Spis treści

Zdzisław Pawlak

ORGANIZACJA MASZYN MATEMATYCZNYCH - CZĘŚĆ I

Pojęcie maszyny matematycznej

mentarz do wykładu dla studentów matematyki

ROZDZIAŁ I

Pojęcie maszyny

Dla określenia maszyny cyfrowej musimy najpierw określić podstawowe pojęcia niezbędne do definicji maszyny. Są to: pamięć, meta-instrukcje i sterowanie. Te trzy pojęcia wystarczająco charakteryzują z naszego punktu widzenia każdą maszynę. Definiując więc jakąkolwiek maszynę musimy opisać jej pamięć, zbiór meta-instrukcji oraz sterowanie. Podany w ten sposób opis maszyny będzie zawierał wszystkie interesujące nas informacje o maszynie.

§ 1. Pamięć

1. Definicja.

Ogólnie biorąc pamięć maszyny służy do magazynowania w niej napisów z określonego zbioru wyrażeń(słów pewnego języka). Nie będziemy interesować się szczegółami technicznymi pamięci, a podamy ogólną charakterystykę jej działania, tak abyśmy potrafili badać i definiować różnego rodzaju pamięci stosowane w maszynach jedynie z ogólnego punktu widzenia sposobu magazynowania w nich informacji.

Dobrym modelem pamięci maszyny jest sieć magazynów, w których musimy przechowywać określone przedmioty oraz każdy przedmiot możemy do magazynu włożyć oraz z magazynu

pobrać. Ponadto przedmioty magazynowane mogą zmieniać swoje położenie w magazynie. Magazyn jest oczywiście scharakteryzowany ilością miejsc, w których można magazynować (pojemnością), rodzajem przedmiotów, które można w nim magazynować, a przede wszystkim sposobem wkładania i pobierania z niego magazynowanych przedmiotów - i także oczywiście sposobem przemieszczania przedmiotów w magazynie. Podanie tych charakterystyk jest wystarczającym opisem ogólnej struktury każdego magazynu. Również pamięci maszyn cyfrowych będziemy opisywać w podobny sposób; podając jej wielkość, rodzaj magazynowanych wyrażeń, sposób wprowadzania i wyprowadzania wyrażeń do pamięci oraz sposób, w jaki będziemy zmieniać stan pamięci (tą ostatnią sprawą zajmiemy się dopiero w następnym paragrafie).

Każda pamięć składa się ze skończonej liczby <u>rejestrów</u> $r_1, r_2, \ldots, r_n$. (Czasem zamiast nazwy rejestr używane są terminy <u>miejsce</u> lub <u>komórka</u>. Będziemy tych terminów używać również na równi ze słowem rejestr). Dla ułatwień formalnych wygodnie jest rozpatrywać pamięci o nieskończonej liczbie miejsc. Tak też często będziemy postępować w naszych rozważaniach.

Liczbę miejsc pamięci będziemy nazywali jej pojemnością.

Miejsca (rejestry) służą do przechowywania pojedyńczych symboli ustalonego, skończonego <u>alfabetu</u> $\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_n\}$ $n > 1$. σ_0 - będziemy zawsze nazywali symbolem pustym i oznaczali przez Λ. W każdym miejscu pamięci może być więc zanotowany jeden symbol (również pusty) z alfabetu Σ.

(Czasem rozważa się takie pamięci, że w każdym jej miejscu można magazynować nie pojedynczy symbol a ciąg symboli $\sigma_{i_1}, \sigma_{i_2} \ldots \sigma_{i_k}$, $\sigma_{i_j} \in \Sigma$, o ustalanej długości. Nie ma to oczywiście znaczenia, gdyż zawsze możemy łatwo przejść od symboli pojedyńczych do skończonych ciągów).

Przyjmiemy ponadto, że rejestry pamięci są jakoś ponazywane poprzez przyporządkowanie im symboli z ustalonego zbioru $A = \{a_1\ a_2 \ldots\}$ zwanego zbiorem <u>adresów</u> pamięci. Dla uproszczenia zamiast mówić o przyporządkowaniu rejestrom symboli alfabetu Σ - oraz adresów, wygodnie jest mówić o przyporządkowaniu adresom symboli alfabetu Σ i pomijać rejstry. Jeżeli więc będziemy mówili, że adresowi $a \in A$ przyporządkowany jest symbol $\sigma \in \Sigma$, to rozumiemy, że w rejestrze r_i o adresie a znajduje się symbol σ.

Funkcję $C: A \to \Sigma$ [1] będziemy nazywali <u>zawartością</u> pamięci. Przez C będziemy rozumieli zbiór wszystkich dopuszczalnych zawartości danej pamięci, tj. $C \subseteq \Sigma^A$ (ogólnie biorąc nie zawsze $C = \Sigma^A$. Mogą istnieć jakieś ograniczenia natury tachnicznej czy też innej, tak że nie każda funkcja $C \in \Sigma^A$ może być zawartością pamięci.

Z każdą pamięcią skojarzymy skończony zbiór $W = \{w_0, w_1, \ldots, w_p\}$ elementów zwanych <u>wskaźnikami</u> pamięci.

---

1) W dalszym ciągu f będzie oznaczało zawsze funkcję, a' f(x) jej wartość.

Wskaźniki będą nam służyły do oznaczania wybranych miejsc pamięci. Niech $l: W \to A$. Funkcję l będziemy nazywali <u>znacznikiem</u>

Przez L będziemy oznaczali zbiór wszystkich dopuszczalnych znaczników w danej pamięci, tj. $L \subseteq A^W$ (Oczywiście , podobnie jak zawartości, nie każda funkcja $l \in A^W$ jest dopuszczalna w pamięci). Zbiór $U = \{l(w_o), .., l(w_p)\}$ będziemy nazywali zbiorem miejsc <u>wyróżnionych</u> lub <u>obserwowanych</u> pamięci. W szczególnym przypadku wszystkie wskaźniki mogą być ustawiane na jednym miejscu, tj. $l(w_o) = l(w_1) = \ldots = l(w_p)$.
Przyjmiemy jednak, że $U \neq \phi$ t.zn. że w każdej pamięci jest zawsze obserwowane co najmniej jedno miejsce.

<u>Stanem</u> pamięci m będziemy nazywali parę uporządkowaną $m = \langle c, l \rangle$. Stan pamięci jest więc parą funkcji, z których pierwsza mówi, jaka jest zawartość pamięci - a druga określa zbiór miejsc obserwowanych pamięci .

Zbiór wszystkich dopuszczalnych stanów pamięci oznaczamy przez M. Oczywiście $M \subseteq C \times L$.

Definiując pamięć powinniśmy więc określać zbiory C i L tak, abyśmy widzieli dokładnie, jakie stany pamięci są dopuszczalne. W rozpatrywanych jednakże przykładach wszystkie możliwe stany będą dopuszczalne tak że C i L nie będziemy podawać.

Tak więc każdą pamięć określamy za pomocą następujących pojęć podstawowych :

A - skończony lub nieskończony zbiór adresów

Σ - skończony alfabet pamięci

W - skończony zbiór wskaźników

$C \subseteq \Sigma^A$ - zbiór zawartości

$L \subseteq A^W$ - zbiór znaczników.

Ponadto będziemy używali pojęć :

$M = C \times L$ - stan pamięci

$U = \{l(w_o), \ldots, l(w_p)\}$ - zbiór miejsc obserwowanych

Pojemność pamięci równa jest oczywiście $|A|$ [1].

2. <u>Funkcje wejścia i wyjścia pamięci</u> .

Dla określenia sposobu wprowadzania i wyprowadzania do i z pamięci wprowadzimy dwie funkcje I oraz O zwane odpowiednio funkcjami <u>wejścia</u> i <u>wyjścia</u> pamięci :

$$I : \Sigma \times A \times M \to M$$

$$O : A \times M \to M \times \Sigma$$

Funkcje I oraz O można rozszerzyć na skończone ciągi symboli w alfabecie Σ . Wtedy :

$$I_r : \Sigma^p \times A^p \times M \to M$$

$$O_r : A^p \times M \to M \times \Sigma^p \qquad (n \geq 1),$$

---

1) $|A|$ - oznacza moc zbioru A

2) Funkcje I oraz O nie muszą być określone dla wszystkich wartości argumentów.

gdzie $\Sigma^p = \underbrace{\Sigma \times \ldots \times \Sigma}_{p}$ i podobnie

$A^n = \underbrace{A \times \ldots \times A}_{p}$.

Funkcje rozszerzalne określimy następująco :

1°. $I_r(\sigma, a, m) = I(\sigma, a, m)$

2°. $I_r(\sigma_1, \ldots, \sigma_p, a_1, \ldots, a_p, m) =$
$= I(\sigma_p, a_p, I_r(\sigma_1, \ldots, \sigma_{p-1}, a_1, \ldots, a_{p-1}, m))$.

1°. $\theta_r(a, m) = \theta(a, m)$

2°. $\theta_r(a_1, \ldots, a_n, m) =$
$= \theta\{a_n, \alpha_1[\theta_r(a_1, \ldots, a_{n-1}, m)]\}$,

gdzie $\alpha_1 \langle a, b \rangle = a$.

Czasem będziemy rozpatrywali pamięci z uproszczonymi funkcjami wejścia lub wyjścia, nie zawierającymi jako argumentów adresów

$I : \Sigma \times M \to M$

$\theta : M \to M \times \Sigma$

oraz

$I_r : \Sigma^* \times M \to M$

$\theta_r : M \times N \to M \times \Sigma^*$,

gdzie N - zbiór liczb naturalnych, a Σ^* zbiór wszystkich słów w alfabecie Σ .

Funkcje takie będziemy nazywali bezadresowymi. Bezadresowe funkcje rozszerzane możemy określić jak niżej :

1°. $I_r(\sigma, m) = I(\sigma, m)$,

2°. $I_r(\sigma_1, \ldots, \sigma_p, m) = I(\sigma_p I_r(\sigma_1, \ldots, \sigma_{p-1}, m))$.

