1

Maszyny licznikowe

Teoria obliczeń składa się z dwóch głównych części: *teorii obliczalności* poświęconej rozwiązywalności problemów obliczeniowych za pomocą komputera, oraz *teorii 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.

Teza Churcha-Turinga

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

Powyższa teza nie jest twierdzeniem w ściśle 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 nieograniczonym (ale skończonym) czasem oraz nieograniczonymi (ale skończonymi) zasobami pamięci. Istnieją również problemy, których nie da się obliczyć przy pomocy wspomnianych urządzeń maszynowych. Są to jednak problemy, które nie mieszczą się nawet teoretycznie w granicach możliwości obliczeniowych komputera.

Podstawowym obiektem badań teorii obliczeń są funkcje obliczalne. Stanowią one odpowiednik intuicyjnego pojęcia algorytmu.

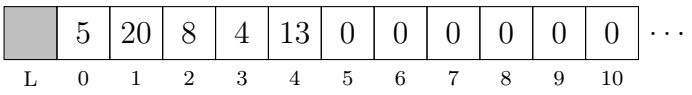
W teorii obliczalności zazwyczaj rozważamy funkcje *częściowe* $f : \mathbb{N}^k \to \mathbb{N}$, które nie muszą być określone dla wszystkich argumentów. Funkcję określoną

dla wszystkich argumentów nazywamy *totalną*.

Precyzyjne 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.

Maszyna licznikowa

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 rejestry, poza skończoną liczbą używanych przez program, zawierają wartość 0. Zawartość rejestru o numerze n oznaczamy przez $R[n]$.



Rysunek 1.1: Schemat pamięci maszyny licznikowej

Maszyna licznikowa może wykonywać następujące instrukcje:

Typ	Instrukcja	Semantyka	Opis
0	`Z(n)`	`R[n]` ← `0`	zerowanie zawartości rejestru
1	`S(n)`	`R[n]` ← `R[n]+1`	następnik wartości rejestru
2	`T(m,n)`	`R[n]` ← `R[m]`	kopiowanie wartości rejestru
3	`I(m,n,q)`	`if R[m] = R[n] then GOTO q`	skok warunkowy

Tabela 1.1: 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 1.1

Programem na maszynę licznikową (ML-programem) nazywamy dowolny *niepusty* 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, wartością 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 $1,\ldots,k$, natomiast wartość zwracana jest do rejestru 0, nazywanego rejestrem wynikowym.

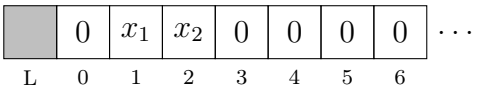
Uwaga

Wszystkie instrukcje oraz rejestry maszyny licznikowej muszą być adresowane bezpośrednio.

Przykład 1.1

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

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



Działanie programu P będzie polegać na zwiększeniu o 1 (inkrementowaniu) wartości rejestru nr 1 x_2 razy. W celu kontroli liczby operacji zwiększania wykorzystany zostanie rejestr nr 3. Porównanie zawartości rejestrów nr 2 oraz nr 3 będzie stanowiło warunek zakończenia głównej pętli programu P.

Program P:

```
0: I(2,3,4)  // warunek końca pętli
1: S(1)      // zwiększenie wartości pierwszego argumentu
2: S(3)      // zwiększenie wartości rejestru kontrolnego
3: I(0,0,0)  // skok na początek pętli
4: T(1,0)    // skopiowanie sumy do rejestru wynikowego
```

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 1.2

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

Definicja 1.3

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

Każdy program P na maszynę licznikową oblicza pewną k-argumentową $(k > 0)$ funkcję częściową $\phi_P^{(k)} : \mathbb{N}^k \to \mathbb{N}$. 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
- $\overline{x} = (x_1, \ldots, x_k)$
- $P(\overline{x}) \downarrow$ – program P zatrzymuje się na danych $\overline{x}$
- $P(\overline{x}) \downarrow y$ – program P zatrzymuje się na danych $\overline{x}$ zwracając wartość y
- $P(\overline{x}) \uparrow$ – program P nie zatrzymuje się na danych $\overline{x}$

Ćwiczenie 1.1

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. W tym celu zdefiniujemy bijektywne kodowanie ML-programów, co pozwoli udowodnić równoliczność zbioru programów na maszyny licznikowe ze zbiorem liczb naturalnych.

