przez pewną maszynę Turinga.

### Uwaga

Zauważmy, że każdy język rozstrzygalny w sensie Turinga jest również rozpoznawalny w sensie Turinga. Implikacja w druga stronę nie jest prawdziwa.

Opisana w Przykładzie 8 maszyna Turinga rozstrzyga język palindromów. Może ona zostać zamieniona na maszynę (tylko) rozpoznającą, jeśli po stwierdzeniu niezgodności pierwszej i ostatniej litery słowa znajdującego się na taśmie wpadnie w nieskończoną pętlę zamiast kończyć działanie w stanie odrzucającym.

Działanie maszyny Turinga możemy rozumieć na dwa sposoby:

- akceptowanie/odrzucanie języków (podobnie jak automaty)
- obliczanie wartości funkcji (podobnie jak maszyny licznikowe)

Dla maszyny obliczającej wartość funkcji znalezienie się w stanie akceptującym oznacza zakończenie obliczeń i zwrócenie wyniku. Zatrzymanie w stanie odrzucającym oznacza natomiast odrzucenie niepoprawnych danych wejściowych.

### Przykład 9

Maszyna Turinga licząca funkcję następnika:  $f(x) = x + 1$ .

Na taśmie maszyny  $M$  znajduje się zapis binarny liczby naturalnej  $n$ . Po zakończeniu działania maszyny  $M$  na taśmie powinien znajdować się zapis binarny liczby  $n + 1$ .

Opis działania:

- Znajdź prawy koniec zapisu binarnego liczby.
- Przesuwaj głowicę w lewo jednocześnie zamieniając napotkane 1 na 0.
- Pierwsze od prawej 0 zmień na 1 i zakończ działanie w stanie akceptującym.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
FUNDUSZ SPOŁECZNY



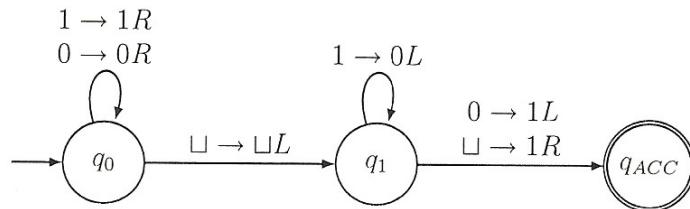
Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych” realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

- Po osiągnięciu lewego końca słowa (w zapisie binarnym liczby nie występowały 0) dodaj 1 na jego początek.

Definicja funkcji przejścia:

$$\begin{aligned}
 q_00 &\longrightarrow q_00R && // szukaj prawego końca \\
 q_01 &\longrightarrow q_01R \\
 q_0\sqcup &\longrightarrow q_1\sqcup L && // stan dodawania 1 \\
 q_11 &\longrightarrow q_10L && // zamiana zer na 1 \\
 q_10 &\longrightarrow q_{ACC}1L && // pierwsze od prawej 0 \\
 q_1\sqcup &\longrightarrow q_{ACC}1L && // same 1
 \end{aligned}$$

Diagram stanów:



### Uwagi do przykładu

Zauważmy, że opisana powyżej maszyna akceptuje zapis liczby binarnej z dowolną liczbą wiodących zer. Ponadto, słowo puste jest traktowane jak prawidłowa reprezentacja liczby 0. Zapewnienie, aby w takich przypadkach maszyna zatrzymywała się w stanie odrzucającym, wymaga niewielkiej modyfikacji jej konstrukcji.

### Ćwiczenie 12

Popraw opisaną powyżej maszynę Turinga tak, aby odrzucała nieprawidłowo sformatowane dane wejściowe.

### Uwagi dotyczące stanów końcowych

W niektórych wypadkach, wygodne jest dodatkowe założenie jednego z późniejszych wymagań, dotyczących sposobu działania maszyny Turinga:

- Taśma maszyny jest pusta, kiedy zatrzyma się ona w stanie akceptującym – wymaga to rozszerzenia definicji maszyny o instrukcje czyszczenia taśmy po zakończeniu obliczeń.
- Maszyna kończąc działanie drukuje na taśmie odpowiedź *TAK* lub *NIE* – wymaga to rozszerzenia definicji maszyny o instrukcje drukujące odpowiedź.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

- Po zakończeniu działania przez maszynę obliczającą wartość funkcji jej głowica znajduje się nad pierwszym znakiem wyniku – wymaga to rozszerzenia definicji maszyny o instrukcję powrotu na początek wyniku.

Żadne z opisywanych powyżej rozszerzeń nie ma wpływu na moc obliczeniową wyjściowej maszyny.



## Wykład 5

### Obliczalność w sensie Turinga

W czasie wykładu zaprezentowanych zostanie kilka wariantów maszyny Turinga równoważnych podstawowej definicji przedstawionej w czasie poprzedniego wykładu. Udowodniona również zostanie niezależność mocy obliczeniowej maszyny Turinga od wariantu jej definicji. Własność powyższa nosi nazwę odporności modelu. Aby pokazać, że dwa modele obliczeń są równoważne, należy udowodnić, że potrafimy symulować jeden model za pomocą drugiego.

#### Maszyna Turinga z ruchem „stój w miejscu”

Najprostszą modyfikacją podstawowej definicji maszyny Turinga jest dodanie instrukcji umożliwiających zmianę stanu i zawartości taśmy przy jednoczesnym braku ruchu głowicy czytająco/piszącej.

Niech  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{ACC}, q_{REJ})$  będzie maszyną Turinga z funkcją przejścia określona jako

$$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R, S\},$$

gdzie  $L, R, S$  oznaczają ruch głowicy odpowiednio w lewo, w prawo lub pozostanie w tej samej pozycji.

Zauważmy, że maszyna dopuszczająca ruch „stój w miejscu” może symulować działanie maszyny nie dopuszczającej tego ruchu. Pozostaje więc udowodnić, że możliwa jest symulacja w drugą stronę.

#### Twierdzenie 11

Dla dowolnej maszyny Turinga  $M$  dopuszczającej ruch  $S$  (stój w miejscu) głowicy czytająco/piszącej istnieje równoważna jej maszyna Turinga  $M'$  nie dopuszczająca ruchu  $S$ .

#### Dowód

Symulacja ruchu  $S$  przez maszynę  $M'$  polega na wykonaniu dwóch następujących bezpośrednio po sobie ruchów  $L+R$  lub  $R+L$ . Pozostałe zasady działania maszyny  $M$  nie ulegają zmianie.  $\square$



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



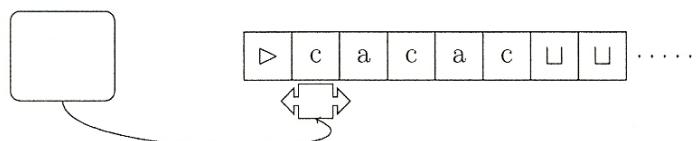
UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Maszyna Turinga z taśmą jednostronnie ograniczoną

Bardzo często spotykanym wariantem maszyny Turinga jest maszyna z taśmą ograniczoną z jednej strony. Na lewym końcu taśmy znajduje się specjalny symbol  $\triangleright$  oznaczający jej koniec. Każda instrukcja wymagająca ruchu w lewo poza symbol  $\triangleright$  jest ignorowana.



Zauważmy, że obliczenie każdej maszyny Turinga z taśmą jednostronnie ograniczoną może być wykonane na maszynie z taśmą dwustronnie nieograniczoną (część taśmy znajdująca się na lewo od symbolu  $\triangleright$  nie jest używana). Wystarczy zatem udowodnić, że obliczenia maszyny z taśmą dwustronnie nieograniczoną mogą być symulowane na maszynie z taśmą jednostronnie ograniczoną.

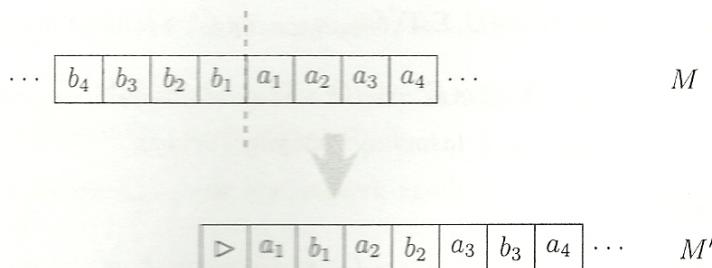
### Twierdzenie 12

Dla dowolnej maszyny Turinga  $M$  z taśmą dwustronnie nieograniczoną istnieje równoważna jej maszyna Turinga  $M'$  z taśmą jednostronnie ograniczoną.

### Dowód

Symulacja obliczeń maszyny  $M$  przez maszynę  $M'$  przebiega następująco:

- Ustalamy dowolny punkt cięcia nieskończonej taśmy maszyny  $M$ .
- Zawartość taśmy maszyny  $M$  po prawej stronie punktu cięcia zapisywana jest w „nieparzystych” komórkach taśmy maszyny  $M'$ , natomiast rewers zawartości po jego lewej stronie zapisywany jest w komórkach „parzystych”.





KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



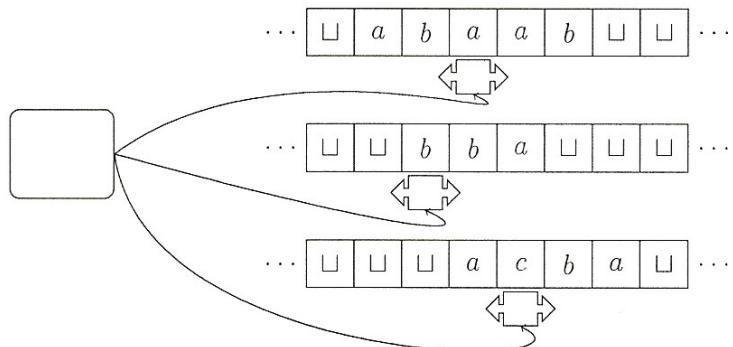
Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych” realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

- Każdy ruch  $L, R$  maszyny  $M$  odpowiada dwóm ruchom maszyny  $M'$  (ruch w komórkach parzystych odbywa się w przeciwnym kierunku).
- Każdy ruch poza symbol końca taśmy  $\triangleright$  powoduje zmianę „parzystości” czytanych/zapisywanych komórek.
- Pozostałe reguły działania maszyny  $M$  nie ulegają zmianie.

□

### Maszyna Turinga z wieloma taśmami

Kolejnym rozszerzeniem maszyny Turinga jest wykorzystanie więcej niż jednej taśmy. Każda taśma ma oddzielną głowicę czytającą/piszącą, która porusza się niezależnie od innych. W konfiguracji początkowej dane wejściowe znajdują się na pierwszej taśmie, pozostałe taśmy są puste. Funkcja przejścia musi zostać zmodyfikowana tak, aby umożliwić jednoczesne przesuwanie wielu głowic oraz czytanie i pisanie na wielu taśmach.



Rysunek 8: Schemat trzytaśmowej maszyny Turinga

### Definicja 16

Maszynę Turinga  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{ACC}, q_{REJ})$  z funkcją przejścia określono jako:

$$\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R, S\}^k,$$

gdzie  $k > 0$ , nazywamy  $k$ -taśmową maszyną Turinga.

Wyrażenie

$$\delta(q_i, a_1, a_2, \dots, a_k) = (q_j, b_1, b_2, \dots, b_k, L, R, \dots, R)$$



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

oznacza, że maszyna Turinga  $M$  będąc w stanie  $q_i$  i widząc na kolejnych taśmach symbole  $a_1, \dots, a_k$  przechodzi w stan  $q_j$ , zapisuje na kolejnych taśmach symbole  $b_1, \dots, b_k$  oraz przesuwa każdą z głowic w odpowiednim kierunku.

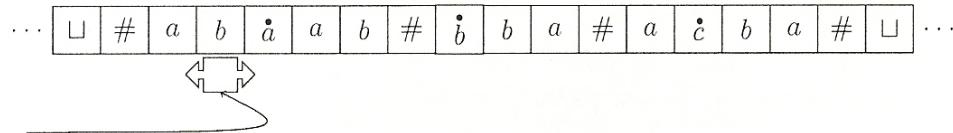
Każda jednotaśmowa maszyna Turinga jest szczególnym przypadkiem maszyny wielotaśmowej. Pozostaje więc udowodnić, że każdą wielotaśmową maszynę Turinga można symulować na maszynie jednotaśmowej.

### Twierdzenie 13

*Dla dowolnej wielotaśmowej maszyny Turinga  $M$  istnieje równoważna jej jednotaśmowa maszyna Turinga  $M'$ .*

#### Dowód

Działanie maszyny  $M$  będziemy symulować na maszynie  $M'$  za pomocą wirtualnych taśm i głowic. Zawartość  $k$  taśm maszyny  $M$  zapisujemy sekwencyjnie na taśmie maszyny  $M'$  rozdzielaając zawartość poszczególnych taśm specjalnym symbolem  $\#$  (nie należącym do alfabetu taśmy maszyny  $M$ ). Ponadto, dla każdej litery  $a \in \Gamma$  dodajemy do alfabetu taśmy symbol  $\dot{a}$  oznaczający pozycję głowicy na taśmie.



Rysunek 9: Konfiguracja jednotaśmowej maszyny Turinga symulującej działanie maszyny trzytaśmowej przedstawionej na Rysunku 8

Dla słowa wejściowego  $w = w_1 w_2 \dots w_n$  działanie maszyny  $M'$  przebiega według następującego schematu:

- Przekształcenie zawartości taśmy w zapis reprezentujący zawartość  $k$  taśm maszyny  $M$ :

$$\square \# \dot{w}_1 w_2 \dots w_n \# \dot{\square} \# \dots \# \dot{\square} \# \square$$

- Przeczytanie zawartości taśmy między skrajnymi znakami  $\#$  w celu odczytania pozycji głowic wirtualnych.
- Zmiana zawartości taśm wirtualnych zgodnie z funkcją przejścia maszyny  $M$ .



- Zapisanie znaku w miejscu symbolu specjalnego # (rozszerzenie długości taśmy wirtualnej) jest realizowane przez kopiowanie „w prawo” całej zawartości taśmy maszyny  $M'$  znajdującej się za symbolem #.

□

## Niedeterministyczna maszyna Turinga

Niedeterministyczna maszyna Turinga jest rozszerzeniem modelu podstawowego o możliwość wyboru więcej niż jednej instrukcji dla ustalonej konfiguracji maszyny (stan+litera). Obliczenie maszyny niedeterministycznej jest więc drzewem zawierającym wszystkie możliwe ścieżki jej obliczeń. Chociaż model ten wydaje się daleki od rzeczywistości, w teorii obliczalności odgrywa bardzo ważną rolę.

### Definicja 17

*Maszynę Turinga  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{ACC}, q_{REJ})$  z relacją przejścia określona:*

$$\delta \subset (Q \times \Gamma) \times (Q \times \Gamma \times \{L, R, S\}),$$

*nazywamy niedeterministyczną maszyną Turinga.*

Dla tak zdefiniowanej maszyny Turinga również pojęcia rozpoznawania i rozstrzygania języków muszą zostać zmodyfikowane.

### Definicja 18

*Powiemy, że niedeterministyczna maszyna Turinga  $M$  rozpoznaje język  $L$ , jeśli dla dowolnego słowa  $w \in L$  istnieje skończona ścieżka jej obliczeń kończąca się w stanie akceptującym.*

### Definicja 19

*Powiemy, że niedeterministyczna maszyna Turinga  $M$  rozstrzyga język  $L$ , jeśli dla dowolnego słowa  $w \in \Sigma^*$  każda ścieżka jej obliczeń jest skończona, oraz  $w \in L$  wtedy i tylko wtedy, gdy istnieje ścieżka obliczeń kończąca się w stanie akceptującym.*

### Uwaga

Zauważmy, że w przypadku maszyn niedeterministycznych do zaakceptowania słowa wystarczy istnienie jednej ścieżki obliczeń kończącej się w stanie akceptującym. Natomiast aby słowo było odrzucane, żadna ścieżka jej obliczeń nie może prowadzić do stanu akceptującego.

Mogłoby się wydawać, że takie rozszerzenie modelu obliczeń powinno znaczco wpływać na jego moc obliczeniową. Okazuje się jednak, że maszyny



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

deterministyczne i niedeterministyczne rozstrzygają dokładnie tę samą klasę języków. Zauważmy, że każda deterministyczna maszyna Turinga jest szczególnym przypadkiem maszyny niedeterministycznej. Wystarczy zatem pokazać, że obliczenia dowolnej maszyny niedeterministycznej mogą być symulowane za pomocą maszyny deterministycznej.

#### Twierdzenie 14

Dla dowolnej niedeterministycznej jednotaśmowej maszyny Turinga  $M$  istnieje równoważna jej deterministyczna trzytaśmowa maszyna Turinga  $M'$ .

#### Dowód

Obliczenie maszyny  $M$  dla słowa  $w$  ma postać drzewa. Każdy jego rozgałęzienie reprezentuje niedeterministyczne rozgałęzienie ścieżki obliczeń. Każdy węzeł drzewa odpowiada pewnej konfiguracji maszyny  $M$  (korzeń odpowiada konfiguracji początkowej  $M$ ). Symulacja obliczeń maszyny  $M$  na maszynie  $M'$  polega na przejrzeniu drzewa obliczeń maszyny  $M$  (metodą BFS) i zakończeniu działania po osiągnięciu stanu końcowego na dowolnej ścieżce.

Taśma pierwsza zawiera słowo wejściowe (tylko do odczytu). Taśma druga zawiera kopię zawartości taśmy maszyny  $M$  będącej na ustalonej ścieżce obliczeń. Taśma trzecia zawiera informacje o pozycji maszyny  $M'$  w drzewie niedeterministycznych obliczeń maszyny  $M$ . Niech  $s$  będzie rozmiarem maksymalnego zbioru możliwych wyborów kolejnego kroku obliczeń maszyny  $M$  wynikającego określenia jej relacji przejścia. Każdy węzeł drzewa obliczeń  $M$  jest adresowany za pomocą słowa nad alfabetem  $\Sigma_s = \{1, 2, \dots, s\}$ . Zbiór adresów będziemy przeglądać w porządku (będącym modyfikacją standartowego porządku leksykograficznego), w którym słowa krótsze poprzedzają dłuższe, zaś słowa równej długości przeglądamy alfabetycznie:

$$\{\varepsilon, 1, 2, \dots, s, 11, 12, \dots, 1s, 21, 22, \dots, ss, 111, 112, \dots\}$$

Symulacja obliczeń maszyny  $M$  przez maszynę  $M'$  przebiega następująco:

1. Pierwsza taśma zawiera słowo wejściowe  $w$ , pozostałe taśmy są puste.
2. Kopiowanie słowa  $w$  z taśmy pierwszej na drugą.
3. Symulacja pojedynczej ścieżki obliczeń maszyny  $M$  na drugiej taśmie.
  - Wybór kroku spośród wielu możliwych dokonywany jest na podstawie kolejnego symbolu odczytanego z taśmy trzeciej.
  - Po osiągnięciu stanu końcowego przez maszynę  $M$  zakończenie działania w stanie akceptującym/odrzucającym.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych” realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

- Po osiągnięciu końca słowa znajdującego się na taśmie trzeciej lub napotkaniu niedozwolonego ruchu porzucenie aktualnej ścieżki obliczeń i powrót do punktu (2).
- 4. Zastąpienie słowa na taśmie trzeciej jego następciem w porządku ustalonym powyżej.
- 5. Powrót do punktu (2) i symulacja kolejnej ścieżki obliczeń.

Zauważmy, że istotne jest założenie o przeglądaniu drzewa obliczeń symulowanej maszyny niedeterministycznej metodą BFS. Daje to gwarancję zbadań wszystkich ścieżek ustalonej długości przed rozpatrzeniem ścieżek dłuższych. Zatem jeśli w rozważanym drzewie obliczeń istnieje ścieżka skończona, na pewno zostanie znaleziona. □

## Obliczalność w sensie Turinga vs. ML-obliczalność

Jako podsumowanie bieżącego wykładu udowodnimy równoważność poznanych wcześniej uniwersalnych modeli obliczeń z maszynami Turinga. Rozpoczniemy od pokazania, że działanie maszyny Turinga może być symulowane przez program na maszynę licznikową.

### Twierdzenie 15

Dla dowolnej maszyny Turinga  $MT$  z taśmą jednostronnie ograniczoną istnieje program  $P$  na maszynę licznikową  $ML$  symulujący jej działanie.

### Dowód

Symulacja działania maszyny Turinga  $MT$  przez program  $P$  na maszynę licznikową  $ML$  przebiega następująco:

- Ustalone jest kodowanie zbioru stanów  $Q$  oraz alfabetu taśmy  $\Gamma$  maszyny  $MT$ .
- Konfiguracja maszyny Turinga  $MT$  postaci  $uqv$  kodowana jest jako piątka  $(|u|, |v|, q, \text{kod}(u), \text{kod}(v))$ , gdzie  $\text{kod}$  jest dowolnym różnowartościowym kodowaniem.
- Dla każdej instrukcji maszyny  $MT$  tworzona jest procedura na maszynę licznikową  $ML$  realizującą jej wykonanie.
- Program  $P$  ma postać pętli zawierającej wszystkie procedury symulujące instrukcje maszyny  $MT$ .



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

- Każda iteracja pętli polega na wykonaniu instrukcji zgodnej z funkcją przejścia maszyny  $MT$ .

□

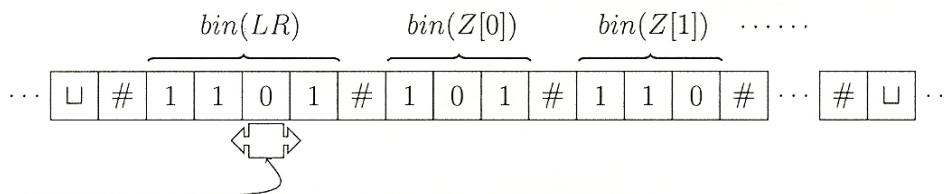
Zauważmy, że na mocy faktów prezentowanych wcześniej, oraz Twierdzenia 15, działanie dowolnej maszyny Turinga może być symulowane za pomocą programu na maszynę licznikową. Pozostało tylko udowodnić, że działanie dowolnego programu na maszynę licznikową może być symulowane za pomocą maszyny Turinga.

### Twierdzenie 16

Dla dowolnego programu  $P$  na maszynę licznikową  $ML$  istnieje (dwutaśmowa) maszyna Turinga  $MT$  symulująca jego działanie.

#### Dowód

Pierwsza taśma maszyny  $MT$  zawiera binarne reprezentacje zawartości rejestrów maszyny  $ML$  oraz licznika rozkazów. Druga taśma wykorzystywana jest do obliczeń pomocniczych.



Rysunek 10: Zawartość pierwszej taśmy maszyny Turinga symulującej wykonanie programu na maszynie licznikowej.

Wykonanie instrukcji programu  $P$  przez maszynę  $MT$  przebiega następująco:

- $Z(m)$  – Odnalezienie na pierwszej taśmie kodu zawartości rejestru o adresie  $m$  i zapisanie w jego miejsce wartości 0.
- $S(m)$  – Odnalezienie na pierwszej taśmie kodu zawartości rejestru o adresie  $m$  i zwiększenie go o 1.
- $T(m, n)$  – Odnalezienie na pierwszej taśmie kodu zawartości rejestru  $m$  i skopiowanie go na drugą taśmę. Następnie odnalezienie na pierwszej taśmie kodu zawartości rejestru  $n$  i zastąpienie go kodem zapamiętanym na drugiej taśmie.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

- $I(m, n, q)$  – Porównanie kodów zawartości rejestrów o adresach  $m$  oraz  $n$  (np. przy pomocy drugiej taśmy). W przypadku pozytywnego wyniku porównania, odnalezienie na pierwszej taśmie kodu licznika rozkazów i zamiana jego wartości na kod  $q$ .
- Po wykonaniu każdej instrukcji (poza skokiem) zwiększenie o 1 kodu licznika rozkazów.

Zauważmy, że adresy rejestrów używanych przez maszynę licznikową mogą być dowolnie duże. Nie ma więc możliwości zapamiętania ich w stanach maszyny Turinga  $MT$ . Maszyna  $MT$  może jednak zapamiętywać adresy (np. ich reprezentacje binarne) na drugiej taśmie.

□

W czasie jednego z poprzednich wykładów udowodniliśmy równość zbioru funkcji obliczalnych przez maszyny licznikowe ze zbiorem funkcji częściowo rekurencyjnych. Powyższe twierdzenie umożliwia nam sformułowanie następującego wniosku:

### **Wniosek**

Funkcje częściowo rekurencyjne są obliczalne w sensie Turinga.



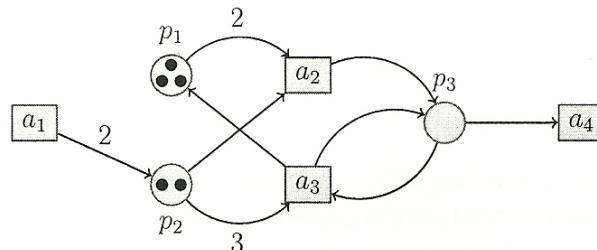
## Wykład 6

### Sieci liczące

Kolejnym modelem obliczeń, z którym zapoznamy się w czasie wykładu będą sieci liczące. Zaprezentowane zostaną dwa modele: PT-sieci oraz ich rozszerzenie – sieci inhibitorowe. Przekonamy się również, że rozszerzenie to jest bardzo istotne.

#### PT-sieci

Sieć Petriego (PT-sieć) składa się z uporządkowanego zbioru miejsc: oraz zbioru akcji: połączonych łukami  $\longrightarrow$  z określoną wagą. Domyślnie, wagi o wartości 1 nie są zaznaczane w reprezentacji graficznej sieci.



Rysunek 11: Przykładowa sieć Petriego

Akcja  $a$  jest umożliwiona jeśli każde jej miejsce wejściowe  $p_i$  zawiera liczbę żetonów równą co najmniej wadze łuku  $p_i \longrightarrow a$ . W każdym kroku obliczeń mogą być wykonywane wyłącznie akcje umożliwiające. Wykonanie umożliwiającej akcji  $a$  polega na zdjęciu z każdego jej miejsca wejściowego  $p_i$  liczby żetonów równej wadze łuku  $p_i \longrightarrow a$  oraz umieszczeniu na każdym jej miejscu wyjściowym  $p_j$  żetonów w liczbie równej wadze łuku  $a \longrightarrow p_j$ . W szczególnych przypadkach akcja może nie mieć miejsc wejściowych (zawsze umożliwiona) lub nie mieć miejsc wyjściowych. Przykładowo, w sieci prezentowanej na rysunku 11, akcje  $a_1$  i  $a_2$  są możliwe, natomiast akcje  $a_3$  i  $a_4$  nie są możliwe.

Markingiem w sieci  $S$  nazywamy wektor  $M = [p_1, \dots, p_k]$  opisujący zawartość poszczególnych miejsc sieci  $S$ . Powiemy, że marking  $M_1$  jest osiągalny z markingiem  $M_2$  (oznaczenie  $M_2 \Rightarrow M_1$ ) jeśli istnieje ciąg wykonan akcji prowadzących z  $M_2$  do  $M_1$ .



Definicja formalna:

**Definicja 20**

Siecią Petriego (PT-siecią) nazywamy czwórkę  $S = (P, T, F, M_0)$ , gdzie  $P$  jest skończonym zbiorem miejsc,  $T$  jest skończonym zbiorem akcji (tranzycji),  $F : P \times T \cup T \times P \rightarrow \mathbb{N}$  jest funkcją wagową, zaś  $M_0$  jest wyróżnionym markingiem początkowym.