1°. $\theta_r(m, 1) = \theta(m)$

2°. $\theta_r(m, n) = \theta(\alpha_1(\theta_r(m, n-1)))$.

3. Przykłady pamięci.

Przykład 1. Pamięć stała.

$A = \{0, 1, \ldots\}$

$\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_k\}$

$W = \{w\}$.

Funkcja wejściowa nie zmienia stanu pamięci, t.zn.

$I(\sigma, a, m) = m$.

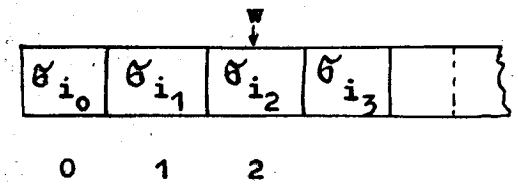
Funkcja wyjściowa dla pamięci stałej ma postać :

$O(a, m) = \langle m', \sigma \rangle$,

gdzie $m' = \langle c', l' \rangle$, $m = \langle c, l \rangle$ oraz $c' = c$,

$l'(w) = a$ natomiast $\sigma = c(l(w)) = c(a)$.

A więc odczytywanie pamięci stałej nie powoduje zmiany jej zawartości, zmienia tylko stan pamięci przez ustawienie jedynego wskaźnika w na miejscu o adresie a oraz powoduje odczytanie zawartego pod tym adresem symbolu. Zwróćmy uwagę, że miejsce które chcemy odczytać, wyznaczamy każdorazowo przez podanie jako argumentu funkcji O adresu interesującego nas miejsca. Pamięć taką możemy przedstawić jak na rysunku:



Na taśmie podzielonej na kratki mamy zapisane symbole alfabetu Σ (w każdej kratce jeden symbol) . Za pomocą funkcji wyjścia możemy ustawiać w różnych kratkach wskaźnik w i odczytać występujący pod nim symbol. Zawartości pamięci nie możemy zmieniać.

Przykład 2. Pamięć jednoadresowa.

Zbiory A, Σ, W jak w przykładzie 1. Funkcja wyjścia również taka sama jak w przykładzie 1. Natomiast funkcja wejścia dla tej pamięci jest inna.

$$I(\sigma, a, m) = m'$$

$$m' = \langle c', l' \rangle$$

$$c'(x) = \begin{cases} \sigma & \text{, jeżeli } x = l'(w) \\ c(x) & \text{, jeżeli } x \neq l'(w) \end{cases}$$

$$l'(w) = a$$

a więc w tej pamięci symbol σ jest wpisywany na miejsce a.

Tę pamięć możemy również przedstawić w postaci taśmy, jak w poprzednim przykładzie, jednakże obecnie w kratkę wskazaną przez wskaźnik w wpisujemy dowolny symbol alfabetu Σ .

Przykład 3. Pamięć z przesuwaniem zawartości.

$A = \{0,1,\ldots,n\}$

$\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_k\}$

$W = \{p,k\}$

Wejście i wyjście tej pamięci jest bezadresowe.

$I(\sigma, m) = m'$, gdzie $m = \langle c,l \rangle$,

$m' = \langle c', l' \rangle$, natomiast

$$c'(x) = \begin{cases} \sigma & \text{, jeżeli } x = l(p) \\ c(x-1) & \text{, jeżeli } x \neq l(p) \end{cases}$$

$$l'(y) = \begin{cases} 0, & \text{jeżeli } y = p \\ n, & \text{jeżeli } y = k \end{cases}$$

lub jeszcze krócej

$$c'(x) = \begin{cases} \sigma & \text{, jeżeli } x = 0 \\ c(x-1) & \text{, jeżeli } x \neq 0 \end{cases}$$

Funkcja wyjścia dla tej pamięci będzie określona w ten sposób :

$$\vartheta(m) = \langle m', \sigma \rangle = \langle \langle c', l' \rangle, \sigma \rangle ,$$

gdzie

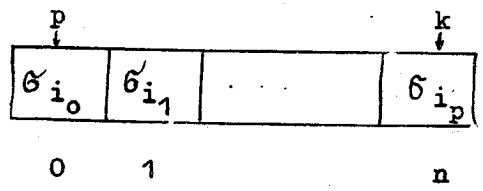
$$c'(x) = \begin{cases} \Lambda & \text{, jeżeli } x = l(p) \\ c(x-1) & \text{, jeżeli } x \neq l(p) \end{cases}$$

$$l'(y) = \begin{cases} 0, & \text{jeżeli } y = p \\ n, & \text{jeżeli } y = k \end{cases}$$

$$\sigma = c(l(k)) .$$

Pamięć ta ma na stałe ustawiony jeden wskaźnik p (początek) na miejscu o adresie zero, drugi zaś wskaźnik k (koniec) na stałe jest ustawiony na miejscu o adresie ostatnim n. Wprowadzany symbol zapisujemy na miejscu zerowym, a odczytujemy zawsze symbol z ostatniego miejsca pamięci o adresie n. Zarówno przy zapisie jak odczycie

przesuwamy całą zawartość o jedno miesjce. Na rysunku pamięć taką przedstawilibyśmy tak :



Przykład 4. Pamięć cykliczna.

A, Σ ,W,0 jak w przykładzie 3. Natomiast inna jest funkcja wejścia.

$I(\sigma, m) = m'$, gdzie $m' = \langle c', l' \rangle$

$$c'(x) = \begin{cases} c(n), & \text{jeżeli } x = l(p) \\ \sigma, & \text{jeżeli } x = l(p) + 1 \\ c(x-1), & \text{jeżeli } 2 \leq x \leq n \end{cases}$$

$$l'(y) = \begin{cases} 0, & \text{jeżeli } y = p \\ n, & \text{jeżeli } y = k \end{cases}$$

Pisząc krócej:

$$c'(x) = \begin{cases} c(n), & \text{jeżeli } x = 0 \\ \sigma, & \text{jeżeli } x = 1 \\ c(x), & \text{jeżeli } 2 \leq x \leq n. \end{cases}$$

Pamięć ta działa więc w ten sposób, że na miejsce 0 wprowadzamy symbol σ , następnie zawartość wszystkich miejsc przesuwamy o jedno miejsce w prawo, przy czym zawartość ostatniego miejsca w pamięci jest wpisywana na miejsce zerowe.

Przykład 5. Pamięć liniowa (w języku angielskim zwana Push - up lub first in - last out).

$A = \{0, 1, \ldots\}$

$\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_k\}$

$W = \{z, 0\}$

$I(\sigma, m) = m' = \langle c', l' \rangle$

$$c'(x) = \begin{cases} \sigma, & \text{jeżeli } x = l(z) \\ c(x), & \text{jeżeli } x \neq l(z) \end{cases}$$

$$l'(y) = \begin{cases} l(y) + 1, & \text{jeżeli } y = z \\ l(y), & \text{jeżeli } y = 0 \end{cases}$$

$\sigma'(m) = \langle m', \sigma \rangle$, gdzie

$m' = \langle c', l' \rangle$ oraz $c'_+ = c$, natomiast

$$l'(x) = \begin{cases} l(x), & \text{jeżeli } x = z \\ l(x) + 1, & \text{jeżeli } x = 0 \end{cases}$$

zaś $\sigma = c(l(0))$.

Pamięć liniowa działa więc następująco: zapisywany symbol umieszczamy w pamięci w miejsce wskazane przez wskaźnik z (zapis) i następnie wskaźnik z przesuwamy na następne miejsce, zaś wskaźnika 0 nie ruszamy.

Przy odczycie podobnie; najpierw odczytujemy symbol z miejsca wskazanego przez z i z przesuwamy na następne miejsce, zaś wskaźnika 0 nie ruszamy.

Przykład 6. Pamięć rewersyjna - zwana też stosem lub stogiem. (W literaturze angielskiej - Push - down, lub first in - first out).

Zbiory A i Σ jak w przykładzie 5. $W = \{w\}$.

$I(\sigma, m) = m' = \langle c', l' \rangle$

$$c'(x) = \begin{cases} \sigma, & \text{jeżeli } x = l(w) \\ c(x), & \text{jeżeli } x \neq l(w) \end{cases}$$

$$l'(w) = l(w) + 1$$

$$O(m) = \langle m', \sigma \rangle = \langle \langle c', l' \rangle, \sigma \rangle, \text{ gdzie}$$

$$c' = c$$

$$l'(w) = l(w) - 1 \quad \text{oraz}$$

$$\sigma = c(l(w))$$

Jeżeli chcemy zapisać jakiś symbol σ w pamięci, to najpierw wskaźnik w przesuwamy na następne miejsce, a następnie w miejsce wskazane przez wskaźnik w wpisujemy σ. Odczyt z pamięci odbywa się w ten sposób, że odczytujemy symbol wskazany przez wskaźnik w i następnie wskaźnik w cofamy o jedno miejsce.

Przykład 7. Licznik modulo n+1.

$A = \{a\}$

$\Sigma = \{0,1,\ldots,n\}$

$W = \{w\}$

$\Sigma' = \{0,1\}$

$I : \Sigma' \times M \to M$

$O : M \to M \times \Sigma \ , \ \langle c', l' \rangle$

$$I(\sigma, m) = \begin{cases} m', & \text{jeżeli } \sigma = 1 \\ m, & \text{jeżeli } \sigma = 0 \end{cases}$$

gdzie $\sigma \in \Sigma'$ oraz

$$c'(a) = \begin{cases} c(a) + 1, & \text{jeżeli } c(a) < n \\ 0, & \text{jeżeli } c(a) = n \end{cases}$$

natomiast

$$l'(w) = \text{const} = a$$

$$O(m) = \langle m, \sigma \rangle \quad , \text{ gdzie}$$

$$\sigma = c(l(w))^{1)}$$

Przykład 8. Akumulator.

