ALGORYTMY

VOL. V • NO. 10 • 1969

INSTYTUT MASZYN MATEMATYCZNYCH

A L G O R Y T M Y

Vol. V N° 10 1969

Adres Redakcji: Warszawa, ul. Krzywickiego 34, tel. 28-37-29

T R E Ś Ć

ALGORYTMY, V, N° 10 /1969/, 5-19

MASZYNY PROGRAMOWANE

Zdzisław PAWLAK

Pracę złożono 18.4.1968 r.

Celem pracy jest sprecyzowanie podstawowych pojęć z zakresu maszyn matematycznych, takich jak pamięć maszyny, obliczenie, program itp. Punktem wyjścia pracy jest pojęcie procesu obliczania . Zdefiniowano maszynę jako funkcję częściową, której argumentami i wartościami są stany pamięci maszyny. Udowodniono pewne elementarne własności maszyn oraz wskazano na dalsze problemy związane z ich teorią.

Dla stworzenia teorii maszyn matematycznych przydatnej do formułowania i rozwiązywania rzeczywistych problemów prawdziwych wych maszyn matematycznych konieczne jest sprecyzowanie podstawowych pojęć takich jak, pamięć maszyny, maszyna cyfrowa, obliczenie, rozkaz, program etc.

Celem niniejszej notatki jest właśnie próba sprecyzowania tych pojęć. Punktem wyjścia jest pojęcie procesu obliczania. Pojęcie to pozwala na zdefiniowanie maszyny, jako funkcji częściowej, której argumentami i wartościami są stany pamięci maszyny. Taka definicja maszyny jest zgodna z intuicjami technicznymi, z drugiej zaś strony pozwala na łatwe formalne definiowanie i badanie różnych maszyn. Zdefiniowanie bowiem każdej maszyny sprowadza się do określenia pewnej funkcji, zaś badanie własności maszyn sprowadza się do badania odpowiednich własności funkcji. Przyjęta definicja jest na tyle ogólna, że

mieszczą się w niej wszystkie znane pojęcia maszyny jak np. maszyny Turinga, automaty skończone itp., jednakże tego rodzaju maszynami nie zajmujemy się tutaj: interesować nas będą takie maszyny, które są możliwie bliskie rzeczywistym maszynom matematycznym.

1. Procesy

Niech $\pi : T \to T$ będzie funkcją częściową. Ciąg $t_0, t_1, \ldots$ taki, że $t_i \in T$ dla każdego i, $0 \leqslant i$, $t_{i+1} = \pi(t_i)$ nazwiemy procesem[1] nieskończonym, t_i nazwiemy stanami procesu, zaś π funkcją przejścia procesu. Ciąg $t_0, t_1, \ldots, t_k$ nazwiemy procesem skończonym, jeżeli $t_i \in T$ oraz dla każdego i, $t_{i+1} = \pi(t_i)$, natomiast $t_k \notin D_\pi$, gdzie D_π oznacza dziedzinę funkcji π. t_0 nazwiemy stanem początkowym procesu, zaś t_k stanem końcowym. Proces ze stanem początkowym t_0 i funkcją przejścia π oznaczać będziemy przez $P_\pi(t_0)$.

Lemat 1. Jeżeli w procesie $P_\pi(t)$ istnieją dwa stany jednakowe tj. $(t_i = t_j,\ i \neq j)$, to dla każdego $k > 0$, $t_{i+k} = t_{j+k}$.

Dowód. Prawdziwość lematu 1 wykażemy przez indukcję względem k. Z założenia lemat 1 jest prawdziwy dla $k = 0$. Załóżmy, że lemat 1 jest prawdziwy dla $k = n$, tj. $t_{j+n} = t_{i+n}$. Wobec tego $\pi(t_{i+n}) = \pi(t_{j+n})$. Ponieważ $\pi(t_{i+n}) = t_{i+n+1}$, zaś $\pi(t_{j+n}) = t_{j+n+1}$, więc $t_{i+n+1} = t_{j+n+1}$, co kończy dowód.

Twierdzenie 1. Jeżeli proces $P_\pi(t_0) = t_0, t_1, \ldots, t_r$ jest skończony, to dla każdego i,j $(0 \leqslant i,j < r)$, $t_i \neq t_j$.

Dowód. Udowodnimy przez indukcję twierdzenie przeciwstawne, tj. jeżeli w procesie $P_\pi(t_0)$ istnieją takie i,j, że $i < j$ oraz $t_i = t_j$, to proces $P_\pi(t_0)$ nie jest skończony.

---

[1] Proces ten można uważać za szczególny przypadek procesu Markowa.

Jeżeli proces $P_{\pi}(t_0)$ jest skończony, to istnieje takie k, że $t_{j+k} \notin D_{\pi}$ i odwrotnie, jeżeli proces $P_{\pi}(t_0)$ jest nieskończony, to nie istnieje takie k, że $t_{j+k} \notin D_{\pi}$.

Pokażemy przez indukcję względem k, że dla każdego $k > 0$, $t_{j+k} \in D_{\pi}$. Jeżeli $k = 0$, to z założenia $t_i = t_j$. Ponieważ $t_i \in D_{\pi}$, więc i $t_j \in D_{\pi}$. Przyjmijmy, że $t_{j+k} \in D_{\pi}$ dla pewnego $k > 0$. Pokażemy, że $t_{j+k+1} \in D_{\pi}$.

Na podstawie lematu 1 jest $t_{j+k} = t_{i+k}$. Wobec tego $\pi(t_{j+k}) = \pi(t_{i+k})$. Ponieważ $\pi(t_{j+k}) = t_{j+k+1}$ oraz $\pi(t_{i+k}) = t_{i+k+1}$, więc $t_{j+k+1} = t_{i+k+1}$. Jednakże $t_{i+k+1} \in D_{\pi}$ na podstawie definicji procesu, więc $t_{j+k+1} \in D_{\pi}$ - co kończy dowód.