Siecią liczącą nazywamy dowolną PT-sieć z wyróżnionym zbiorem miejsc argumentowych (nie mających wejść), wyróżnionym zbiorem miejsc roboczych oraz wyróżnionym miejscem wynikowym (nie mającym wyjścia). Markingi w sieciach liczących zapisujemy w postaci

$$M = [\text{miejsc argumentowe} ; \text{miejsc robocze} ; \text{miejscie wynikowe}].$$

Markingiem początkowym sieci liczącej  $S$  nazywamy dowolny marking postaci  $M = [x_1, \dots, x_n; 0, \dots, 0; 0]$  natomiast markingiem końcowym dowolny marking postaci  $[0, \dots, 0; 0, \dots, 0; y]$ .

Powiemy, że sieć  $S$  oblicza funkcję  $f : \mathbb{N}^n \rightarrow \mathbb{N}$ , jeśli

$$[x_1, \dots, x_n; 0, \dots, 0; 0] \implies [0, \dots, 0; 0, \dots, 0; y] \iff f(x_1, \dots, x_n) = y$$

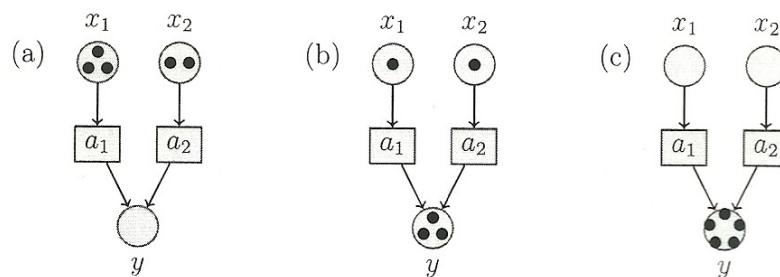
(marking końcowy jest osiągalny z markingu początkowego, miejsce wynikowe zawiera właściwą wartość zaś pozostałe miejsca są puste).

**Uwaga**

W sieci liczącej  $S$  może istnieć wiele ścieżek obliczeń nie prowadzących do markingu końcowego.

**Przykład 10**

Sieć licząca sumę  $y = x_1 + x_2$ .



Na rysunku (a) przedstawiony został marking początkowy sieci, na rysunku (c) – jej marking końcowy, natomiast rysunek (b) obrazuje jeden z pośrednich kroków obliczeń.



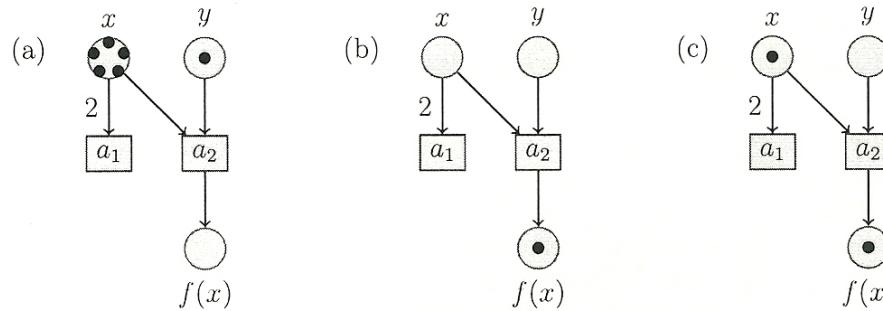
Akcje  $a_1$  oraz  $a_2$  kopią żetony z miejsc  $x_1$  oraz  $x_2$  na miejsce  $y$ . Zatem po zakończeniu działania sieci miejsce  $y$  będzie zawierało sumę żetonów z miejsc  $x_1$  oraz  $x_2$ . Zauważmy, że akcje  $a_1$  oraz  $a_2$  mogą się wykonywać w dowolnym porządku, jednak tylko do momentu kiedy ich miejsca wejściowe nie staną się puste. Ponadto, w przypadku powyższej sieci każda ścieżka obliczeń prowadzi do poprawnego markingu końcowego.

### Przykład 11

Sieć sprawdzająca czy liczba  $x$  jest nieparzysta.

Konstruujemy sieć obliczającą następującą funkcję częściową:

$$f(x) = \begin{cases} 1 & \Leftrightarrow x \text{ jest nieparzysta} \\ \text{nieokreślona w przeciwnym przypadku} \end{cases}$$



Na rysunku (a) przedstawiony jest marking początkowy rozważanej sieci. Miejsce wejściowe  $x$  odpowiada argumentowi obliczanej funkcji  $f(x)$ , natomiast miejsce  $y$  jest dodatkowym argumentem zapewniającym, że akcja  $a_2$  wykona się co najwyżej raz. Ponadto, wyłącznie akcja  $a_2$  może umieścić żeton na miejscu wynikowym.

Zauważmy, że każde wykonanie akcji  $a_1$  usuwa dwa żetony z miejsca  $x$ . Może się ona wykonywać dopóki na miejscu  $x$  znajdują się co najmniej dwa żetony. Akcja  $a_2$  natomiast usuwa z miejsca  $x$  dokładnie jeden żeton i w dowolnym obliczeniu rozważanej sieci może się wykonać się co najwyżej raz. Jeśli więc w markingu początkowym liczba żetonów na miejscu  $x$  jest nieparzysta, wykonanie akcji  $a_2$  oraz  $a_1$  w dowolnym porządku prowadzi do oczekiwanej markingu końcowego, przedstawionego na rysunku (b).

Jeśli w markingu początkowym liczba żetonów na miejscu  $x$  jest parzysta, nie będzie możliwe usunięcie wszystkich żetonów z miejsc  $x$  oraz  $y$ . Jeśli akcja  $a_2$  wykona się, nie będzie możliwe usunięcie ostatniego żetonu z miejsca  $x$ . Jeśli natomiast akcja  $a_2$  nie wykona się, po opróżnieniu miejsca  $x$  przez akcję  $a_1$



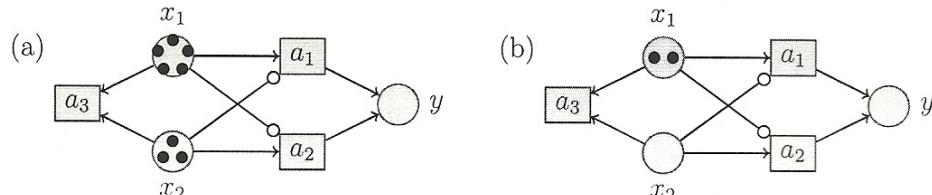
nie będzie możliwe usunięcie żetonu z miejsca  $y$ . W tym przypadku wartość obliczanej funkcji będzie więc nieokreślona. Na rysunku (c) przedstawiona jest sytuacja po wykonaniu akcji  $a_2$ .

## Sieci inhibitorowe

Sieci inhibitorowe są rozszerzeniem PT-sieci. Powstają przez dodanie łuków oznaczanych  $\rightarrow$  prowadzących od miejsc do akcji oraz modyfikację reguł umożliwienia akcji. W sieci inhibitorowej akcja  $a$  jest umożliwiona jeśli każde jej miejsce wejściowe  $p_i$  połączone z  $a$  łukiem  $p_i \rightarrow a$  zawiera liczbę żetonów równą co najmniej wadze łuku  $p_i \rightarrow a$ , zaś każde miejsce wejściowe  $p_j$  połączone z  $a$  łukiem  $p_j \rightarrow a$  nie zawiera żadnego żetonu. Wykonanie akcji oraz obliczanie funkcji w sieciach inhibitorowych określa się identycznie jak w PT-sieciach.

### Przykład 12

Sieć inhibitorowa licząca funkcję  $y = |x_1 - x_2|$ .



Zauważmy, że wykonanie akcji  $a_1$  jest możliwe wyłącznie w przypadku kiedy miejsce  $x_2$  jest puste. Podobnie, warunkiem koniecznym do wykonania akcji  $a_2$  jest pustaść miejsca  $x_1$ . Akcja  $a_3$  jest więc wykonywana dopóki jedno z miejsc  $x_1$  lub  $x_2$  nie stanie się puste (rysunek (a)). Wówczas, dokładnie jedna z akcji  $a_1$  lub  $a_2$  skopiuje pozostałe żetony na miejsce wynikowe (rysunek (b)). Jeśli miejsca  $x_1$  oraz  $x_2$  zawierają tyle samo żetonów lub oba są puste w markingu początkowym, żadna z akcji  $a_1$  oraz  $a_2$  nigdy nie stanie się umożliwiona. Miejsce wynikowe pozostanie puste, co jest prawidłową wartością funkcji w tym przypadku.

## Sieci liczące vs maszyny licznikowe

Moc obliczeniowa PT-sieci jest słabsza od mocy obliczeniowej sieci inhibitorowych. Przykładowo, w PT-sieciach nie jest możliwe obliczenie funkcji  $y = x_1 \cdot x_2$ , co jest możliwe w sieciach inhibitorowych ze względu na możliwość „testowania zera”. Moc obliczeniowa sieci inhibitorowych jest równa mocy obliczeniowej maszyn licznikowych, lub równoważnie mocy obliczeniowej maszyn Turinga.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

### Twierdzenie 17

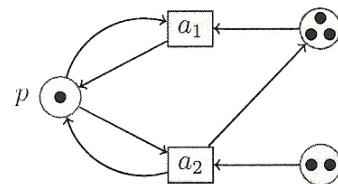
*Działanie dowolnego ML-programu  $P$  może być symulowane za pomocą sieci inhibitorowej.*

#### Dowód

Przypomnijmy, że maszyna licznikowa posiada nieskończenie wiele rejestrów pamięci, jednak w czasie obliczeń dowolnego programu używa tylko ich skończonej liczby. Rejestrom maszyny  $ML$  używanym przez program  $P$  będą odpowiadały wyróżnione miejsca sieci inhibitorowej  $S$ . Miejscami wejściowymi sieci  $S$  będą miejsca odpowiadające rejestrów zawierającym argumenty programu  $P$ , natomiast miejscem wynikowym – miejsce odpowiadające rejestrowi wynikowemu nr 0.

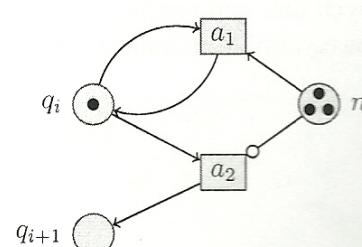
Poniżej przedstawionych jest kilka podstawowych konstrukcji wykorzystywanych do symulacji wykonania instrukcji  $ML$ -programu  $P$ .

#### 1. Przełącznik



Przełącznikiem dla wyróżnionego fragmentu sieci  $S_0 \subseteq S$  nazywamy miejsce  $p$ , które dla każdej akcji  $a \in S_0$  jest miejscem zarówno wejściowym jak i wyjściowym. Na rysunku po prawej  $S_0$  składa się z akcji  $a_1$  oraz  $a_2$ . Niezależnie od struktury  $S_0$  warunkiem koniecznym wykonania dowolnej akcji  $a \in S_0$  jest niepustość miejsca  $p$ . Ponadto, każda akcja po wykonaniu umieszcza żeton ponownie na miejscu  $p$ . Umieszczając zatem żeton na miejscu  $p$  „włączamy” rozważany fragment sieci  $S_0$ , natomiast usuwając żeton z miejsca  $p$  „wyłączamy” go.

#### 2. Instrukcja $Z(n)$ .

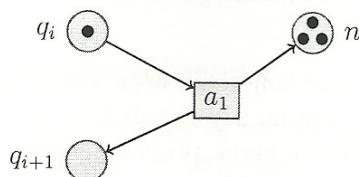


Miejsce  $q_i$  jest przełącznikiem odpowiadającym za wykonanie instrukcji o numerze  $q_i$ , natomiast miejsce  $n$  odpowiada rejestrowi nr  $n$  maszyny licznikowej.

Akcja  $a_1$  jest umożliwiona wyłącznie gdy miejsca  $q_i$  oraz  $n$  są niepuste. W wyniku jej sekwencyjnego wykonania, wszystkie żetony z miejsca  $n$  zostaną usunięte. Wówczas akcja  $a_2$  przeniesie żeton z miejsca  $q_i$  na miejsce  $q_{i+1}$  powodując wykonanie kolejnej instrukcji programu  $P$ .



### 3. Instrukcja $S(n)$ .



Miejsce  $q_i$  jest przełącznikiem odpowiadającym za wykonanie instrukcji o numerze  $q_i$ , natomiast miejsce  $n$  odpowiada rejestrowi nr  $n$  maszyny licznikowej.

Wykonanie akcji  $a_1$  spowoduje dodanie żetonu do miejsca  $n$ , usunięcie żetonu z miejsca  $q_i$  oraz umieszczenie żetonu na miejscu  $q_{i+1}$ , co spowoduje wykonanie kolejnej instrukcji programu  $P$ .

Instrukcje  $T(m, n)$  oraz  $I(m, n, q)$  mogą być symulowane w sposób podobny do opisanego powyżej.

Konstrukcja sieci inhibitorowej  $S$  symulującej działanie  $ML$ -programu  $P$  polega na użyciu powyższych konstrukcji dla każdej instrukcji programu  $P$  oraz dodaniu zestawu przełączników zapewniających właściwą kolejność wykonania instrukcji.  $\square$

### Ćwiczenie 13

Uzupełnij dowód Twierdzenia 17 o brakujące konstrukcje dla instrukcji  $T(m, n)$  oraz  $I(m, n, q)$ .

### Twierdzenie 18

*Obliczenia dowolnej sieci inhibitorowej  $S$  mogą być symulowane za pomocą programu na maszynie licznikowej.*

### Dowód (konsepcja)

Zawartość poszczególnych miejsc sieci  $S$  może być zapamiętana w wyróżnionych rejestrach maszyny licznikowej.  $ML$ -program symuluje wykonanie obliczeń sieci  $S$  przeglądając drzewo jej możliwych obliczeń metodą przeszukiwania wszerz. Po osiągnięciu markingu końcowego program kończy działanie zwracając wynik.  $\square$



## Wykład 7

# Funkcje, programy i maszyny uniwersalne

Komputer może być zaprogramowany do wykonywania wielu różnorodnych zadań. Programy na maszynę licznikową oraz maszyny Turinga są tworzone w celu rozwiązywania jednego wyspecjalizowanego problemu. W czasie tego wykładu poznamy funkcje, programy oraz maszyny uniwersalne, które mogą symulować obliczenia dowolnej innej funkcji, programu lub maszyny (również siebie samej).

### Twierdzenie o parametryzacji

Twierdzenie o parametryzacji, zwane również s-m-n twierdzeniem, jest intuicyjnie tak naturalne, że zwykle posługujemy się nim nawet tego nie zauważając. Poniższe twierdzenie mówi, że funkcja powstająca przez ustalenie wartości części argumentów dowolnej funkcji obliczalnej jako parametry również jest obliczalna. Ponadto, istnieje algorytm pozwalający znaleźć numer jednego z obliczających ją programów.

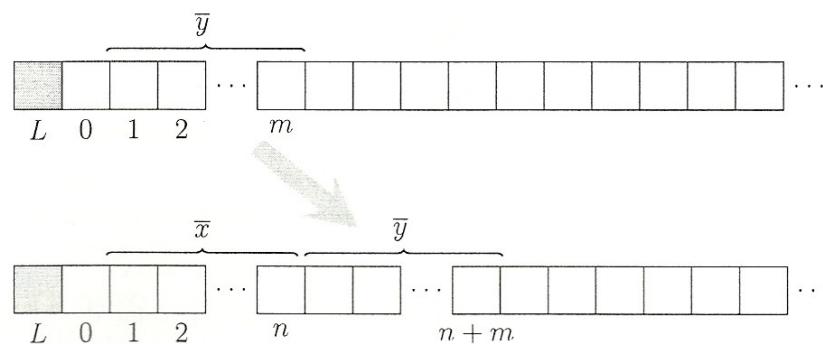
### Twierdzenie 19 (O parametryzacji)

Dla dowolnych liczb naturalnych  $m, n \geq 1$  istnieje totalna i obliczalna funkcja  $s_n^m : \mathbb{N}^{m+1} \rightarrow \mathbb{N}$  taka, że dla dowolnych  $e \in \mathbb{N}$ ,  $\bar{x} \in \mathbb{N}^m$  oraz  $\bar{y} \in \mathbb{N}^n$  zachodzi:

$$\phi_e^{(m+n)}(\bar{x}, \bar{y}) = \phi_{s_n^m(e, \bar{x})}^{(n)}(\bar{y}).$$

### Dowód

Niech  $\phi_e^{(m+n)} : \mathbb{N}^{(m+n)} \rightarrow \mathbb{N}$  będzie funkcją obliczalną,  $P_e$  obliczającym ją programem (o numerze  $e$ ), zaś  $\bar{x} = \{x_1, \dots, x_n\}$  oraz  $\bar{y} = \{y_1, \dots, y_m\}$  jego argumentami.





**KAPITAŁ LUDZKI**  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Wykonanie programu  $P_s$  obliczającego funkcję  $\phi_{s_n^m(e, \bar{x})}^{(n)}$  dla argumentu  $\bar{y}$  polega na skopiowaniu wartości  $\bar{y}$  o  $n$  pozycji „w prawo”, umieszczeniu wartości  $\bar{x}$  w pierwszych  $n$  rejestrach oraz uruchomieniu programu  $P_e'$  powstającego z  $P_e$  przez modyfikację instrukcji warunkowych.

Kod programu  $P_s$  obliczającego funkcję  $\phi_{s_n^m(e, \bar{x})}^{(n)}$  dla argumentu  $\bar{y}$  jest przedstawiony poniżej:

```

 $T(m, m + n)$       // kopiowanie argumentu  $\bar{y}$ 
⋮
 $T(1, n + 1)$ 
 $Z(1)$            // przygotowanie miejsca na argument  $\bar{x}$ 
⋮
 $Z(n)$ 
 $S(1)$            //  $x_1$  razy
⋮
 $S(n)$            //  $x_n$  razy
 $P_e'$             // uruchomienie programu  $P_e'$ 
```

Zauważmy, że wartość funkcji  $s_n^m(e, \bar{x})$  możemy obliczyć na podstawie numeru programu  $P_e$  przez zakodowanie instrukcji kopujących  $\bar{y}$  oraz ustalających wartość  $\bar{x}$ .  $\square$

### Funkcje uniwersalne

#### Definicja 21

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

$$\forall e \in \mathbb{N} \quad \forall \bar{x} \in \mathbb{N}^n \quad \Psi_U^{(n)}(e, \bar{x}) \simeq \phi_e^{(n)}(\bar{x}).$$

#### Twierdzenie 20 (O funkcjach uniwersalnych)

*Każda funkcja uniwersalna  $\Psi_U^{(n)}$  jest częściowo rekurencyjna, a zatem jest obliczalna.*

#### Ćwiczenie 14

Uzasadnij, że funkcja uniwersalna jest częściowo rekurencyjna.

*ML*-obliczalność funkcji uniwersalnej dowiedziemy konstruując dla niej program uniwersalny.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Programy uniwersalne

### Definicja 22

Programem uniwersalnym nazywamy dowolny program  $P_U$  obliczający funkcję uniwersalną.

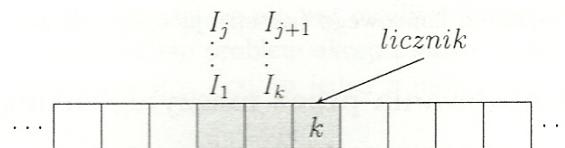
Dla programu uniwersalnego liczącego  $n$ -argumentową funkcję uniwersalną, możemy umieścić jego argumenty w wybranych  $n$  rejestrach. Dla programu uniwersalnego liczącego funkcje uniwersalne o dowolnej liczbie argumentów, umieszczamy w wybranym rejestrze liczbę całkowitą będącą kodem listy argumentów  $\tau(x_1, \dots, x_n)$ .

Przypomnijmy, że numer programu jest kodem listy jego instrukcji. Wykonanie programu  $P$  przez program uniwersalny  $P_U$  polega więc na sekwencyjnym odkodowaniu i wykonaniu kolejnych instrukcji programu  $P$ .

Aby efektywnie obliczyć funkcje  $\tau$  oraz  $\tau^{-1}$  dla listy kodów instrukcji  $I_1, \dots, I_k$  (argumentów programu) możemy wykorzystać następujące podprogramy:

- podprogram liczący potęgi dwójką  $2^j$ ,
- podprogram liczący wykładnik potęgi  $2^j$  ( $\lfloor \log_2 2^j \rfloor$ ),
- podprogram sumujący liczby całkowite.

Szybki dostęp do kodów instrukcji możemy uzyskać wykorzystując 3 rejesty pamięci do przechowywania 2 stosów (instrukcji) oraz licznika. Stosy przechowują listy kodów instrukcji w postaci:  $\tau(I_j, \dots, I_1)$  oraz  $\tau(I_{j+1}, \dots, I_k)$ . Przeniesienie kodu instrukcji  $I_j$  polega na wyodrębnieniu jej z kodu jednego ze stosów i dodaniu jej do kodu drugiego ze stosów.



## Uniwersalna maszyna Turinga

Obliczalność w sensie Turinga jest równoważna  $ML$ -obliczalności. Istnieje zatem uniwersalna maszyna Turinga  $U$ , mogąca symulować obliczenia dowolnej innej maszyny Turinga  $M$ .

Ponieważ maszyna uniwersalna będzie symulować działanie dowolnej maszyny Turinga nie może mieć z góry ograniczonej liczby stanów oraz symboli



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

taśmowych. Z tego powodu stany oraz symbole taśmowe będą kodowane za pomocą liczb naturalnych. Podobnie, ruchy głowicy symulowanej maszyny mogą zostać zakodowane za pomocą liczb naturalnych. Symulację działania maszyny  $M$  na danych wejściowych  $x$  przez maszynę uniwersalną będziemy oznaczać:  $U(M, x)$ .

Maszyna uniwersalna  $U$  jest dwutaśmową maszyną Turinga. Pierwsza taśma zawiera kod symulowanej maszyny  $M$ , druga zawiera aktualną konfigurację maszyny  $M$ . Ponieważ stany oraz alfabet taśmy maszyny  $U$  są kodowane za pomocą liczb całkowitych, wartości  $|Q|$  oraz  $|\Gamma|$  wystarczają do ich jednoznacznego określenia. Kod maszyny  $M$  zapisany jest na pierwszej taśmie maszyny uniwersalnej w potaci:

$$\sqcup \# bin(|Q|) \# bin(|\Gamma|) \# kod(\delta) \# \sqcup$$

Funkcja przejścia kodowana jest w postaci:

$$||stan||litera||stan||litera||ruch||$$

Działanie maszyny uniwersalnej przebiega według schematu:

- Odczytanie z drugiej taśmy kodu aktualnego stanu maszyny  $M$  oraz znaku aktualnie widzianego przez jej głowicę.
- Odnalezienie kodu odczytanego stanu i znaku widzianego przez głowicę w definicji funkcji przejścia maszyny  $M$  na pierwszej taśmie.
- Zmiana zapisanej na drugiej taśmie konfiguracji maszyny  $M$  zgodnie z odczytaną wartością funkcji przejścia  $\delta$ .
- Zakończenie działania (zwrócenie wyniku) w momencie osiągnięcia przez maszynę  $M$  stanu końcowego (akceptującego bądź odrzucającego).

## Problem akceptowania przez maszynę Turinga

**Problem:** Weryfikacja poprawności działania programu.

Rozważając program napisany w dowolny języku programowania (również ML-program oraz maszynę Turinga) możemy przeprowadzić formalny dowód poprawności jego działania oraz poprawności zwracanych przez niego wyników. Program oraz specyfikacja jego działania są ścisłymi pojęciami matematycznymi. Czy w związku z tym możliwa jest automatyzacja procesu weryfikacji programów? Czy możliwe jest stworzenie narzędzia, które dla dowolnego programu podanego na wejściu będzie rozstrzygać czy jest on poprawny?



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Uproścmy trochę nasz problem: czy możliwe jest stworzenie narzędzia, które dla dowolnego opisu maszyny Turinga podanego na wejściu potrafi rozstrzygnąć czy maszyna ta akceptuje ustalone dane wejściowe  $x$ ?

Zdefiniujmy problem formalnie:

$$\mathcal{P}_{ACC} = \{(M, x) : \text{Maszyna Turinga } M \text{ akceptuje dane wejściowe } x\}.$$

Założymy, że potrafimy skonstruować maszynę Turinga  $M_R$  rozstrzygającą powyższy problem:

$$M_R(M, x) = \begin{cases} \text{akceptuj} & \text{jeśli } M \text{ akceptuje } x \\ \text{odrzucaj} & \text{jeśli } M \text{ nie akceptuje } x \end{cases}$$

Używając maszyny  $M_R$  jako podprogramu zdefiniujemy maszynę  $M_O$  zwieraczącą wynik przeciwny niż maszyna  $M_R$ :

$$\begin{aligned} M_O(M, x) &= \begin{cases} \text{akceptuj} & \text{jeśli } M_R \text{ odrzuca } x \\ \text{odrzucaj} & \text{jeśli } M_R \text{ akceptuje } x \end{cases} \\ &= \begin{cases} \text{akceptuj} & \text{jeśli } M \text{ nie akceptuje } x \\ \text{odrzucaj} & \text{jeśli } M \text{ akceptuje } x \end{cases} \end{aligned}$$

Jak zachowa się maszyna  $M_O$  jeśli uruchomimy ją na jej własnym kodzie?

$$M_O(M_O, x) = \begin{cases} \text{akceptuj} & \text{jeśli } M_O \text{ nie akceptuje } x \\ \text{odrzucaj} & \text{jeśli } M_O \text{ akceptuje } x \end{cases}$$

Uzyskana sprzeczność dowodzi, że nie jest możliwe skonstruowanie rozstrzygającej maszyny  $M_R$ , a zatem problem akceptowania wejścia przez maszynę Turinga jest nierzestrzygalny. Jest to jeden z najważniejszych wyników w teorii obliczeń.

### Twierdzenie 21

Problem akceptowania danych wejściowych przez maszynę Turinga jest nierzestrzygalny.

Zauważmy, że dla ustalonej pary  $(M, w)$  rozstrzygnięcie czy maszyna  $M$  zatrzymuje się na słowie  $w$  może być bardzo proste. Nierozstrzygalność problemu  $\mathcal{P}_{ACC}$  oznacza, że nie istnieje algorytm pozwalający rozstrzygnąć ten problem dla dowolnej pary  $(M, w)$ .



## Wykład 8

### Rozstrzygalność

Większość zagadnień informatyki związana jest z algorytmicznym rozwiązywaniem problemów. Istnieją jednak problemy, których nie da się rozwiązać niezależnie od ilości dostępnej pamięci oraz czasu dostępnego dla rozwiązujuącego je urządzenia. Przyczyną takiej sytuacji nie jest konieczność podania skomplikowanego wyniku lecz brak możliwości zautomatyzowania działań koniecznych do jego uzyskania. Dla ułatwienia analizy ograniczymy się więc do badania *problemów decyzyjnych*, które na podstawie danych wejściowych dają odpowiedź *TAK* lub *NIE*.

Intuicyjnie, rozstrzygalność problemu oznacza istnienie rozwiązyującego go algorytmu. Jeśli algorytm o tej własności nie istnieje – problem jest nierostrzygalny. Z formalnego punktu widzenia wygodnie jest reprezentować problemy obliczeniowe za pomocą języków nad ustalonym alfabetem  $\Sigma$ . Słowo  $w \in \Sigma^*$  nazywamy instancją problemu. Rozstrzygnięcie problemu  $P$  polega wówczas na udzieleniu odpowiedzi (twierdzącej lub przeczącej) na pytanie o należenie do odpowiadającego mu języka  $L_P$ .