A, Σ, W jak w poprzednim przykładzie.

$$I(\sigma, m) = m' = \langle c', l' \rangle \quad , \text{ gdzie}$$

$$c'(a) = \begin{cases} c(a) + \sigma & , \text{jeżeli } c(a) + \sigma \leq n \\ c(a) + \sigma - (n+1), & \text{jeżeli } c(a) + \sigma > n \end{cases}$$

$$l'(w) = a$$

$$\vartheta(m) = \langle m, \sigma \rangle, \quad \text{gdzie}$$

$$\sigma = c(w).$$

Przykład 9. Przerzutnik.

$$A = \{a\} \ , \quad \Sigma = \{0,1\} \ , \ W = \{w\}$$

$$I(\sigma, m) = \begin{cases} m' = \langle c', l' \rangle & , \text{jeżeli } \sigma = 0 \\ m'' = \langle c'', l'' \rangle & , \text{jeżeli } \sigma = 1 \end{cases}$$

gdzie

$$c'(a) = 0$$

i $l'(w) = a$

$$c''(a) = 1 \ ; \ l''(w) = a.$$

---

1) W dalszym ciągu zamiast c(l(w)) będziemy pisać krótko c(w)

Stosowany tu język dla niektórych rodzajów pamięci jest oczywiście za bogaty. Np. licznik, akumulator czy przerzutnik można opisać znacznie prościej. Jednakże wydaje się, że z uwagi na jednolitość wygodnie jest mieć jeden sposób opisu, który obejmuje wszystkie interesujące nas przypadki pamięci, mimo że w niektórych przypadkach jest on zbyt obszerny. Dobieranie bowiem dla różnych klas pamięci odpowiednich języków jest chyba równie kłopotliwe.

4. Własności pamięci

W ustępie tym podamy kilka elementarnych własności pamięci, które pozwalają na zorientowanie się w niektórych podobieństwach czy różnicach pamięci.

Ważną cechą pamięci jest następująca własność :

W_1: $\Lambda\, \sigma\, \Lambda\, a\, \Lambda\, m \left\{\alpha_2 \left[\vartheta(a, I(\sigma, a, m))\right]\right\} = \sigma$,

gdzie $\alpha_2 \langle a,b \rangle = b$.

Własność ta wyraża następujący fakt: jeżeli do pamięci pod adres a wprowadzimy symbol σ , to odczytując następnie zawartość a, otrzymamy ten sam symbol σ . Oczywiście, nie wszystkie pamięci posiadają tę własność. Nie ma jej np. licznik czy akumulator.

Własność W_1 dla wejść i wyjść bezadresowych przyjmie postać :

W'_1: $\Lambda\sigma\, \Lambda\, m \left[\alpha_2(\vartheta(I(\sigma, m)))\right] = \sigma$.

Ta własność przysługuje pamięci rewersyjnej, lecz nie przysługuje pamięci liniowej ani pamięci z przesuwaniem

zawartości.

Własność W_1 i W'_1 można zmodyfikować wprowadzając rozszerzone funkcje wejścia lub wyjścia.

Otrzymamy wtedy:

W_2: $\Lambda\, X \in \Sigma^p\ \Lambda\, Y \in A^p\ \Lambda\, m \in M$

$\left\{\alpha_2(\vartheta_r[Y, I_r(X,Y,m)]\right\} = X$

W'_2: $\Lambda X \in \Sigma^n\ \Lambda\, m \in M \left\{\alpha_2 \vartheta(I(X,m))\right\} = X.$

Każda pamięć wyznacza więc pewną funkcję P

$$P : \Sigma^* \rightarrow \Sigma^*,$$

której argumentami są słowa wprowadzane do pamięci, a wartościami - słowa wyprowadzane z pamięci, określona następująco :

$P(X) = \alpha_2 \left\{\vartheta_r[Y, I_r(X,Y,m)]\right\}$, gdzie

$X \in \Sigma^n$, $Y \in A^n$, $m \in M$.

Funkcja ta dla wejść i wyjść bezadresowych przyjmie postać :

$$P(X) = \alpha_2 \left[\vartheta_r(I_r, (X,m))\right] .$$

Funkcję P(X) będziemy nazywać funkcją charakterystyczną pamięci.

Jeżeli dla każdego X, P(X) = X, to pamięć nazwiemy prostą.

Jeżeli dla każdego X $P(X) = X^{-1}$, to pamięć nazwiemy odwrotną, (X^{-1} oznacza słowo odwrotne do X, tj. słowo składające się z symboli słowa X napisanych w odwrotnym porządku.

Dwie pamięci nazwiemy równoważnymi, jeżeli ich funkcje charakterystyczne są identyczne.

Pamięci równoważne są to więc takie pamięci, które z punktu widzenia wprowadzania i wyprowadzania słów są nierozróżnialne.

Oczywiście, dla pamięci o skończonej pojemności problem równoważności jest rozstrzygalny. Wystarczy po prostu kolejno wprowadzać do obu pamięci wszystkie możliwe słowa z Σ^* i sprawdzać wyjścia, czy są zawsze jednakowe. Dla pamięci o nieskończonej pojemności problem równoważności jest w ogólnym przypadku nierozstrzygalny.

Jest rzeczą oczywistą, że każda pamięć jest automatem skończonym.

Ćwiczenia.

1. Podać funkcje I_r i O_r dla maszyn w przykładach 1 - 9.
2. Określić funkcje charakterystyczne dla pamięci podanych w przykładach.
3. Zbadać, które z podanych przykładów pamięci są równoważne.
4. Pokazać, że pamięć z wejściem i wyjściem adresowym jest automatem skończonym.

   Uwaga: Przyjąć następującą definicję automatu skończonego :

$$A = \langle Q, \Sigma, \varphi, \psi \rangle$$

Q - zbiór stanów,

Σ - alfabet

$\varphi: Q \times \Sigma \to Q$

$\psi: Q \to \Sigma$.

§ 2. Instrukcje

1. Semantyczne określenie instrukcji.

Każda maszyna matematyczna może wykonywać pewną liczbę elementarnych czynności zwanych operacjami elementarnymi (lub podstawówymi) maszyny. Dla krótkości będziemy je nazywać operacjami. Wykonanie każdej operacji jest związane z realizacją odpowiedniej meta-instrukcji (zwanej też rozkazem). Meta-instrukcje będziemy krótko nazywać instrukcjami. Do terminu „meta-instrukcja" wrócimy dopiero przy omawianiu maszyn uniwersalnych. Tak więc z każdą maszyną związany jest pewien skończony zbiór operacji $F = \{f_1, \ldots, f_s\}$ oraz skończony zbiór instrukcji $R = \{r_1, \ldots, r_k\}$.

Operacje są to funkcje

$$f_i : \Sigma^{n_i} \to \Sigma \quad , \quad n_i \geqslant 0 \quad ^{1)}$$

Będziemy przyjmować, że wśród operacji znajdują się następujące funkcje :

1° $f(x_1, \ldots, x_i, \ldots, x_n) = x_i$,

2° $f'(x_1, \ldots, x_n) = \text{const}$.

$f, f' \in F$.

W dalszym ciągu przyjmiemy, że alfabet rozważanych

---

1) W rzeczywistych maszynach $n_i \leqslant 2$.

maszyn jest zbiorem liczb naturalnych 0,1,2,... . Dla wygody przyjmiemy, że zbiór ten jest nieskończony, choć uprzednio zakładaliśmy, że alfabet jest zawsze zbiorem skończonym. Podobnie jako zbiór adresów będziemy też przyjmowali zbiór liczb naturalnych. Jeżeli od tych założeń będziemy odstępowali, wyraźnie to zaznaczymy.

Przyjmiemy ponadto, że wśród operacji jest dodawanie $x + y$, następnik $x + 1$, oraz poprzednik $x - 1$.

Instrukcje są to funkcje :

$$r : A^k \times M \longrightarrow M, \quad k \geqslant 0$$ [1)]

Instrukcje takie będziemy nazywali k-adresowymi. Instrukcje zero-adresowe będziemy też nazywali instrukcjami bezadresowymi.

Oznaczmy $a_1,\dots,a_k$ przez $\bar{a}_k$. Oczywiście dla każdej instrukcji i dowolnych $\bar{a}_k \in A^k$ oraz $m \in M$ zachodzi

$$r(\bar{a}_k,m) = I\left\{\bar{a}_k, \alpha_1\left[\sigma_r(\bar{a}_k,m)\right], f\left[\alpha_2(\sigma_r(\bar{a}_k,m))\right]\right\},$$

gdzie I oraz σ_r są funkcjami wejścia i wyjścia pamięci, w której instrukcja r jest realizowana. Znaczy to, że najpierw za pomocą funkcji wyjścia σ_r (rozszerzonej) musimy odczytać z pamięci argumenty operacji f i po wykonaniu tej operacji wyniki jej wpisujemy do pamięci za pomocą funkcji I.

Dwie instrukcje r i r' są <u>równoważne</u>, jeżeli dla dowol-

---

1) W budowanych maszynach $k \leq 3$.

nych adresów $\bar{a}_k$ oraz dla dowolnego stanu pamięci m

$$r(\bar{a}_k,m) = r'(\bar{a}_k,m).$$

Oczywiście <u>instrukcje równoważne realizują jednakowe operacje</u>.

Instrukcję r taką, że

$$\bigwedge \bar{a}_k \in A^k \bigwedge m \in M \left[r(\bar{a}_k,m) = m\right]$$

nazywamy <u>instrukcją tożsamościową</u> i oznaczymy ją przez r_o.

Przez <u>złożenie</u> instrukcji r i r' będziemy rozumieli zrealizowanie najpierw instrukcji r a następnie r'. Złożenie oznaczymy przez r'r. Wartością tego złożenia będzie stan pamięci:

$$r'\left[\bar{a}'_k, r(\bar{a}_k,m)\right] .$$

Również n-krotne powtórzenie instrukcji r jest instrukcją, przy czym powtórzenie określamy następująco :

1^o $\quad r(1,\bar{a}_k,m) = r'(\bar{a}_k,m)$,

2^o $\quad r(n+1,\bar{a}_k,m) = r'(\bar{a}_k,m)\, r(n,\bar{a}_k,m) =$

$$= r'\left[\bar{a}_k, r(n,\bar{a}_k,m)\right] ,$$

gdzie n liczba powtórzeń instrukcji r'.

2. <u>Przykłady instrukcji</u>.