Proces $P_{\pi}(t_0)$ nazwiemy cyklicznym, jeżeli istnieją takie liczby p,r, że dla wszystkich $i \geq r$, $t_i = t_{i+p}$; najmniejszą liczbę p taką, że dla każdego $i \geq r$, $t_i = t_{i+p}$, nazwiemy cyklem procesu $P_{\pi}(t_0)$.

Z lematu 1 i twierdzenia 1 wynika

Wniosek 1. Jeżeli w procesie $P_{\pi}(t)$ istnieją takie i,j, że $i \neq j$ oraz $t_i = t_j$, to $\pi(t)$ jest cykliczny.

Jeżeli istnieją takie n,t, że

$$\underbrace{\pi \ldots \pi}_{n}(t) = t,$$

to π nazwiemy funkcją cykliczną, a liczbę n cyklem funkcji π i zapiszemy $\pi^n(t) = t$.

Jeżeli w procesie $P_{\pi}(t)$ dla każdego i,j, $t_i \neq t_j$, gdy $i \neq j$, to proces $P_{\pi}(t)$ nazwiemy różnowartościowym.

Twierdzenie 2. Dla wszystkich t proces $P_{\pi}(t)$ jest różnowartościowy wtedy i tylko wtedy, gdy jego funkcja przejścia π nie jest cykliczna.

Dowód. Pokażemy najpierw, że jeżeli dla każdego t proces $P_\pi(t)$ jest różnowartościowy, to funkcja przejścia π nie jest cykliczna. Jeżeli bowiem funkcja π jest cykliczna, to z definicji istnieją takie t i n, że $\pi^n(t) = t$, a więc istnieje proces $t, \pi(t), \pi^2(t), \ldots, \pi^n(t)$, w którym $t = \pi^n(t)$, co przeczy założeniu.

Pokażemy teraz, że jeżeli funkcja przejścia nie jest cykliczna, to dla każdego t proces $P_\pi(t)$ jest różnowartościowy. Załóżmy, że proces $P_\pi(t)$ nie jest różnowartościowy. Wtedy $P_\pi(t)$ zawiera dwa wyrazy jednakowe, a więc funkcja przejścia π jest cykliczna, co przeczy założeniu.

Twierdzenie 3. Jeżeli dla dowolnych $P_\pi(t_0)$ i $P_\pi(t'_0)$ istnieją takie i,j, że $t_i = t'_j$, to dla każdego $k \geq 0$ $t_{i+k} = t'_{j+k}$.

Dowód. Twierdzenie to udowodnimy przez indukcję.

Jeżeli $k = 0$, to twierdzenie jest prawdziwe na podstawie założenia.

Załóżmy, że twierdzenie 3 jest prawdziwe dla $k = n$ $(n > 0)$, tj. $t_{i+n} = t'_{j+n}$. Pokażemy, że jest ono prawdziwe dla $k = n+1$. Ponieważ $t'_{j+n+1} = \pi(t_{j+n})$ oraz $t_{i+n+1} = \pi(t_{i+n})$, zaś na podstawie założenia indukcyjnego mamy $t_{i+n} = t'_{j+n+1}$, co należało pokazać.

Z twierdzenia 3 wynikają następujące wnioski:

Wniosek 2. Jeżeli procesy $P_\pi(t_0) = t_0, t_1, \ldots, t_k$ oraz $P_\pi(t'_0) = t'_0, t'_1, \ldots, t'_p$ są skończone oraz istnieją takie i,j, że $i \neq j$ oraz $t_i = t'_j$, to $t_k = t'_p$ oraz $k-i = p-j$. Procesy $P_\pi(t_0)$ i $P_\pi(t'_0)$ są równe $(P_\pi(t_0) = P_\pi(t'_0))$, jeżeli dla każdego i, $t_i = t'_j$.

Wniosek 3. Jeżeli $t_0 = t'_0$, to $P_\pi(t_0) = P_\pi(t'_0)$.

2. Ogólne pojęcie maszyny

Powiemy, że stany t i t' T są w relacji M_π wtedy i tylko wtedy, gdy istnieje skończony proces $P_\pi = t_0, t_1, \ldots, t_k$ taki, że $t_0 = t$ i $t_k = t'$.

Twierdzenie 4. Relacja M_π jest funkcją.

Dowód. Prawdziwość tego twierdzenia wynika bezpośrednio z wniosku 2.

Każdą funkcję M_π będziemy nazywać maszyną.

Zbiór T nazwiemy zbiorem stanów albo pamięcią maszyny M_π; zbiór $T' = T - D_\pi$ - zbiorem stanów końcowych maszyny M_π, zaś funkcję π - funkcją przejścia albo sterowania maszyny M_π.

Proces $P_\pi(t)$ nazwiemy obliczeniem w maszynie M_π. Jeżeli ciąg $t_0, t_1, \ldots$ jest obliczeniem w maszynie M, to zapiszemy $(t_0, t_1, \ldots) \in M$.