Formalnie:

#### Definicja 23

Język  $L \subseteq \Sigma^*$  nazywamy rozstrzygalnym jeśli istnieje maszyna Turinga, która zatrzymuje się dla każdego słowa  $w \in \Sigma^*$  odpowiednio w stanie akceptującym jeśli  $w \in L$  oraz w stanie odrzucającym jeśli  $w \notin L$ .

#### Przykład 13

Problem  $P$ : „liczba  $x \in \mathbb{N}$  jest parzysta” jest rozstrzygalny. Problem ten jest reprezentowany przez język:

$$L_P = \{w \in \{0,1\}^* : w \text{ jest binarnym zapisem parzystej liczby naturalnej}\}.$$

Maszyna Turinga  $M_P$  rozstrzygająca język  $L_P$  sprawdza ostatnią (najmniej znaczącą) cyfrę zapisu binarnego  $w$ . Jeśli cyfra ta jest zerem  $M_P$  zwraca odpowiedź *TAK*, w przeciwnym przypadku zwraca odpowiedź *NIE*.

Zauważmy, że dla niektórych instancji  $w \in \Sigma^*$  rozstrzygnięcie problemu „ $w \in L_P$ ” może być bardzo proste. Nie oznacza to jednak automatycznie jego rozstrzygalności. Aby język był rozstrzygalny wymagamy istnienia maszyny Turinga dającej odpowiedź twierdzącą lub przeczącą dla dowolnej instancji  $w \in \Sigma^*$ . Nierostrzygalność języka nie oznacza, że rozstrzygający go



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

algorytm nie jest znany, lecz że taki algorytm nie istnieje i nie będzie możliwe jego stworzenie również w przyszłości.

Zaproponujemy teraz przykłady problemów decyzyjnych, dla których nie istnieją algorytmy rozstrzygające.

### Twierdzenie 22

Problem „ $\phi_x$  jest totalna” jest nierozstrzygalny.

#### Dowód

Załóżmy, że problem „ $\phi_x$  jest totalna” jest rozstrzygalny. Zatem obliczalna jest funkcja:

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

Wówczas obliczalna jest również funkcja symulująca działanie programów liczących funkcje totalne:

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

Rozważmy funkcję  $g(x) = t(x, x) + 1$ , która w oczywisty sposób jest totalna. Przypuśćmy, że  $i$  jest numerem programu obliczającego funkcję  $g$ , czyli  $g(x) = t(i, x)$ . Wówczas, z definicji  $t$  mamy  $g(i) = t(i, i)$ . Jednocześnie z określenia  $g$  mamy  $g(i) = t(i, i) + 1$ , co daje sprzeczność.  $\square$

### Twierdzenie 23

Problem „ $x \in W_x$ ” jest nierozstrzygalny.

#### Dowód

Załóżmy, że problem „ $x \in W_x$ ” jest rozstrzygalny. Wówczas funkcja

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

jest obliczalna. Niech  $m$  będzie numerem dowolnego programu ją obliczającego ( $g = \phi_m$ ). Wówczas:

$$m \in W_m \iff m \in D_g \iff m \notin W_m$$

co daje sprzeczność.  $\square$



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych” realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Zbiory rekurencyjne

Zbiory rekurencyjne są obiektami matematycznymi ścisłe związanymi z pojęciem rozstrzygalności problemów obliczeniowych.

### Definicja 24

Niech  $A \subseteq \mathbb{N}^n$  będzie zbiorem, zaś  $c_A : \mathbb{N}^n \rightarrow \mathbb{N}$  jego funkcją charakterystyczną określoną:

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

Zbiór  $A$  nazywamy rekurencyjnym, jeśli jego funkcja charakterystyczna jest obliczalna ( $c_A \in C_n$ ).

### Przykład 14

1. Zbiór liczb parzystych (podzbiór  $\mathbb{N}$ ) jest zbiorem rekurencyjnym.
2. Zbiór  $\{x \in \mathbb{N} : x \in W_x\}$  nie jest zbiorem rekurencyjnym.

Zauważmy, że jeśli zbiór  $A$  jest rekurencyjny (jego funkcja charakterystyczna  $c_A$  jest obliczalna), problem „ $\bar{x} \in A$ ” jest rozstrzygalny. Z drugiej strony, jeśli problem „ $\bar{x} \in A$ ” jest nierozstrzygalny, funkcja  $c_A$  nie jest obliczalna, a zatem zbiór  $A$  nie jest rekurencyjny. Procedurę obliczającą wartość funkcji  $c_A$  nazywa się często procedurą rozstrzygającą. Relację między rekurencyjnością zbiorów oraz rozstrzygalnością problemów możemy zapisać w skrócie:

$A$  jest rekurencyjny  $\iff c_A \in C_n \iff \text{„}\bar{x} \in A\text{” jest rozstrzygalny}$

Podstawowe własności zbiorów rekurencyjnych opisuje poniższe twierdzenie.

### Twierdzenie 24

Dla ustalonego  $n \geq 1$  zbiór wszystkich podzbiorów rekurencyjnych  $A \subseteq \mathbb{N}^n$  jest zamknięty na operacje teoriomnogościowe.

### Dowód

W sposób oczywisty zbiory  $\emptyset$  oraz  $\mathbb{N}^n$  są rekurencyjne. Niech  $A, B \subseteq \mathbb{N}^n$  będą rekurencyjne. Wówczas funkcje charakterystyczne następujących zbiorów:

- dopełnienie zbioru  $A$ :  $c_{\mathbb{N}^n \setminus A}(\bar{x}) = 1 - c_A(\bar{x})$
- suma zbiorów  $A$  i  $B$ :  $c_{A \cup B}(\bar{x}) = \max(c_A(\bar{x}), c_B(\bar{x}))$
- przekrój zbiorów  $A$  i  $B$ :  $c_{A \cap B}(\bar{x}) = c_A(\bar{x}) \cdot c_B(\bar{x})$

są również obliczalne. □



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Redukowalność problemów obliczeniowych

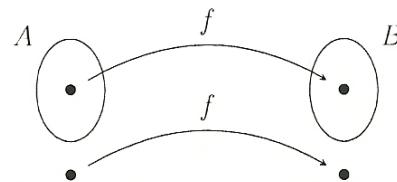
Redukcją nazywamy procedurę przekształcającąinstancję jednego problemu obliczeniowego w instancję drugiego tak, by rozwiązanie drugiego problemu można było wykorzystać przy rozwiązaniu pierwszego. Jest to narzędzie bardzo często stosowane w dowodzeniu rozstrzygalności oraz nierozstrzygalności.

### Definicja 25

Powiemy, że język (problem)  $A$  jest redukowalny do języka  $B$  jeśli istnieje totalna i obliczalna funkcja  $f : \Sigma^* \rightarrow \Sigma^*$  taka, że

$$\forall_{x \in \Sigma^*} x \in A \iff f(x) \in B.$$

Funkcję  $f$  nazywamy redukcją  $A$  do  $B$ .



Rysunek 12: Graficzna interpretacja redukcji problemów obliczeniowych

### Przykład 15

Problem znalezienia drogi w nieznanym mieście można zredukować do problemu znalezienia mapy tego miasta.

#### Obserwacja:

Zauważmy, że jeśli problem obliczeniowy  $A$  redukuje się do problemu  $B$ , to:

- Możliwe jest wykorzystanie rozwiązania problemu  $B$  w celu rozwiązania problemu  $A$ .
- Rozwiązanie problemu  $A$  nie może być trudniejsze niż rozwiązanie problemu  $B$  (ponieważ rozwiązanie  $B$  daje rozwiązanie  $A$ ).

Powyższe rozważania sformułujemy w postaci twierdzenia.

### Twierdzenie 25

Niech  $A$  i  $B$  będą językami, zaś funkcja  $f$  redukcją  $A$  do  $B$ . Jeśli język  $B$  jest rozstrzygalny, również język  $A$  jest rozstrzygalny.



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

### Dowód

Niech  $M_B$  będzie maszyną Turinga rozstrzygającą język  $B$ . Maszyna  $M_A$  rozstrzygająca problem „ $x \in A$ ” działa według następującego schematu:

- Oblicz wartość  $f(x)$ .
- Uruchom maszynę  $M_B$  na wejściu  $f(x)$ .
- Jako wynik zwróć wynik działania maszyny  $M_B$ .

□

### Wniosek

Jeśli nierostrzygalny problem  $A$  jest redukowalny do problemu  $B$ , to problem  $B$  również jest nierostrzygalny.

Powyższy wniosek wykorzystywany jest bardzo często w dowodach nierostrzygalności przebiegających według następującego schematu. Zakładamy rozstrzygalność problemu  $B$  i redukujemy do niego problem  $A$ , o którym wiemy, że jest nierostrzygalny. Wówczas z rozstrzygalności problemu  $B$  musiałaby wynikać rozstrzygalność problemu  $A$  co prowadzi do sprzeczności.

### Uwaga

Istnienie redukcji problemu  $A$  do problemu  $B$  nic nie mówi na temat rozstrzygalności żadnego z nich. Daje ona jedynie możliwość rozwiązania problemu  $A$  korzystając z rozwiązania problemu  $B$  oraz uzależnia rozstrzygalność  $A$  od rozstrzygalności  $B$ .

W czasie jednego z wcześniejszych wykładów pokazaliśmy dowód nierostrzygalności problemu akceptowania słowa wejściowego przez maszynę Turinga. Pokażemy teraz za pomocą redukcji, że określenie czy maszyna Turinga zatrzyma się dla danych wejściowych (tzw. *problem stopu*) jest problemem nierostrzygalnym.

### Twierdzenie 26 (Problem stopu)

#### *Problem stopu*

$$\mathcal{P}_{STOP} = \{(M, x) : \text{maszyna } M \text{ zatrzymuje się na danych wejściowych } x\}.$$

*jest nierostrzygalny.*

### Dowód

Załóżmy, że problem  $\mathcal{P}_{STOP}$  jest rozstrzygalny przez maszynę Turniga  $M_S$ . Skonstruujemy maszynę  $M_A$  rozstrzygającą problem  $\mathcal{P}_{ACC}$  akceptowania słowa wejściowego przez maszynę Turinga.

Maszyna  $M_A$  działa według schematu:



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

- Uruchom maszynę  $M_S$  na danych wejściowych  $(M, x)$
- Jeśli  $M_S$  odrzuci dane wejściowe ( $M$  nie zatrzymuje się na wejściu  $x$ ), to odrzuć.
- Jeśli  $M_S$  zaakceptuje dane wejściowe ( $M$  zatrzymuje się na wejściu  $x$ ), symuluj działanie maszyny  $M$  do jej zatrzymania.
- Zwróć wynik (akceptacja/odrzucenie) zwrócony przez maszynę  $M$ .

Jeśli maszyna  $M_S$  rozstrzygałaby problem stopu, maszyna  $M_A$  rozstrzygałaby problem akceptowania wejścia. Wiemy jednak, że problem ten jest nierozstrzygalny. Uzyskana sprzeczność dowodzi, że również problem stopu jest nierozstrzygalny.  $\square$

Z pomocą redukcji można również udowodnić nierozstrzygalność następujących problemów:

- Problem pustoci języka akceptowanego przez maszynę Turinga.
- Problem, czy język rozpoznawany przez maszynę Turinga jest regularny (bezkontekstowy, skończony, pusty, rozstrzygalny).
- Problem równości języków akceptowanych przed dwie różne maszyny Turinga.

### Ćwiczenie 15

Sformułuj formalne dowody (redukcje) dla powyższych problemów.

### Twierdzenie Rice'a

Kolejnym ważnym narzędziem pozwalającym badać rozstrzygalność problemów obliczeniowych jest twierdzenie Rice'a. Mówiąc o nim, że sprawdzenie dowolnej nietrywialnej własności funkcji obliczalnych jest nierozstrzygalne. Własność nazwiemy nietrywialną, jeśli istnieje co najmniej jedna funkcja obliczalna posiadająca tę własność oraz co najmniej jedna funkcja obliczalna nie posiadająca tej własności.

### Twierdzenie 27 (Rice, wersja 1 – funkcje)

Niech  $\mathcal{B}$  będzie właściwym i niepustym podzbiorem zbioru wszystkich funkcji obliczalnych. Wówczas problem „ $\phi_x \in \mathcal{B}$ ” jest nierozstrzygalny. Równoważnie, zbiór  $B = \{x \in N : \phi_x \in \mathcal{B}\}$  nie jest rekurencyjny.



Możemy również sformułować równoważne twierdzenie mówiące, że wszystkie nietrywialne własności języków akceptowanych przez maszyny Turinga są nierostrzygalne. Własność nazwiemy nietrywialną, jeśli istnieje co najmniej jeden język mający tę własność oraz co najmniej jeden język nie mający tej własności.

### Twierdzenie 28 (Rice, wersja 2 – języki)

Niech  $\mathcal{B}$  będzie właściwym i niepustym podzbiorem zbioru wszystkich języków rozpoznawalnych przez maszyny Turinga. Wówczas problem „ $L(M) \in \mathcal{B}$ ” jest nierostrzygalny (zbior  $B = \{M : L(M) \in \mathcal{B}\}$  nie jest rekurencyjny).

#### Dowód

Bez straty ogólności możemy założyć, że język pusty  $\emptyset \in \mathcal{B}$ . W przeciwnym przypadku wykorzystamy obserwację, że zbiór jest rekurencyjny wtedy i tylko wtedy, gdy jego dopełnienie jest rekurencyjne, i przeprowadzimy dowód dla dopełnienia zbioru  $\mathcal{B}$ .

Ponieważ zbiór  $\mathcal{B}$  jest niepusty, możemy również założyć, że istnieje język  $L \in \mathcal{B}$  rozpoznawany przez maszynę Turinga  $M_L$ .

Niech  $M_A$  będzie maszyną Turinga rozpoznającą problem akceptowania wejścia  $P_{ACC}$ . Konstruujemy maszynę  $M_x$ , dla której  $L(M_x) = L$  lub  $L(M_x) = \emptyset$ . Maszyna  $M_x$  na słowie wejściowym  $y$  działa według następującego schematu:

- Symuluje działanie maszyny  $M_A$  na wejściu  $x$ .
- Jeśli maszyna  $M_A$  zaakceptuje  $x$ ,  $M_x$  rozpoczyna symulację maszyny  $M_L$  na wejściu  $y$ .
- Jeśli maszyna  $M_L$  zatrzyma się,  $M_x$  zwraca wynik jej działania.

Jeśli maszyna  $M_A$  zaakceptuje  $x$ , maszyna  $M_x$ , symulując maszynę  $M_L$ , akceptuje  $y$  lub nie zatrzymuje się. Zatem językiem rozpoznawanym przez maszynę  $M_x$  jest  $L \in \mathcal{B}$ .

Jeśli maszyna (rozpoznająca)  $M_A$  nie zatrzyma się na słowie  $x$ , maszyna  $M_x$  nie zatrzyma się na słowie  $y$ . Zatem językiem rozpoznawanym przez maszynę  $M_x$  będzie język pusty  $\emptyset \notin \mathcal{B}$ .

Otrzymaliśmy równoważność:  $L(M_x) \in \mathcal{B}$  wtedy i tylko wtedy, gdy maszyna  $M_A$  akceptuje słowo  $x$ . Ponieważ problem akceptowania wejścia przez maszynę Turinga jest nierostrzygalny, również problem „ $L(M_x) \in \mathcal{B}$ ” jest nierostrzygalny.  $\square$

Ponieważ zbiory  $\emptyset$  oraz  $\mathbb{N}^n$  są rekurencyjne, twierdzenie Rice'a możemy zapisać równoważnie jako:



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

### Twierdzenie 29 (Rice, wersja 3 – zbiory)

Zbiór  $B = \{x \in \mathbb{N} : \phi_x \in \mathcal{B}\}$  jest rekurencyjny (problem „ $\phi_x \in \mathcal{B}$ ” jest rozstrzygalny) wtedy i tylko wtedy, gdy  $B = \emptyset$  lub  $B = \mathbb{N}^n$ .

#### Przykład 16

Problem wejścia: „ $y \in W_x$ ” jest nierozstrzygalny.

Rozważmy zbiór  $\mathcal{B} = \{f \in C_1 : y \in D_f\}$ . Na mocy twierdzenia Rice'a wystarczy pokazać, że zbiór  $\mathcal{B}$  jest właściwym i niepustym podzbiorem zbioru wszystkich funkcji obliczalnych. Istotnie, funkcja  $g_0$  tożsamościowo równa 0 należy do zbioru  $\mathcal{B}$ , natomiast funkcja pusta  $f_\emptyset$  do niego nie należy.

#### Przykład 17

Problem wyjścia: „ $y \in E_x$ ” jest nierozstrzygalny.

Rozważmy zbiór  $\mathcal{B} = \{f \in C_1 : y \in f(D_f)\}$ . Zbiór  $\mathcal{B}$  jest niepusty, ponieważ należy do niego funkcja  $g_y$  tożsamościowo równa  $y$ . Ponadto  $\mathcal{B}$  jest właściwym podzbiorem  $C_1$ , ponieważ nie zawiera funkcji pustej  $f_\emptyset$ . Zatem na mocy twierdzenia Rice'a problem wyjścia jest nierozstrzygalny.



Projekt pn. „*Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych*”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Wykład 9

### Częściowa rozstrzygalność

W czasie poprzedniego wykładu poznaliśmy pojęcie rozstrzygalności. Aby rozważany język  $L$  (problem decyzyjny należenia do  $L$ ) był rozstrzygalny, wymagamy istnienia algorytmu zatrzymującego się dla każdych danych wejściowych i dającego odpowiedź  $TAK$  dla słów  $w \in L$  oraz  $NIE$  dla słów  $w \notin L$ . Problemy nie spełniające powyższego warunku nazywamy nierostrzygalnymi. Możemy wśród nich znaleźć zarówno takie, dla których nie istnieją algorytmy dające odpowiedź w którymkolwiek przypadku ( $w \in L$  lub  $w \notin L$ ), jak i takie, dla których istnieją algorytmy dające odpowiedź wyłącznie w jednym przypadku:  $TAK$  dla  $w \in L$  lub  $NIE$  dla  $w \notin L$ . Problemy, dla których istnieją algorytmy dające pozytywną odpowiedź w przypadku  $w \in L$  nazywamy częściowo rozstrzygalnymi (lub rozpoznawalnymi).

#### Definicja 26

Język  $L \subseteq \Sigma^*$  nazywamy rozpoznawalnym (częściowo rozstrzygalnym) jeśli istnieje maszyna Turinga, która go rozpoznaje (t.j. zatrzymuje się w stanie akceptującym dla każdego  $x \in L$ ).

#### Przykład 18

Problem stopu jest częściowo rozstrzygalny. Odpowiada mu język:

$$L_S = \{(M, w) : \text{maszyna Turinga } M \text{ zatrzymuje się na } w\}.$$

Maszyna Turinga  $M_S$  rozpoznająca ten język symuluje działanie maszyny  $M$  na wejściu  $w$ . Jeśli maszyna  $M$  zatrzyma się,  $M_S$  zatrzyma się w stanie akceptującym. Jeśli natomiast maszyna  $M$  się nie zatrzyma,  $M_S$  również się nie zatrzyma.

Zauważmy, że rozpoznawalność jest własnością silniejszą niż rozstrzygalność. Wymaganie, aby maszyna Turinga zatrzymywała się dla każdych danych wejściowych ogranicza klasę języków, które może ona rozpoznać.

### Zbiory rekurencyjnie przeliczalne

Przypomnijmy, że językom rozstrzygalnym odpowiadają zbiory rekurencyjne. Podobnie, językom rozpoznawalnym odpowiadają zbiory rekurencyjnie przeliczalne.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁCZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

### Definicja 27

Niech  $A \subseteq \mathbb{N}^n$  będzie zbiorem, zaś  $cc_A : \mathbb{N}^n \rightarrow \mathbb{N}$  jego częściową funkcją charakterystyczną określoną:

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

Zbiór  $A$  nazywamy rekurencyjnie przeliczalnym, jeśli jego częściowa funkcja charakterystyczna jest obliczalna ( $cc_A \in C_n$ ).

### Przykład 19

1. Każdy zbiór rekurencyjny jest rekurencyjnie przeliczalny.
2. Język  $L_1 = \{(M, x) : \text{maszyna Turinga } M \text{ akceptuje słowo } x\}$  jest rekurencyjnie przeliczalny.
3. Język  $L_2 = \{(M, x) : \text{maszyna Turinga } M \text{ nie akceptuje słowa } x\}$  nie jest rekurencyjnie przeliczalny. Dopełnienie  $L_2$  jest językiem rekurencyjnie przeliczalnym.
4. Język  $L_3 = \{x \in \mathbb{N} : \phi_x \text{ jest totalna}\}$  nie jest rekurencyjnie przeliczalny. Jego dopełnienie również nie jest językiem rekurencyjnie przeliczalnym.

Zauważmy, że jeśli zbiór  $A$  jest rekurencyjnie przeliczalny (jego częściowa funkcja charakterystyczna  $cc_A$  jest obliczalna), problem „ $\bar{x} \in A$ ” jest częściowo rozstrzygalny (rozpoznawalny). Z drugiej strony, jeśli problem „ $\bar{x} \in A$ ” jest częściowo rozstrzygalny, funkcja  $cc_A$  jest obliczalna, a zatem zbiór  $A$  jest rekurencyjnie przeliczalny. Relację między rekurencyjną przeliczalnością zbiorów oraz rozpoznawalnością problemów możemy zapisać w skrócie:

$A$  jest rek. przeliczalny  $\iff cc_A \in C_n \iff \text{„}\bar{x} \in A\text{” jest rozpoznawalny}$

Bezpośrednio z definicji otrzymujemy następujące twierdzenie:

### Twierdzenie 30

Język  $L$  jest rekurencyjnie przeliczalny wtedy i tylko wtedy, gdy istnieje maszyna Turinga  $M$  taka, że  $L = L(M)$ .

Równoważnie:

### Twierdzenie 31

Zbiór  $A \subseteq \mathbb{N}^n$  jest rekurencyjnie przeliczalny wtedy i tylko wtedy, gdy jest dziedziną pewnej funkcji obliczalnej.



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Podstawowe własności zbiorów rekurencyjnie przeliczalnych opisuje poniższe twierdzenie:

### Twierdzenie 32

Dla ustalonego  $n \geq 1$  zbiór wszystkich podzbiorów rekurencyjnie przeliczalnych  $A \subseteq \mathbb{N}^n$  jest zamknięty ze względu na sumę i przekrój zbiorów.

#### Dowód

Niech  $A, B \subseteq \mathbb{N}^n$  będą zbiorami rekurencyjnie przeliczalnymi, zaś  $M_A$  i  $M_B$  maszynami Turinga obliczającymi ich częściowe funkcje charakterystyczne. Konstruujemy trzytasmowe maszyny Turinga  $M_{A \cup B}$  oraz  $M_{A \cap B}$  obliczające częściowe funkcje charakterystyczne zbiorów  $A \cup B$  oraz  $A \cap B$ . W obu konstruowanych maszynach na pierwszej taśmie znajduje się słowo wejściowe  $x$ , druga taśma służy do symulacji działania maszyny  $M_A$ , zaś trzecia do symulacji działania maszyny  $M_B$ . W czasie symulacji kolejne instrukcje maszyn  $M_A$  oraz  $M_B$  wykonywane są naprzemiennie.

Schemat działania maszyny  $M_{A \cup B}$  na wejściu  $x$ :

- Uruchom „współbieżnie” maszyny  $M_A$  oraz  $M_B$  na danych  $x$ .
- Zaakceptuj  $x$  jeśli jedna z maszyn  $M_A, M_B$  zatrzyma się w stanie akceptującym.

Schemat działania maszyny  $M_{A \cap B}$  na wejściu  $x$ :

- Uruchom „współbieżnie” maszyny  $M_A$  oraz  $M_B$  na danych  $x$ .
- Zaakceptuj  $x$  jeśli obie maszyny  $M_A, M_B$  zatrzymają się w stanie akceptującym.

□

#### Uwaga:

Zauważmy, że zbiory rekurencyjnie przeliczalne nie są domknięte ze względu na operację dopełnienia. Przykładowo, język

$$L = \{(M, x) : \text{maszyna Turinga } M \text{ zatrzymuje się na słowie } x\}$$

jest rekurencyjnie przeliczalny, natomiast jego dopełnienie

$$\bar{L} = \{(M, x) : \text{maszyna Turinga } M \text{ nie zatrzymuje się na słowie } x\}$$

nie jest językiem rekurencyjnie przeliczalnym.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁCZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmacnianie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

### Twierdzenie 33

Język  $L$  jest rekurencyjny wtedy i tylko wtedy, gdy zarówno język  $L$  jak i jego dopełnienie  $\bar{L}$  są rekurencyjnie przeliczalne.

#### Dowód

==>:

Założymy, że  $L$  jest zbiorem rekurencyjnym. Wówczas również jego dopełnienie  $\bar{L}$  jest zbiorem rekurencyjnym. Ponieważ każdy zbiór rekurencyjny jest rekurencyjnie przeliczalny,  $L$  oraz  $\bar{L}$  są rekurencyjnie przeliczalne.

==<:

Niech  $L$  będzie językiem rekurencyjnie przeliczalnym takim, że jego dopełnienie  $\bar{L}$  również jest językiem rekurencyjnie przeliczalnym. Niech  $M_1$  oraz  $M_2$  będą maszynami Turinga rozpoznającymi odpowiednio języki  $L$  oraz  $\bar{L}$ . Skonstruujmy trzytaśmową maszynę Turinga  $M_L$  rozstrzygającą język  $L$ . Na pierwszej taśmie  $M_L$  znajduje się słowo wejściowe  $x$ , druga taśma służy do symulacji działania maszyny  $M_1$ , natomiast trzecia do symulacji działania maszyny  $M_2$ . W czasie symulacji kolejne instrukcje maszyn  $M_1$  oraz  $M_2$  wykonywane są naprzemiennie.

Schemat działania maszyny  $M_L$  na wejściu  $x$ :

- Uruchom „współbieżnie” maszyny  $M_1$  oraz  $M_2$  na danych  $x$ .
- Zaakceptuj  $x$  jeśli maszyna  $M_1$  zatrzymała się w stanie akceptującym.
- Odrzuć  $x$  jeśli maszyna  $M_2$  zatrzymała się w stanie akceptującym.

□

#### Zależności między klasami rozstrzygalności

Zaprezentowane powyżej rozważania dotyczące zbiorów rekurencyjnych oraz rekurencyjnie przeliczalnych pozwalają określić relację zależności między klasami rozstrzygalności problemów obliczeniowych.

Dla dowolnej pary wzajemnie się dopełniających języków  $L$  oraz  $\bar{L}$  zachodzi dokładnie jeden z warunków:

1. Oba języki  $L$  oraz  $\bar{L}$  są rekurencyjne.
2. Język  $L$  jest rekurencyjnie przeliczalny (ale nie rekurencyjny), zaś język  $\bar{L}$  nie jest rekurencyjnie przeliczalny.
3. Język  $\bar{L}$  jest rekurencyjnie przeliczalny (ale nie rekurencyjny), zaś język  $L$  nie jest rekurencyjnie przeliczalny.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI

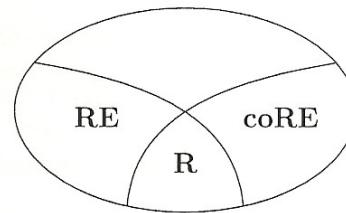


UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

4. Żaden z języków  $L$  oraz  $\bar{L}$  nie jest rekurencyjnie przeliczalny. rekurencyjnie przeliczalny.