<u>Przykład 1</u>. Przesuwanie wskaźnika w lewo i w prawo. Oznaczmy obie te instrukcje odpowiednio przez l i p. W niech będzie wskaźnikiem, który chcemy przesuwać. Wtedy

$$l(m) = m' = \langle c', l' \rangle \text{ , gdzie } c' = c$$

$$l'(x) = \begin{cases} l(x) - 1, & \text{jeżeli } x = w \\ l(x), & \text{jeżeli } x \neq w. \end{cases}$$

$p(m) = m' = \langle c', l' \rangle$, gdzie $c' = c$

$$l'(x) = \begin{cases} l(x) + 1, & \text{jeżeli } x = w \\ l(x), & \text{jeżeli } x \neq w. \end{cases}$$

Powtarzając podane instrukcje n razy otrzymamy przesunięcie wskaźnika w o n pozycji w lewo bądź w prawo.

Niech l_n i p_n $(n \geqslant 0)$ oznaczają przesunięcie wskaźnika w o n miejsc w lewo bądź w prawo. Przesuwanie w lewo określimy :

1° $l_o = l$,

2° $l_{n+1} = ll_n$,

zaś przesuwanie w prawo :

1° $p_o = p$,

2° $p_{n+1} = pp_n$.

Przykład 2. Wpisywanie stałej k pod adres a.

$s_k(a) : k(a,m) = m' = \langle c', l' \rangle$, gdzie

$$c'(x) = \begin{cases} k, & \text{jeżeli } x = l'(w) \\ c(x), & \text{jeżeli } x \neq l'(w) \end{cases}$$

$$l'(y) = \begin{cases} a, & \text{jeżeli } y = w \\ l(y), & \text{jeżeli } y \neq w \end{cases}$$

lub pisząc krócej

$$c'(x) = \begin{cases} k, & \text{jeżeli } x = a \\ c(x), & \text{jeżeli } x \neq a \end{cases}$$

Gdy k jest równe zero, to instrukcję tę nazywamy zerowaniem pamięci. Jest to instrukcja jednoadresowa.

Podany przykład jest raczej schematem instrukcji, aniżeli pojedynczą instrukcją.

Przykład 3. Następnik i poprzednik : N(a) i P(a).

Do zawartości miejsca o adresie a należy dodać (odjąć) liczbę 1.

$N(a) : N(a,m) = m' = \langle c', l' \rangle$, gdzie

$$c'(x) = \begin{cases} c(x) + 1, & \text{jeżeli } x = l'(w) \\ c(x), & \text{jeżeli } x \neq l'(w) \end{cases}$$

$$l'(y) = \begin{cases} a, & \text{jeżeli } y = w \\ l(y), & \text{jeżeli } y \neq w \end{cases}$$

lub pisząc krócej

$$c'(x) = \begin{cases} c(x) + 1, & \text{jeżeli } x = a \\ c(x), & \text{jeżeli } x \neq a. \end{cases}$$

Podobnie określamy poprzednik (odjęcie jedynki od zawartości miejsca o adresie a).

Przykład 4. Przesyłanie.

Instrukcja przesyłania P(a,b) powoduje wpisanie zawartości miejsca a w miejsce b.

$P(a,b) : P(a,b,m) = \langle m' \rangle = \langle c', l' \rangle$; gdzie

$$c'(x) = \begin{cases} c(a), \text{ jeżeli } x = 1'(w_2) \\ c(x), \text{ jeżeli } x \neq 1'(w_2) \end{cases}$$

$$1'(y) = \begin{cases} a, \text{ jeżeli } y = w_1 \\ b, \text{ jeżeli } y = w_2 \end{cases}$$

(Zakładamy, że pamięć posiada tylko dwa wskaźniki w_1 i w_2) Pisząc krócej

$$c'(x) = \begin{cases} c(a), \text{ jeżeli } x = b \\ c(x), \text{ jeżeli } x \neq b. \end{cases}$$

Przykład 5. Dodawanie. Można badać różne instrukcje realizujące dodawanie. Podamy tu dla ilustracji trzy różne instrukcje dodawania.

1° $D(a_1,a_2,a_3) : D(a_1,a_2,a_3,m) = m' = \langle c',1' \rangle$, gdzie

$$c'(x) = \begin{cases} c(a_1) + c(a_2), \text{ jeżeli } x = 1'(w_3) \\ c(x), \text{ jeżeli } x \neq 1'(w_3) \end{cases}$$

$$1'(y) = \begin{cases} a_1, \text{ jeżeli } y = w_1 \\ a_2, \text{ jeżeli } y = w_2 \\ a_3, \text{ jeżeli } y = w_3 \end{cases}$$

lub krócej

$$c'(x) = \begin{cases} c(a_1) + c(a_2), \text{ jeżeli } x = a_3 \\ c(x), \text{ jeżeli } x \neq a_3 \end{cases}$$

2° $D(a_1,a_2) : D'(a_1,a_2,m) = m' = \langle c',1' \rangle$, gdzie

$$c'(x) = \begin{cases} c(a_1) + c(a_2), \text{ jeżeli } x = a_2 \\ c(x), \text{ jeżeli } x \neq a_2. \end{cases}$$

3° $D(a) : D''(a,m) = m' = \langle c',1' \rangle$

$$c'(x) = \begin{cases} c(a) + c(a'), \text{ jeżeli } x = a' \\ c(x), \text{ jeżeli } x \neq a' \end{cases}$$

$a' = \text{const.}$

W pierwszej instrukcji dodawania mamy trzy wskaźniki w pamięci. Pierwsze dwa z nich wskazują miejsca argumentów, trzeci miejsce wyniku.

W drugiej instrukcji dodawania w pamięci mamy dwa wskaźniki, wskazujące argumenty dodawania. Wynik dodawania jest wpisywany w miejsce drugiego argumentu.

W trzeciej instrukcji dodawania mamy również dwa wskaźniki w pamięci, jednakże jeden z nich jest na stałe ustawiony na adres a'(np. 0), drugi zaś może być ustawiony dowolnie. Poza tym instrukcja ta działa podobnie, jak instrukcja poprzednia. Miejsce a' jest nazywane akumulatorem.

3. Syntaktyczne określenie instrukcji.

Instrukcje można również określić w ten sposób, że najpierw budujemy pewien język formalny, a następnie określamy znaczenie wyrażeń tego języka w ten sposób, aby znaczeniem były właśnie instrukcje.

Będziemy rozpatrywali język $J = \langle A, \Phi \rangle$, gdzie A jest alfabetem, zaś Φ zbiorem formuł tego języka.

Jako alfabet przyjmiemy symbole :

$$A = \{N,W,F,1,c,C,), \rightarrow\},$$

gdzie

$N = \{0,1,2,\ldots\}$ - symbole liczb naturalnych

$W = \{w_1,\ldots,w_n\}$ - nazwy wskaźników

$F = \{f_1,\ldots,f_k\}$ - nazwy operacji maszyny

Użyliśmy tu jednakowych oznaczeń na przedmioty i ich nazwy.

w_i jest bądź wskaźnikiem, bądź nazwą wskaźnika w_i. Ustawiczne rozróżnianie przedmiotów i ich nazw jest uciążliwe, dlatego rozróżnienie to będziemy robili tylko wtedy, gdy to będzie niezbędne dla jednoznaczności. Będziemy używali cudzysłowu " " dla zaznaczenia nazwy, np. "w_i" będzie nazwą w_i, "n" nazwą liczby n itp. l i c są pewnymi literami.

Elementy W będziemy nazywać stałymi.

Określimy teraz indukcyjnie pojęcie termu.

1° "n" jest termem

2° "l(w)" - jest termem

3° Jeżeli τ, $\tau_1,\ldots,\tau_n$ są termami, to "c(τ)" oraz "f($\tau_1,\ldots,\tau_n$)" są termami, gdzie f jest symbolem operacji n argumentowej.

Przykłady termów(pisane bez cudzysłowów): l(w), c(x), c(l(w)), +(c(x),1), +(c(x),c(c(y))).

Dla uproszczenia będziemy opuszczać nawiasy zgodnie z ogólnie przyjętymi zasadami, oraz w przypadku funktorów dwuargumentowych symbol operacji będziemy pisać między argumentami. Podane wyżej przykłady będą więc miały postać: lw, cx, clw, cn+1, c+ccy itp.

Jeżeli τ jest termem, zaś τ' termem lub stałą, to wyrażenie $\tau \to \tau'$ nazwiemy formułą. Przykłady formuł : ca → w, ca → b, 0 → 1, cx+1 → lw. Litery a,b,x w tych przykładach uważamy za nazwy pewnych liczb a nie za zmienne.

Aby określić, w jaki sposób przyporządkowujemy formułom instrukcje, wprowadzimy odpowiednie funkcje wartościujące. Najpierw określimy, co rozumiemy przez <u>wartość</u> termu lub stałej dla ustalanego stanu zadanej pamięci P wprowadzając funkcję v: T → N, gdzie

T - zbiór termów i stałych.

1° v("w") = w

2° v("l(w)") = l(w) (wartość funkcji l dla wskaźnika w, a więc pewna liczba naturalna)

3° v("c τ") = cv("τ")

4° v("f($\tau_1,\ldots,\tau_n$)") = f(v("τ_1"),...,v("τ_n"))

5° v("n") = n.

Np. jeżeli C5 = 2, to v(C5 + 7) = 9.

Wprowadzimy teraz funkcję $\bar{v} : \Phi \to R$ przyporządkowującą formułom instrukcje. W tym celu przyjmiemy, że pamięć posiada zbiór wskaźników $W = \{w_1,\ldots,w_k\}$, oraz k + 1 funkcji wprowadzających $I_0,I_1,\ldots,I_k$ zdefiniowanych jak niżej :

$I_0(a,m,\sigma) = \langle m' \rangle = \langle c', l' \rangle$, gdzie

$$c'(x) = \begin{cases} \sigma, & \text{jeżeli } x = a \\ c(x), & \text{jeżeli } x \neq a \end{cases}$$

$$l'(y) = \begin{cases} a, & \text{jeżeli } y = w_k \\ l''(y), & \text{jeżeli } y \neq w_k, \end{cases}$$

zaś pozostałe funkcje wprowadzające dla i > 0 mają postać :

$I_i(m,\sigma) = m' = \langle c', l' \rangle$, gdzie $c' = c$,

$$l'(y) = \begin{cases} \sigma, & \text{jeżeli } y = w_i \\ l''(y), & \text{jeżeli } y \neq w_i \end{cases} \qquad 1)$$

Funkcja I_o wprowadza więc symbol σ na miejsce o adresie a, zaś i-ta funkcja wprowadzająca ustawia wskaźnik w_i na miejscu o adresie σ.