Z definicji maszyny wynika, że każdą maszynę M możemy przedstawić w postaci:

$$M_\pi(t) = h\,(t, k_\mu),$$

gdzie

$$h\,(t,0) = t$$

$$h\,(t,k+1) = \pi\,(h\,(t,k)),$$

zaś k_μ - oznacza najmniejsze takie k, że

$$h\,(t,k) \notin D_\pi .$$

Niech M_1 i M_2 oznaczają odpowiednie maszyny o funkcjach przejścia π_1 i π_2 oraz pamięciach T_1 i T_2. Powiemy, że maszyna M_1 jest φ - naśladowana /symulowana/ przez maszynę M_2 (symbolicznie: $M_1 \overset{\subset}{\varphi} M_2$), wtedy i tylko wtedy, gdy istnieje takie wzajemnie jednoznaczne odwzorowanie $\varphi : T_1 \longrightarrow T_2$, że

$$1^\circ \quad \bigwedge_{t \in D_{M_1}} \varphi M_1\,(t) = M_2\,(\varphi\,(t)) \quad ,$$

$$2^{\circ} \quad \bigwedge_{t \in D_{\pi_1}} \bigvee_{k \geq i} \varphi^{\pi_1}(t) = \pi^{k}_{2}\left(\varphi(t)\right)$$

Jeżeli $M_1 \overset{\subset}{\underset{\varphi}{\leq}} M_2$ oraz $M_2 \overset{\subset}{\underset{\varphi^{-1}}{\leq}} M_1$, to powiemy, że maszyny M_1 i M_2 są izomorficzne i zapiszemy $M_1 \overset{\sim}{\underset{\varphi}{}} M_2$.

Powiemy, że maszyna M_π oblicza funkcję $f : X \longrightarrow X$ wtedy i tylko wtedy, gdy istnieją takie odwzorowania $\varkappa : X \to T$ oraz $\delta : T \longrightarrow X$, że dla każdego $x \in X$, $f(x) = \delta\left(M_\pi(\varkappa(x))\right)$.

Jeżeli M_π oblicza funkcję f przy ustalonym δ i $\varkappa$ to powiemy, że f jest obliczalna przez maszynę M_π i zapiszemy f_{M_π} lub krócej f_M (zakładając, że $\delta, \varkappa, \pi$ są ustalone i znane).

Niech M_1 oznacza maszynę M_{π_1}, zaś M_2 - maszynę M_{π_2}. Powiemy, że maszyny M_1 i M_2 są równoważne, symbolicznie $M_1 \equiv M_2$ wtedy i tylko wtedy, gdy $f_{M_1} = f_{M_2}$.

Bezpośrednio z definicji zawierania i równoważności maszyn wynikają następujące wnioski:

<u>Wniosek 4.</u> Jeżeli $M_1 \overset{\subset}{\underset{\varphi}{\leq}} M_2$, to

1°. Dla każdego obliczenia $P_1(t_0) = t_0, \ldots, t_n$ w M_1 istnieje obliczenie $P_2(t'_0) = t'_0, \ldots, t'_m$ w M_2 takie, że :

$$\bigwedge_{0 \leq i \leq n} \bigwedge_{i \leq j \leq m} \left[t'_j = \varphi(t_i) \,\&\, t'_0 = \varphi(t_0) \,\&\, t'_m = \varphi(t_n) \right],$$

$$2^{\circ}. \quad \bigwedge_{t \in D_{M_1}} d\left(P_1(t)\right) \leq d\left(P_2(\varphi(t))\right),$$

gdzie $d\left(P_1(t)\right)$ oznacza długość obliczenia $P_1(t)$.

Jeżeli obliczenie jest nieskończone, to nie zachodzi trzeci człon koniunkcji.

<u>Wniosek 5.</u> Jeżeli $M_1 \overset{\sim}{\underset{\varphi}{}} M_2$, to

1° . Dla każdego obliczenia $P_1(t_0) = t_0,\ldots,t_n$ w M_1 istnieje obliczenie $P_2(t'_0) = t'_0,\ldots,t'_n$ w M_2 takie, że

$$\bigwedge_{0 \leqslant i \leqslant n} \left[t'_i = \varrho\ (t_i) \right] ,$$

2° . $$\bigwedge_{t \in D_{M_1}} d\big(P_1\ (t)\big) = d\big(P_2\ (\varrho(t))\big) .$$

3. Maszyny programowane

Określimy najpierw pamięć maszyn programowanych.

Niech Σ i A będą dowolnymi zbiorami takimi, że $\Sigma \cap A \neq 0$. Σ będziemy nazywać alfabetem, zaś A zbiorem adresów. Zbiór funkcji $C \subset \Sigma^A$ nazwiemy pamięcią adresową. Ponieważ w dalszym ciągu będziemy zajmować się tylko pamięciami adresowymi, zamiast "pamięć adresowa" będziemy używali zwrotu "pamięć". Każdą funkcję $c \in C$ będziemy nazywać zawartością pamięci. Przyjmiemy w dalszym ciągu, że pamięć spełnia następujące warunki:

W 1. $\bigwedge_{c \in C}$ Dziedzina D_c jest skończona.

W 2. W zbiorze adresów A istnieje wyróżniony element l , zwany licznikiem rozkazów taki, że $\bigwedge_{c \in C} l \in D_c$.

W ten sposób określimy pamięć maszyn programowanych.

Zanim określimy sterowanie tego rodzaju maszyn, wprowadzimy najpierw kilka pojęć pomocniczych. Każdą funkcję postaci:

$$r : A^n \times C \longrightarrow C , \quad n \geqslant 0$$

nazwiemy schematem instrukcji n adresowej w pamięci C . Jeżeli w schemacie instrukcji adresy traktować będziemy jako

parametry, to funkcję

$$r_{\bar{a}_n} : C \longrightarrow C , \quad \bar{a}_n \in A^n$$

nazwiemy instrukcją n adresową w pamięci C . Instrukcje będziemy krótko oznaczali przez r^* .

Przyjmiemy w dalszym ciągu, że instrukcje spełniają następujący warunek:

W 3. $$\bigwedge_{r^*} \bigwedge_{c \in C} \bigvee_{a \in A} r^*(c) \,/A- l \cup a = c/A- l \cup a .$$

Znaczy to, że każda instrukcja może zmienić zawartość pamięci co najwyżej w dwu adresach l i a.