Rysunek 13: Relacje między zbiorami języków rekurencyjnych (**R**), rekurencyjnie przeliczalnych (**RE**) oraz dopełnień języków rekurencyjnie przeliczalnych (**coRE**).

### Twierdzenie Rice'a-Shapiro

Twierdzenie Rice'a-Shapiro jest ważnym narzędziem pozwalającym badać częściową rozstrzygalność problemów obliczeniowych. Jest ono uogólnieniem twierdzenia Rice'a przedstawionego w czasie poprzedniego wykładu. Mówiąc o nim, że jeśli pewna częściowo rozstrzygalna własność funkcji obliczalnych  $\mathcal{P}$  jest spełniana przez funkcję  $f$ , to jest ona spełniana również przez jej pewne skończone obcięcie. Równoważnie, jeśli pewna skończona funkcja  $g$  spełnia  $\mathcal{P}$ , muszą ją spełniać również wszystkie funkcje obliczalne, dla których  $g$  jest skończonym obcięciem.

#### Definicja 28

Niech  $f, g : \mathbb{N}^n \rightarrow \mathbb{N}$  będą funkcjami. Powiemy, że funkcja  $g$  jest obcięciem funkcji  $f$  (ozn.  $g \subseteq f$ ) jeśli:

$$\forall \bar{x} \in \mathbb{N}^n \quad \bar{x} \in D_g \Rightarrow \bar{x} \in D_f \wedge g(\bar{x}) = f(\bar{x}).$$

#### Twierdzenie 34 (Rice & Shapiro)

Niech  $\mathcal{A}$  będzie zbiorem funkcji obliczalnych takim, że zbiór ich indeksów  $A = \{x \in \mathbb{N} : \phi_x \in \mathcal{A}\}$  jest rekurencyjnie przeliczalny. Wówczas  $f \in \mathcal{A}$  wtedy i tylko wtedy, gdy istnieje skończona funkcja  $\Theta \in \mathcal{A}$  taka, że  $\Theta \subseteq f$ .

#### Dowód

$\Rightarrow$ :

Załóżmy, że zbiór  $A$  jest rekurencyjnie przeliczalny, zaś  $f \in \mathcal{A}$  jest funkcją



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

taką, że żadna skończona funkcja  $\Theta \subseteq f$  nie należy do  $\mathcal{A}$ . Ponadto, niech  $P = P_e$  będzie programem obliczającym częściową funkcję charakterystyczną zbioru

$$K = \{x \in \mathbb{N} : x \in W_x\},$$

oraz

$H(e, z, t) = „P_e zatrzymuje się na z w co najwyżej t krokach”$ .

Zauważmy, że zbiór  $K$  jest rekurencyjnie przeliczalny, zaś problem  $H(e, z, t)$  jest rozstrzygalny.

Rozważmy funkcję:

$$g(z, t) = \begin{cases} f(t) & \text{jeśli } H(e, z, t) \text{ nie jest prawdziwy} \\ \infty & \text{jeśli } H(e, z, t) \text{ jest prawdziwy} \end{cases}$$

Z rozstrzygalności  $H(e, z, t)$  oraz obliczalności funkcji  $f$  i funkcji pustej wynika obliczalność funkcji  $g$ . Zatem, na mocy twierdzenia o parametryzacji, istnieje totalna i obliczalna funkcja  $s$ , tż.

$$\forall_{z,t} g(z, t) = \phi_{s(z)}(t).$$

Wówczas:

- Z określenia  $\phi_{s(z)} \subseteq f$ .
- Jeśli  $z \in K \implies \phi_{s(z)} = \phi_\emptyset \notin \mathcal{A}$ .
- Jeśli  $z \notin K \implies \phi_{s(z)} = f \in \mathcal{A}$ .

Zatem  $z \notin K \iff \phi_{s(z)} \in \mathcal{A}$ , więc zbiór  $A$  nie może być rekurencyjnie przeliczalny, co jest sprzeczne z założeniem.

$\Leftarrow$ :

Przypuśćmy, że istnieje obliczalna funkcja  $f \notin \mathcal{A}$ , dla której  $\Theta \in \mathcal{A}$  dla pewnej skończonej funkcji  $\Theta \subseteq f$ . Rozważmy funkcję:

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

Zauważmy, że:

- $D_\Theta$  – dziedzina funkcji  $\Theta$  jest zbiorem skończonym, więc jest zbiorem rekurencyjnym, a zatem również rekurencyjnie przeliczalnym.
- Zbiór  $K = \{x \in \mathbb{N} : x \in W_x\}$  jest rekurencyjnie przeliczalny.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

- Zbiór  $B = D_\Theta \cup K$  również jest rekurencyjnie przeliczalny.
- Funkcja  $g(z, t) = f(t) \cdot cc_B(z, t)$  jest obliczalna.

Zatem na mocy twierdzenia o parametryzacji istnieje totalna i obliczalna funkcja  $s$ , tż.

$$\forall_{z,t} g(z, t) = \phi_{s(z)}(t).$$

Wówczas:

- Jeśli  $z \in K \implies \phi_{s(z)} = f \notin \mathcal{A}$ .
- Jeśli  $z \notin K \implies \phi_{s(z)} = \Theta \in \mathcal{A}$ .

Zatem  $z \notin K \iff \phi_{s(z)} \in \mathcal{A}$ , więc zbiór  $A$  nie może być rekurencyjnie przeliczalny, co daje sprzeczność z założeniem.  $\square$

### Przykład 20

Rozważmy zbiór  $\mathcal{A} = \{\phi \in C_n : \phi \text{ jest totalna}\}$ . Zauważmy, że do zbioru  $\mathcal{A}$  nie należy żadna funkcja skończona. Zatem na mocy twierdzenia Rice'a-Shapiro zbiór  $\mathcal{A}$  nie jest rekurencyjnie przeliczalny (problem „ $\phi_x$  jest totalna” nie jest częściowo rozstrzygalny).

Podobnie, zbiór  $\bar{\mathcal{A}} = \{\phi \in C_n : \phi \text{ nie jest totalna}\}$  nie jest rekurencyjnie przeliczalny, ponieważ wszystkie funkcje skończone do niego należą (w szczególności funkcja pusta), ale zbiór ten jest różny od całego  $C_n$ .



## Wykład 10

# Przykłady problemów nierostrzygalnych

W czasie poprzednich wykładów zapoznaliśmy się z definicją rozstrzygalności problemów obliczeniowych. Poznaliśmy również przykłady problemów nierostrzygalnych oraz kilka metod dowodzenia nierostrzygalności. W czasie dzisiejszego wykładu skupimy się na dwóch problemach nierostrzygalnych o bardzo prostym sformułowaniu. Pierwszy z nich – problem odpowiedniości Posta – dotyczy operacji na słowach. Drugi – nazywany X problemem Hilbertha – jest związany ze znajdowaniem rozwiązań równań diofantycznych, t.j. równań o współczynnikach całkowitych posiadających rozwiązania wyłącznie w zbiorze liczb całkowitych

Przypomnijmy, rozstrzygalność języka  $L$  oznacza istnienie algorytmu, który dla dowolnego słowa  $w$  daje odpowiedź (twierdzącą lub przeczącą) na pytanie o należenie  $w$  do  $L$ . Nierostrzygalność natomiast oznacza, że nie jest możliwe stworzenie takiego algorytmu niezależnie od długości czasu oraz rozmiaru pamięci którymi dysponujemy.

Aby udowodnić nierostrzygalność rozważanego problemu  $P$ , wystarczy znaleźć redukcję do  $P$  dla znanego już problemu nierostrzygalnego  $P_N$ . Wówczas z rozstrzygalności  $P$  wynikłaby rozstrzygalność  $P_N$ , co prowadzi do sprzeczności.

## Problem odpowiedniości Posta (PCP)

Rozważmy zbiór par  $P = \{(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)\}$ , gdzie  $x_i, y_i \in \Sigma^*$ . Powiemy, że dla zbioru  $P$  istnieje dopasowanie, jeśli istnieje ciąg indeksów  $i_1, i_2, \dots, i_n$ , gdzie  $n \geq 1$  oraz  $1 \leq i_j \leq k$ , taki że

$$x_{i_1} \cdot x_{i_2} \cdot \dots \cdot x_{i_n} = y_{i_1} \cdot y_{i_2} \cdot \dots \cdot y_{i_n}.$$

Problem ten możemy również przedstawić jako rodzaj układanki, w której pary  $(x_i, y_i)$  interpretujemy jako kamienie domino

$$P = \left\{ \left[ \begin{array}{c} x_1 \\ y_1 \end{array} \right], \left[ \begin{array}{c} x_2 \\ y_2 \end{array} \right], \left[ \begin{array}{c} x_3 \\ y_3 \end{array} \right], \dots, \left[ \begin{array}{c} x_k \\ y_k \end{array} \right] \right\}.$$

Dopasowanie dla tak zdefiniowanego zbioru  $P$  oznacza takie ułożenie kamieni, aby słowa utworzone przez połączenie ich górnych i dolnych etykiet były



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

identyczne. Podczas tworzenia dopasowania możliwe jest kilkukrotne wybranie tego samego kamienia.

### Przykład 21

Dla zbioru

$$P_1 = \left\{ \left[ \frac{aa}{a} \right], \left[ \frac{ba}{ab} \right], \left[ \frac{c}{ac} \right] \right\}$$

istnieje dopasowanie postaci:

$$\left[ \frac{aa}{a} \right] \cdot \left[ \frac{ba}{ab} \right] \cdot \left[ \frac{ba}{ab} \right] \cdot \left[ \frac{c}{ac} \right],$$

natomiast dla zbioru

$$P_2 = \left\{ \left[ \frac{ab}{aba} \right], \left[ \frac{baa}{bb} \right] \right\}$$

nie istnieje dopasowanie.

Problem odpowiedniości Posta (Post correspondence problem) polega na rozstrzygnięciu, czy dla podanego zbioru  $P$  istnieje dopasowanie

$$PCP = \{P : \text{istnieje dopasowanie dla } P\}.$$

W dowodzie nierostrzygalności problemu odpowiedniości Posta wykorzystamy jego nieco zmodyfikowaną wersję

$$MPCP = \left\{ P : \text{istnieje dopasowanie dla } P \text{ rozpoczynające się od } \left[ \begin{array}{c} x_1 \\ y_1 \end{array} \right] \right\}$$

Rozpoczniemy od udowodnienia nierostrzygalności problemu MPCP przez sprowadzenie do niego problemu akceptowania słowa przez maszynę Turinga, którego nierostrzygalność pokazaliśmy na jednym z wcześniejszych wykładów. Następnie, udowodnimy nierostrzygalność problemu PCP wykorzystując nierostrzygalność MPCP.

### Twierdzenie 35

Zmodyfikowany problem odpowiedniości Posta (MPCP) jest nierostrzygalny.

#### Dowód

Pokażemy, że dla dowolnej maszyny Turinga  $M$  oraz słowa  $w \in \Sigma^*$  potrafimy skonstruować instancję  $P_{(M,w)}$  problemu MPCP, dla której dopasowaniem jest akceptująca historia obliczeń maszyny  $M$  na słowie  $w$ .

Przypomnijmy, że konfigurację maszyny Turinga zapisujemy w postaci słowa  $uqv$  oznaczającego, że na taśmie maszyny znajduje się słowo  $uv$ , maszyna



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

jest w stanie  $q_i$ , a jej głowica czyta pierwszy znak słowa  $v$ . Historię obliczeń maszyny Turinga zapisujemy jako ciąg jej kolejnych konfiguracji rozdzielonych znakiem specjalnym  $\#$ .

Niech  $M = (Q, \Sigma, \Gamma, \delta, q_o, q_{ACC}, q_{REJ})$  będzie maszyną Turinga z taśmą lewostronnie ograniczoną oraz  $w = w_1 w_2 \dots w_n$ . Odpowiadającą jej instancję  $P_{(M,w)}$  problemu MPCP konstruujemy według następującego schematu.

1. Pierwszym elementem zbioru  $P$  jest  $\left[ \frac{\#}{\# q_0 w_1 \dots w_n \#} \right]$ .

Element ten musi rozpoczęta zarówno dopasowanie problemu MPCP, jak i akceptującą historię obliczeń maszyny  $M$  na słowie  $w$ . Dolne słowo zawiera konfigurację początkową maszyny  $M$ . Konieczne jest teraz rozszerzenie górnego słowa i umożliwienie jego dopasowania do dolnego.

2. Dla dowolnych  $a, b \in \Gamma$  oraz stanów  $q_1, q_2 \in Q$  ( $q_1 \neq q_{REJ}$ ), jeśli  $\delta(q_1, a) = (q_2, b, R)$  dodajemy do zbioru  $P$  element  $\left[ \frac{q_1 a}{b q_2} \right]$
3. Dla dowolnych  $a, b \in \Gamma$  oraz stanów  $q_1, q_2 \in Q$  ( $q_1 \neq q_{REJ}$ ), jeśli  $\delta(q_1, a) = (q_2, b, L)$  dodajemy do zbioru  $P$  element  $\left[ \frac{c q_1 a}{q_2 c b} \right]$
4. Dla każdego znaku  $a \in \Gamma$  dodajemy do zbioru  $P$  element  $\left[ \frac{a}{a} \right]$

Elementy dodane w krokach 2-5 pozwalają na rozszerzenie dopasowania o drugą konfigurację maszyny  $M$  wynikającą z pierwszej zgodnie z jej funkcją przejścia  $\delta$ .

5. Do zbioru  $P$  dodajemy elementy  $\left[ \frac{\#}{\#} \right]$  oraz  $\left[ \frac{\#}{\sqcup \#} \right]$ . Pozwalają one skopiać znak  $\#$  rozdzielający kolejne konfiguracje maszyny  $M$  oraz dodać znak pusty na końcu konfiguracji. Dzięki temu możliwe jest rozszerzenie dopasowania o kolejne konfiguracje maszyny  $M$ .
6. Dla każdego  $a \in \Gamma$  dodajemy do  $P$  elementy  $\left[ \frac{a q_{ACC}}{q_{ACC}} \right]$  oraz  $\left[ \frac{q_{ACC} a}{q_{ACC}} \right]$  odpowiadające czyszczeniu taśmy po zakończeniu obliczeń.
7. Do zbioru  $P$  dodajemy element  $\left[ \frac{q_{ACC} \# \#}{q_{ACC} \#} \right]$  kończący dopasowanie.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmacnianie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Zauważmy, że konstrując dopasowanie dla zbioru  $P_{(M,w)}$  symulujemy działanie maszyny  $M$  na słowie  $w$ . Po osiągnięciu przez  $M$  stanu akceptującego, używamy elementów czyszczących taśmę w celu zrównania zawartości górnego i dolnego wiersza. Dopasowanie dla rozważanego zbioru  $P_{(M,w)}$  istnieje więc dokładnie wtedy, gdy maszyna  $M$  akceptuje słowo  $w$ . Gdyby więc problem MPCP był rozstrzygalny, również problem akceptowania wejścia byłby rozstrzygalny. Wiemy jednak, że problem akceptowania wejścia jest nierostrzygalny, zatem problem MPCP również jest nierostrzygalny.  $\square$

Nierostrzygalność MPCP wykorzystamy teraz w dowodzie nierostrzygalności oryginalnego problemu PCP.

### Twierdzenie 36

*Problem odpowiedniości Posta (PCP) jest nierostrzygalny.*

#### Dowód

Dla dowolnego słowa  $u = u_1 u_2 \dots u_n$  definiujemy następujące operacje:

- $\circledast u = *u_1 * u_2 \dots * u_n$
- $u \circledast = u_1 * u_2 \dots * u_n *$
- $\circledast u \circledast = *u_1 * u_2 \dots * u_n *$

Niech

$$P = \left\{ \left[ \frac{x_1}{y_1} \right], \left[ \frac{x_2}{y_2} \right], \left[ \frac{x_3}{y_3} \right], \dots, \left[ \frac{x_k}{y_k} \right] \right\},$$

gdzie  $x_i, y_i \in \Sigma^*$ , będzie zbiorem, dla którego chcemy rozstrzygnąć problem MPCP. Konstruujemy odpowiadający mu zbiór

$$P' = \left\{ \left[ \frac{\circledast x_1}{\circledast y_1 \circledast} \right], \left[ \frac{\circledast x_2}{y_2 \circledast} \right], \left[ \frac{\circledast x_3}{y_3 \circledast} \right], \dots, \left[ \frac{\circledast x_k}{y_k \circledast} \right], \left[ \frac{\circledast \diamond}{\diamond} \right] \right\},$$

gdzie  $\circledast, \diamond \notin \Sigma$ . i rozważamy dla niego problem PCP.

Zauważmy, że

- Jedynym elementem mogącym rozpoczęć dopasowanie jest  $\left[ \frac{\circledast x_1}{\circledast y_1 \circledast} \right]$ .
- Dopuszczenie  $*$  nie ma wpływu na pozostałą część dopasowania, ponieważ oryginalne symbole są rozdzielone znakami  $*$  i występują w obu słowach wyłącznie na parzystych pozycjach.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

- Elementem kończącym dopasowanie (uzupełniającym ostatni znak \* w górnym słowie) może być wyłącznie  $\frac{[\oplus\Diamond]}{\Diamond}$ .

Zauważmy, że zbiór  $P'$  ma dopasowanie dokładnie wtedy, gdy zbiór  $P$  ma dopasowanie rozpoczynające się od pierwszego elementu. Z rozstrzygalności problemu PCP wynikałaby więc rozstrzygalność problemu MPCP. Otrzymana sprzeczność dowodzi, że problem PCP jest nierożstrzygalny.  $\square$

### Warianty problemu PCP

Modyfikacja problemu PCP przez dołożenie wymagania, aby dopasowanie rozpoczęło się od pierwszego elementu zbioru  $P$  nadal jest problemem nierożstrzygalnym. Warto wspomnieć również o następujących modyfikacjach problemu PCP:

- Ograniczenie rozmiaru alfabetu – jeśli  $|\Sigma| = 1$ , problem PCP jest rozstrzygalny.
- Ograniczenie liczby elementów zbioru  $P$ :
  - jeśli  $|P| \leq 2$ , problem PCP jest rozstrzygalny,
  - jeśli  $|P| \geq 7$  problem PCP jest nierożstrzygalny,
  - dla  $3 \leq |P| \leq 6$  status problemu PCP nie jest znany.
- Ograniczony PCP – szukanie dopasowania za pomocą co najwyżej  $k$  elementów. Problem w tej wersji jest rozstrzygalny (NP-zupełny).
- Marked PCP – każde słowo  $x_i$  oraz  $y_i$  zaczyna się inną literą. Problem w tej wersji jest rozstrzygalny (EXPTIME).
- 2-marked PCP – dowolne dwa słowa  $x_i$  i  $x_j$  lub  $y_i$  i  $y_j$  różnią się prefiksem długości 2. Problem w tej wersji jest nierożstrzygalny.

### X problem Hilberta (1900)

W roku 1900 na Międzynarodowym Kongresie Matematycznym David Hilbert przedstawił listę 23 nierożwiązanych problemów matematycznych nazywanych później *Problemami Hilberta*. Niektórych z nich nie udało się rozwiązać do dziś. Problem oznaczony numerem X dotyczy istnienia algorytmicznej metody pozwalającej rozwiązać dowolne równanie diofantyczne. Mimo prostoty sformułowania problem ten czekał na rozwiązanie do lat 70-tych XX wieku.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

### Definicja 29

Równaniem diofantycznym nazywamy równanie wielomianowe o współczynnikach całkowitych, którego rozwiązań szukamy wyłącznie w zbiorze liczb całkowitych.

### Przykład 22

Równanie  $2x^2 + 3y + 2z = 7$  ma rozwiązanie ( $x = 1, y = 1, z = 1$ ), natomiast równanie  $4x + 6y = 5$  nie posiada rozwiązań w zbiorze liczb całkowitych.

X problem Hilberta polega na rozstrzygnięciu, czy rozważane równanie diofantyczne posiada rozwiązania

$$X_H = \{f : f - \text{równanie diofantyczne posiadające rozwiązanie}\}.$$

### Definicja 30

Zbiór  $A \subseteq \mathbb{N}$  nazywamy diofantycznym, jeśli

$$A = \{x \in \mathbb{N} : \exists_{y_1, \dots, y_n \in \mathbb{N}} f_A(x, y_1, \dots, y_n) = 0\}$$

dla pewnego wielomianu  $f_A$  o współczynnikach całkowitych.

Zauważmy, że dla ustalonych wartości  $y_1, \dots, y_n \in \mathbb{N}$  zbiór rozwiązań równania  $f_A(x, y_1, \dots, y_n) = 0$  jest rekurencyjny. Ponadto, klasa zbiorów rekurencyjnie przeliczalnych jest domknięta ze względu na kwantyfikator egzystencjalny ( $\exists$ ). Możemy zatem sformułować twierdzenie.

### Twierdzenie 37

Każdy zbiór diofantyczny jest rekurencyjnie przeliczalny.

W latach 70-tych XX-wieku Yuri Matiyasevich opierając się na wynikach badań Martina Davisa, Hilary'ego Putnama oraz Julii Robinson udowodnił następujące twierdzenie.

### Twierdzenie 38 (Matiyasevich, 1970)

Każdy zbiór rekurencyjnie przeliczalny jest diofantyczny.

Zastanówmy się nad konsekwencjami powyższego twierdzenia. Wynika z niego, że zbiór  $K = \{x \in \mathbb{N} : x \in W_x\}$  jest diofantyczny, czyli

$$K = \{x \in \mathbb{N} : \exists_{y_1, \dots, y_n \in \mathbb{N}} f_K(x, y_1, \dots, y_n) = 0\}$$



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

dla pewnego wielomianu  $f_K$  o współczynnikach całkowitych. Otrzymujemy więc równoważność

$$x \in W_x \iff \exists_{y_1, \dots, y_n \in \mathbb{N}} f_K(x, y_1, \dots, y_n) = 0.$$

Zatem, gdyby X problem Hilberta był rozstrzygalny, wykorzystując procedurę rozstrzygającą istnienie rozwiązań równania  $f_K(x, y_1, \dots, y_n) = 0$  otrzymalibyśmy procedurę rozstrzygającą problem „ $x \in W_x$ ”. Wiemy jednak, że znalezienie takiej procedury nie jest możliwe, ponieważ problem ten jest nierozstrzygalny. Otrzymana sprzeczność dowodzi, że X problem Hilberta jest nierozstrzygalny.

Podsumowując powyższe rozważania otrzymujemy następujące twierdzenie.

### Twierdzenie 39

X problem Hilberta

$$X_H = \{f : f - \text{równanie diofantyczne mające rozwiązanie}\}$$

jest nierozstrzygalny.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Wykład 11

### Czasowa złożoność obliczeniowa

Przy założeniu nieograniczonej dostępnych zasobów (czas oraz pamięć) wszystkie poznane do tej pory modele obliczeń są równoważne. Jednak w zastosowaniach praktycznych komputery dysponują pamięcią ograniczonych rozmiarów. Również akceptowalny czas działania algorytmu jest ograniczony.

Jeśli problem jest rozstrzygalny, możliwe jest jego rozwiązanie w sensie obliczeniowym. Może ono jednak wymagać niezwykle długiego czasu lub niezwykle dużego rozmiaru pamięci czyniąc problem nierozwiązywalnym w praktyce. W trakcie wykładu zapoznamy się z metodami analizy czasu potrzebnego do rozwiązania określonych problemów obliczeniowych. Pokażemy również jak klasyfikować problemy obliczeniowe w zależności od czasu wymaganego do ich rozwiązania. Ponieważ czas działania algorytmu zależy od wielu parametrów, dla uproszczenia będziemy wyznaczać go jako funkcję długości danych wejściowych.

#### Definicja 31

Niech  $M$  będzie maszyną Turinga zatrzymującą się na każdym wejściu. Złożonością czasową (czasem działania)  $M$  nazywamy funkcję  $f : \mathbb{N} \rightarrow \mathbb{N}$ , gdzie  $f(n)$  jest równa największej liczbie kroków wykonywanych przez maszynę  $M$  dla dowolnego słowa długości  $n$ .

Zauważmy, że badanie maksymalnej liczby kroków obliczeń maszyny Turin-ga ma sens wyłącznie dla maszyn, które zatrzymują się dla każdych danych wejściowych. W analizie złożoności obliczeniowej algorytmów zazwyczaj nie rozpatrujemy dokładnego czasu działania algorytmu lecz pewne jego przybliżenie. Analiza asymptotyczna polega na zrozumieniu, jak działa algorytm na bardzo dużych danych wejściowych. W tym celu rozpatrujemy najbardziej znaczące składniki w wyrażeniu opisującym czas działania algorytmu. W szczególności pomijamy stałe oraz składniki mniej znaczące.

#### Definicja 32

Niech  $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$ . Powiemy, że  $f(n) = O(g(n))$  jeśli istnieją dodatnie stałe  $c$  oraz  $n_0$  takie, że

$$\forall_{n \geq n_0} f(n) \leq c \cdot g(n).$$

Funkcję  $g$  nazywamy asymptotycznym ograniczeniem górnym funkcji  $f$ .



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

### Przykład 23

- $f(n) = 2n^2 + 2n - 1$  jest  $O(n^2)$  ale nie jest  $O(n)$
- $f(n) = 2n - 1$  jest  $O(n^2)$ ,  $O(n)$  ale nie  $O(\log n)$
- $f(n) = \log_2 n$  jest  $O(n)$ ,  $O(\log n)$
- $f(n) = 4$  jest  $O(1)$

### Uwaga

Z twierdzenia o zmianie podstawy logarytmu mamy

$$\log_a x = \frac{\log_b x}{\log_b a}.$$

Ponieważ w notacji asymptotycznej pomijamy stałe współczynniki  $\left(\frac{1}{\log_b a}\right)$ , w przypadku złożoności logarytmicznej  $O(\log n)$  nie ma potrzeby określania podstawy logarytmu.

Notacja  $f = O(g)$  oznacza, że funkcja  $f$  jest asymptotycznie *nie większa* od funkcji  $g$ . Aby określić, że funkcja  $f$  jest asymptotycznie *mniejsza* od funkcji  $g$ , używamy notacji  $o$ .

### Definicja 33

Niech  $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$ . Powiemy, że  $f(n) = o(g(n))$  jeśli

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

Innymi słowy dla każdej liczby rzeczywistej  $c > 0$  istnieje taka liczba  $n_0$ , że

$$\forall_{n \geq n_0} f(n) \leq c \cdot g(n).$$

### Przykład 24

- $f(n) = n^2$  jest  $o(n^3)$
- $f(n) = \sqrt{n}$  jest  $o(n)$
- $f(n) = 3n$  nie jest  $o(n)$



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych” realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Spróbujmy teraz zbadać złożoność maszyny Turinga rozstrzygającej przykładowy problem obliczeniowy.

### Problem

Rozważmy język  $L = \{a^k b^k : k \geq 0\}$ . Nie jest trudno skonstruować maszynę Turinga  $M$  zatrzymującą się dla każdego słowa  $w \in \{a, b\}^*$  i rozstrzygającą problem „ $w \in L$ ”. Zastanówmy się więc, ile operacji musi wykonać maszyna  $M$  aby dać prawidłową odpowiedź.