Możemy teraz określić przyporządkowanie formułom instrukcji.

$$\bar{v}(\tau \to \tau') = r(\bar{a}_k, m) =$$

$$= \begin{cases} I_o(v(\tau'), m, v(\tau)), & \text{jeżeli } \tau' \text{ jest termem} \\ I_i(m, v(\tau)), & \text{jeżeli } v(\tau') = w_i \end{cases}$$

Złożeniem formuł $f(\tau_1, \ldots, \tau_i, \ldots, \tau_{n-1}) \to \tau_n$ oraz $\tau \to \tau_i$ będzie oczywiście formuła

$f(\tau_1, \ldots, \tau, \ldots, \tau_{n-1}) \to \tau_n$.

W ten sposób otrzymanej formule odpowiada instrukcja, która jest złożeniem instrukcji przyporządkowanych formule $f(\tau_1, \ldots, \tau_i, \ldots, \tau_{n-1}) \to \tau_n$ oraz $\tau \to \tau_i$. Formuły możemy więc uważać za nazwy instrukcji. W dalszym ciągu dla uproszczenia formuły będziemy też nazywali instrukcjami.

---

1) Można by tu mówić o jednej funkcji wprowadzającej, w której wskaźnik jest parametrem, $I(\bar{a}_2, m, \sigma)$, gdzie a_1 jest parametrem takim, że

$$I(a, a_2, m, \sigma) = I_{a_1}(a_2, m, \sigma).$$

Ćwiczenia.

1. Podać funkcje wejścia i wyjścia dla instrukcji podanych w przykładach 1 - 5.
2. Podać instrukcje realizowane przez następujące formuły : $cx \to x$, $l(w) + 1 \to w$, $c(ca + k) \to c(a - 1)$, $c(cx + cy) \to c(cy + 1)$.

§ 3. Sterowanie maszyny

Mając już określone pojęcia pamięci oraz instrukcji możemy przystąpić do zdefiniowania ostatniego elementu potrzebnego nam do określenia maszyny - sterowania. Przed podaniem właściwej definicji wprowadzimy najpierw kilka pojęć pomocniczych.

Niech X oznacza skończony zbiór liczb naturalnych $1,2,\ldots,n$. Grafem G będziemy nazywali uporządkowaną czwórkę $G = \langle X, Y, g_0, g_1 \rangle$, gdzie $Y \subset X$, $g_0 : X - Y \longrightarrow X$ oraz $g_1 : X - Y \longrightarrow X$. Elementy X nazywamy punktami grafu G, elementy Y - punktami końcowymi grafu G. Zbiory X i Y będziemy też oznaczali czasem X(G) i Y(G) zaznaczając w ten sposób, że są to punkty określonego grafu G.

Ciąg $x_1,\ldots,x_r$, $x_i \in X$, $1 \leq i \leq r$, będziemy nazywali drogą z x_1 do x_r w G, jeżeli dla każdego i, $1 \leq i < r$ $x_{i+1} = g_j(x_i)$, $j = 0,1$. Jeżeli istnieje droga z x do y, to zapiszemy D(x,y).

Zbiorem punktów początkowych grafu G będziemy nazywali zbiór $Z(G) \subset X(G)$ określony następująco :

$Z(G) = \{x : x \in X(G) - Y(G) \wedge (\bigwedge y \in X(G))(x \neq g_i(y))$,i=0,1.

Grafem przepływu nazwiemy każdy graf spełniający warunki:

1°. $\overline{\overline{Z(G)}} = 1$, $Z(G) = \{x_0\}$,

2°. $Y(G) \neq \phi$,

3°. $\bigwedge x \left[(x \in X(G) - x_0) \longrightarrow D(x_0,x)\right]$

4°. $\bigwedge x \{x \in X(G) - Y(G) \longrightarrow (\bigvee y \in Y(G))\ D(x,y)\}$.

T.zn. graf przepływu jest to taki graf, który posiada

jeden punkt początkowy, co najmniej jeden punkt końcowy; z punktu początkowego istnieje droga do każdego punktu grafu oraz z każdego punktu nie końcowego istnieje droga do jakiegoś punktu końcowego.

Niech $K = \{K_1,\ldots K_s\}$, gdzie $K_i \subseteq M$. K_i będziemy nazywali warunkiem. Jeżeli $m \in M$ i $m \in K_i$, to powiemy, że m spełnia warunek K_i; jeżeli zaś $m \in M - K_i$ powiemy, że m nie spełnia warunku K_i.[1)]

Sterowanie maszyny określimy jako graf przepływu, w którym każdemu punktowi przyporządkujemy instrukcje z ustalonego zbioru instrukcji R, oraz warunek z ustalonego zbioru warunków K.

Sterowanie możemy więc przedstawić w postaci

---

1) Czasem jest wygodnie na zbiorach K_i wprowadzić działania sumy " $\cup$ ", iloczynu " $\cap$ ", uzupełnienia " - " i mówić, że m spełnia warunek f jeżeli $m \in f(K_{i_1},..,K_{i_n})$ gdzie $f(K_{i_1},\ldots,K_{i_n})$ oznacza zbiór otrzymany ze zbiorów $K_{i_1},\ldots,K_{i_n}$ za pomocą działań $\cup$, $\cap$, - . Np. niech $f(K_1,K_2,K_3)$ będzie zbiorem $(K_1 \cup K_2) \cap (-K_3)$. Wtedy"m spełnia warunek f" znaczy, że

$$m \in (K_1 \cup K_2) \cap (-K_3).$$

W dalszym ciągu będziemy rozumieli spełnienie warunku jako należenie do jednego ze zbiorów K_i, chyba że wyraźnie powiemy inaczej.

$S = \langle G, R, K, \varphi, \psi \rangle$, gdzie

$\varphi : X(G) \longrightarrow R$ oraz $\psi : X(G) \to K$.

Punkty grafu będziemy też nazywali <u>stanami sterowania</u>.

§ 4. <u>Maszyny</u>

Maszyną $\mathcal{M}$ będziemy nazywali funkcję

$$\mathcal{M} : M \times N \longrightarrow M \times X(G)$$

zdefiniowaną następująco :

1°. $\mathcal{M}(m,1) = \langle m', x_0 \rangle$, $m' = \varphi(x_0)(m)$

2°. $\mathcal{M}(m,k+1) = \begin{cases} \langle m_1, x \rangle & \text{,jeżeli } \alpha_1(\mathcal{M}(m,k)) \notin \psi(x') \\ \langle m_2, y \rangle & \text{,jeżeli } \alpha_1(\mathcal{M}(m,k)) \in \psi(x') \end{cases}$

gdzie

$x' = \alpha_2(\mathcal{M}(m,k))$

$x = g_0(x')$

$y = g_1(x')$

$m_1 = \varphi(x)(\alpha_1(\mathcal{M}(m,k)))$

$m_2 = \varphi(y)(\alpha_1(\mathcal{M}(m,k)))$

Zbiór $E = M \times X(G)$ będziemy nazywać zbiorem <u>stanów maszyny</u>.

Powiemy, że stany pamięci m i m' spełniają funkcję $\mathcal{M}$, jeżeli w p-tym kroku

1°. $m = \alpha_1(\mathcal{M}(m',p))$,

2°. $\alpha_2(\mathcal{M}(m',p)) \in Y(G)$.

Wprowadzimy funkcję zdaniową $\Theta(\mathcal{M},m,m',k)$ zdefiniowaną jak następuje :

$$S(\mathcal{M},m,m',p) = \begin{cases} 1 \text{ , jeżeli m i m' spełniają w kroku p, } \mathcal{M} \\ 0 \text{ , jeżeli m i m' nie spełniają w p-tym kroku } \mathcal{M} . \end{cases}$$

Twierdzenie 1.

$$(\wedge\, m_1,m_2,m_3,p)\ \{S(\mathcal{M},m_2,m_3,p) = 1 \wedge m_1 \neq m_2 \rightarrow S(\mathcal{M},m_1,m_3,p) = 0\}\ .$$

Twierdzenie 2.

$$(\wedge\, \mathcal{M},m_1,m_2,p,q)\ \{S(\mathcal{M},m_1,m_2,p) = 1 \wedge p \neq q \rightarrow S(\mathcal{M},m_1,m_2,q) = 0\}\ .$$

Obliczeniem realizowanym przez maszynę $\mathcal{M}$ będziemy nazywali ciąg jej stanów

1) $e_0,e_1,\ldots\ e_i \in E$, taki że $e_0 = \langle m_0,x_0\rangle$ zaś dla każdego $i > 1$ $e_{i+1} = \mathcal{M}(m_0,i)$.

Obliczenie jest nieskończone, jeżeli ciąg 1) jest nieskończony, zaś skończone, jeżeli ciąg 1) jest skończony,

$$e_0,e_1,\ldots,e_k\ ,$$

oraz $\alpha_2(e_k) \in Y(G)$.

Powiemy, że $\phi_{\mathcal{M}}$ jest funkcją obliczalną przez maszynę $\mathcal{M}$, jeżeli $\phi_{\mathcal{M}} : \Sigma^r \rightarrow \Sigma^p$ oraz dla każdego $\sigma_1,\ldots,\sigma_r$, $(\sigma_i \in \Sigma)$ istnieje skończone obliczenie takie, że

$$\alpha_1(e_0) = I(\sigma_1,\ldots,\sigma_r,m) \text{ oraz}$$
$$\phi_{\mathcal{M}}(\sigma_1,\ldots,\sigma_r) = \alpha_2(\vartheta(\alpha_1(e_k)))\ 1)$$

---

1) Dla uproszczenia pominęliśmy w funkcjach wejścia i wyjścia pamięci adresy.