Niech $R_C^* \subset R_C$, gdzie R_C jest zbiorem wszystkich instrukcji w pamięci C , zaś $\varphi : \sum \longrightarrow R_C^*$ niech będzie wzajemnie jednoznaczną funkcją, zwaną w dalszym ciągu kodowaniem. Każdą maszynę posiadającą pamięć C spełniającą warunki W 1 i W 2 oraz funkcję przejścia określoną jak niżej

$$\bigwedge_{c \in C} c' = \left[\varphi\left(c^*(l)\right)\right](c) ,$$

gdzie $c^*(l) = c\left(c(l)\right)$,

nazwiemy maszyną programowaną.

Przykład. Rozpatrzymy maszynę trójadresową o pamięci $C = \sum^A$, gdzie $A = l \cup N$, zaś $N = 0,1,2,\ldots$ o następującym zbiorze schematów instrukcji :

$+(\bar{a}_3)$, $-(\bar{a}_3)$, $.(\bar{a}_3)$, $/(\bar{a}_3)$, $!(\bar{a}_3)$, $?(\bar{a}_3)$, $\text{stop}(\bar{a}_3)$,

gdzie $\bar{a}_3 = a_1 a_2 a_3$, $a_1 \in N$.

Schematy te są określane następująco:

$$c' = \left[+(\bar{a}_3)\right](c) ,$$

gdzie

$$c'(x) = \begin{cases} c(a_1) + c(a_2), & \text{gdy } x = a_3,\ x \in A \\ c(1) + 1, & \text{gdy } x = 1 \\ c(x), & \text{gdy } x \neq a_3 \text{ i } x \neq 1. \end{cases}$$

Podobnie możemy określić instrukcje odejmowania, mnożenia i dzielenia. Instrukcję $!(\bar{a}_3)$ nazwiemy przejściem bezwarunkowym i określimy ją następująco:

$$c' = \left[! (\bar{a}_3) \right](c) \text{ , gdzie}$$

$$c'(x) = \begin{cases} c(a_3) \text{ , gdy } x = 1 \\ c(x) \text{ , gdy } x \neq 1. \end{cases}$$

Instrukcję $?(\bar{a}_3)$ zwaną przejściem warunkowym zdefiniujemy jak niżej:

$$c' = \left[? (\bar{a}_3) \right](c) \text{ , gdzie}$$

$$c'(x) = \begin{cases} c(1) + 1 \text{ , gdy } x = 1 \text{ oraz } c(a_1) = 0, \\ c(a_3) \text{ , gdy } x = 1 \text{ oraz } c(a_1) \neq 0, \\ c(x) \text{ , gdy } x \neq 1. \end{cases}$$

Instrukcja $\text{stop}(\bar{a}_3)$ ma postać

$$c' = \left[\text{stop } (\bar{a}_3) \right] (c)$$

i jest nieokreślona dla żadnego $c \in C$.

Dla pełnego określenia maszyny musimy podać jeszcze kodowanie φ. Gdy zbiór instrukcji jest nieskończony, możemy przyjąć tu Gödelowską zasadę numeracji instrukcji. Dla skończonego zbioru adresów sprawa jest o wiele prostsza; wystarczy symbolom

+ , - , . , / , ! , ? , stop przyporządkować numery od 1 do 7. Numerem instrukcji $d_1, a_1, a_2 a_3$ będzie liczba, której cyfry rozwinięcia dziesiętnego są kolejnymi cyframi liczb d_1, a_1, a_2, a_3 d_1 jest numerem symbolu +, - , . , / , ? , ! , stop - uzupełniając odpowiednio liczby a_1, a_2, a_3 zerami, tak aby maksymalna ilość pozycji przeznaczonych na jeden adres pozwalała na zapisanie każdego adresu.

Jeżeli maszyna M spełnia warunek

W 4. $$\bigwedge_{(c_0,c_1,\ldots)\in M} \bigwedge_{0\leqslant i,j} \left[c_i(1) = c_j(1) \rightarrow c_i^*(1) = c_j^*(1) \right], \quad i \neq j,$$

to maszynę M nazwiemy stałoprogramową.

Jeżeli maszyna M spełnia warunek

W 5. $$\bigvee_{(c_0,c_1,\ldots)\in M} \bigvee_{0\leqslant i,j} \left[c_i(1) = c_j(1) \wedge c_i^*(1) \neq c_j^*(1) \right], \quad i \neq j,$$

to maszynę M nazwiemy zmiennoprogramową.

Niech K_s i K_z oznaczają odpowiednio klasy maszyn stało- i zmiennoprogramowych. Powiemy, że klasa K_s jest zawarta w klasie K_z wtedy i tylko wtedy, gdy

1) $$\bigwedge_{M\in K_s} \bigvee_{M'\in K_z} \bigvee_{\varphi} (M \subset_{\varphi} M').$$

Przyjmiemy, że dla maszyn programowanych odwzorowanie spełnia następujące warunki:

1. $$\bigwedge_{C,C'} C(x) = C'(x) \Leftrightarrow \varphi C(\varphi'(x)) = \varphi C'(\varphi'(x)),$$

gdzie $\varphi' : A \rightarrow A'$ jest wzajemnie jednoznacznym odwzorowaniem zbioru adresów pamięci maszyn M i M',

2. $1' = \varrho'(1)$, gdzie 1 i $1'$ są odpowiednio licznikami rozkazów w maszynach M i M'.*)

Twierdzenie 5. Klasa maszyn stałoprogramowych jest istotnie zawarta w klasie maszyn zmiennoprogramowych.

Dowód. Pokażemy najpierw, że dla każdej maszyny stałoprogramowej istnieje maszyna zmiennoprogramowa spełniająca warunek 1).

Niech M_3 będzie maszyną stałoprogramową z pamięcią C, zbiorem instrukcji R_s^* oraz funkcją przejścia π_s, zaś M_2 niech będzie maszyną zmiennoprogramową o pamięci C, zbiorze instrukcji R_z^* takim że $R_z^* \supset R_s^*$ i funkcji przejścia

$$\pi_z = \varphi_z(c(c(1))),$$

gdzie

$$\varphi_z : \Sigma \longrightarrow R_z^*$$

takiej, że jeżeli $\varphi(a) \in R^*$, to $\varphi(a) = \varphi_z(a)$.