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.

Definicja 1.4

Kodowaniem zbioru X nazywamy dowolną totalną funkcję $f : X \longrightarrow I\!N$, która jest różnowartościowa (injekcja).

Definicja 1.5

Numeracją (wyliczeniem) zbioru X nazywamy dowolną funkcję $f : I\!N \longrightarrow X$, która jest „na” (surjekcja).

W przypadku zbiorów nieskończonych (przeliczalnych) będziemy zainteresowani bijektywnymi (odwracalnymi) kodowaniami i numeracjami. Poniżej prezentujemy kilka przykładów bijektywnych kodowań par i ciągów liczb naturalnych.

1. Bijektywne kodowanie par liczb naturalnych: $\pi : I\!N \times I\!N \to I\!N$

$$\pi(x, y) \;=\; 2^x \cdot (2y + 1) - 1.$$

Funkcja odwrotna (numeracja) $\pi^{-1} : I\!N \to I\!N \times I\!N$ określona jest wzorem:

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

gdzie

$$\begin{cases} \pi_1(z) &= \max\Big\{k \in I\!N :\ 2^k | (z+1)\Big\} \\ \pi_2(z) &= \dfrac{1}{2}\left(\dfrac{z+1}{2^{\pi_1(z)}} - 1\right) \end{cases}.$$

2. Kodowanie Eulera $\mathcal{E} : I\!N \times I\!N \to I\!N$

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

3. Bijektywne kodowanie trójek liczb naturalnych $\beta : I\!N^3 \to I\!N$

$$\beta(x,y,z) \ = \ \pi\Big(\pi(x,y), z\Big).$$

4. Bijektywne kodowanie ciągów liczb naturalnych: $\tau : \bigcup_{k>0} I\!N^k \to I\!N$

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

Uwaga

Kodowanie τ jest określone dla ciągów skończonej długości.

Ćwiczenie 1.2

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

Ćwiczenie 1.3

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

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 1.1.

Rozważmy ML-program $P = \{I_0, I_1, \ldots, I_k\}$ składający się z $k+1$ instrukcji. Każdą instrukcję P zakodujemy za pomocą wzoru

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

W przypadku instrukcji $Z(n)$ oraz $S(n)$ kodem argumentów jest adres rejestru, w przypadku instrukcji $T(m,n)$ używamy bijektywnego kodowania par π, natomiast w przypadku instrukcji $I(m,n,q)$ używamy bijektywnego kodowania trójek β. Ponieważ typy instrukcji 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 τ:

$$K_P(P) \;=\; \tau\Big(K_I(I_0), K_I(I_1), \ldots, K_I(I_k)\Big).$$

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.

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

Przykład 1.2

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

Rozważany program składa się z dwóch instrukcji:

I_0: `S(1)`
I_1: `T(1,0)`

Kodami kolejnych instrukcji programu P są:

$$\begin{aligned} K_I(I_0) &= 4 \cdot 1 + 1 &&= 5 \\ K_I(I_1) &= 4 \cdot \pi(1,0) + 2 &&= 4 \cdot 1 + 2 \;=\; 6 \end{aligned}$$

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 obliczana przez nieskończenie wiele programów – tym samym może mieć nieskończenie wiele numerów.

Przykład negatywny

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

Twierdzenie 1.1

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

Dowód

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

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

i przypuśćmy, 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 $e \in D_e$ oraz z określenia $f(e) = \phi_e(e) + 1$. Otrzymana sprzeczność dowodzi, że funkcja f nie może być obliczalna. ■