Znaczy to, że stan początkowy pamięci maszyny jest uzależniony przez funkcję wejścia od wartości argumentów funkcji $\phi_{\mathcal{M}}$, zaś wartość funkcji $\phi_{\mathcal{M}}$ jest otrzymana poprzez funkcję wyjścia z końcowego stanu pamięci.

Będziemy mówili, że zbiór $\Sigma' \subseteq \Sigma^*$ jest rozstrzygalny przez maszynę $\mathcal{M}$, jeżeli dla każdego $\sigma \in \Sigma^*$

$$\phi_{\mathcal{M}}(\sigma) = \begin{cases} 0, \text{ jeżeli } & \sigma \in \Sigma' \\ \neq 0, \text{ jeżeli } & \sigma \notin \Sigma' \end{cases} \quad 1)$$

Będziemy mówili, że zbiór $\Sigma' \subseteq \Sigma^*$ jest akceptowalny przez maszynę $\mathcal{M}$, jeżeli dla każdego $\sigma \in \Sigma^*$ $\phi_{\mathcal{M}}(\sigma) = 0$ wtedy i tylko wtedy, gdy $\sigma \in \Sigma'$.

Powiemy, że zbiór $\Sigma' \subseteq \Sigma^*$ jest generowalny przez maszynę $\mathcal{M}$, jeżeli dla każdego $\sigma \in \Sigma'$ istnieją takie $\tau_1,\ldots,\tau_k$, $\tau_i \in \Sigma^*$, że $\phi_{\mathcal{M}}(\tau_1,\ldots,\tau_k) = \sigma$.

Wymienione definicje podają zasadnicze zadania, które mogą wykonywać maszyny. Mogą one więc obliczać wartości funkcji, decydować (rozstrzygać) o przynależności elementu do zbioru, akceptować pewne zbiory i generować pewne zbiory.

---

1) Przypominamy o umowie, że alfabetem są liczby naturalne.

W zależności od tego, które z wymienionych zadań może maszyna realizować, będziemy mówili o maszynach liczących (obliczających wartości funkcji), maszynach decydujących (rozstrzygających), maszynach akceptujących bądź wreszcie o maszynach generujących (zwanych czasem też generatorami).

Gdy zadane są pamięć, instrukcje i sterowanie maszyny $\mathcal{M}$ a szukamy funkcji $\Phi_{\mathcal{M}}$ - mówimy o analizie maszyny $\mathcal{M}$.

Gdy zaś zadana jest funkcja Φ i szukamy pamięci, instrukcji i sterowania maszyny $\mathcal{M}$ takiej, że $\Phi = \Phi_{\mathcal{M}}$ to mówimy o syntezie maszyny $\mathcal{M}$.

W teorii maszyn matematycznych bada się różne zależności między rozmaitymi maszynami a funkcjami, które one obliczają, czy też zbiorami, które mogą one rozstrzygać, generować czy akceptować.

Będziemy mówili, że dwie maszyny należą do tej samej klasy, jeżeli posiadają one jednakową pamięć oraz listę instrukcji. Maszyny tej samej klasy różnią się więc tylko sterowaniem [1].

Można wprowadzić różne pojęcia równoważności maszyn. Pierwsza grupa pojęć równoważności dotyczy pojedynczych maszyn.

---

1) Można oczywiście przyjąć również inną zasadę klasyfikacji maszyn.

Powiemy, że maszyny $\mathcal{M}$ i $\mathcal{M}'$ są o-równoważne, jeżeli

$$\Phi_{\mathcal{M}} = \Phi_{\mathcal{M}'}.$$

Maszyny $\mathcal{M}$ i $\mathcal{M}'$ są d-równoważne, jeżeli $\Delta_{\mathcal{M}} = \Delta_{\mathcal{M}'}$ gdzie $\Delta_{\mathcal{M}}$ i $\Delta_{\mathcal{M}'}$ oznaczają zbiory rozstrzygalne przez maszyny $\mathcal{M}$ i $\mathcal{M}'$.

Maszyny $\mathcal{M}$ i $\mathcal{M}'$ są a-równoważne, jeżeli $\Lambda_{\mathcal{M}} = \Lambda_{\mathcal{M}'}$ gdzie $\Lambda_{\mathcal{M}}$ i $\Lambda_{\mathcal{M}'}$ są zbiorami akceptowalnymi przez maszyny $\mathcal{M}$ i $\mathcal{M}'$.

I wreszcie maszyny $\mathcal{M}$ i $\mathcal{M}'$ nazwiemy g-równoważnymi, jeżeli

$$\Gamma_{\mathcal{M}} = \Gamma_{\mathcal{M}'}, \text{ gdzie}$$

$\Gamma_{\mathcal{M}}$ i $\Gamma_{\mathcal{M}'}$ oznaczają zbiory generowalne przez maszyny $\mathcal{M}$ i $\mathcal{M}'$.

Możemy również mówić o zawieraniu się maszyn $\mathcal{M}$ i $\mathcal{M}'$ ($\mathcal{M} \subset \mathcal{M}'$), jeżeli odpowiednie funkcje lub zbiory się zawierają.

Drugie pojęcie równoważności dotyczy klasy maszyn.

Powiemy, że klasy maszyn $\mathcal{K}$ i $\mathcal{K}'$ są o(g,d,a)-równoważne wtedy i tylko wtedy, gdy dla każdej maszyny $\mathcal{M}$ należącej do klasy $\mathcal{K}$ istnieje maszyna $\mathcal{M}'$ równoważna $\mathcal{M}$ należąca do klasy $\mathcal{K}'$ i na odwrót.

Zasadniczym problemem teorii maszyn jest badanie równoważności maszyn.

ROZDZIAŁ II

Przykłady maszyn

§ 5. Przykłady klas maszyn liczących

1. Maszyny jednoadresowe

Pamięć tych maszyn jest następująca:

$$A = \Sigma = N = \{0,1,2\ldots\}$$

$$W = \{w_o, w_1\} \qquad 1(w_o) = \text{const} = 0$$

Miejsce wskazywane przez wskaźnik w_o jest nazywane akumulatorem.

Funkcje wejścia i wyjścia dla tej pamięci mają postać :

$$\sigma_r : A \times M \to M \times N^2$$

$$\sigma_r(a,m) = \langle m', \bar{n}_2 \rangle = \langle\langle c, 1' \rangle, \bar{n}_2 \rangle \text{ , gdzie}$$

$$1'(x) = \begin{cases} 1(x), \text{ jeżeli } x = w_o \\ a, \text{ jeżeli } x = w_1 \end{cases}$$

zaś

$$\bar{n}_2 = \langle c\,1'w_1 , c\,1'w_o \rangle = \langle c(a), c(0) \rangle \text{ .}$$

Pamięć tych maszyn będzie miała dwie funkcje wejścia I oraz I' [1].

---

1) Przyjmowaliśmy do tej pory, że w pamięci jest tylko

jedna funkcja wejścia lub wyjścia. Nic nie stoi na przeszkodzie, aby przyjmować większą liczbę tych funkcji dla jednej pamięci. Znaczy to, że w danej pamięci możemy w różny sposób wpisywać bądź odczytywać symbole.

$I(a,m,n) = m' = \langle c', 1' \rangle$, gdzie

$$c'(x) = \begin{cases} n, \text{ jeżeli } x = 1'(w_o) = 0 \\ c(x), \text{ jeżeli } x \neq 1'(w_o) \neq 0 \end{cases}$$

$$1'(y) = \begin{cases} 1(y), \text{ jeżeli } y = w_o \\ a, \text{ jeżeli } y \neq w_o \end{cases}$$

Funkcja I powoduje więc wpisanie symbolu n do akumulatora, niezależnie od podanego w tej funkcji adresu i wskaźnik w_1 ustawi na miejscu o adresie a.

$I'(a,m,n) = m' = \langle c', 1' \rangle$, gdzie

$$c'(x) = \begin{cases} n, \text{ jeżeli } x = 1'(w_1) = a \\ c(x), \text{ jeżeli } x \neq 1'(w_1) \neq a \end{cases}$$

$$1'(y) = \begin{cases} a, \text{ jeżeli } y = w_1 \\ 1(y), \text{ jeżeli } y = w_o. \end{cases}$$

Ta z kolei funkcja wejścia wpisuje symbol n pod adres a podany jako argument funkcji I'.

W maszynach jednoadresowych przyjmuje się na ogół następujący zbiór operacji :

$$F = \{+, -, \cdot, /, \rightarrow, \leftarrow, N, P, 0\} \text{ , gdzie}$$

+ (x,y) = x + y , podobnie - , · , / oznaczają różnice, iloczyn i iloraz (część całkowita ilorazu) ; funkcja $\rightarrow(x,y) = f(x,y) = y$ zaś $\leftarrow(x,y) = g(x,y) = x$. 1, N i P oznaczają następnik N(x,y) = x + 1, i poprzednik P(x,y) = x - 1 (dla jednolitości przyjęliśmy tu, że funkcje N i P są również dwuargumentowe). O(x,y) = const = 0.

Instrukcje w maszynach jednoadresowych mają więc postać :

$$r(a,m) = \begin{cases} I(a,m,f[\alpha_2(\sigma_r(a,m))]), & \text{jeżeli } f=+,-,\cdot,/,\leftarrow \\ I'(a,m,f[\alpha_2(\sigma_r(a,m))]), & \text{jeżeli } f=\rightarrow,N,P,O \end{cases}$$

gdzie

$$m' = \alpha_1[\sigma_r(a,m)] .$$

Operacje asymetryczne (dodawanie, odejmowanie, mnożenie, dzielenie) są więc wykonywane na argumentach znajdujących się pod adresem a i w akumulatorze, zaś wynik tych operacji umieszczany jest zawsze w akumulatorze. Funkcje N i P powodują dodanie lub odjęcie jedynki w miejscu a. Operacja $\rightarrow$ powoduje przepisanie liczby spod adresu a do akumulatora, zaś operacja $\leftarrow$ - przepisanie liczby z akumulatora do miejsca o adresie a. Operacja O powoduje wpisanie O pod adres a.