Maszyna M_s jest wtedy oczywiście zawarta w maszynie M_z.

Pokażemy teraz, że jeżeli M_z jest maszyną zmiennoprogramową, to nie istnieje maszyna stałoprogramowa M_s taka, że $M_z \subset_{\varrho} M_s$. Mówiąc inaczej, jeżeli M_z jest maszyną zmiennoprogramową oraz $M_z \subset_{\varrho} M'$, to M' jest też maszyną zmiennoprogramową dla dowolnych maszyn M_z, M' i odwzorowania ϱ.

Ponieważ maszyna M_z jest zmiennoprogramowa, możemy więc napisać

1) $\bigvee_{c_0, c_1, \ldots \in M_z} \bigvee_{i,j} \left[c_i(1) = c_j(1) \,\&\, c_i^*(1) \neq c_j^*(1) \right]$.

Na podstawie założenia, że $M_z \subset_{\varrho} M'$, wniosku 4 warunków i i 2 spełnianych przez odwzorowanie ϱ - 1) możemy przepisać w postaci

*) Na konieczność umieszczenia tych warunków zwrócili mi uwagę dr G. Rozenberg oraz mgr Z. Sozańska.

2) $$\bigvee_{c'_0, c'_1, \ldots \in M'} \bigvee_{p,q} \left[c'_p(1') = c'_q(1') \wedge c''_p(1') \neq c''_q(1'), \right]$$

$$p \neq q$$

gdzie

$$c'_p = \varphi(c_i) \quad \text{oraz} \quad c'_q = \varphi(c_j) \quad \text{zaś} \quad 1' = \varphi'(1) .$$

A więc M' jest zmiennoprogramowa.

W podobny sposób można udowodnić wiele innych twierdzeń o maszynach programowych. Zostaną one pokazane w dalszych pracach.

*

Rozpatrywany tu zbiór stanów T można interpretować zależnie od potrzeb, jako zbiór liczb naturalnych skończony lub nieskończony , zbiór liczb rzeczywistych, zbiór funkcji czy też nawet bardziej złożonych pojęć matematycznych. Interpretacja zbioru T jako zbioru liczb rzeczywistych może być przydatna do opisu maszyn analogowych. Dla dobrego opisu istniejących maszyn wydaje się rzeczą konieczną wprowadzenie jeszcze bardziej złożonych pojęć matematycznych do opisu stanu maszyny.

Rozpatrzmy dla przykładu pamięć rzeczywistej maszyny cyfrowej. Pamięć taką można uważać za pewnego rodzaju graf, punkty którego są interpretowane jako komórki pamięci. Ponieważ w każdej komórce może znajdować się pewna liczba, możemy mówić o funkcji, której dziedziną jest zbiór punktów grafu, zaś przeciwdziedziną są liczby naturalne. Taki model pamięci tj. graf oraz funkcja zawartości określona na grafie dość dobrze oddaje techniczną strukturę wielu rodzajów pamięci.

Ponieważ techniczne przesyłanie zawartości komórek pamięci może odbywać się jedynie pomiędzy tymi komórkami, które są bezpośrednio połączone - każdą instrukcję maszyny możemy uważać za funkcję, która zmienia zawartość pamięci w ten sposób, że nowa zawartość zależy co najwyżej od zawartości komórek z nią sąsiadujących tj. komórek, które są z nią bezpośrednio połączone.

Dla dobrego określenia maszyny konieczne jest więc przyjęcie, że w zbiorze adresów określona jest pewna relacja binarna określająca strukturę połączeń w pamięci maszyny.

Ten sposób opisu pamięci wydaje się jednak dość niewygodny do formalnego traktowania, dlatego rzeczą wygodniejszą będzie przyjąć, że pamięć jest pewną przestrzenią ciągłą, w której jest określone pojęcie odległości. Na przestrzeni tej określona jest funkcja zawartości. Instrukcjami będą wtedy pewne operatory zmniejszające zawartość pamięci w ten sposób, że nowa zawartość dowolnego punktu przestrzeni może zależeć co najwyżej od zawartości punktów nie dalszych niż pewna z góry ustalona odległość.

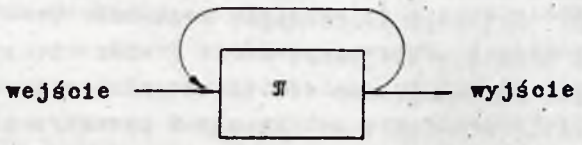
Inaczej mówiąc działanie instrukcji jest lokalne: zmiana zawartości dowolnego miejsca pamięci może zależeć jedynie od zawartości miejsc najbliższych. Miejsca dostatecznie dalekie nie mają bezpośrednio wpływu na zmianę zawartości miejsc najbliższych. Miejsca dostatecznie dalekie nie mają bezpośrednio wpływu na zmianę zawartości. Program powoduje propagację zmiany zawartości pamięci od punktu sąsiedniego do punktu sąsiedniego. Przypomina to rozchodzenie się zaburzenia w przestrzeni fizycznej.

Wydaje się, że zastosowanie modelu ciągłego dla maszyn cyfrowych może doprowadzić do głębszych i bliższych praktyce wyników, chociaż maszyny te są w swej istocie nieciągłe. Próba takiego ujęcia zostanie przedstawiona w jednym z najbliższych artykułów.