### Rozwiązań 1

Rozważmy jednotaśmową maszynę Turinga  $M_1$  działającą według następującego schematu:

1. Przeczytaj dane na taśmie i odrzuć jeśli nie są postaci  $a \dots ab \dots b$
2. Dopóki na taśmie znajdują się znaki  $a$  i  $b$ , usuwaj naprzemiennie  $a$  z początku oraz  $b$  z końca słowa.
3. Zaakceptuj jeśli taśma jest pusta
4. Odrzuć jeśli na taśmie zostały wyłącznie litery  $a$  lub wyłącznie litery  $b$

Policzmy ile operacji wykonuje maszyna  $M_1$  dla słowa długości  $n$ :

- Pierwsze przeczytanie taśmy:  $n + 1$  kroków
- Usunięcie skrajnych liter słowa długości  $n$ :  $n + 2$  kroki
- Usuwanie skrajnych liter wykonywane  $\frac{n}{2}$  razy

Otrzymujemy zatem złożoność  $O(n) + O(n^2) = O(n^2)$ .

### Rozwiązań 2

Rozważmy jednotaśmową maszynę Turinga  $M_2$  działającą według następującego schematu:

1. Przeczytaj dane na taśmie i odrzuć jeśli nie są postaci  $a \dots ab \dots b$
2. Dopóki taśma nie jest pusta:
  - Przeczytaj dane na taśmie sprawdzając czy liczba liter jest parzysta - odrzuć jeśli jest nieparzysta
  - Usuń z taśmy co drugie wystąpienie  $a$  oraz co drugie wystąpienie  $b$
3. Zaakceptuj jeśli taśma jest pusta



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Zanim oszacujemy złożoność obliczeniową maszyny  $M_2$  musimy uzasadnić poprawność jej działania. Przy każdym przejściu przez taśmę liczba wystąpień  $a$  oraz liczba wystąpień  $b$  zmniejsza się o połowę (pomijamy w tym miejscu resztę z dzielenia). Sprawdzanie czy liczba wszystkich liter na taśmie jest parzysta jest równoważne sprawdzeniu czy liczba wystąpień  $a$  oraz liczba wystąpień  $b$  mają tę samą parzystość. Jeśli przy każdym dzieleniu przez 2 liczba wystąpień  $a$  oraz liczba wystąpień  $b$  mają tę samą parzystość, to są one równe.

Oszacujmy teraz czas działania maszyny  $M_2$  dla słowa długości  $n$ :

- Pierwsze przeczytanie taśmy:  $O(n)$  kroków
- Przeczytanie taśmy ze sprawdzeniem parzystości liczby liter:  $O(n)$
- Przeglądanie taśmy z usuwaniem co drugiej litery:  $O(n)$
- W każdym kroku usuwamy co najmniej połowę liter – maksymalnie  $\log_2 n$  powtórzeń

Otrzymujemy zatem złożoność  $O(n) + O(n \log n) = O(n \log n)$ .

### Rozwiążanie 3

Rozważmy dwutaśmową maszynę Turinga  $M_3$  działającą według następującego schematu:

1. Czytaj słowo na pierwszej taśmie kopując litery  $a$  na drugą taśmę (do napotkania pierwszego  $b$ ).
2. Kontynuuj czytanie pierwszej taśmy dla każdego  $b$  usuwając kolejne  $a$  na drugiej taśmie.
3. Zaakceptuj jeśli po osiągnięciu prawego końca słowa wszystkie  $a$  z pierwszej taśmy zostały usunięte.
4. Odrzuć jeśli:
  - po osiągnięciu końca słowa na pierwszej taśmie na drugiej taśmie znajdują się jeszcze litery  $a$
  - ostatnie  $a$  zostało usunięte z drugiej taśmy przed osiągnięciem końca słowa na pierwszej taśmie
  - za ostatnim  $b$  znajduje się  $a$



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Maszyna  $M_3$  przegląda taśmę wejściową dokładnie raz i kończy działanie w stanie akceptującym lub odrzucającym. Zatem całkowity czas jej działania jest rzędu  $O(n)$ .

Zauważmy, że chociaż w teorii obliczalności z tezy Churcha-Turinga wynika, że wszystkie sensowne modele obliczalności są sobie równoważne, w teorii złożoności obliczeniowej wybór modelu obliczeń ma istotny wpływ na złożoność języka. Skonstruowaliśmy trzy maszyny Turinga rozwiązujące problem „ $w \in L$ ” znacząco różniące się czasem działania. Okazuje się jednak, że zmiana liczb taśm maszyny Turinga powoduje co najwyżej wielomianową zmianę złożoności obliczeniowej.

#### Twierdzenie 40

Niech  $f(n)$  będzie funkcją spełniającą warunek  $f(n) \geq n$ . Wówczas dla każdej  $k$ -taśmowej maszyny Turinga  $M_1$  o złożoności czasowej  $O(f(n))$  istnieje równoważna jej jednotaśmowa maszyna  $M_2$  działająca w czasie  $O(f^2(n))$ .

#### Dowód

Przypomnijmy, że obliczenia  $k$ -taśmowej maszyny Turniga  $M_1$  mogą być symulowane na maszynie  $M_2$  z jedną taśmą (patrz Twierdzenie 13 Wykład 5). Zawartość  $k$  taśm maszyny  $M_1$  zapisujemy na taśmie maszyny  $M_2$  w postaci taśm wirtualnych, natomiast położenie głowic czytająco/piszących  $M_1$  kodujemy za pomocą wyróżnionych symboli dodanych do alfabetu. Maszyna  $M_2$  zmienia zawartość taśm wirtualnych oraz aktualnia pozycje głowic wirtualnych zgodnie ze schematem działania maszyny  $M_1$ .

Załóżmy, że dane wejściowe mają rozmiar  $n$ , zaś maszyna  $M_1$  ma złożoność obliczeniową  $O(f(n))$ . Oszacujmy liczbę operacji wykonywanych przez maszynę  $M_2$ .

- Symulacja pojedynczego kroku maszyny  $M_1$  polega dwukrotnym przejściem taśmy oraz co najwyżej  $k$ -krotnym skopiowaniem jej zawartości.
- Maszyna  $M_1$  działając w czasie  $O(f(n))$  może zapisać co najwyżej  $O(f(n))$  różnych komórek na każdej taśmie, zatem symulacja pojedynczego kroku  $M_1$  wykonywana jest w czasie  $O(f(n))$ .
- Maszyna  $M_2$  musi zasymulować wykonanie  $O(f(n))$  kroków obliczeń maszyny  $M_1$ .

Podsumowując, złożoność obliczeniowa maszyny  $M_2$  wynosi  $O(f^2(n))$ .  $\square$

Ponieważ obliczenia  $k$ -taśmowej maszyny Turinga możemy symulować na maszynie jednotaśmowej w czasie jedynie wielomianowo większym, modele



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

te nazywamy *wielomianowo równoważnymi*. Podobna własność zachodzi dla dowolnych sensownych *deterministycznych* modeli obliczeń. Z tego powodu najwygodniejszym modelem używanym do badania złożoności obliczeniowej jest  $(k+2)$ -taśmowa maszyna Turinga zbudowana według schematu:

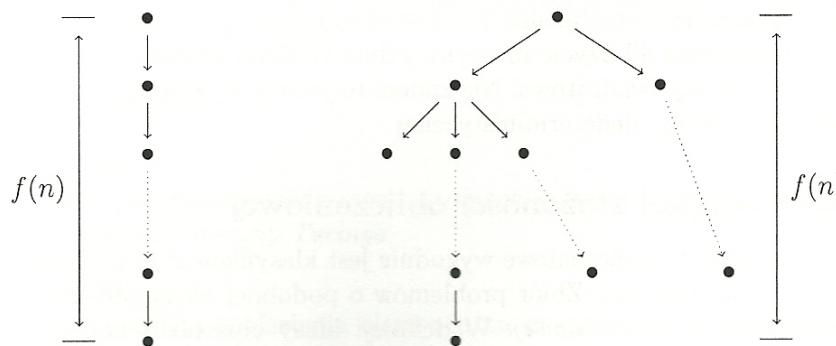
- pierwsza taśma zawiera dane wejściowe (tylko do odczytu)
- ostatnia taśma zawiera wynik (tylko do zapisu)
- $k$  taśm roboczych wykorzystywanych do obliczeń

### Ćwiczenie 16

Przypomnij dowód równoważności maszyn Turinga oraz maszyn liczeniowych. Uzasadnij, że modele te są wielomianowo równoważne.

### Niedeterministyczne modele obliczeń

Każdy krok obliczeń deterministycznej maszyny Turinga jest jednoznacznie wyznaczony przez jej aktualną konfigurację. Maszyna niedeterministyczna dopuszcza kilka możliwości wyboru kolejnego kroku. Obliczenia tworzą więc drzewo, którego gałęzie odpowiadają różnym sposobom obliczeń. Definicja złożoności obliczeniowej maszyny niedeterministycznej musi ten fakt uwzględniać.



### Definicja 34

Niech  $M$  będzie niedeterministyczną maszyną Turinga rozstrzygającą pewien język. Złożonością czasową (czasem działania) maszyny  $M$  nazywamy funkcję  $f : \mathbb{N} \rightarrow \mathbb{N}$ , gdzie  $f(n)$  jest równa największej liczbie kroków wykonywanych przez maszynę  $M$  na dowolnej ścieżce obliczeń dla dowolnego słowa długości  $n$ .



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁCZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Przypomnijmy, że obliczenia niedeterministycznej maszyny Turinga mogą być symulowane przez maszynę deterministyczną. Istotnym pytaniem jest złożoność obliczeniowa.

#### Twierdzenie 41

Niech  $f(n)$  będzie funkcją spełniającą warunek  $f(n) \geq n$ . Wówczas dla każdej jednotaśmowej niedeterministycznej maszyny Turinga  $M_1$  o złożoności czasowej  $O(f(n))$  istnieje równoważna jej jednotaśmowa deterministyczna maszyna Turinga  $M_2$  działająca w czasie  $2^{O(f(n))}$ .

#### Dowód

Symulacja działania maszyny  $M_1$  polega na przeglądaniu (metodą wszerz) jej drzewa obliczeń. Dla słowa wejściowego długości  $n$  każda niedeterministyczna ścieżka obliczeń maszyny  $M_1$  ma długość co najwyżej  $O(f(n))$ , zatem dojście od korzenia do dowolnego węzła wymaga maksymalnie czasu rzędu  $O(f(n))$ . Ponadto, każdy węzeł drzewa obliczeń  $M_1$  ma co najwyżej  $b$  potomków, gdzie  $b$  jest maksymalną liczbą wyborów dopuszczanych przez funkcję przejścia  $M_1$ . Maksymalna liczba węzłów w drzewie obliczeń  $M_1$  (nie większa niż podwojona liczba liści) jest rzędu  $O(b^{f(n)})$ . Maszyna  $M_2$  przeglądając wszerz drzewo obliczeń maszyny  $M_1$  wykonuje więc co najwyżej  $O(f(n)) \cdot O(b^{f(n)}) = 2^{O(f(n))}$  operacji.  $\square$

#### Uwaga

W dowodzie Twierdzenia 13 do symulacji obliczeń niedeterministycznej maszyny Turinga wykorzystaliśmy trzytaśmową maszynę deterministyczną. Na mocy Twierdzenia 40 użycie maszyny jednotaśmowej zwiększy czas jej działania co najwyżej kwadratowo. Nie zmieni to jednak wykładniczej złożoności symulacji maszyny niedeterministycznej.

### Klasy czasowej złożoności obliczeniowej

Badając problemy obliczeniowe wygodnie jest klasyfikować je pod względem trudności obliczeniowej. Zbiór problemów o podobnej złożoności nazywamy *klasą złożoności obliczeniowej*. W definicji klasy złożoności konieczne jest określenie trybu obliczeń (deterministyczny lub niedeterministyczny) oraz funkcji złożoności ograniczającą liczbę możliwych kroków obliczeń.

#### Definicja 35

*Klasę złożoności  $DTIME(f(n))$  definiujemy jako zbiór wszystkich języków, które są rozstrzygane w czasie  $O(f(n))$  przez pewną deterministyczną maszynę Turinga.*

Podobnie określamy klasy złożoności dla modeli niedeterministycznych.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

### Definicja 36

*Klasę złożoności  $NTIME(f(n))$  definiujemy jako zbiór wszystkich języków, które są rozstrzygane w czasie  $O(f(n))$  przez pewną niedeterministyczną maszynę Turinga.*

Zdefiniujemy teraz kilka najważniejszych klas złożoności obliczeniowej oraz podamy przykłady problemów należących do każdej z nich.

### Klasa PTIME

### Definicja 37

*PTIME (w skrócie  $P$ ) jest klasą języków rozstrzygalnych w czasie wielomianowym przez deterministyczne maszyny Turinga:*

$$P = \bigcup_k DTIME(n^k)$$

Klasa  $P$  określa mniej więcej zakres problemów, które mogą być rozwiązane (w sensownym czasie) na rzeczywistych komputerach. Zauważmy ponadto, że definicja klasy  $P$  nie zależy od wyboru modelu obliczeniowego, jeśli tylko jest on wielomianowo równoważny deterministycznej maszynie Turinga.

### Przykład 25

Rozważmy problem znalezienia w grafie skierowanym  $G$  ścieżki łączącej dwa ustalone wierzchołki  $v_1$  i  $v_2$ . Problem ten możemy rozwiązać za pomocą algorytmów przeglądania grafu wszerz (BFS) oraz wgłąb (DFS) działających w czasie liniowo (a więc wielomianowo) zależnym od sumarycznej liczby wierzchołków oraz krawędzi grafu  $G$ .

### Klasa LOGTIME

### Definicja 38

*LOGTIME jest klasą języków rozstrzygalnych w czasie logarytmicznym przez deterministyczne maszyny Turinga.*

### Przykład 26

Rozważmy problem znalezienia elementu na posortowanej liście. Używając algorytmu wyszukiwania binarnego możemy rozwiązać ten problem w czasie logarytmicznie zależnym od rozmiaru przeszukiwanej listy.

*LOGTIME* jest klasą problemów, dla których istnieją bardzo wydajne algorytmy działających w czasie krótszym niż liniowy. Przypomnijmy, że przy złożoności logarytmicznej nie ma potrzeby określania podstawy logarytmu. Ponadto, z własności funkcji logarytmicznej otrzymujemy  $\log(n^k) = k \cdot \log(n)$  oraz  $\log(k \cdot n) = k + \log(n)$ . A zatem logarytm dowolnego wielomianu jest rzędu  $O(\log(n))$ .



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych” realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Klasa EXPTIME

### Definicja 39

*EXPTIME jest klasą języków rozstrzygalnych w czasie wykładniczym przez deterministyczne maszyny Turinga:*

$$EXPTIME = \bigcup_k DTIME(2^{n^k})$$

### Przykład 27

Przypomnijmy, że problem stopu dla maszyn Turinga

$$HALT = \{(M, w) : M \text{ zatrzymuje się na } w\}$$

jest nierożstrzygalny. Rozważmy zatem jego zmodyfikowaną wersję:

$$HALT_K = \{(M, w, k) : M \text{ zatrzymuje się po co najwyżej } k \text{ krokach}\}$$

Zauważmy, że jeśli maszyna  $M$  zatrzymuje się po co najwyżej  $k$  krokach, może mieć na to wpływ wyłącznie zawartość pierwszych  $k$  komórek taśmy. Aby rozstrzygnąć problem  $HALT_K$  wystarczy zasymulować działanie maszyny  $M$  na każdym możliwym wejściu długości  $k$ .

Klasa  $EXPTIME$  zawiera problemy trudne obliczeniowo. Istnieją jednak problemy jeszcze trudniejsze o ponadwykładniczej złożoności obliczeniowej. Przykładem takiego problemu jest algorytm obliczający wartość funkcji Ackermanna.

## Klasa NPTIME

### Definicja 40

*NPTIME (w skrócie NP) jest klasą języków rozstrzygalnych w czasie wielomianowym przez niedeterministyczne maszyny Turinga:*

$$NP = \bigcup_k NTIME(n^k)$$

### Przykład 28

Ścieżką Hamiltona w grafie skierowanym  $G$  nazywamy ścieżkę skierowaną przechodzącą przez każdy jego wierzchołek dokładnie raz. Rozważmy problem rozstrzygnięcia czy w grafie skierowanym  $G$  istnieje ścieżka Hamiltona łącząca dwa ustalone wierzchołki  $v_1$  oraz  $v_2$ .

$$HAMPATH = \{(G, v_1, v_2) : G \text{ zawiera ścieżkę Hamiltona } v_1 \rightsquigarrow v_2\}$$



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Zauważmy, że nie jest trudno znaleźć deterministyczny algorytm rozwiązujujący ten problem działający w czasie wykładniczym. Z drugiej strony, gdyby obliczenia wykonywała maszyna niedeterministyczna i w każdym kroku dokonywała wyboru właściwej krawędzi, rozwiązałaby problem w czasie wielomianowym. Zatem problem *HAMPATH* ma niedeterministyczną złożoność wielomianową.

Problem *HAMPATH* ma jeszcze jedną bardzo istotną własność. Jeśli znamy opis ścieżki  $v_1 \rightsquigarrow v_2$  w grafie skierowanym  $G$ , weryfikacja czy jest ona ścieżką Hamiltona jest możliwa w czasie wielomianowym przez deterministyczną maszynę Turinga. Co ciekawe, własność tę posiadają wszystkie problemy z klasy *NP*. Możemy zatem sformułować alternatywną definicję tej klasy:

#### Definicja 41

*NP (NP TIME)* jest klasą języków posiadających wielomianowe algorytmy weryfikujące.

#### Uwaga

Zauważmy, że każdy język rozstrzygalny w czasie wielomianowym przez deterministyczną maszynę Turinga, jest również rozstrzygalny w czasie wielomianowym przez maszynę niedeterministyczną. Otrzymujemy zatem inkluzyję  $P \subseteq NP$ . Odpowiedź na pytanie, czy jest to inkluzyja właściwa jest jednym z najważniejszych problemów informatyki teoretycznej. Bardziej szczegółowo analizą tego zagadnienia zajmiemy się w czasie kolejnych wykładów.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Wykład 12

### Pamięciowa złożoność obliczeniowa

Najważniejszymi parametrami branymi pod uwagę podczas analizy wydajności algorytmu są czas jego działania oraz rozmiar wykorzystywanej przez niego pamięci. W czasie poprzedniego wykładu poznaliśmy metody klasyfikacji problemów pod względem ich złożoności czasowej. Bieżący wykład poświęcony będzie badaniu złożoności pamięciowej.

Przypomnijmy, że jako modelu do badania złożoności czasowej używaliśmy  $(k+2)$ -taśmowej maszyny Turinga zbudowanej według schematu:

- pierwsza taśma zawiera dane wejściowe (tylko do odczytu)
- ostatnia taśma zawiera wynik (tylko do zapisu)
- $k$  taśm roboczych wykorzystywanych do obliczeń

Ten sam model wykorzystamy do badania rozmiaru pamięci używanej w trakcie obliczeń. Podobnie jak w przypadku złożoności czasowej, konieczne jest rozdzielenie definicji złożoności pamięciowej maszyn deterministycznych oraz niedeterministycznych.

#### Definicja 42 (Złożoność deterministyczna)

*Niech  $M$  będzie deterministyczną maszyną Turinga zatrzymującą się dla każdych danych wejściowych. Złożonością pamięciową maszyny  $M$  nazywamy funkcję  $f : \mathbb{N} \rightarrow \mathbb{N}$ , gdzie  $f(n)$  jest maksymalną liczbą komórek taśm roboczych wykorzystywanych (odczyt/zapis) przez maszynę  $M$  uruchomioną na dowolnym słowie długości  $n$ .*

#### Definicja 43 (Złożoność niedeterministyczna)

*Niech  $M$  będzie niedeterministyczną maszyną Turinga zatrzymującą się na wszystkich ścieżkach obliczeń. Złożonością pamięciową maszyny  $M$  nazywamy funkcję  $f : \mathbb{N} \rightarrow \mathbb{N}$ , gdzie  $f(n)$  jest maksymalną liczbą komórek taśmy wykorzystywanych (odczyt/zapis) przez maszynę  $M$  na dowolnej ścieżce obliczeń po uruchomieniu na dowolnym słowie długości  $n$ .*

Pamięciową złożoność obliczeniową, podobnie jak złożoność czasową, określamy jako funkcję rozmiaru danych wejściowych. Zazwyczaj jesteśmy zainteresowani efektywnością działania algorytmu dla danych dużego rozmiaru.



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych” realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Przeprowadzamy więc analizę asymptotyczną i zapisujemy złożoność za pomocą notacji  $O$ .

Problemy obliczeniowe wygodnie jest klasyfikować pod względem zbliżonego rozmiaru wykorzystywanej pamięci. Definiując klasę złożoności pamięciowej konieczne jest określenie trybu obliczeń (deterministyczny lub niedeterministyczny) oraz funkcji złożoności ograniczającej rozmiar pamięci używanej w czasie obliczeń.

#### Definicja 44

Niech  $f : \mathbb{N} \rightarrow \mathbb{R}^+$ . Klasę złożoności pamięciowej definiujemy następująco:

$$\text{SPACE}(f(n)) = \left\{ L : L - \text{język rozstrzygany przez deterministyczną maszynę Turinga w pamięci } O(f(n)) \right\}$$

$$\text{NSPACE}(f(n)) = \left\{ L : L - \text{język rozstrzygany przez niedeterministyczną maszynę Turinga w pamięci } O(f(n)) \right\}$$

Przeanalizujmy teraz złożoność pamięciową przykładowego problemu obliczeniowego.

#### Przykład 29

Formułę logiczną nazywamy wyrażenie składające się ze zmiennych oraz operacji logicznych (alternatywa, koniunkcja oraz negacja), np.

$$\phi = (\neg x \vee y) \wedge (z \vee \neg y).$$

Powiemy, że formuła logiczna  $\phi$  jest spełnialna jeśli istnieje takie wartościowanie zmiennych zawartych w  $\phi$ , dla którego cała formuła jest prawdziwa. Problem spełnialności polega na sprawdzeniu czy dana formuła logiczna jest spełnialna.

$$SAT = \{\phi : \phi \text{ jest spełnialna}\}.$$

Założymy, że formuła  $\phi$  ma rozmiar  $n$  i zawiera  $m$  różnych zmiennych logicznych  $x_1, \dots, x_m$  ( $m \leq n$ ). Maszyna Turinga rozstrzygająca problem  $SAT$  może działać według następującego schematu

- Generuj kolejno wszystkie wartościowania zmiennych  $x_1, \dots, x_m$
- Dla każdego wartościowania oblicz wartość formuły  $\phi$
- Zaakceptuj, jeśli  $\phi$  ma wartość  $TRUE$
- Odrzuć, jeśli dla żadnego wartościowania  $\phi$  nie miała wartości  $TRUE$



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Zauważmy, że dla formuły zawierającej  $m$  różnych zmiennych logicznych konieczne może być sprawdzenie  $2^m$  możliwych wartościowań. Maszyna deterministyczna rozstrzygająca problem  $SAT$  będzie więc działała w czasie  $O(2^n)$  (gdzie  $n$  jest rozmiarem formuły). Z drugiej strony, do obliczenia wartości formuły  $\phi$  maszyna  $M$  musi zapamiętać wyłącznie aktualne wartościowanie zmiennych. W każdej iteracji pętli może więc wykorzystać ten sam obszar taśmy. Liczba zmiennych jest ograniczona z góry przez długość formuły  $n$ . Maszyna  $M$  będzie więc działać w pamięci  $O(n)$ , czyli  $SAT \in SPACE(n)$ .

### Obserwacja

Zauważmy, że w tym samym czasie maszyna Turinga może wykonać dokładnie jeden krok obliczeń. Z drugiej strony, każda komórka taśmy może być w czasie obliczeń używana wielokrotnie (zapis i odczyt). A zatem pamięć jest zasobem silniejszym niż czas.

### Twierdzenie Savitcha

Przypomnijmy, że obliczenia dowolnej niedeterministycznej maszyny Turinga mogą być symulowane na maszynie deterministycznej. Symulacja taka powoduje jednak wykładniczy wzrost czasu obliczeń. Okazuje się jednak, że maszyna deterministyczna potrzebuje do jej przeprowadzenia zaskakująco mało pamięci.

### Twierdzenie 42 (Savitch, 1970)

Dla dowolnej funkcji  $f : \mathbb{N} \rightarrow \mathbb{R}^+$  spełniającej warunek  $f(n) \geq \log(n)$  zachodzi inkluzja:

$$NSPACE(f(n)) \subseteq SPACE(f^2(n)).$$

### Dowód

Niech  $M_1$  będzie niedeterministyczną maszyną Turinga rozstrzygającą język  $A$  działającą w pamięci  $f(n)$ . Skonstruujemy deterministyczną maszynę Turinga  $M_2$  rozstrzygającą język  $A$ . Bez straty ogólności możemy założyć, że maszyna  $M_1$  ma dokładnie jedną konfigurację akceptującą  $c_{ACC}$ .

Zauważmy, że niedeterministyczna ścieżka w drzewie obliczeń maszyny  $M_1$  wykorzystująca pamięć rozmiaru  $f(n)$  może składać się z  $2^{O(f(n))}$  kroków. Naiwne sekwencyjne przeglądanie ścieżki obliczeń wymagające zapisania wszystkich dokonywanych niedeterministycznych wyborów może wymagać pamięci rozmiaru wykładniczego. Do symulacji działania maszyny  $M_1$  za pomocą  $M_2$  wykorzystamy więc rozwiązywanie problemu osiągalności – sprawdzenia czy z konfiguracji  $c_1$  maszyny  $M_1$  możemy dojść do konfiguracji  $c_2$  w co najwyżej  $t$  krokach. Problem ten będzie rozwiązywany przez procedurę  $REACH(c_1, c_2, t)$ .



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Symulacja maszyny  $M_1$  będzie równoważna rozwiązyaniu przez maszynę  $M_2$  problemu osiągalności dla konfiguracji początkowej, konfiguracji końcowej oraz maksymalnej liczby kroków obliczeń, które może wykonać maszyna  $M_1$ .

Procedura REACH( $c_1, c_2, t$ ) działa według następującego schematu:

- Jeśli  $t = 0$ , zwróć  $TRUE$  jeśli  $c_1 = c_2$  oraz  $FALSE$  w przeciwnym przypadku.
- Jeśli  $t = 1$ , zwróć  $TRUE$  jeśli  $c_1 = c_2$  lub jeśli jest możliwe przejście w jednym kroku z konfiguracji  $c_1$  do konfiguracji  $c_2$  zgodnie z funkcją przejścia maszyny  $M_1$ .
- Jeśli  $t > 1$ , dla każdej konfiguracji  $c_m$  maszyny  $M_1$  (wykorzystując pamięć rozmiaru  $f(n)$ ):
  - Wykonaj  $\text{REACH}(c_1, c_m, \lfloor \frac{t}{2} \rfloor)$
  - Wykonaj  $\text{REACH}(c_m, c_2, \lceil \frac{t}{2} \rceil)$
  - Zwróć  $TRUE$  jeśli oba powyższe wywołania zwróciły  $TRUE$ , oraz  $FALSE$  w przeciwnym przypadku.
- Zwróć  $FALSE$ , jeśli do tej pory nie zostało zwrócone  $TRUE$ .

Niech  $c_0$  będzie konfiguracją początkową maszyny  $M_1$ , oraz założymy że  $2^{d \cdot f(n)}$ , dla pewnego  $d \in \mathbb{N}$ , jest górnym ograniczeniem liczby konfiguracji maszyny  $M_1$  (a zatem również długości ścieżki w drzewie obliczeń). Wówczas, działanie niedeterministycznej maszyny  $M_1$  na słowie wejściowym  $w$  może być zasymulowane przez deterministyczną maszynę  $M_2$  za pomocą wywołania procedury  $\text{REACH}(c_0, c_{ACC}, 2^{d \cdot f(n)})$ .