Możemy więc wszystkie instrukcje maszyny jednoadresowej przedstawić w postaci tablicy.

skrót instr.	instrukcja
+ a	ca + c0 → 0
- a	ca - c0 → 0
· a	ca · c0 → 0
/ a	ca / c0 → 0
Na	ca + 1 → a
Pa	ca - 1 → a
← a	ca → 0
→ a	c0 → a
Oa	0 → a

2. Maszyny dwuadresowe

W maszynach dwuadresowych przyjmiemy również, że

$$A = \Sigma = N = \{0,1,2,\ldots\}$$

$$W = \{w_1, w_2\} .$$

Funkcją wyjścia dla tych maszyn jest

$$\sigma_r = A^2 \times M \rightarrow M \times N^2$$

$$\sigma_r(\bar{a}_2,m) = \langle m', \bar{n}_2 \rangle = \langle \langle c,l' \rangle, \bar{n}_2 \rangle \text{ gdzie}$$

$$l'(x) = \begin{cases} a_1, & \text{jeżeli } x = w_1 \\ a_2, & \text{jeżeli } x = w_2 , \end{cases}$$

natomiast

$$\bar{\bar{n}}_2 = \langle cl'w_1 , cl'w_2 \rangle = \langle c(a_1), c(a_2) \rangle .$$

Funkcje wejścia pamięci będą również jak poprzednio dwie : I, I' :

$I(\bar{a}_2,m,n) = m' = \langle c',l' \rangle$, gdzie

$$c'(x) = \begin{cases} n, \text{ jeżeli } x = l'(w_1) \\ c(x), \text{ jeżeli } x \neq l'(w_2) \end{cases}$$

$$l'(y) = \begin{cases} a_1, \text{ jeżeli } y = w_1 \\ a_2, \text{ jeżeli } y = w_2 \end{cases}$$

Funkcja I powoduje więc wpisanie symbolu n na miejsce a_2. Adres a_1 nie jest tu istotny.

Drugą funkcję wejścia I' określimy w ten sposób :

$I'(\bar{a}_2,m,n) = m' = \langle c',l' \rangle$, gdzie

$$c'(x) = \begin{cases} n, \text{ jeżeli } x = l'(w_1) \\ c(x), \text{ jeżeli } x \neq l'(w_1) \end{cases}$$

$$l'(y) = \begin{cases} a_1, \text{ jeżeli } y = w_1 \\ a_2, \text{ jeżeli } y = w_2 \end{cases}$$

Ta funkcja umożliwia więc wpisanie symbolu n pod pierwszy adres występujący jako argument tej funkcji.

Zbiór operacji dla maszyn dwuadresowych przyjmiemy identyczny jak w maszynach jednoadresowych.

$$F = \{+, -, \cdot, /, \rightarrow, \leftarrow, N, P, 0\} .$$

Instrukcje więc będą miały postać :

$$r(\bar{a}_2,m) = \begin{cases} I(\bar{a}_2,m';f[\alpha_2(\sigma_r(\bar{a}_2,m)] \text{ ,jeżeli } f=+,-,\cdot,/,\leftarrow \\ I'(\bar{a}_2,m';f[\alpha_2(\sigma_r(\bar{a}_2,m)] \text{ ,jeżeli } f=\rightarrow,N,P,0 \end{cases}$$

gdzie

$$m' = \alpha_1 \sigma_r(\bar{a}_2,m) .$$

Wszystkie instrukcje maszyn dwuadresowych możemy więc przedstawić w postaci tablicy :

Skrót instrukcji	Instrukcja
$+ a_1 a_2$	$ca_1 + ca_2 \rightarrow a_2$
$- a_1 a_2$	$ca_1 - ca_2 \rightarrow a_2$
$\cdot a_1 a_2$	$ca_1 \cdot ca_2 \rightarrow a_2$
$/ a_1 a_2$	$ca_1 / ca_2 \rightarrow a_2$
$\rightarrow a_1 a_2$	$ca_1 \rightarrow a_2$
$\leftarrow a_1 a_2$	$ca_2 \rightarrow a_1$
$N a_1 a_2$	$ca_1 + 1 \rightarrow a_1$
$P a_1 a_2$	$ca_1 - 1 \rightarrow a_1$
$0 a_1 a_2$	$0 \rightarrow a_1$

3. Maszyny trójadresowe

Postępując podobnie jak w obu poprzednich przykładach możemy otrzymać dla maszyn trójadresowych następujące instrukcje :

Skrót instrukcji	Instrukcja
$+ a_1 a_2 a_3$	$ca_1 + ca_2 \rightarrow a_3$
$- a_1 a_2 a_3$	$ca_1 - ca_2 \rightarrow a_3$
$\cdot a_1 a_2 a_3$	$ca_1 \cdot ca_2 \rightarrow a_3$
$/ a_1 a_2 a_3$	$ca_1 / ca_2 \rightarrow a_3$
$\rightarrow a_1 a_2 a_3$	$ca_1 \rightarrow a_3$
$\leftarrow a_1 a_2 a_3$	$ca_3 \rightarrow a_1$
$N a_1 a_2 a_3$	$ca_3 + 1 \rightarrow a_3$
$P a_1 a_2 a_3$	$ca_3 - 1 \rightarrow a_3$
$0 a_1 a_2 a_3$	$0 \rightarrow a_3$

4. Uwagi o maszynach adresowych

Maszyny jedno, dwu i trójadresowe itd. będziemy nazywali maszynami adresowymi.

Twierdzenie 1. Jeżeli $f(x_1,\ldots,x_n)$ oraz $g(y_1,\ldots,y_i,\ldots,y_k)$ są funkcjami obliczalnymi przez maszyny adresowe, to również funkcja

$$g(y_1,\ldots,f(x_1,\ldots,x_n),\ldots,y_k)$$

jest obliczalna przez maszyny adresowe.

Twierdzenie to jest oczywiste, wartość funkcji f możemy bowiem umieścić za pomocą instrukcji przesyłania w dowolnym miejscu pamięci, w szczególności w miejscu, w którym ma być zapisana wartość zmiennej y_i.

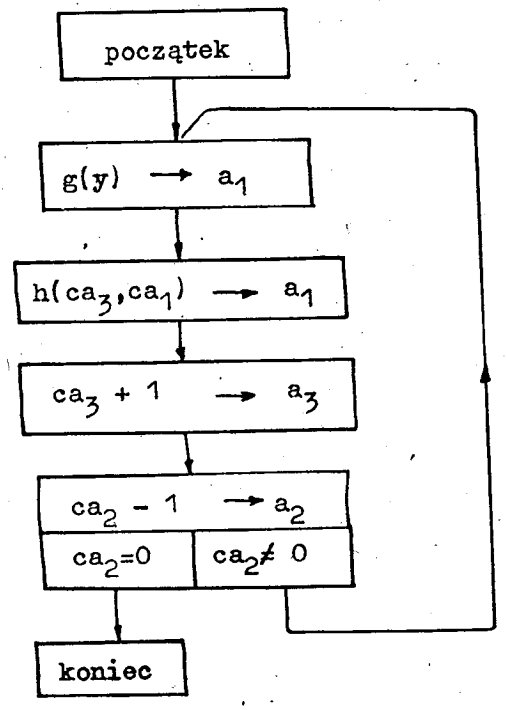
Twierdzenie 2. Jeżeli funkcje g i h są obliczalne za pomocą maszyn adresowych, to również funkcja h określona następująco :

1°. $f(0,y) = g(y)$

2°. $f(x+1,y) = h(x,f(x,y))$

jest obliczalna za pomocą maszyn adresowych.

Dodów. Przyjmijmy, że wartość funkcji g jest umieszczona w miejscu a_1, wartość funkcji h w miejscu a_1, wartość zmiennej x w miejscu a_2. W miejscu a_3 znajduje się na początku zero. Rozpatrzmy maszynę o następującym sterowaniu:



Po obliczeniu wartości funkcji g(y) umieszczamy w miejscu a_1. Następnie liczymy wartość funkcji h i wynik umieszczamy w a_1 oraz ca_3 zwiększamy o 1. Dalej wartość zmiennej x zmniejszamy o 1 i sprawdzamy, czy jest ona równa zeru. Jeżeli tak, to obliczenie jest zakończone ; jeżeli nie - powtarzamy jeszcze raz obliczanie wartości funkcji h przyjmując jako drugi argument poprzednio obliczoną wartość. W ten sposób otrzymamy w miejscu a_1 po x-krotnym powtórzeniu funkcji h wartość funkcji f. Wartość tę możemy oczywiście odczytać z pamięci poprzez funkcje

wyjścia. Wartości y i x przed rozpoczęciem liczenia możemy wprowadzić w odpowiednie miejsce pamięci poprzez funkcje wejścia.

Twierdzenie 3. Jeżeli funkcja f jest obliczalna przez maszyny adresowe, to również funkcja

$x_i = g(x_1,..,x_{i-1},x_{i+1},...,x_n) = \mu x_i[f(x_1,..,x_i,..,x_n)=0]$,

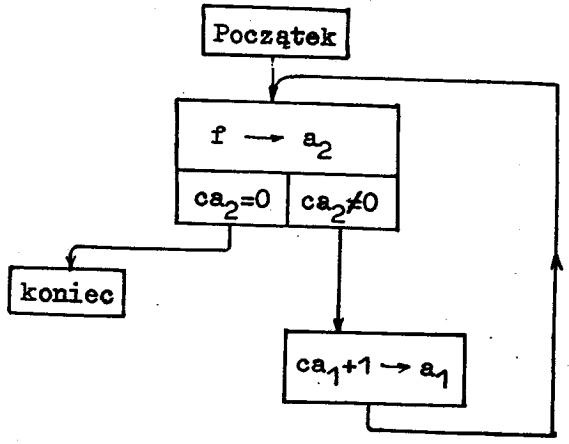
jest obliczalna przez maszyny adresowe.

x_i - oznacza najmniejsze takie x_i, że

$$f(x_1,...,x_i,...,x_n) = 0.$$

Dowód. Poprzez funkcje wejścia możemy wprowadzić do pamięci wartość zmiennych $x_1,..,x_n$ przy czym wartość $x_i=0$ jest umieszczona w miejscu a_1, zaś wartość funkcji f będzie umieszczona w miejscu a_2.