Na rozpatrywane tu sprawy można spojrzeć z nieco innego punktu widzenia. Zbiór T możemy interpretować jako alfabet skończony albo nieskończony , którego elementami mogą być liczby naturalne, liczby rzeczywiste, funkcje, przestrzenie itp. zależnie od potrzeby. Słowem w alfabecie T nazwiemy każdy ciąg skończony lub nieskończony elementów zbioru T . Będziemy mówili, że słowo W jest akceptowane przez maszynę o sterowaniu π wtedy i tylko wtedy, gdy $W = P_{\pi}(t)$, gdzie t jest pierwszym elementem słowa W . W ten sposób każda maszyna wyznacza pewien język.

Wynika stąd, że maszynę można zdefiniować nie jako funkcję, jak to uczyniliśmy na początku niniejszej pracy, a jako język zdefiniowany przez zadaną funkcję przejścia może sensowniej byłoby wtedy nazywać maszyną nie język a samą funkcję przejścia . Taka definicja może być do wielu celów nawet lepsza niż poprzednia. Maszyna jest bowiem wtedy utożsamiana ze zbiorem wszystkich wykonywanych przez nią obliczeń, również nieskończonych, i na takim zbiorze możemy wykonywać operacje teorio-mnogościowe otrzymując nowe maszyny. W ten sposób otrzymujemy pełniejszą charakterystykę pracy maszyny, gdyż wyłączone są również przypadki, kiedy maszyna się nie zatrzyma. Natomiast przy rozumieniu maszyny jako funkcji możemy rozpatrywać tylko przypadki pracy maszyny prowadzące do zatrzymania. Dla wielu zastosowań jest nie wystarczające.

Warto tu zwrócić jeszcze uwagę na następującą interpretację pojęcia maszyny : każdą maszynę można przedstawić w postaci jak niżej



Prostokątne pudełko symbolizuje urządzenie realizujące funkcję przejścia. Podany na wejściu pudełka sygnał reprezentuje stan początkowy maszyny. Urządzenie realizujące funkcję przejścia po pewnym czasie na wyjściu wyprodukuje sygnał reprezentujący następny stan maszyny. Sygnał ten ponownie jest poprzez sprzężenie zwrotne kierowany na wejście i jeżeli należy on do dziedziny funkcji przejścia produkowany jest następny stan maszyny.

W ten sposób na wyjściu maszyny pojawiają się kolejne stany procesu. Jest rzeczą ciekawą, że podany schemat maszyny jest na tyle ogólny, że mieści się w nim zarówno pojedynczy przerzutnik jak i cała maszyna całą maszynę można zresztą uważać za bardzo złożony przerzutnik . W schemacie tym mieszczą się także inne urządzenia elektroniczne, jak np. filtry, wzmacniacze, a także maszyny analogowe.

Wydaje się, że badania tego rodzaju mogą mieć również znaczenie dla dowodzenia twierdzeń, bowiem proces dowodzenia twierdzeń mieści się również w podanym schemacie maszyny i obliczenia.

Z podanych tu interpretacji płynie bardzo bogata problematyka badawcza, którą trudno omówić w tak krótkiej notatce.

STORED PROGRAM COMPUTERS

Summary

This note contains general definition of a computer and on this basis stored program computer is introduced. As a primitive notion we use the set of states T of a computer and a transition function $\pi : T \to T$. /π is partial function/. Sequence $t_0, t_1, \ldots$ is called process if for all i $t_{i+1} = \pi(t_i)$. The process is finite if it has the form $t_0, t_1, \ldots, t_k$ and t_k does not belong to the domain of a transition function π. Some elementary properties of processes are proved. Than the relation $M \subset T \times T$ is introduced by all finite process /by fixed transition function π / and it is shown that this relation is a function. This function is called machine. Some elementary properities of machines are shown, the inclusion /the simulation of one machine by another machine/ is defined, as well as the equivalence of machines is introduced.

In the second part of this note the stored program computer is defined. The notion of adressable memory, the notion of scheme of instruction and the instruction itself are defined for such kind of machines. It is shown that the class of stored program computers with modification of instructions is essentialy wider /in the sense of the concept of inclusion/ than the class of stored program computers without the modification. Thus the idea of von Neumann computer is essentialy different from those not containing the ability of modification.

Some further problem related to the theory of computers are outlined.

Warszawa, ul. Krzywickiego 34.

Z.P.

ALGORYTMY, V, N° 10 /1969/, 21-24

ON SOME PROPERTIES OF MACHINES

by W. KWASOWIEC

Received on Oct. 18th.1968 r.

This paper contains some results concerning mathematical model for computers. Each computers is represented by a function with arguments and values from the set of memory states. Some properties of functions representing computers are investigated.

In this note we shall treat the class of machines the concept of which was introduced by Z. Pawlak [2].

By $D\,f$ (or D_f) and $R\,f$ we shall denote the domain and the range of the function f respectively.

Let there be given any set T, the partial function $f: T \to T$ and the set $D_f \subset T$. First we shall quote the definition of a machine from [2].

Definition: By a machine we understand the function

$$M_f(t) = h(t, k_\mu), \qquad t \in T,\ k_\mu \in N$$

where

$$h(t,0) = t,$$

$$h(t,k+1) = f(h(t,k)),$$

$$k_\mu = \min_{k>0} \{ h(t,k) \in D_f \}.$$

Of course, the domain of the machine M_f is

$$D\,M_f = \left\{ t \in D_f : \bigvee_{k_\mu} \bigwedge_{k<k_\mu} f^k(t) \in D_f \wedge f^{k_\mu}(t) \notin D_f \right\},$$

where by $f^k(t)$ we understand $f(f\ldots f(t))$ (k times).

We shall try to characterize the class of machines.

Theorem 1. The class of machines is given by all functions g such that $D_g \cap R_g = 0$.