Dla każdego wywołania procedury  $\text{REACH}(c_1, c_2, t)$  maszyna  $M_2$  musi zapamiętać na taśmie roboczej  $c_1$ ,  $c_2$  oraz  $t$ , aby móc je odtworzyć po zakończeniu wywołania rekurencyjnego. Każde wywołanie rekurencyjne wymaga więc  $O(f(n))$  dodatkowej pamięci. Ponieważ liczba konfiguracji maszyny  $M_1$  jest ograniczona z góry przez  $2^{d \cdot f(n)}$ , zaś przy każdym wywołaniu rekurencyjnym dzielimy wartość  $t$  przez 2, głębokość rekursji wynosi  $O(\log(2^{d \cdot f(n)})) = O(f(n))$ . Zatem całkowity rozmiar pamięci używanej przez maszynę  $M_2$  w czasie symulacji obliczeń maszyny  $M_1$  jest rzędu  $O(f^2(n))$ .  $\square$

## Klasy złożoności pamięciowej

Zdefiniujmy teraz kilka najważniejszych klas pamięciowej złożoności obliczeniowej oraz podamy przykłady problemów należących do każdej z nich.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Klasy PSPACE oraz NPSPACE

### Definicja 45

*PSPACE jest klasą języków rozpoznawanych przez deterministyczne maszyny Turinga w pamięci wielomianowej*

$$PSPACE = \bigcup_k SPACE(n^k).$$

### Definicja 46

*NPSPACE jest klasą języków rozpoznawanych przez niedeterministyczne maszyny Turinga w pamięci wielomianowej*

$$NPSPACE = \bigcup_k NSPACE(n^k).$$

### Przykład 30

W przykładzie 29 pokazaliśmy, że problem spełnialności formuł logicznych (SAT) może być rozwiązyany w pamięci liniowej. A zatem:  $SAT \in PSPACE$ .

### Uwaga

Ponieważ wielomian podniesiony do dowolnej potęgi nadal pozostaje wielomianem, bezpośrednio z twierdzenia Savitcha wynika równość klas

$$PSPACE = NPSPACE.$$

### Przykład 31

*Formułą logiczną z kwantyfikatorami* nazywamy formułę logiczną składającą się ze zmiennych, operatorów logicznych  $\neg$ ,  $\vee$  i  $\wedge$ , oraz kwantyfikatorów: uniwersalnego  $\forall$  i egzystencjalnego  $\exists$ . Powiemy, że formuła  $\phi$  nie ma zmiennych wolnych, jeśli każda zmienna jest w zasięgu pewnego kwantyfikatora. Powiemy, że formuła  $\phi$  jest w *preneksowej postaci normalnej* jeśli wszystkie kwantyfikatory występują na początku  $\phi$  (Dowolną formułę logiczną  $\phi$  możemy przekształcić do równoważnej formuły  $\phi'$  w prenksowej postaci normalnej). Przykładowo, formuła

$$\phi = \forall_x \exists_y ((x \vee y) \wedge (\neg x \vee \neg y))$$

jest formułą w prenksowej postaci normalnej bez zmiennych wolnych.

Rozważmy problem:

$$TBQF = \{(\phi) : \phi \text{ jest prawdziwą formułą logiczną bez zmiennych wolnych}\}.$$

Procedura  $\mathcal{P}$  na maszynę Turinga rozstrzygająca problem  $TBQF$  działa według następującego schematu:



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNA



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

- Jeśli formuła  $\phi$  nie zawiera kwantyfikatorów, składa się wyłącznie ze stałych. Oblicz wartość  $\phi$  i zaakceptuj jeśli jest ona równa TRUE oraz odrzuć w przeciwnym przypadku.
- Jeśli formuła  $\phi$  jest postaci  $\exists_x \psi$ , wywołaj  $\mathcal{P}$  rekurencyjnie na formule  $\psi$  podstawiając za  $y$  wartość 0 a następnie 1. Zaakceptuj, jeśli którekolwiek wywołanie  $\mathcal{P}$  zwróciło wartość TRUE oraz odrzuć w przeciwnym przypadku.
- Jeśli formuła  $\phi$  jest postaci  $\forall_x \psi$ , wywołaj  $\mathcal{P}$  rekurencyjnie na formule  $\psi$  podstawiając za  $y$  wartość 0 a następnie 1. Zaakceptuj, jeśli oba wywołania  $\mathcal{P}$  zwróciły wartość TRUE oraz odrzuć w przeciwnym przypadku.

Zauważmy, że głębokość wywołań rekurencyjnych jest ograniczona liczbą zmiennych formuły  $\phi$ . Ponadto, dla każdego wywołania musimy zapamiętać wartość zmiennej. Rozmiar wykorzystanej pamięci nie przekroczy więc rozmiaru  $\phi$ . A zatem  $TBQF \in PSPACE$ .

### Klasa L (LSPACE)

#### Definicja 47

*Klasa L to klasa języków rozpoznawanych przez deterministyczne maszyny Turinga w pamięci logarytmicznej:*

$$L = SPACE(\log(n)).$$

Zauważmy, że maszyna Turinga działająca w pamięci subliniowej może przeczytać całe słowo wejściowe, nie ma jednak miejsca, aby zapisać je na taśmie roboczej. Z drugiej strony, subliniowe ograniczenie czasu działania maszyny uniemożliwia jej przeczytanie słowa wejściowego w całości. Z tego powodu klasy o złożoności poniżej liniowej rozwija się wyłącznie dla złożoności pamięciowej. Ponieważ długość słowa wejściowego jest większa niż rozmiar dostępnej pamięci roboczej, konfigurację maszyny Turinga określamy w tym przypadku za pomocą zawartości taśm roboczych oraz pozycji jej głowic. Tak zdefiniowaną konfigurację możemy zapisać za pomocą słowa rozmiaru  $c \log n$ .

#### Przykład 32

Rozważmy język  $A = \{a^k b^k : k \geq 0\}$ . Rozstrzygająca go maszyna Turinga  $M$  liczy wystąpienia liter  $a$  oraz  $b$  zapisując reprezentacje binarne liczby wystąpień na taśmie roboczej. Ponieważ binarny zapis liczby całkowitej wymaga pamięci logarytmicznej, maszyna  $M$  działa w pamięci logarytmicznej. Wobec tego  $A \in L$ .



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Klasa NL (NLSPACE)

### Definicja 48

*Klasa NL to klasa języków rozpoznawanych przez niedeterministyczne maszyny Turinga w pamięci logarytmicznej:*

$$NL = NSPACE(\log(n)).$$

### Przykład 33

Rozważmy problem istnienia ścieżki między dwoma wyróżnionymi wierzchołkami w grafie skierowanym.

$$PATH = \{(G, v_1, v_2) : \text{w grafie skierowanym } G \text{ istnieje ścieżka } v_1 \rightsquigarrow v_2\}$$

Niedeterministyczna maszyna Turinga  $M$  rozstrzygająca problem PATH rozpoczyna przeszukiwanie grafu  $G$  od wierzchołka  $v_1$ . W każdym kroku wybiera w sposób niedeterministyczny kolejne wierzchołki tworzące ścieżkę  $v_1 \rightsquigarrow v_2$ . Maszyna  $M$  zatrzymuje się w stanie akceptującym po osiągnięciu wierzchołka  $v_2$ , lub w stanie odrzucającym po przejrzeniu wszystkich wierzchołków  $G$  osiągalnych z  $v_1$ . Jeśli maszyna  $M$  pamięta na taśmie roboczej wyłącznie numer aktualnie przetwarzanego wierzchołka, procedura ta może być zrealizowana w pamięci logarytmicznej. Zatem  $PATH \in NL$ .

## Klasy EXPSPACE oraz NEXPSPACE

### Definicja 49

*EXPSPACE jest klasą języków rozpoznawanych przez deterministyczne maszyny Turinga w pamięci wykładniczej*

$$EXPSPACE = \bigcup_k SPACE(2^{n^k}).$$

### Definicja 50

*NEXPSPACE jest klasą języków rozpoznawanych przez niedeterministyczne maszyny Turinga w pamięci wykładniczej*

$$NEXPSPACE = \bigcup_k NSPACE(2^{n^k}).$$

### Przykład 34

Wyrażenia regularne konstruujemy za pomocą operacji konkatenacji, sumy oraz domknięcia (gwiazdki) Kleene'go. Jeśli rozszerzymy listę operacji o potęgowanie:

$$R^k = \underbrace{R \cdot R \cdot \dots \cdot R}_k$$



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

otrzymamy *uogólnione* wyrażenia regularne. Zauważmy, że rozszerzenie to zmienia tylko zapis wyrażenia, ponieważ każde potęgowanie możemy zastąpić odpowiednią liczbą konkatenacji.

Rozważmy problem równoważności dwóch uogólnionych wyrażeń regularnych (równości języków przez nie opisywanych):

$$EQREX \uparrow = \{(R, S) : \text{uogólnione wyrażenia regularne t.z } R \text{ równoważne } S\}$$

Równoważność dwóch wyrażeń regularnych możemy rozstrzygnąć w pamięci wielomianowej przez równoległą symulację odpowiadających im automatów skończonych. Aby zastosować powyższą procedurę do uogólnionych wyrażeń regularnych, musimy najpierw usunąć występujące w nich potęgi zastępując je wymaganą liczbą konkatenacji. Zauważmy jednak, że operacja ta może spowodować wykładniczy wzrost rozmiaru rozważanych wyrażeń regularnych. A zatem  $EQREX \uparrow \in EXPSPACE$ .

### Uwaga

Ponieważ funkcja wykładnicza podniesiona do kwadratu nadal jest funkcją wykładniczą, na mocy twierdzenia Savitcha otrzymujemy równość:

$$EXPSPACE = NEXPSPACE.$$

### Złożoność czasowa vs złożoność pamięciowa

Zauważmy, że maszyna Turinga działająca w czasie  $O(f(n))$  (zarówno deterministyczna jak i niedeterministyczna) wykonuje co najwyżej  $O(f(n))$  kroków obliczeń. Nie może więc używać więcej niż  $O(f(n))$  komórek pamięci. Możemy zatem sformułować wniosek:

$$TIME(f(n)) \subseteq SPACE(f(n))$$

oraz

$$NTIME(f(n)) \subseteq NSPACE(f(n)).$$

Bardziej szczegółowa analiza relacji między znymi klasami złożoności obliczeniowej (zarówno czasowej jak i pamięciowej) zostanie omówiona w czasie kolejnego wykładu.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Wykład 13

# Trudność i zupełność problemów obliczeniowych

W czasie poprzednich wykładów zapoznaliśmy się z pojęciem klasy złożoności obliczeniowej. Poznaliśmy również kilka ważniejszych klas złożoności czasowej i pamięciowej wraz z przykładami problemów do nich należących. Podczas dzisiejszego wykładu zajmiemy się analizą problemów, których złożoność obliczeniowa jest ściśle związana ze złożonością obliczeniową całej klasy. Problemy takie nazywać będziemy trudnymi oraz zupełnymi.

Intuicyjnie problemami *trudnymi* dla ustalonej klasy złożoności (czasowej lub pamięciowej) będziemy nazywać problemy, których rozwiązanie jest co najmniej tak trudne, jak rozwiązanie każdego innego problemu z tej klasy. Jeśli dodatkowo problem taki należyć będzie do rozważanej klasy złożoności, będziemy nazywać go problemem *zupełnym* dla tej klasy.

## Efektywne redukcje problemów obliczeniowych

W celu porównywania trudności obliczeniowej problemów będziemy potrzebować narzędzi pozwalających zamieniać instancję jednego problemu na instancję innego problemu. Procedurę taką nazywać będziemy *redukcją*. Istotne jest jednak, aby używana redukcja nie wpływała na zmianę rozpatrywanej klasy złożoności obliczeniowej. Dlatego też złożoność samej redukcji musi być „łatwa” w porównaniu ze złożonością typowego problemu rozważanej klasy złożoności.

Przypomnijmy, że wszystkie poznane przez nas deterministyczne modele obliczeń są wielomianowo równoważne, natomiast złożoność czasowa problemów mierzona na maszynach deterministycznych i niedeterministycznych może się różnić wykładniczo. Traktowanie wielomianowych różnic w złożoności jako nieistotnych i pomijanie ich pozwala budować teorię niezależną od wyboru modelu obliczeń. Z tego właśnie powodu w większości zastosowań przydatne są redukcje w czasie wielomianowym.

### Definicja 51

*Powiemy, że funkcja  $f : \Sigma^* \rightarrow \Sigma^*$  jest obliczalna w czasie wielomianowym, jeśli istnieje maszyna Turinga  $M$  działająca w czasie wielomianowym, która dla dowolnego słowa  $w \in \Sigma^*$  zatrzymuje się zwracając wartość  $f(w)$ .*



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Przykładem takiej funkcji może być dowolny wielomian. Obliczenie jego wartości przez maszynę Turinga będzie wymagało co najwyżej wielomianowej liczby kroków.

### Definicja 52

Język  $A \subseteq \Sigma^*$  jest wielomianowo redukowalny do języka  $B \subseteq \Sigma^*$ , jeśli istnieje funkcja  $f : \Sigma^* \rightarrow \Sigma^*$  obliczalna w czasie wielomianowym taka, że dla każdego  $w \in \Sigma^*$

$$w \in A \iff f(w) \in B.$$

Funkcję  $f$  nazywamy wielomianową redukcją  $A$  do  $B$ .

Redukcja w czasie wielomianowym pozwala zastosować efektywne rozwiązań jednego problemu do rozwiązywania innych problemów w następujący sposób. Przypuśćmy, że  $f : \Sigma^* \rightarrow \Sigma^*$  jest redukcją w czasie wielomianowym języka  $A$  do języka  $B$ , oraz potrafimy efektywnie (np. w czasie wielomianowym) rozwiązać problem „ $w \in B$ ”. Wówczas, aby rozstrzygnąć problem „ $w \in A$ ”, wyznaczamy wartość  $f(w)$ , a następnie rozstrzygamy problem „ $f(w) \in B$ ”. Zauważmy, że czasowa złożoność obliczeniowa tak uzyskanego rozwiązania problemu „ $w \in A$ ” będzie co najwyżej wielomianowo większa od złożoności rozwiązania problemu „ $w \in B$ ”.

Wiemy, że maszyna Turinga działająca w czasie wielomianowym może wykorzystać wielomianową liczbę komórek pamięci. Zatem redukcja w czasie wielomianowym może nie być wystarczająca dla problemów należących do klas LOGSPACE oraz NLOGSPACE. W tym przypadku musimy rozważać redukcję w pamięci logarytmicznej.

### Definicja 53

Powiemy, że funkcja  $f : \Sigma^* \rightarrow \Sigma^*$  jest obliczalna w pamięci logarytmicznej jeśli istnieje maszyna Turinga  $M$  wykorzystująca co najwyżej logarytmiczną liczbę komórek na taśmach roboczych, która dla dowolnego słowa  $w \in \Sigma^*$  zatrzymuje się zwracając wartość  $f(w)$ .

Przykładem takiej funkcji może być  $f(n) = n^2$ . Jeśli maszyna Turinga licząca wartość funkcji  $f$  będzie przetwarzała liczby w zapisie binarnym użyje co najwyżej  $O(\log n)$  komórek na taśmach roboczych.

### Definicja 54

Język  $A \subseteq \Sigma^*$  jest redukowalnym do języka  $B \subseteq \Sigma^*$  w pamięci logarytmicznej, jeśli istnieje funkcja  $f : \Sigma^* \rightarrow \Sigma^*$  obliczalna w pamięci logarytmicznej taka, że dla każdego  $w \in \Sigma^*$

$$w \in A \iff f(w) \in B.$$

Funkcję  $f$  nazywamy redukcją  $A$  do  $B$  w pamięci logarytmicznej.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Zauważmy, że maszyna Turinga używająca pamięci rozmiaru  $O(\log n)$  może mieć co najwyżej  $n \cdot c^{O(\log n)}$  różnych konfiguracji, dla pewnego  $c \in \mathbb{N}$ . Zatem redukcja w pamięci logarytmicznej jest również redukcją w czasie wielomianowym. Implikacja w drugą stronę nie jest niestety prawdziwa.

## Problemy trudne i zupełne

Możemy teraz podać formalną definicję problemu trudnego dla ustalonej klasy złożoności obliczeniowej.

### Definicja 55

Niech  $\mathcal{C}$  będzie klasą złożoności obliczeniowej (czasowej lub pamięciowej). Język  $A$  nazywamy  $\mathcal{C}$ -trudnym jeśli dla dowolnego języka  $B \in \mathcal{C}$  istnieje efektywna redukcja (w czasie wielomianowym lub pamięci logarytmicznej) do języka  $A$ .

Jeśli dodamy wymaganie, aby rozważany problem należał do rozpatrywanej klasy złożoności, otrzymamy formalną definicję problemu zupełnego dla klasy złożoności obliczeniowej.

### Definicja 56

Niech  $\mathcal{C}$  będzie klasą złożoności obliczeniowej (czasowej lub pamięciowej). Język  $A$  nazywamy  $\mathcal{C}$ -zupełnym jeśli:

1.  $A \in \mathcal{C}$
2.  $A$  jest  $\mathcal{C}$ -trudny

Dowodzenie  $\mathcal{C}$ -zupełności języka  $A$  polega zatem na wykazaniu, że należy on do klasy  $\mathcal{C}$  oraz każdy inny problem z tej klasy może być sprowadzony do  $A$  za pomocą efektywnej redukcji w czasie wielomianowym lub pamięci logarytmicznej.

## Przykład problemu zupełnego

Przypomnijmy rozważany w czasie poprzedniego wykładu problem istnienia ścieżki między dwoma wyróżnionymi wierzchołkami w grafie skierowanym

$$\text{PATH} = \{(G, v_1, v_2) : \text{w grafie skierowanym } G \text{ istnieje ścieżka } v_1 \rightsquigarrow v_2\}.$$

Pokazaliśmy, że  $\text{PATH} \in \text{NL}$ . Okazuje się, że problem jest również NL-trudny. Jest zatem przykładem problemu NL-zupełnego.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

### Twierdzenie 43

Problem PATH jest NL-zupełny.

#### Dowód

Przypomnijmy, że problem PATH  $\in$  NL. Niedeterministyczna maszyna Turinga może rozstrzygnąć istnienie ścieżki między wierzchołkami  $v_1$  oraz  $v_2$  grafu  $G$  dokonując w każdym kroku obliczeń niedeterministycznego wyboru kolejnego wierzchołka ją tworzącego. Zapamiętując na taśmie roboczej wyłącznie binarną reprezentację numeru aktualnie przetwarzanego wierzchołka, może rozstrzygnąć ten problem w pamięci logarytmicznej. Aby udowodnić jego NL-zupełność wystarczy więc pokazać, że jest on NL-trudny, a więc dla dowolnego języka  $A \in$  NL istnieje redukcja w pamięci logarytmicznej do problemu PATH.

Niech  $A \in$  NL. Język  $A$  jest więc rozstrzygalny przez niedeterministyczną maszynę Turinga  $M$  używającą pamięci logarytmicznej. Redukcja języka  $A$  do problemu PATH polega na skonstruowaniu grafu skierowanego  $G$  reprezentującego drzewo możliwych obliczeń maszyny  $M$  na słowie wejściowym  $w$ . Jego wierzchołkami są poszczególne konfiguracje rozważanej maszyny. W grafie  $G$  istnieje krawędź  $c_1 \rightarrow c_2$ , jeśli z konfiguracji  $c_1$  maszyna  $M$  może przejść bezpośrednio do konfiguracji  $c_2$ .

Niech  $v_S$  oraz  $v_{ACC}$  oznaczają wierzchołki odpowiadające konfiguracji początkowej oraz konfiguracji akceptującej maszyny  $M$ . Jeśli maszyna  $M$  akceptuje słowo  $w$ , w drzewie jej obliczeń istnieje ścieżka prowadząca z konfiguracji początkowej do konfiguracji akceptującej. Z drugiej strony, jeśli w grafie  $G$  istnieje ścieżka  $v_S \rightsquigarrow v_{ACC}$ , istnieje ciąg konfiguracji maszyny  $M$  prowadzący z konfiguracji początkowej do konfiguracji akceptującej. Zatem problem rozstrzygnięcia problemu „ $w \in A$ ” jest równoważny rozstrzygnięciu problemu istnienia ścieżki  $v_S \rightsquigarrow v_{ACC}$  w grafie  $G$ .

Pozostaje tylko uzasadnić, że opisana powyżej redukcja jest możliwa w pamięci logarytmicznej. Opiszemy maszynę Turinga  $M_G$  generującą graf obliczeń rozważanej maszyny  $M$  w pamięci logarytmicznej. Kodowanie grafu składa się z listy wierzchołków oraz listy krawędzi. Przypomnijmy, że dla maszyn Turinga z wyróżnionymi taśmami wejściową oraz wyjściową, konfiguracja opisana jest za pomocą zawartości taśm roboczych oraz pozycji jej głowic na wszystkich taśmach. Ponieważ maszyna  $M$  działa w pamięci rozmiaru  $O(\log n)$ , każda jej konfiguracja może być opisana za pomocą słowa rozmiaru  $c \log n$ . Maszyna  $M_G$  musi więc przejrzeć wszystkie słowa rozmiaru  $c \log n$  i umieścić na taśmie wynikowej wyłącznie poprawne konfiguracje maszyny  $M$ . W podobny sposób maszyna  $M_G$  generuje zbiór krawędzi grafu  $G$ . Aby rozstrzygnąć istnienie krawędzi  $c_1 \rightarrow c_2$  w grafie  $G$ , maszyna  $M_G$



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

sprawdza, czy zgodnie z funkcją przejścia maszyny  $M$  możliwe jest bezpośrednie przejście do konfiguracji  $c_2$ .  $\square$

### Problem P vs NP

Przypomnijmy, że P jest klasą języków rozstrzygalnych w czasie wielomianowym przez *deterministyczne* maszyny Turinga, natomiast NP jest klasą języków rozstrzyganych w czasie wielomianowym przez *niedeterministyczne* maszyny Turinga. Rozwiązywanie problemów z klasy NP na maszynach deterministycznych zazwyczaj powoduje wykładowiczy wzrost czasu obliczeń. Problemy należące do klasy NP mają jednak bardzo ważną własność nazywaną własnością wielomianowej weryfikacji. Jeśli znamy jest rozwiązanie takiego problemu, jego poprawność może być zweryfikowana na deterministycznej maszynie Turinga w czasie wielomianowym.

Dla przykładu przypomnijmy problem polegający na rozstrzygnięciu czy w grafie skierowanym  $G$  istnieje ścieżka Hamiltona łącząca dwa ustalone wierzchołki  $v_1$  oraz  $v_2$ .

$$\text{HAMPATH} = \{(G, v_1, v_2) : G \text{ zawiera ścieżkę Hamiltona } v_1 \rightsquigarrow v_2\}.$$

Na maszynie niedeterministycznej problem ten może być rozwiązany w czasie wielomianowym, natomiast maszyna deterministyczna będzie potrzebowała czasu wykładowicznego. Jeśli jednak dostaniemy opis ścieżki  $v_1 \rightsquigarrow v_2$  (wyznaczonej w dowolny sposób), maszyna deterministyczna będzie w stanie zweryfikować w czasie wielomianowym czy jest ona ścieżką Hamiltona.

Zauważmy jednak, że własność wielomianowej weryfikacji nie jest cechą wszystkich problemów rozstrzygalnych w czasie wykładowiczym za pomocą deterministycznych maszyn Turinga. Rozważmy dopełnienie problemu HAMPATH:

$$\overline{\text{HAMPATH}} = \{(G, v_1, v_2) : G \text{ nie zawiera ścieżki Hamiltona } v_1 \rightsquigarrow v_2\}$$

Znając rozwiązanie tego problemu (wyznaczone w czasie wykładowiczym), aby zweryfikować jego poprawność nadal musimy wykonać wykładowiczą liczbę kroków.

Każda deterministyczna maszyna Turinga jest szczególnym przypadkiem maszyny niedeterministycznej, zatem zachodzi oczywista inkluza  $P \subseteq NP$ . Nie wiemy jednak czy jest ona właściwa (tzn. czy klasy P oraz NP są istotnie różne). Dlatego też wśród problemów należących do klasy NP na szczególną uwagę zasługują problemy NP-zupełne. Ich czasowa złożoność obliczeniowa jest ściśle związana ze złożonością całej klasy NP.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
FUNDUSZ SPOŁECZNY

Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Każdy problem z klasy NP (również NP-zupełny) może być w czasie wielomianowym zredukowany do dowolnego problemu NP-zupełnego. Zatem, z rozwiązywalności w deterministycznym czasie wielomianowym dowolnego problemu NP-zupełnego wynika rozwiązywalność w deterministycznym czasie wielomianowym dla dowolnego problemu z klasy NP. Podobnie, aby udowodnić, że żadnego problemu z klasy NP nie da się rozwiązać w deterministycznym czasie wielomianowym, wystarczy wykazać ten fakt dla dowolnie wybranego problemu NP-zupełnego.

W klasie NP znajduje się wiele problemów o znaczeniu praktycznym. Przykładem może być problem rozkładu liczby na czynniki pierwsze, Jego trudność obliczeniowa jest podstawą powszechnie używanego algorytmu szyfrującego RSA. Dlatego też problem rozstrzygnięcia równości  $P = NP$  jest jednym z najważniejszych problemów współczesnej informatyki teoretycznej. Za jego poprawne rozwiązanie ufundowano nawet nagrodę w wysokości miliona dolarów. Jednak mimo ogromnego wysiłku wielu badaczy status tego problemu nadal pozostaje nieznany. Z kilkoma przykładami problemów NP-zupełnych zapoznamy się w czasie kolejnych wykładów.

## Relacje między klasami złożoności obliczeniowej

Klasa złożoności obejmuje grupę problemów o podobnej trudności obliczeniowej (czasowej lub pamięciowej). Wiemy już, że klasy złożoności tworzą hierarchię: niektóre z nich pokrywają się ze sobą, niektóre są istotnie różne, dla jeszcze innych wzajemna relacja nie jest jeszcze znana. Intuicyjnie oczekujemy, że klasy z bardziej ograniczonymi zasobami będą zawierały mniej języków niż klasy z mniejszymi ograniczeniami. Poniżej spróbujemy podsunąć znane fakty dotyczące relacji między klasami złożoności obliczeniowej.

Przypadki ogólne:

- $\text{DTIME}(f(n)) \subseteq \text{DSPACE}(f(n))$  oraz  $\text{NTIME}(f(n)) \subseteq \text{NSPACE}(f(n))$   
Dowolna maszyna Turinga (deterministyczna lub niedeterministyczna) działająca w czasie  $f(n)$  będzie w stanie wykorzystać co najwyżej  $f(n)$  komórek pamięci.
- $\text{NTIME}(f(n)) \subseteq \text{DTIME}(c^{f(n)})$ , dla pewnego  $c \in \mathbb{N}$   
Każda ścieżka obliczeń maszyny niedeterministycznej  $M_1$  działającej w czasie  $f(n)$  ma długość co najwyżej  $f(n)$ . Jeśli  $c \in \mathbb{N}$  oznacza maksymalną liczbę niedeterministycznych wybór w każdym kroku obliczeń maszyny  $M_1$ , to symulująca jej obliczenia maszyna deterministyczna wykoną co najwyżej  $c^{f(n)}$  kroków.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

- $\text{DSPACE}(f(n)) \subseteq \text{DTIME}(c^{f(n)})$ , dla pewnego  $c \in \mathbb{N}$

Maszyna Turinga, która dla słowa wejściowego długości  $n$  wykorzystuje  $f(n)$  komórek taśmy, może mieć co najwyżej  $c^{f(n)}$  różnych konfiguracji, gdzie  $c \in \mathbb{N}$  zależy od słowa wejściowego oraz alfabetu taśmy. Ponieważ rozważana maszyna zatrzymuje się dla każdego wejścia, żadna jej konfiguracja nie może się powtórzyć.