Rozpatrzmy następujące sterowanie :



Łatwo sprawdzić, że maszyna o takim sterowaniu oblicza wartość funkcji g. Poprzez funkcję wyjściową możemy wartość funkcji g odczytać z pamięci.

Z twierdzeń 1,2,3 wynika bezpośrednio

Twierdzenie 4. Maszyny adresowe są równoważne.

5. Maszyny bezadresowe

Rozpatrzmy następującą maszynę bezadresową. Pamięć tej maszyny określimy jak następuje :

$A = \Sigma = N = \{0,1,2,....\}$

$\mathcal{W} = \{w_0,w_1\}$

$l(w_0) = const = 0.$

W pamięci tej będą dwie funkcje wyjścia $\mathcal{O}_r$ oraz $\mathcal{O}$.

$\mathcal{O}_r = M \to M \times N^2$

$\mathcal{O}_r(m) = < m,\bar{n}_2 >$ gdzie

$\bar{n}_2 = < clw_1, clw_0 >$

$\mathcal{O}(m) = < m,n >$, gdzie

$n = l(w_1)$

Określimy trzy funkcje wejścia I, I', I" .

$I(m,n) = m' = < c',l' >$, gdzie

$$c'(x) = \begin{cases} n, \text{ jeżeli } x = l'(w_0) = 0 \\ c(x), \text{ jeżeli } x \neq l'(w_0) \end{cases}$$

$l' = l.$

$I'(m,n) = m' = < c',l' >$, gdzie

$$c'(x) = \begin{cases} n, \text{ jeżeli } x = l'(w_1) \\ c(x), \text{ jeżeli } x \neq l'(w_1) \end{cases}$$

$l = l'.$

$I''(m,n) = m' = < c',l' >$,gdzie

$c' = c$

$$l'(y) = \begin{cases} n, \text{ jeżeli } y = w_1 \\ l(y), \text{ jeżeli } y = w_0 \end{cases}$$

Sens wszystkich trzech funkcji wprowadzających jest oczywisty.

Jako zbiór operacji przyjmiemy :

$$F = \{+ , - , - , / , \rightarrow , \leftarrow , N , P , 0 \} ,$$

gdzie znaczenie symboli jest takie same jak w maszynie jednoadresowej.

Instrukcje realizujące operacje F określimy następująco :

$$r(m) = \begin{cases} I(m', f[\alpha_2(\sigma_r(m)] \text{ ,jeżeli } f=+,-,\cdot, / , \leftarrow \\ I'(m', f[\alpha_2(\sigma_r(m)] \text{ ,jeżeli } f= \rightarrow, N, P, 0 \end{cases} ,$$

gdzie

$$m' = \alpha_1 [\sigma_r(m)] .$$

Przyjmiemy ponadto, że w maszynie bazadresowej są dwie instrukcje przesuwania wskaźnika w_1 w lewo i w prawo o jedno miejsce. Instrukcje te oznaczymy literami p i l oraz określimy je następująco :

$r'(m) = I''(m', f(\sigma'(m)))$, gdzie $f = N$ dla przesuwania w prawo, zaś $f = P$ dla przesuwania w lewo.

Maszyna ta posiada następujące instrukcje :

Skrót	Instrukcja
+	$clw_1 + c0 \rightarrow 0$
-	$clw_1 - c0 \rightarrow 0$
.	$clw_1 \cdot c0 \rightarrow 0$
/	$clw_1 / c0 \rightarrow 0$
←	$clw_1 \rightarrow 0$
→	$c0 \rightarrow lw_1$
N	$clw_1 + 1 \rightarrow lw_1$
P	$clw_1 - 1 \rightarrow lw_1$
0	$0 \rightarrow lw_1$
l	$lw_1 + 1 \rightarrow w_1$
p	$lw_1 - 1 \rightarrow w_1$

Można łatwo wykazać, że zachodzi

Twierdzenie. Maszyny bazadresowe i adresowe są równoważne.

Ćwiczenia

1. Zbadać, czy jeżeli w maszynach jednoadresowych usuniemy operacje
   a) N,P
   b) 0
   c) ←
   d) →

   to otrzymamy maszyny równoważne ?
2. Zbadać to samo zagadnienie dla maszyn dwu i trójadres.

§ 6. Przykłady klas maszyn rozstrzygających (akceptujących)

1. Automaty Rabina-Scotta

Określimy teraz klasę maszyn zwanych automatami skończonymi Rabina-Scotta. Maszyny te odgrywają dużą rolę w teorii maszyn matematycznych oraz mają liczne zastosowania praktyczne.

Automaty te określimy przez podanie funkcji obliczalnych przez tę klasę automatów.

Niech Σ będzie skończonym alfabetem, Σ^* zbiorem wszystkich słów nad alfabetem Σ, oraz $\Sigma' \subset \Sigma$ niech będzie zbiorem symboli końcowych alfabetu Σ, zaś ψ niech będzie funkcją charakterystyczną zbioru Σ', tzn.

$$\psi(\sigma) = \begin{cases} 0, & \text{jeżeli } \sigma \in \Sigma' \\ 1, & \text{jeżeli } \sigma \notin \Sigma' \end{cases}$$

$\sigma_0 \in \Sigma$ jest wyróżnionym elementem Σ.

Wprowadzimy funkcje

$$\varphi : \Sigma \times \Sigma \to \Sigma$$

i rozszerzymy funkcje φ na skończone słowa nad alfabetem Σ w następujący sposób:

1°. $\varphi^*(\sigma) = \varphi(\sigma_0, \sigma)$,

2° $\varphi^*(\sigma_1, \ldots, \sigma_k) = \varphi(\varphi^*(\sigma_1, \ldots, \sigma_{k-1}), \sigma_k)$.

Następnie wprowadzimy funkcje

$$\Theta : \Sigma^* \to \{0,1\},$$

określoną jak niżej

$$\Theta(\sigma_1, \ldots, \sigma_n) = \psi(\varphi^*(\sigma_1, \ldots, \sigma_k)).$$

Każdą maszynę $\mathcal{M}$ taką, że $\Phi_{\mathcal{M}} = \Theta$ będziemy nazywać automatem skończonym Rabina i Scotta [1].

Powiemy, że automat Rabina-Scotta A akceptuje słowo $P \in \Sigma^*$, wtedy i tylko wtedy, gdy

$$\Phi_A = \Theta(P) = 0.$$

Można więc uważać, że automat taki czyta symbol po symbolu słowo P. Jeżeli po przeczytaniu ostatniego symbolu, wartość funkcji φ^* jest symbolem końcowym, to słowo jest akceptowane przez automat R-S.

Podobnie można wprowadzić pojęcie nieskończonego automatu R-S.

Niech $\Sigma = \{0,1,\ldots\}$ będzie nieskończonym zbiorem liczb naturalnych i niech $\Sigma' \subset \Sigma$ będzie obliczalnym podzbiorem Σ. Pozostałe elementy definicji pozostają bez zmiany. Każdy taki automat definiuje więc pewien zbiór relacji jedno-, dwu- ... itd. członowych. Relacje definiowane przez automaty nieskończone R-S będziemy nazywali relacjami R-S rozstrzygalnymi. Pytanie: czy każda relacja obliczalna jest R-S rozstrzygalna ?

Odpowiedź na to pytanie jest nie znana.

---

1) W literaturze automaty Rabina i Scotta są określone nieco inaczej.

2. Maszyny stosowe (push - down).

Niech $\Sigma = \{0,1,\ldots,k\}$ będzie skończonym alfabetem i niech $w : \Sigma \to \{0,1,-1\}$. Wartość funkcji $w(\sigma)$ będziemy nazywali wagą symbolu σ. Rozszerzmy wagę na słowa nad alfabetem Σ w następujący sposób :

1°. $w^*(\sigma) = w(\sigma)$

2°. $w^*(\sigma_1,\ldots,\sigma_k) = w^*(\sigma_1,\ldots,\sigma_{k-1}) + w(\sigma_k)$.

Każdą maszynę $\mathcal{M}$ taką, że $\varphi_{\mathcal{M}} = w^*$ nazwiemy maszyną stosową.

Powiemy, że maszyna stosowa $\mathcal{M}$ akceptuje słowo $\sigma_1,\ldots,\sigma_k$, gdy $\varphi_{\mathcal{M}}(\sigma_1,\ldots,\sigma_k) = 1$ oraz dla każdego i, $1 \leq i \leq k$ $\varphi_{\mathcal{M}}(\sigma_1,\ldots,\sigma_i) > 0$ [1].

Wiadowo, że maszyny stosowe stanowią szerszą klasę maszyn aniżeli automaty Rabina-Scotta.

Podaną definicję można również uogólnić na przypadek nieskończony. Przyjmiemy wtedy, że $\Sigma = \{0,1,\ldots\}$ jest zbiorem liczb naturalnych, C - zbiorem liczb całkowitych, zaś $w : \Sigma \to C$. w jest oczywiście funkcją obliczalną. Funkcja rozszerzona w może mieć również nieco ogólniejszą postać :

1°. $w^*(\sigma) = w(\sigma)$

2°. $w^*(\sigma_1,\ldots,\sigma_k) = h(w^*(\sigma_1,\ldots,\sigma_{k-1}), w(\sigma_k))$

gdzie h jest też pewną funkcją obliczalną.

---

1) W literaturze maszyny te są określone w inny sposób.

Nieskończone maszyny stosowe definiują więc również pewne relacje na liczbach naturalnych. Można również postawić pytanie, czy każda relacja obliczalna jest akceptowalna przez odpowiednią maszynę stosową.

Ćwiczenia

1. Podać maszynę stosową akceptującą formuły beznawiasowe Łukasiewicza.