Proof. Let us consider an arbitrary machine M_f.
If $t \in D\,M_f$ then $M_f(t) \notin D_f$ by the definition of a machine. Because $D\,M_f \subset D_f$ therefore $M_f(t) \notin D\,M_f$ for every $t \in D\,M_f$. Consequently $D\,M_f \cap R\,M_f = 0$ for every machine M_f.
Now let $g:T \to T$ be an arbitrary function safisfying the condition $D_g \cap R_g = 0$. Then obviously $M_g(t) = h(t,1) = g(t)$ for some t, where h is the function obtained from the definition of a machine by the substitution of g for f.
Therefore we have obtained the machine M_g which is equal to the function g.

Corollary 1. For every function f

$$R\,M_f \subset T - D_f \qquad \text{and} \qquad M_{M_f} = M_f.$$

Let $Z(f) \overset{df}{=} \{ g : M_g = M_f \}$. By corollary 1 follows at once that $Z(M_f) = Z(f)$. We shall try to characterize the set $Z(f)$ only by means of the function f.

Theorem 2. $g \in Z(f)$ if and only if for every $t \in T$ either $\bigvee_{n_\mu>0} \bigvee_{k_\mu>0} \bigwedge_{n<n_\mu} \bigwedge_{k<k_\mu} f^n(t) \in D_f \wedge f^{n_\mu}(t) \notin D_f \wedge g^{k_\mu}(t) \in D_g \wedge$

$$\wedge\, g^{k_\mu}(t) \notin D_g \wedge f^{n_\mu}(t) = g^{k_\mu}(t), \text{ or}$$

$$\left(t \in D_f \vee \bigwedge_{n>0} f^n(t) \in D_f\right) \wedge \left(t \in D_g \vee \bigwedge_{n>0} g^n(t) \in D_g\right).$$

The proof is obvious because either $t \in D\,M_f \cap D\,M_g$ and then $M_f(t) = M_g(t)$, or $t \notin D\,M_f$ and $t \notin D\,M_g$ (since $DM_f = DM_g$).

The same problem may be given for the machine M_f (instead of the function f).

Corollary 2.

$$Z(f) = Z(M_f) = \left\{ g: \bigwedge_t \left[t \in D\,M_f \Leftrightarrow \bigvee_{k_\mu > 0} \bigwedge_{k < k_\mu} \left(g^k(t) \in D_g \wedge g^{k_\mu}(t) \notin D_g \wedge \right.\right.\right.$$

$$\left.\left.\left. \wedge g^{k_\mu}(t) = M_f(t) \right) \right] \right\}.$$

Theorem 3. For every functions f, g either $Z(f) = Z(g)$, or $Z(f) \cap Z(g) = 0$.

Proof. It suffices to prove that if the sets $Z(f)$, $Z(g)$ are non-empty, and $Z(f) \cap Z(g) \neq 0$, then $Z(f) = Z(g)$.
We suppose that $Z(f) \cap Z(g) \neq 0$. There exists the function f_1 such that $M_{f_1} = M_g$ and $M_{f_1} = M_f$.
From here $M_g = M_f$ and $Z(g) = \left\{ f_2 : M_{f_2} = M_g \right\} =$

$$= \left\{ f_2 : M_{f_2} = M_f \right\} = Z(f)$$

Consequently $Z(g) = Z(f)$.

If $T = N$ (the set of all positive integers), then the following theorem holds:

Theorem 4. If f is a primitive recursive function, and D_f and $N - D_f$ are recursively enumerable sets, then M_f is a partially computable function.

Proof. f^n is a partial recursive function (because it is the superposition). The quantification of a recursively enumerable relation by means of the existential quantifier as well as by means of the bounded universal quantifier is a recursively enumerable relation.

Then the sets $\left\{(t_1,t_2) : \bigvee_{n_o} \bigwedge_{n<n_o} f^n(t_1) \in D_f\right\}$,

$\left\{(t_1,t_2) : \bigvee_{n_o} f^{n_o}(t_1) \in N-D_f\right\}$ and $\left\{(t_1,t_2) : \bigvee_{n_o} f^{n_o}(t_1) = t_2\right\}$

are recursively enumerable sets. Therefore the set

$$\left\{(t_1,t_2) : \bigvee_{n_o} \bigwedge_{n<n_o} f^n(t_1) \in D_f \wedge f^{n_o}(t_1) \in D_f \wedge f^{n_o}(t_1) = t_2\right\}$$

is a recursively enumerable set (as intersection). Then the set $\left\{(t_1,t_2) : t_2 = M_f(t_1)\right\}$ is a recursively enumerable set. Consequently M_f is a computable function (because its diagram is a recursively enumerable set).

We are going to show now, that the machines exist.

Example. Let T be a set consisting of at least two elements and $t_o \in T$. A very simple machine is the function f: T→T such that $f(t) = t_o$ for all $t \in D f$, where $D f = T - t_o$.

The author wishes to express his thanks to Dr. Pawlak for his constant encouragement during the course of his research and for his careful review of the manuscript.

References

[1] MAL'CEV A.I. : Algoritmy i rekursivnye funkcii. Moskva 1965.
[2] PAWLAK Z.: Maszyny programowane. ALGORYTMY № 10,1969.

ALGORYTMY, V, N° 10 /1969/, 25-43

FORMALIZACJA POJĘCIA PROGRAMU

Henryka ROSSNER

Pracę złożono Oct. 10th,1968 r.

Praca ta jest próbą formalizacji pojęcia programu. Określono w niej pojęcia funkcji obliczalnych przez program i pojęcia programów równoważnych. Ponadto pokazano pewne własności programów, jak jednoznaczność obliczania i możliwości upraszczania grafu programu, czy też upraszczania instrukcji. Wreszcie określono składanie programów i pojęcia podprogramów oraz programów zawierających się, a także zależność między programami o grafach izomorficznych.

SPIS RZECZY

1. WSTĘP

Praca ta jest próbą formalizacji pojęcia programu. W szczególności starałam się zbadać niektóre zależności między programami równoważnymi /tzn. obliczającymi takie same funkcje/.