- $\text{NSPACE}(f(n)) \subseteq \text{DSPACE}(f^2(n))$

Wynika bezpośrednio z twierdzenia Savitcha.

Przypadki szczególne:

- $P \subsetneq \text{EXPTIME}$

Każdy problem rozstrzygalny w czasie wielomianowym jest w oczywisty sposób rozstrzygalny również w czasie wykładniczym. Z drugiej strony, potrafimy wskazać problemy rozstrzygalne w czasie wykładniczym, których nie da się rozstrzygnąć w czasie wielomianowym. Przykładem takiego problemu może być  $\text{HALT}_K$  – rozstrzygalność zatrzymywania się maszyny Turinga na dowolnych danych w co najwyżej  $k$  krokach. A zatem inkluзja ta jest właściwa.

Podobnie możemy uzasadnić, że  $NP \subsetneq \text{NEXPTIME}$ .

- $\text{PSPACE} = \text{NPSPACE}$

Wynika bezpośrednio z twierdzenia Savitcha.

- $P \subseteq \text{PSPACE}$  oraz  $NP \subseteq \text{NPSPACE}$

Dowolna maszyna Turinga działająca w czasie wielomianowym (deterministycznym lub niedeterministycznym) może wykorzystać co najwyżej wielomianową liczbę komórek taśmy. Zatem, na mocy twierdzenia Savitcha,  $NP \subseteq \text{PSPACE}$ .

- $\text{PSPACE} \subseteq \text{EXPTIME}$

Zauważmy, że dla  $f(n) \geq n$ , maszyna Turinga działająca w pamięci  $f(n)$  może mieć co najwyżej  $f(n) \cdot 2^{O(f(n))}$  różnych konfiguracji. Ponadto, jeśli maszyna ta zatrzymuje się dla każdego wejścia, żadna konfiguracja nie może wystąpić dwukrotnie.

- $L \subseteq NL$

Dowolny problem rozstrzygalny w pamięci logarytmicznej przez maszynę deterministyczną jest również rozstrzygalny w pamięci logarytmicznej przez maszynę niedeterministyczną. Nie wiemy natomiast, czy



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

inkluzja ta jest właściwa. Aby rozstrzygnąć ten problem trzeba rozstrzygnąć pozytywnie lub negatywnie problem należenia do klasy L dla dowolnego problemu NL-zupełnego.

- $P \subseteq NP$

Dowolny problem rozstrzygalny w czasie wielomianowym przez maszynę deterministyczną jest również rozstrzygalny w czasie wielomianowym przez maszynę niedeterministyczną. Nie wiemy natomiast, czy inkluzja ta jest właściwa. Aby rozstrzygnąć ten problem trzeba rozstrzygnąć pozytywnie lub negatywnie problem należenia do klasy P dla dowolnego problemu NP-zupełnego.

- $NL \subseteq P$

Niedeterministyczna maszyna Turinga rozstrzygająca dowolny problem w pamięci logarytmicznej może mieć co najwyżej wielomianową liczbę konfiguracji. Ponieważ rozważana maszyna musi zatrzymywać się dla każdych danych wejściowych, żadna konfiguracja nie może się powtarzać. Zatem działa ona w czasie wielomianowym.

### Problem P vs NP raz jeszcze

Na podstawie powyższych rozważań wiemy, że zachodzą następujące inkluzje  
 $P \subseteq NP \subseteq EXPTIME \subseteq NEXPTIME$ .

Wiemy jednak również, że klasy P oraz EXPTIME są istotnie różne. Zatem co najmniej jedna z inkluzji  $P \subseteq NP$  lub  $NP \subseteq EXPTIME$  jest właściwa (być może obie). Gdyby udało się udowodnić równość  $P = NP$ , oznaczałoby to również równość  $EXPTIME = NEXPTIME$ .



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Wykład 14

### Problemy NP-zupełne

W czasie poprzedniego wykładu rozważaliśmy zależności między klasami złożoności obliczeniowej. Szczególną uwagę zwróciliśmy na dwie klasy złożoności czasowej: klasę P składającą się z problemów rozstrzygalnych w czasie wielomianowym na *deterministycznych* maszynach Turinga, oraz klasę NP składającą się z problemów rozstrzygalnych w czasie wielomianowym na *niedeterministycznych* maszynach Turinga. Klasę NP możemy również określić jako klasę problemów posiadających własność wielomianowej weryfikacji. Oznacza to, że poprawność rozwiązania rozważanego problemu może zostać zweryfikowana w czasie wielomianowym na maszynie deterministycznej.

W klasie NP wyróżniliśmy grupę problemów, których czasowa złożoność obliczeniowa jest ściśle związana ze złożonością obliczeniową całej klasy – tzw. problemy NP-zupełne. Przypomnijmy, problem nazywamy NP-zupełnym jeśli jest w klasie NP oraz każdy problem z klasy NP może być do niego zredukowany w czasie wielomianowym. W czasie bieżącego wykładu poznamy dwa problemy, które będą stanowiły podstawę dowodzenia NP-zupełności innych problemów.

### Spełnialność formuł logicznych – SAT

*Formułę logiczną* nazywamy wyrażenie składające się ze zmiennych oraz operacji logicznych: negacji, koniunkcji oraz alternatywy. Powiemy, że formuła logiczna  $\phi$  jest *spełnialna* jeśli istnieje takie wartościowanie zawartych w niej zmiennych, dla którego cała formuła jest prawdziwa. Przykładowo, formuła

$$\phi = (\neg x \vee y) \wedge (z \vee \neg y) \wedge (\neg z \vee \neg y)$$

jest prawdziwa dla wartościowania ( $x = \text{false}$ ,  $y = \text{false}$ ,  $z = \text{true}$ ), a więc jest spełnialna.

Pierwszym przykładem problemu NP-zupełnego jest *problem spełnialności* polegający na sprawdzeniu czy dana formuła logiczna jest spełnialna.

$$SAT = \{\phi : \phi \text{ jest spełnialna}\}.$$

Zauważmy, że maszyna deterministyczna może rozwiązać ten problem w czasie wykładniczym testując wszystkie możliwe wartościowania zmiennych. Maszyna niedeterministyczna może natomiast rozwiązać go w czasie wielomianowym wybierając od razu właściwe wartościowanie. Pozostało udowodnić,



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

że każdy problem z klasy NP może być w czasie wielomianowym zredukowany do problemu SAT.

#### Twierdzenie 44 (Cooke-Levin)

Problem SAT jest NP-zupełny. ( $SAT \in P$  wtedy i tylko wtedy, gdy  $P=NP$ ).

#### Dowód

(1)  $SAT \in NP$ .

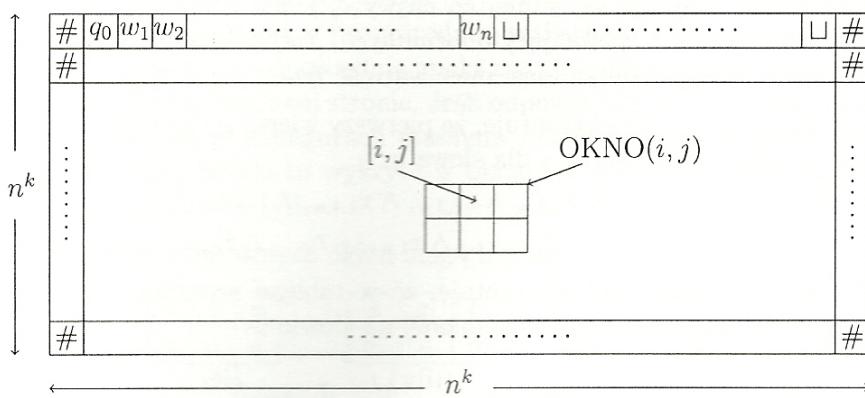
Niedeterministyczna maszyna Turinga może w czasie wielomianowym wybrać wartościowanie zmiennych formuły  $\phi$  i zaakceptować jeśli dla tego wartościowania formuła jest prawdziwa. Równoważnie, dla znanego wartościowania maszyna deterministyczna może w czasie wielomianowym zweryfikować prawdziwość  $\phi$ .

(2) SAT jest NP-trudny.

Musimy udowodnić, że dowolny język  $A \in NP$  jest redukowalny do problemu SAT w czasie wielomianowym. Niech  $M_A$  będzie niedeterministyczną maszyną Turinga rozstrzygającą język  $A$  w czasie  $n^k$  dla pewnej stałej  $k \in \mathbb{N}$ . Dla maszyny  $M_A$  oraz słowa  $w \in \Sigma^*$  skonstruujemy formułę  $\phi_{M_A}(w)$  spełnialną wtedy i tylko wtedy, gdy maszyna  $M_A$  akceptuje  $w$ .

#### Tableau maszyny Turinga

Tableau dla maszyny  $M_A$  oraz słowa  $w = w_1 w_2 \dots w_n$  nazywamy tablicę rozmiaru  $n^k \times n^k$ , w której wierszach zapisane są konfiguracje ścieżki obliczeń maszyny  $M_A$  dla słowa  $w$ . Dla uproszczenia zakładamy, że wszystkie konfiguracje  $M_A$  mają tę samą długość ( $n^k$ ) oraz rozpoczynają i kończą się znakiem specjalnym #.



Rysunek 14: Tableau dla maszyny Turinga oraz słowa  $w = w_1 w_2 \dots w_n$ .



Pierwszy wiersz tableau zawiera konfiguracje początkową maszyny  $M_A$ , natomiast każdy kolejny wynika z poprzedniego zgodnie z jej funkcją przejścia. Powiemy, że tableau jest akceptujące jeśli zawiera wiersz odpowiadający konfiguracji akceptującej maszyny  $M_A$ . Zauważmy, że każde akceptujące tableau maszyny  $M_A$  dla słowa  $w$  odpowiada akceptującej ścieżce obliczeń maszyny  $M_A$  na  $w$ . Zatem maszyna  $M_A$  akceptuje słowo  $w$  dokładnie wtedy, gdy istnieje akceptujące tableau maszyny  $M_A$  dla  $w$ .

### Redukcja języka $A \in NP$ do problemu SAT

Niech  $C = Q \cup \Gamma \cup \{\#\}$ , gdzie  $Q$  jest zbiorem stanów zaś  $\Gamma$  alfabetem taśmy maszyny  $M_A$ . Dla każdego  $s \in C$  oraz  $1 \leq i, j \leq n^k$  tworzymy zmienną  $x_{i,j,s}$ . Zmienna  $x_{i,j,s}$  ma wartość *true* jeśli w komórce tableau o współrzędnych  $[i, j]$  znajduje się znak  $s$ . Formuła  $\phi_{M_A}(w)$  dla maszyny  $M_A$  oraz słowa  $w$  ma postać

$$\phi_{M_A}(w) = \phi_{cell}(w) \wedge \phi_{start}(w) \wedge \phi_{akceptuj}(w) \wedge \phi_{ruch}(w).$$

Poszczególne składniki formuły  $\phi_{M_A}(w)$  opiszemy szczegółowo poniżej.

(a) Formuła  $\phi_{cell}$  opisuje zawartość komórek tableau zapewniając, że w każdej z nich może znajdować się dokładnie jeden symbol:

$$\phi_{cell}(w) = \bigwedge_{1 \leq i, j \leq n^k} \left[ \left( \bigvee_{s \in C} x_{i,j,s} \right) \wedge \left( \bigwedge_{s,t \in C, s \neq t} (\neg x_{i,j,s} \vee \neg x_{i,j,t}) \right) \right]$$

Formuła  $\phi_{cell}$  składa się z koniunkcji formuł opisujących zawartość każdej komórki  $[i, j]$  tableau dla  $1 \leq i, j \leq n^k$ . Pierwsza część formuły zapewnia, że w każdej komórce znajduje się co najmniej jeden znak. Druga część zapewnia, że znajduje się w niej co najwyżej jeden znak. Zatem w dowolnym wartościowaniu spełniającym formułę  $\phi_{M_A}(w)$  dokładnie jedna zmienna związana z każdą komórką musi mieć wartość *true*.

(b) Formuła  $\phi_{start}(w)$  gwarantuje, że pierwszy wiersz tableau jest konfiguracją początkową maszyny  $M_A$  dla słowa  $w$ :

$$\begin{aligned} \phi_{start}(w) = & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \dots \\ & \dots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\square} \wedge \dots \wedge x_{1,n^k-1,\square} \wedge x_{1,n^k,\#}. \end{aligned}$$

(c) Formuła  $\phi_{akceptuj}(w)$  gwarantuje, że w tableau występuje akceptująca konfiguracja maszyny  $M_A$  uruchomionej na słowie  $w$ :

$$\phi_{akceptuj}(w) = \bigvee_{1 \leq i, j \leq n^k} x_{i,j,q_{ACC}}.$$



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

(d) Formuła  $\phi_{ruch}(w)$  gwarantuje, że każdy wiersz tableau jest opisem konfiguracji wynikającej bezpośrednio z konfiguracji opisanej w wierszu poprzedzającym zgodnie z funkcją przejścia maszyny  $M_A$ . Przypomnijmy, że konfigurację maszyny  $M_A$  możemy opisać za pomocą słowa  $uq_iv$  oznaczającego, że  $M_A$  jest w stanie  $q_i$ , jej taśma zawiera słowo  $uv$ , zaś jej głowica znajduje się nad pierwszym znakiem słowa  $v$ .

Niech  $OKNO(i, j)$  oznacza blok w tableau rozmiaru  $2 \times 3$ :

$$OKNO(i, j) = \{[i, j-1], [i, j], [i, j+1], [i+1, j-1], [i+1, j], [i+1, j+1]\}.$$

Powiemy, że  $OKNO(i, j)$  jest *poprawne* jeśli jego zawartość nie narusza zasad przejścia między kolejnymi konfiguracjami maszyny  $M_A$  zgodnie z jej funkcją przejścia  $\delta$ .

### Przykład 35

Przypuśćmy, że  $\delta(q_1, a) = \{(q_1, b, R)\}$  oraz  $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$ . Przykładami poprawnych okien są wówczas:

a	$q_1$	b
$q_2$	a	c

(i)

a	$q_1$	b
a	a	$q_2$

(ii)

#	b	a
#	b	a

(iii)

a	a	$q_1$
a	a	b

(iv)

a	b	a
a	b	$q_2$

(v)

b	b	b
c	b	b

(vi)

Okna (i) oraz (ii) są w oczywisty sposób zgodne z funkcją przejścia  $\delta$ , zaś okno (iii) odpowiada brakowi zmian w komórkach nie znajdujących się w bezpośrednim sąsiedztwie głowicy. Okna (iv), (v) oraz (vi) odpowiadają natomiast zmianom zawartości w bezpośrednim sąsiedztwie głowicy i są poprawne pod warunkiem, że odpowiednie znaki znajdują się we właściwych miejscach po ich prawej lub lewej stronie. Jeśli odpowiadające im wiersze tableau nie opisują kolejnych konfiguracji maszyny  $M_A$  otrzymanych zgodnie z jej funkcją przejścia, będzie to wykryte w oknach bezpośrednio sąsiadujących z oknami (iv), (v) oraz (vi).

Przykładami niepoprawnych okien mogą być natomiast:

a	b	a
a	a	a

(vii)

a	$q_1$	b
$q_1$	a	a

(viii)

b	$q_1$	b
$q_2$	b	$q_2$

(ix)

Okno (vii) jest niepoprawne, ponieważ zmiana zawartości może nastąpić wyłącznie w komórce znajdującej się bezpośrednio pod głowicą. Okno (viii) jest



niepoprawne, ponieważ jego zawartość jest niezgodna z funkcją przejścia  $\delta$ . Okno (ix) jest niepoprawne, ponieważ jego dolny wiersz zawiera dwa symbole stanów kodujące dwie pozycje jednej głowicy.

Podstawą konstrukcji formuły  $\phi_{ruch}(w)$  będzie następujące stwierdzenie.

### Stwierdzenie 1

*Jeśli pierwszy wiersz tableau zawiera konfigurację początkową maszyny  $M_A$  oraz każde OKNO w tableau jest poprawne, to każdy wiersz w tableau jest konfiguracją  $M_A$  poprawnie wynikającą z konfiguracji znajdującej się w wierszu poprzedzającym.*

Aby uzasadnić prawdziwość powyższego stwierdzenia rozważmy konfiguracje maszyny  $M_A$  opisane przez parę sąsiadujących wierszy tableau. Górnego i dolnego wiersza każdego okna nie znajdującego się w bezpośrednim sąsiedztwie symbolu stanu (kodującego położenie głowicy maszyny  $M_A$ ) powinny być identyczne. Zawartość każdego okna sąsiadującego z symbolem stanu maszyny powinna być natomiast zgodna z jej funkcją przejścia. Spełnienie powyższych warunków gwarantuje, że konfiguracja opisana w dolnym wierszu rozważanej pary wynika bezpośrednio z konfiguracji opisanej w górnym wierszu zgodnie z funkcją przejścia maszyny  $M_A$ .

Formułę  $\phi_{ruch}(w)$  możemy zatem zapisać jako:

$$\phi_{ruch}(w) = \bigwedge_{1 \leq i < n^k, 1 < j < n^k} „OKNO(i, j) jest poprawne”,$$

gdzie dla symboli  $a_1, a_2, a_3, a_4, a_5, a_6$  formuła „OKNO( $i, j$ ) jest poprawne” jest postaci

$$\phi_O = x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6}.$$

### Złożoność obliczeniowa redukcji

Ostatnim etapem dowodu NP-zupełności problemu SAT będzie oszacowanie złożoności czasowej skonstruowanej powyżej redukcji. Zgodnie z założeniem maszyna  $M_A$  rozstrzyga język w czasie  $n^k$

- Tableau zawiera  $n^{2k}$  komórek, dla każdej z nich tworzymy  $|C|$  zmiennych, zatem formuła  $\phi_{M_A}$  zawiera  $O(n^{2k})$  zmiennych.
- Formuła  $\phi_{cell}$  zawiera element stałego rozmiaru dla każdej komórki tableau, ma więc rozmiar  $O(n^{2k})$ .
- Formuła  $\phi_{start}$  zawiera element stałego rozmiaru dla każdej komórki w pierwszym wierszu, ma więc rozmiar  $O(n^k)$ .



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

- Formuła  $\phi_{akceptuj}$  zawiera element stałego rozmiaru dla każdej komórki tableau, ma więc rozmiar  $O(n^{2k})$ .
- Formuła  $\phi_{ruch}$  zawiera element stałego rozmiaru dla każdej komórki tableau, ma więc rozmiar  $O(n^{2k})$ .

Podsumowując, cała formuła  $\phi_{M_A}$  ma rozmiar  $O(n^{2k})$ . Możliwe jest zatem wygenerowanie jej w czasie wielomianowym.  $\square$

## Problem spełnialności 3SAT

Problem spełnialności 3SAT, jest modyfikacją problemu SAT, w której postać formuły logicznej jest dość mocno ograniczona. Jednak mimo tych ograniczeń problem 3SAT również okazuje się być NP-zupełny.

*Literałem* nazywamy zmienną logiczną lub jej zaprzeczenie, np.  $x$  lub  $\neg y$ . *Klauzulą* nazywamy kilka literałów połączonych spójnikiem logicznym  $\vee$ , np.  $(x \vee \neg y \vee z \vee \neg x)$ . Powiemy, że formuła logiczna  $\phi$  jest w *koniunktywnej postaci normalnej* (CNF), jeśli składa się z ciągu klauzul połączonych spójnikiem logicznym  $\wedge$ , np.

$$\phi = (x \vee \neg z) \wedge (x \vee y \vee \neg z \vee \neg y) \wedge y.$$

Powiemy, że formuła logiczna  $\phi$  jest w postaci 3CNF, jeśli każda klaузula zawiera dokładnie 3 literały.

$$3SAT = \{\phi : \phi \text{ jest spełnialną formułą w postaci 3CNF}\}.$$

### Twierdzenie 45

Problem 3SAT jest NP-zupełny.

#### Dowód

Zauważmy, że niedeterministyczna maszyna Turinga może wybrać wartościanie zmiennych i rozstrzygnąć prawdziwość rozważanej formuły w czasie wielomianowym. Zatem  $3SAT \in NP$ .

Musimy pokazać, że dowolny język  $A \in NP$  jest redukowalny w czasie wielomianowym do problemu 3SAT. W tym celu zmodyfikujemy nicco dowód NP-zupełności problemu SAT. Niech  $M_A$  będzie maszyną Turinga rozstrzygającą język  $A \in NP$ ,  $w \in \Sigma^*$ , zaś

$$\phi_{M_A}(w) = \phi_{cell}(w) \wedge \phi_{start}(w) \wedge \phi_{akceptuj}(w) \wedge \phi_{ruch}(w)$$

będzie formułą skonstruowaną w dowodzie Twierdzenia 44. Zauważmy, że:



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

- Formuły  $\phi_{cell}$  oraz  $\phi_{start}$  są w koniunktywnej postaci normalnej (CNF)
- Formuła  $\phi_{akceptuj}$  jest pojedynczą klauzulą (czyli jest w postaci CNF)
- Formuła  $\phi_{ruch}$  jest alternatywą koniunkcji i może być przekształcona w koniunkcję alternatyw (postać CNF) za pomocą praw de Morgana.

Zatem, formuła  $\phi_{M_A}(w)$  może być przekształcona do koniunktywnej postaci normalnej. Ponadto, dowolna formuła w postaci CNF może być zamieniona na równoważną jej formułę w postaci 3CNF za pomocą poniższej procedury.

#### Procedura zamiany formuły w postaci CNF na 3CNF

- Klauzule zawierające mniej niż 3 zmienne rozszerzamy powtarzając jedno- lub dwukrotnie wystąpienie dowolnej z nich.
- Klauzule zawierające więcej niż 3 zmienne dzielimy na klauzule rozmiaru 3 dodając nowe zmienne nie zmieniając jej spełnialności. Formalnie, klauzulę postaci

$$(a_1 \vee a_2 \vee \dots \vee a_m)$$

zamienimy na ciąg klauzul

$$(a_1 \vee a_2 \vee z_1) \wedge (\neg z_1 \vee a_3 \vee z_2) \wedge (\neg z_2 \vee a_4 \vee z_3) \wedge \dots \wedge (\neg z_{m-3} \vee a_{m-1} \vee a_m),$$

gdzie  $z_1, \dots, z_{m-3}$  nie występują w rozważanej formule.

Podsumowując, dla maszyny Turinga  $M_A$  oraz słowa  $w$  konstrujemy formułę logiczną  $\phi_{M_A}(w)$ , która jest prawdziwa wtedy i tylko wtedy, gdy maszyna  $M_A$  akceptuje słowo  $w$ . Przekształcamy  $\phi_{M_A}(w)$  w równoważną jej formułę  $\phi'_{M_A}$  w postaci CNF modyfikując składnik  $\phi_{ruch}$  zgodnie z prawami de Morgana, a następnie w formułę  $\phi''_{M_A}$  w postaci 3CNF za pomocą opisanej powyżej procedury. Ponieważ rozważane przekształcenia mogą być wykonane w czasie wielomianowym, otrzymujemy wielomianową redukcję języka  $A$  do problemu spełnialności 3SAT, co kończy dowód.  $\square$

#### Uwaga

Jeżeli rozmiar klauzul formuły logicznej w koniunktywnej postaci normalnej ograniczymy do dwóch zmiennych, otrzymamy problem 2SAT posiadający rozwiązanie w czasie wielomianowym.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

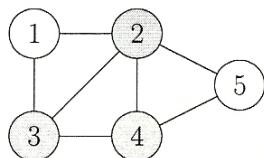
## Wykład 15

# Przykłady problemów NP-zupełnych

W czasie poprzedniego wykładu poznaliśmy dwa ważne problemy NP-zupełne SAT oraz 3SAT związane ze spełnialnością formuł logicznych. W czasie dzisiejszego wykładu wykorzystamy je do wykazania NP-zupełności kilku innych ciekawych problemów obliczeniowych.

Przypomnijmy, problem nazywamy NP-zupełnym jeśli należy do klasy NP oraz każdy inny problem z klasy NP redukuje się do niego w czasie wielomianowym. Dowodząc NP-zupełność nowego problemu wystarczy więc wykazać, że dowolny znany już problem NP-zupełny redukuje się do niego w czasie wielomianowym.

## Problem pokrycia wierzchołkowego



Niech  $G$  będzie grafem nieskierowanym. Podzbiór  $S$  jego wierzchołków nazywamy pokryciem wierzchołkowym  $G$ , jeśli każda krawędź  $G$  ma co najmniej jeden koniec w zbiorze  $S$ . Przykładowo, dla grafu umieszczonego na rysunku pokryciem wierzchołkowym może być zbiór  $S = \{2, 3, 4\}$ .

Problem pokrycia wierzchołkowego polega na rozstrzygnięciu czy graf posiada pokrycie wierzchołkowe ustalonego rozmiaru.

$$\text{VERTEX-COVER} = \{(G, k) : G \text{ ma pokrycie wierzchołkowe rozmiaru } k\}.$$

### Twierdzenie 46

Problem VERTEX-COVER jest NP-zupełny.

#### Dowód

(1) VERTEX-COVER  $\in$  NP

Dla dowolnego podzbioru  $S$  składającego się z  $k$  wierzchołków grafu  $G$  deterministyczna maszyna Turinga jest w stanie w czasie wielomianowym zweryfikować, czy jest on pokryciem wierzchołkowym  $G$ .

(2) VERTEX-COVER jest NP-zupełny

Pokażemy redukcję problemu 3SAT do problemu VERTEX-COVER działającą w czasie wielomianowym. W tym celu dla dowolnej formuły logicznej  $\phi$



w postaci 3CNF zbudujemy graf  $G$  oraz wyznaczymy liczbę  $k$  w taki sposób, że formuła  $\phi$  będzie spełnialna wtedy i tylko wtedy, gdy graf  $G$  będzie miał pokrycie wierzchołkowe rozmiaru  $k$ .

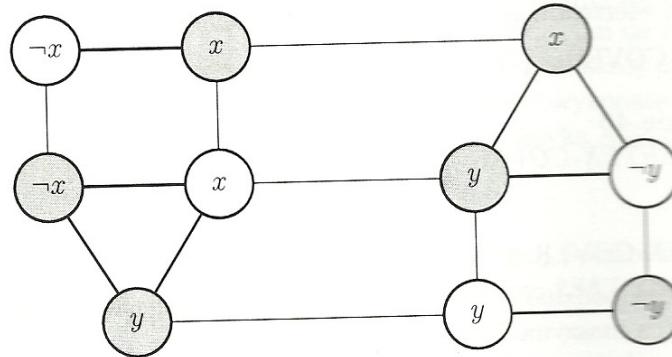
Niech  $\phi$  będzie formułą logiczną w postaci 3CNF składającą się z  $m$  zmiennych oraz  $l$  klauzul. Odpowiadający jej graf  $G$  konstruujemy według następującego schematu:

- Dla każdej zmiennej  $x$  występującej w  $\phi$  tworzymy *podgraf zmiennej* składający się z dwóch wierzchołków etykietowanych  $x$  oraz  $\neg x$  połączonych krawędzią.
- Dla każdej klauzuli  $(x \vee y \vee z)$  tworzymy *podgraf klauzuli* składający się z trzech wierzchołków z etykietami odpowiadającymi występującym w niej literałom połączonych krawędziami każdy z każdym.
- Każdy wierzchołek  $v$  występujący w podgrafie klauzuli łączymy z wierzchołkiem  $v'$  w podgrafie zmiennej o identycznej etykiecie.

