6

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 rozwiązania jednego wyspecjalizowanego problemu. W czasie tego wykładu poznamy funkcje, programy oraz maszyny uniwersalne, które mogą symulować obliczenia dowolnej innej funkcji, programu lub maszyny (równiez 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 funcji obliczalnej jako parametry również jest obliczalna. Ponadto, istnieje algorytm pozwalający znaleźć numer jednego z obliczających ją programów.

Twierdzenie 6.1 (O parametryzacji)

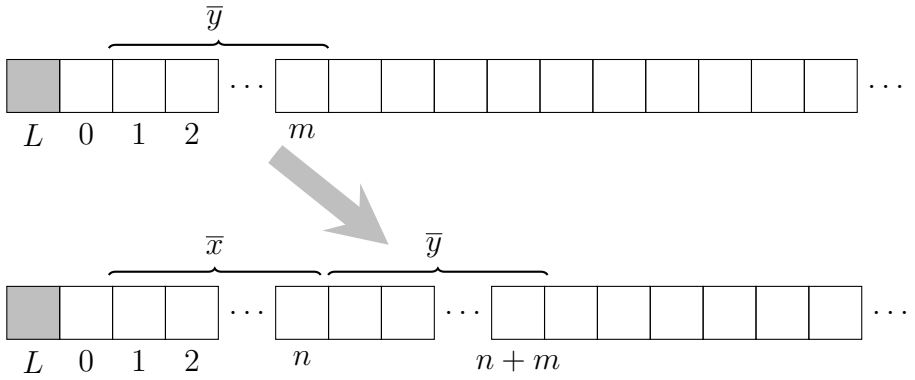
Dla dowolnych liczb naturalnych $m, n \geqslant 1$ istnieje totalna i obliczalna funkcja $s_n^m : \mathbb{N}^{n+1} \to \mathbb{N}$ taka, że dla dowolnych $e \in \mathbb{N}$, $\overline{x} \in \mathbb{N}^n$ oraz $\overline{y} \in \mathbb{N}^m$ zachodzi:

$$\phi_e^{(m+n)}(\overline{x}, \overline{y}) \;=\; \phi^{(m)}_{s_n^m(e,\overline{x})}(\overline{y}).$$

Dowód

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

argumentami.



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

Kod programu P_s obliczającego funkcję $\phi^{(m)}_{s^m_n(e,\overline{x})}$ dla argumentu $\overline{y}$ jest przedstawiony poniżej:

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

Zauważmy, że wartość funkcji $s^m_n(e, \overline{x})$ możemy obliczyć na podstawie numeru programu P_e przez zakodowanie instrukcji kopiujących $\overline{y}$ oraz ustalających wartość $\overline{x}$. ■

Funkcje uniwersalne

Definicja 6.1

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

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

Twierdzenie 6.2 (O funkcjach uniwersalnych)

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

Ćwiczenie 6.1

Uzasadnij, że funkcja uniwersalna jest częściowo rekurencyjna.

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

Programy uniwersalne

Definicja 6.2

Programem uniwersalnym nazywamy dowolny program P_U obliczający funkcję uniwersalną Ψ_U.

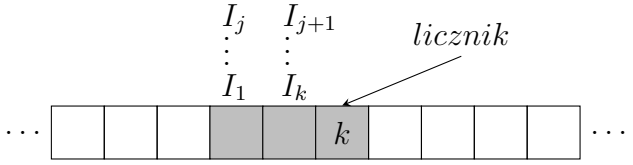
Dla programu uniwersalnego liczącego n-argumentową funkcję uniwersalną, możemy umieścić jego argumenty w wybranych n rejestrach. Dla programu uniwersalnego liczącego funkcje uniwersalne o dowolnej liczbie argumentów, umieszczamy w wybranym rejestrze liczbę całkowitą będącą kodem listy argumentów $\tau(x_1,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

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 τ oraz τ^{-1} dla listy kodów instrukcji $I_1, \ldots I_k$ (argumentów programu) możemy wykorzystać następujące podprogramy:

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

Szybki dostęp do kodów instrukcji możemy uzyskać wykorzystując 3 rejestry pamięci do przechowywania 2 stosów (instrukcji) oraz licznika. Stosy przechowują listy kodów instrukcji w postaci: $\tau(I_j, \ldots, I_1)$ oraz $\tau(I_{j+1}, \ldots, I_k)$. 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 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 δ.
- 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?

Uprośćmy 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:

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

Załóżmy, że potrafimy skonstruować maszynę Turinga M_R rozstrzygającą powyższy problem:

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

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

$$M_O(M) = \begin{cases} \textbf{akceptuj} & \text{jeśli } \mathrm{M_R} \text{ odrzuca } \langle \mathrm{M} \rangle \\ \textbf{odrzucaj} & \text{jeśli } \mathrm{M_R} \text{ akceptuje } \langle \mathrm{M} \rangle \end{cases}$$

$$= \begin{cases} \textbf{akceptuj} & \text{jeśli M nie akceptuje } \langle \mathrm{M} \rangle \\ \textbf{odrzucaj} & \text{jeśli M akceptuje } \langle \mathrm{M} \rangle \end{cases}$$

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

$$M_O\Big(M_O\Big) = \begin{cases} \textbf{akceptuj} & \text{jeśli } \mathrm{M_O} \text{ nie akceptuje } \langle \mathrm{M_O} \rangle \\ \textbf{odrzucaj} & \text{jeśli } \mathrm{M_O} \text{ akceptuje } \langle \mathrm{M_O} \rangle \end{cases}$$

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

Twierdzenie 6.3

Problem akceptowania danych wejściowych przez maszynę Turinga jest nierozstrzygalny.

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).