Tak skonstruowany graf  $G$  ma  $2m + 3l$  wierzchołków. Ustalamy rozmiar pokrycia wierzchołkowego na  $k = m + 2l$ .

### Przykład 36

Niech  $\phi = (x \vee y \vee \neg x) \wedge (x \vee y \vee \neg y)$ . Rozważana formuła przyjmuje wartość *true* dla wartościowania ( $x = \text{true}$ ,  $y = \text{false}$ ), a zatem jest spełnialna. Odpowiadający jej graf nieskierowany z zaznaczonym pokryciem wierzchołkowym rozmiaru  $2 + 2 \cdot 2 = 6$  znajduje się na rysunku poniżej. Podgrupy zmiennych oraz klauzul zostały wyróżnione grubszymi liniami.



Musimy teraz udowodnić, że graf  $G$  skonstruowany dla formuły  $\phi$  według schematu opisanego powyżej ma pokrycie wierzchołkowe rozmiaru  $k = m + 2l$  wtedy i tylko wtedy, gdy formuła  $\phi$  jest spełnialna.



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

==>:

Założymy, że formuła  $\phi$  jest spełnialna. Pokrycie wierzchołkowe  $S$  rozmiaru  $k = m + 2l$  w grafie  $G$  konstruujemy następująco:

- Z każdego podgrafa zmiennej do  $S$  wybieramy wierzchołek odpowiadający literałowi mającemu wartość  $true$ .
- Z każdego podgrafa klauzuli wybieramy jeden literał mający wartość  $true$  i dodajemy do  $S$  pozostałe 2 wierzchołki.

Zauważmy, że skonstruowaliśmy podzbiór  $S$  rozmiaru  $k = m + 2l$  zbioru wszystkich wierzchołków  $G$  o następujących własnościach:

- Każda krawędź należąca do podgrafa zmiennej jest pokryta.
- Każda krawędź należąca do podgrafa klauzuli jest pokryta.
- Każda krawędź łącząca podgraf klauzuli z podgrafem zmiennej jest pokryta przez wierzchołek z wartością  $true$  z podgrafa zmiennej lub jeden z dwóch pozostałych wierzchołków z podgrafa klauzuli.

Zatem  $S$  jest pokryciem wierzchołkowym grafu  $G$  o rozmiarze  $k = m + 2l$ .

$\Leftarrow$ :

Założymy, że zbiór  $S$  jest pokryciem wierzchołkowym grafu  $G$  o rozmiarze  $k = m + 2l$ . Wówczas zbiór  $S$  musi zawierać po jednym wierzchołku z każdego podgrafa zmiennej oraz po dwa wierzchołki z każdego podgrafa klauzuli – razem  $m + 2l$  wierzchołków. Wartościowanie formuły  $\phi$  otrzymujemy przypisując wartość  $true$  wszystkim wierzchołkom wybranym z podgrafów zmiennej. Dzięki temu co najmniej jeden literał w każdej klauzuli jest prawdziwy, a zatem cała formuła również jest prawdziwa.

Pozostało nam jeszcze oszacować złożoność obliczeniową rozważanej redukcji. Dla formuły  $\phi$  składającej się z  $m$  zmiennych oraz  $l$  klauzul tworzymy graf składający się z  $2m+3l$  wierzchołków oraz  $6m+l$  krawędzi. Konstrukcja grafu może więc być wykonana w czasie  $O(n^2)$ , gdzie  $n$  jest długością formuły  $\phi$ . □

## Problem kliki

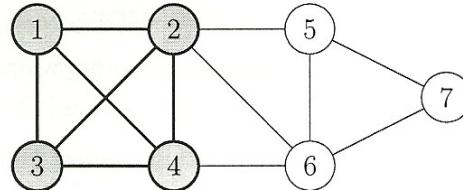
Kliką w grafie nieskierowanym nazywamy podgraf, w którym każde dwa wierzchołki połączone są krawędzią. Klikę rozmiaru  $k$  nazywamy  $k$ -kliką. Problem kliki polega na rozstrzygnięciu, czy graf nieskierowany  $G$  zawiera klikę zadanego rozmiaru.

$$\text{CLIQUE} = \{(G, k) : G \text{ zawiera klikę rozmiaru } k\}.$$



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

Na poniższym rysunku zaznaczona została 4-klika w grafie nieskierowanym.



#### Twierdzenie 47

Problem CLIQUE jest NP-zupełny.

#### Dowód

(1) CLIQUE  $\in$  NP

Dla podanego grafu  $G$  niedeterministyczna maszyna Turinga potrafi w czasie wielomianowym wybrać zbiór  $S$  składający się z  $k$  wierzchołków i rozstrzygnąć czy jest on kliką rozmiaru  $k$ . Podobnie, deterministyczna maszyna Turinga potrafi zweryfikować w czasie wielomianowym, czy wybrany zbiór  $S$  jest  $k$ -kliką w grafie  $G$ .

(2) CLIQUE jest NP-zupełny

Pokażemy redukcję w czasie wielomianowym problemu 3SAT do problemu CLIQUE. Dla formuły logicznej  $\phi$  skonstruujemy graf  $G$ , który będzie zawierał  $k$ -klikę wtedy i tylko wtedy, gdy formuła  $\phi$  jest spełnialna.

Niech  $\phi$  będzie formułą logiczną składającą się z  $k$  klauzul:

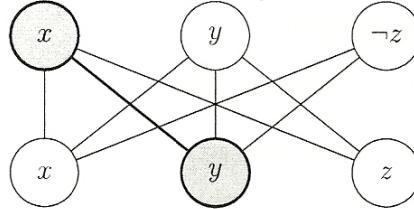
$$\phi = (x_1 \vee y_1 \vee z_1) \wedge \dots \wedge (x_k \vee y_k \vee z_k).$$

Odpowiadający jej graf  $G$  konstruujemy według następującego schematu:

- Tworzymy  $3k$  wierzchołków etykietowanych wystąpieniami wszystkich literałów w formule  $\phi$ .
- Grupujemy wierzchołki trójkami odpowiadającymi klauzulom  $\phi$
- Każdy wierzchołek grafu  $G$  o etykiecie  $x$  łączymy krawędzią ze wszystkimi pozostałymi poza wierzchołkami należącymi do tej samej trójki oraz wierzchołkami etykietowanymi  $\neg x$ .

#### Przykład 37

Niech  $\phi = (x \vee y \vee \neg z) \wedge (x \vee y \vee z)$ . Rozważana formuła jest prawdziwa dla wartościowania  $x = \text{true}, y = \text{true}, z = \text{false}$ , a zatem jest spełnialna. Odpowiadający jej graf znajduje się na poniższym rysunku.



Pogrubionymi liniami zaznaczone zostały wierzchołki 2-kliki. Literał, których prawdziwość gwarantuje prawdziwość całej formuły  $\phi$ , zostały wyróżnione kolorem.

Musimy teraz udowodnić, że w grafie  $G$  skonstruowanym dla formuły  $\phi$  według schematu opisanego powyżej istnieje  $k$ -klika wtedy i tylko wtedy, gdy formuła  $\phi$  jest spełnialna.

≤⇒:

Założymy, że formuła  $\phi$  jest spełnialna. Istnieje zatem wartościowanie, dla którego co najmniej jeden literał z każdej klauzuli ma wartość *true*. Z każdej trójki wierzchołków  $G$  odpowiadającej klauzuli formuły  $\phi$  wybieramy dowolny wierzchołek etykietowany literałem mającym wartość *true*. Otrzymaliśmy zbiór  $S$  składający się z  $k$  wierzchołków. Zauważmy, że żadna para wierzchołków ze zbioru  $S$  nie należy do tej samej trójki. Ponadto, ponieważ wszystkie wybrane literaly mają wartość *true*, żadna para wierzchołków ze zbioru  $S$  nie ma etykiet przeciwnych. Zatem, z konstrukcji grafu  $G$ , każda para wierzchołków ze zbioru  $S$  jest połączona krawędzią. Zbiór  $S$  jest więc  $k$ -kliką w grafie  $G$ .

⇒:

Założymy, że w grafie  $G$  istnieje zbiór wierzchołków  $S$  tworzący  $k$ -klikę. Z konstrukcji  $G$  wynika, że żadna para wierzchołków ze zbioru  $S$  nie należy do tej samej trójki wierzchołków, a tym samym do odpowiadającej jej klauzuli formuły  $\phi$ . Ponadto, żadna para wierzchołków ze zbioru  $S$  nie może mieć etykiet przeciwnych. Przypisując wartość *true* literałom będącym etykietami wierzchołków wybranych do  $S$  zapewniamy, że w każdej klauzuli formuły  $\phi$  dokładnie jeden literał jest prawdziwy. Otrzymujemy więc wartościowanie spełniające formułę  $\phi$ .

Pozostało nam oszacowanie złożoności obliczeniowej opisanej redukcji. Dla formuły  $\phi$  zawierającej  $n$  literałów tworzymy graf składający się z  $n$  wierzchołków oraz  $O(n^2)$  krawędzi. Zatem rozważana redukcja ma złożoność czasową rzędu  $O(n^2)$ . □



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Problem sumy podzbioru

Problem sumy podzbioru polega na rozstrzygnięciu, czy dla danego multizbioru liczb naturalnych  $S = \{x_1, x_2, \dots, x_n\}$  oraz ustalonej liczby naturalnej  $k$  istnieje podzbiór  $Y \subseteq S$ , którego elementy sumują się do  $k$ .

$$\text{SUBSET-SUM} = \{(S, k) : S = \{x_1, \dots, x_n\} \text{ t.z. } \exists_{Y=\{y_1, \dots, y_m\}} \sum y_i = k\}.$$

### Przykład 38

Rozważmy parę  $(X, k) = (\{11, 12, 14, 22, 34, 45\}, 25)$ . Wybierając podzbiór  $Y = \{11, 14\}$  otrzymujemy  $11 + 14 = 25$ , a zatem  $(X, k) \in \text{SUBSET-SUM}$ .

### Twierdzenie 48

Problem SUBSET-SUM jest NP-zupełny.

#### Dowód

##### (1) SUBSET-SUM $\in$ NP

Dla podanego multizbioru  $S$  niedeterministyczna maszyna Turinga potrafi w czasie wielomianowym wybrać zbiór  $Y$ , a następnie wyznaczyć sumę jego elementów. Z drugiej strony, maszyna deterministyczna potrafi w czasie wielomianowym zweryfikować, czy suma elementów wybranego podzbioru  $Y$  jest równa  $k$ .

##### (2) SUBSET-SUM jest NP-zupełny

Pokażemy redukcję w czasie wielomianowym problemu 3SAT do problemu SUBSET-SUM.

Niech  $\phi$  będzie formułą logiczną zawierającą zmienne  $x_1, x_2, \dots, x_m$  oraz klauzule  $c_1, c_2, \dots, c_k$ . Odpowiadający jej zbiór  $S$  tworzymy według następującego schematu. Dla każdej zmiennej  $x_i$  występującej w formule  $\phi$  do zbioru  $S$  dodajemy parę liczb dziesiętnych  $y_i$  oraz  $z_i$ :

$$y_i = 1 \underbrace{0 \dots 0}_{m-i} \underbrace{1 0 \dots 0}_k 1$$

$$z_i = 1 \underbrace{0 \dots 0}_{m-i} \underbrace{0 1 \dots 0}_k 1$$

Lewa część dodawanych liczb odpowiada numeracji zmiennych, natomiast prawa klauzulom. Liczba  $y_i$  zawiera 1 na pozycji  $j$ , jeśli w klauzuli  $c_j$  występuje  $x_i$ , natomiast  $z_i$  zawiera 1 na pozycji  $j$ , jeśli w klauzuli  $c_j$  występuje  $\neg x_i$ . Ponadto, do zbioru  $S$  dodajemy pary liczb  $g_1, h_1, \dots, g_k, h_k$  postaci

$$g_i = h_i = 1 \underbrace{0 \dots 0}_{k-i}$$



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

oraz przyjmujemy

$$z = \underbrace{1 \dots 1}_m \underbrace{3 \dots 3}_k.$$

### Przykład 39

Rozważmy formułę w postaci 3CNF

$$\phi = (x_1 \vee x_2 \vee \neg x_1) \wedge (x_1 \vee x_2 \vee \neg x_2) \wedge (x_1 \vee \neg x_1 \vee \neg x_2).$$

Jest ona prawdziwa dla wartościowania ( $x_1 = \text{false}$ ,  $x_2 = \text{true}$ ), a zatem jest spełnialna. Struktura odpowiadającego jej zbioru  $S$  jest przedstawiona w poniższej tabeli

	$x_1$	$x_2$	$c_1$	$c_2$	$c_3$
$y_1$	1	0	1	1	1
$z_1$	1	0	1	0	1
$y_2$		1	1	1	0
$z_2$	1	0	1	1	1
$g_1$			1	0	0
$h_1$			1	0	0
$g_2$				1	0
$h_2$				1	0
$g_3$					1
$h_3$					1
$z$	1	1	3	3	3

Musimy udowodnić, że zbiór  $S$  zawiera podzbiór  $Y$ , którego elementy sumują się do  $z$ , wtedy i tylko wtedy, gdy formuła  $\phi$  jest spełnialna.

$\Leftarrow$ :

Założymy, że formuła  $\phi$  jest spełnialna. Dla ustalonego wartościowania zmiennych spełniającego  $\phi$  do zbioru  $Y$  wybieramy  $y_i$  jeśli  $x_i$  ma wartość  $true$  lub  $z_i$  jeśli  $x_i$  ma wartość  $false$ . Po zsumowaniu elementów zbioru  $Y$  otrzymujemy liczbę dziesiętną, która na pierwszych  $m$  pozycjach ma cyfrę 1 natomiast na kolejnych  $k$  pozycjach cyfry z zakresu 1-3 (ponieważ co najmniej jeden literal w każdej klauzuli musi mieć wartość  $true$ ). Aby otrzymać sumę elementów równą  $z$ , do zbioru  $Y$  dodajemy niezbędne elementy  $g_i$  oraz  $h_i$ .

Formuła z Przykładu 39 jest spełniona np. dla wartościowania ( $x_1 = \text{false}$ ,  $x_2 = \text{true}$ ). Do zbioru  $Y$  wybieramy więc  $y_1$  oraz  $z_2$  otrzymując sumę jego elementów równą 11122. Aby uzyskać sumę elementów równą 11333 do zbioru  $Y$  dodajemy dodatkowo  $g_1$ ,  $h_1$ ,  $g_2$  oraz  $g_3$ .

⇒:

Załóżmy, że istnieje podzbiór  $Y \subseteq S$ , którego elementy sumują się do  $z$ . Zauważmy, że:

- Wszystkie cyfry liczb tworzących sumę są równe 0 lub 1
- Każda kolumna tablicy opisującej zbiór  $S$  zawiera co najwyżej pięć jedynek, więc przy dodawaniu nie występuje przeniesienie
- Aby otrzymać 1 na  $m$  najbardziej znaczących pozycjach sumy musimy wybrać dokładnie jeden element z każdej pary  $(y_i, z_i)$ .

Wartościowanie formuły  $\phi$  tworzymy przypisując zmiennej  $x_i$  wartość *true* jeśli  $y_i \in Y$  oraz *false* jeśli  $z_i \in Y$ . Ostatnie  $k$  pozycji liczby  $z$  odpowiada poszczególnym klauzulom formuły  $z$ . Na każdej z nich występuje cyfra 3. Wybranie elementów  $g_i$  oraz  $h_i$  daje w sumie co najwyżej 2, zatem każda klauzula  $\phi$  zawiera co najmniej jeden literal ( $x_i$  lub  $\neg x_i$ ) mający wartość *true*. Formuła  $\phi$  jest więc spełnialna.

Pozostało oszacowanie złożoności czasowej opisanej redukcji. Dla formuły  $\phi$  składającej się z  $m$  zmiennych oraz  $k$  klauzul redukcja polega na stworzeniu tabeli zbioru rozmiaru  $O((k+m)^2)$ . Może więc być wykonana w czasie  $O(n^2)$ , gdzie  $n$  jest długością formuły  $\phi$ . □

## Problem ścieżki Hamiltona

Ścieżką Hamiltona w grafie skierowanym  $G$  nazywamy ścieżkę skierowaną przechodzącą przez każdy jego wierzchołek dokładnie raz. Problem ścieżki Hamiltona polega na rozstrzygnięciu, czy w grafie skierowanym  $G$  istnieje ścieżka Hamiltona łącząca dwa ustalone wierzchołki  $v_1$  oraz  $v_2$ .

$$\text{HAMPATH} = \{(G, v_1, v_2) : G \text{ zawiera ścieżkę Hamiltona } v_1 \rightsquigarrow v_2\}.$$

### Twierdzenie 49

Problem HAMPATH jest NP-zupełny.

#### Dowód

(1)  $\text{HAMPATH} \in \text{NP}$

Dla grafu skierowanego  $G$  niedeterministyczna maszyna Turinga może w czasie wielomianowym wybrać ścieżkę łączącą wyróżnione wierzchołki  $v_1$  oraz  $v_2$ . Z drugiej strony, dla wybranej ścieżki maszyna deterministyczna może w czasie wielomianowym zweryfikować, czy jest ona ścieżką Hamiltona.

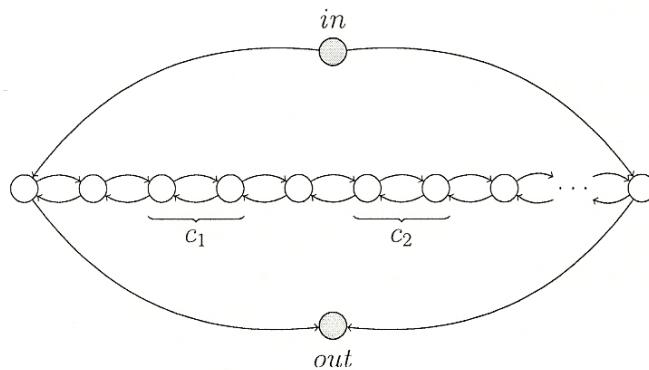
(2) HAMPATH jest NP-zupełny



Pokażemy redukcję problemu spełnialności 3SAT do problemu HAMPATH działającą w czasie wielomianowym. W tym celu dla dowolnej formuły logicznej  $\phi$  w postaci 3CNF zbudujemy graf  $G_\phi$  oraz wybierzemy parę jego wierzchołków  $v_1$  oraz  $v_2$  w taki sposób, że formuła  $\phi$  będzie spełnialna wtedy i tylko wtedy, gdy graf  $G$  będzie miał ścieżkę Hamiltona  $v_1 \rightsquigarrow v_2$ .

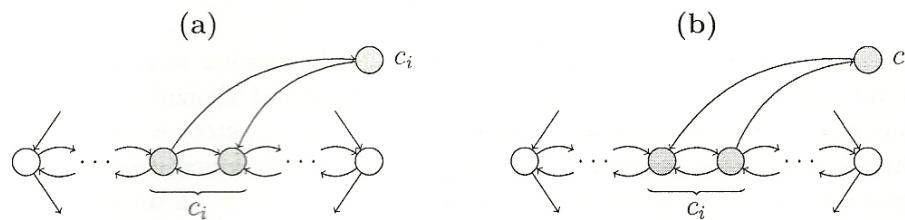
Niech  $\phi$  będzie formułą logiczną w postaci 3CNF zawierającą  $m$  zmiennych oraz  $k$  klauzul. Odpowiadający jej graf skierowany  $G_\phi$  konstruujemy według następującego schematu.

- Każdej klauzuli  $c_i$  formuły  $\phi$  odpowiada pojedynczy wierzchołek  $G_\phi$  z etykietą  $c_i$ .
- Każdej zmiennej  $x_i$  formuły  $\phi$  odpowiada podgraf postaci:



Wyróżnione kolorem wierzchołki *in* oraz *out* służą do łączenia ze sobą podgrafów zmiennych. Przekątna podgrafa odpowiada klauzulom formuły  $\phi$ . Każdej klauzuli odpowiada dokładnie jedna para wierzchołków. Pary odpowiadające poszczególnym klauzulom rozdzielone są dodatkowo pojedynczym wierzchołkiem.

- Wierzchołki odpowiadające klauzulom łączymy z podgrafami odpowiadającym zmiennym następująco:





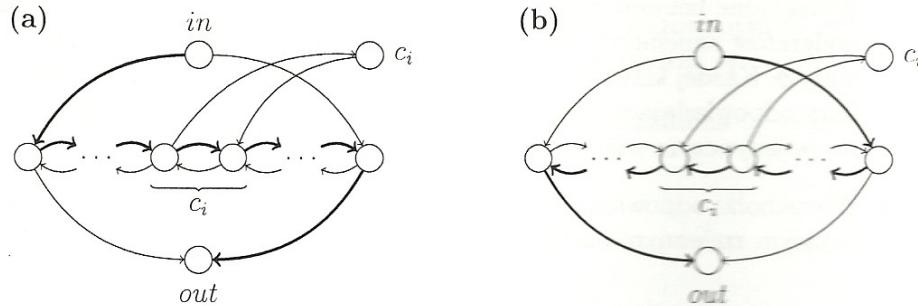
Jeśli klauzula  $c_i$  zawiera literał  $x_j$ , łączymy wierzchołek odpowiadający  $c_i$  z podgrafem odpowiadającym  $x_j$  (dokładnie z wyróżnioną parą wierzchołków na jego przekątnej) w sposób pokazany na rysunku (a). Jeśli natomiast klauzula  $c_i$  zawiera literał  $\neg x_j$ , łączymy wierzchołek odpowiadający  $c_i$  z podgrafem odpowiadającym  $x_j$  w sposób pokazany na rysunku (b).

- Podgrupy odpowiadające kolejnym zmiennym  $x_1, x_2, \dots, x_m$  łączymy w ten sposób, że wierzchołek *out* podgrafa zmiennej  $x_i$  staje się wierzchołkiem *in* podgrafa zmiennej  $x_{i+1}$ .
- Wierzchołkiem początkowym ( $v_1$ ) szukanej ścieżki Hamiltona jest wierzchołek *in* podgrafa odpowiadającego zmiennej  $x_1$ , natomiast wierzchołkiem końcowym ( $v_2$ ) jest wierzchołek *out* podgrafa odpowiadającego zmiennej  $x_m$ .

Musimy teraz udowodnić, że formuła  $\phi$  jest spełnialna wtedy i tylko wtedy, gdy graf  $G_\phi$  zawiera ścieżkę Hamiltona  $v_1 \rightsquigarrow v_2$ .

$\implies$ :

Załóżmy, że formuła  $\phi$  jest spełnialna. Istnieje zatem wartościowanie zmiennych  $x_1, x_2, \dots, x_m$ , dla którego jest ona prawdziwa. Konstruujemy ścieżkę  $v_1 \rightsquigarrow v_2$ . Jeśli w rozważanym wartościowaniu formuły zmienna  $x_i$  ma wartość *true* odpowiadający jej podgraf przechodzimy w kierunku zaprezentowanym na rysunku (a). Jeśli natomiast zmienna  $x_i$  ma wartość *false* odpowiadający jej podgraf przechodzimy w kierunku zaprezentowanym na rysunku (b).



Zauważmy, że skonstruowana w ten sposób ścieżka **zawiera wszystkie wierzchołki** grafu  $G_\phi$  poza wierzchołkami odpowiadającymi klauzulom. Z każdej klauzuli  $c_i$  wybieramy dokładnie jeden literał  $x_j$  lub  $\neg x_j$ , który w rozważanym wartościowaniu ma wartość *true* i dołączamy go do fragmentu konstruowanej ścieżki przechodzącego przez podgraf odpowiadający  $x_j$ . Z konstrukcji  $G_\phi$



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPOŁECZNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

wynika, że krawędzie łączące wierzchołek odpowiadający  $c_i$  z wierzchołkami podgrafa odpowiadającego  $x_j$  mają odpowiedni zwrot. Skonstruowana ścieżka  $v_1 \rightsquigarrow v_2$  jest zatem ścieżką Hamiltona.

⇒:

Założymy, że w grafie  $G_\phi$  istnieje ścieżka Hamiltona  $v_1 \rightsquigarrow v_2$  przechodząca kolejno przez podgrafy wszystkich zmiennych i odstępująca od tej reguły wyłącznie po to, żeby odwiedzić wierzchołki odpowiadające klauzulom  $c_j$ . Wówczas łatwo jest znaleźć wartościowanie spełniające formułę  $\phi$ . Zależnie od kierunku przechodzenia podgrafa odpowiadającego zmiennej  $x_i$  nadajemy jej wartość *true* lub *false*.

Przypuśćmy teraz, że w grafie  $G_\phi$  istnieje ścieżka Hamiltona  $v_1 \rightsquigarrow v_2$  nie spełniająca warunków opisanych powyżej. Musi zatem zawierać fragment  $v_{x_a} \rightarrow v_{c_i} \rightarrow v_{x_b}$ , gdzie  $v_{x_a}$  jest wierzchołkiem należącym do wnętrza podgrafa odpowiadającego zmiennej  $x_a$ ,  $v_{x_b}$  jest wierzchołkiem należącym do wierzchołka podgrafa odpowiadającego zmiennej  $x_b$ , zaś  $v_{c_i}$  jest wierzchołkiem odpowiadającym klauzuli  $c_i$ . Z konstrukcji grafu  $G_\phi$  wynika jednak, że do rozważanej ścieżki nie będzie mógł należeć jeden z wierzchołków sąsiadujących z  $v_{x_a}$ . Zatem ścieżka  $v_1 \rightsquigarrow v_2$  nie będzie ścieżką Hamiltona.

Pozostało oszacowanie złożoności czasowej opisanej redukcji. Dla formuły  $\phi$  składającej się z  $m$  zmiennych oraz  $k$  klauzul redukcja polega na stworzeniu grafu  $G_\phi$  rozmiaru  $O(k \cdot m)$ . Może więc być wykonana w czasie  $O(n^2)$ , gdzie  $n$  jest długością formuły  $\phi$ . □



KAPITAŁ LUDZKI  
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA  
EUROPEJSKI  
FUNDUSZ SPOŁECZNY



Projekt pn. „Wzmocnienie potencjału dydaktycznego UMK w Toruniu w dziedzinach matematyczno-przyrodniczych”  
realizowany w ramach Poddziałania 4.1.1 Programu Operacyjnego Kapitał Ludzki

## Spis treści

Wykład 1. <i>Maszyny licznikowe</i>	3
Wykład 2. <i>Struktura zbioru funkcji ML-obliczalnych</i>	10
Wykład 3. <i>Funkcje częściowo rekurencyjne</i>	17
Wykład 4. <i>Maszyny Turinga</i>	24
Wykład 5. <i>Obliczalność w sensie Turinga</i>	32
Wykład 6. <i>Sieci liczące</i>	41
Wykład 7. <i>Funkcje, programy i maszyny uniwersalne</i>	47
Wykład 8. <i>Rozstrzygalność</i>	52
Wykład 9. <i>Częściowa rozstrzygalność</i>	60
Wykład 10. <i>Przykłady problemów nierozstrzygalnych</i>	67
Wykład 11. <i>Czasowa złożoność obliczeniowa</i>	74
Wykład 12. <i>Pamięciowa złożoność obliczeniowa</i>	84
Wykład 13. <i>Trudność i zupełność problemów obliczeniowych</i>	92
Wykład 14. <i>Problemy NP-zupełne</i>	100
Wykład 15. <i>Przykłady problemów NP-zupełnych</i>	107