W pierwszym rozdziale podano definicje potrzebne do określenia pojęcia programu i funkcji obliczanej przez ten program /a więc definicja grafu przepływu, pojęcie realizacji i obliczenia programu, a w końcu pojęcie równoważności progra-

mu określone przez równość funkcji przez nie obliczanych/.

W rozdziale drugim pokazałam takie własności programów, jak jednoznaczność obliczenia, możliwość uproszczenia grafu programu /bez zmiany jego działania/, czy też uproszczenie instrukcji. Następny rozdział opisuje składanie programów /a więc superponowanie funkcji przez nie obliczanych/ oraz możliwość zamiany programów równoważnych przy wykonywaniu tej operacji. Określono tam także pojęcie zawierania się programów i pojęcie podprogramu. W końcu omówiono zależność między programami obliczającymi funkcje określone na różnych dziedzinach, przy różnych zbiorach funkcji elementarnych /zależnych jednak w pewien sposób od siebie/ i przy grafach izomorficznych.

2. Rozdział I
PODSTAWOWE DEFINICJE

D e f i n i c j a 1:

G r a f e m nazywamy uporządkowaną czwórkę $G=\langle Q, Q^+, Q_p, H\rangle$,

gdzie: $Q = \{q_0, q_1, \ldots, q_l\}$ - skończony zbiór wierzchołków,

$Q^+ \subset Q$ - zbiór wierzchołków końcowych,

$Q_p \subset Q$ - zbiór wierzchołków początkowych,

$H:/Q-Q^+/ \rightarrow 2^Q$ - funkcja następnego wierzchołka /$q \in Q^+$, to $H/q/$ - nieokreślona/.

D e f i n i c j a 2:

D r o g ą $D/q,q'/$ w Q z q do q' nazywamy ciąg $/q_{11}, q_{12}, \ldots, q_{1m}/$ taki, że $q=q_{11}$, $q'=q_{1m}$ i dla $i=1,2,\ldots,m-1$, $q_{1(i+1)} \in H/q_{11}/$.

D e f i n i c j a 3:

G r a f e m p r z e p ł y w u nazywamy graf spełniający warunki:

1^0 $\bar{\bar{Q}}_p = 1$ /oznaczamy $Q_p = \{q_0\}$,

2^0 $Q^+ \neq \emptyset$,

3^0 $\bigwedge_{q \in Q-\{q_0\}} D/q_0, q/$,

4^0 $\bigwedge_{q \in Q-Q^+} \bigvee_{q^+ \in Q^+} D/q, q^+/$.

D e f i n i c j a 4:

P r o g r a m e m nazywamy uporządkowaną piątkę

$S = \langle G, A^*, R, \varphi, h \rangle$, gdzie:

G - graf przepływu,

$A = \{a_1, \ldots, a_p\}$ - skończony alfabet,

$R = \{r_0, \ldots, r_n\}$ - skończony zbiór funkcji elementarnych,

$r_i \in R$, to $r_i : A^* \to A^*$; r_0 - wyróżniona funkcja /tożsamość/,

$\varphi: Q \to R$,

$h: A^* \times /Q - Q^+/ \to Q$ - funkcja przejścia, związana z funkcją następnego wierzchołka: $H/q/ = \{ q' \in Q: \bigvee_{\bar{a} \in A^*} q' = h/\bar{a}, q/ \}$ dla $q \in Q$.

Można też określić funkcję $g: A^* \times Q \to A^*$, gdzie $\bigwedge_{\bar{a} \in A^*} \bigwedge_{q \in Q} g/\bar{a}, q/ = [\varphi/q/]/\bar{a}/$, wtedy $\langle S = G, A^*, R, g, h \rangle$, które to przedstawienie jest odpowiedniejsze przy badaniu automatów. Przyjmijmy, dla prostoty, że jeśli $q \in Q^+$, to $\varphi/q/ = r_0$.

D e f i n i c j a 5:

P r o d u k c j ą programu S nazywamy funkcję $p=/g,h/$:

$$A^* \times Q \to A^* \times Q ,$$

$\bar{a} \in A^*, q \in Q-Q^+; p/\langle \bar{a}, q\rangle/ = \langle g/\bar{a}, q/, h/\bar{a}, q/\rangle = \langle [\varphi/q/]/\bar{a}/, h/\bar{a}, q/\rangle$.

D e f i n i c j a 6:

R e a l i z a c j ą P programu S nazwiemy każdy ciąg $/p_0, p_1, \ldots, p_s/$ taki, że $p_0 = \langle \bar{a}_0, q_0 \rangle$,

$$p_1 = p/\langle \bar{a}_0, q_0\rangle/ = p/p_0/ = \langle \bar{a}_1, q_{11}\rangle ,$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$p_{i+1} = p/\langle \bar{a}_1, q_{1i}\rangle/ = p/p_i/ = \langle \bar{a}_{i+1}, q_{1(i+1)}\rangle$$

dla $i = 0, \ldots, s-i$.

D e f i n i c j a 7:

Realizację $/p_0, p_1, \ldots, p_s/$ nazywamy o b l i c z e n i e m dla $\bar{a}_0$ w programie S, jeżeli $q_s = I_2^2 /p_s/ \in Q^+$. Wtedy $\bar{a}_s$ nazywamy w y n i k i e m obliczenia P i zapisujemy $P_1/\bar{a}_0, q_0/ = \bar{a}_s$, $/P/\bar{a}_0, q_0/ = p_s/$.

Obliczenie jest więc odwzorowaniem $P: A^* \times Q \to A^* \times Q$,

$$/P: A^* \times \{q_0\} \to A^* \times Q^+/.$$

D e f i n i c j a 8:

Mówimy, że słowa $\bar{a}, \bar{b} \in A^*$ s p e ł n i a j ą program S, jeśli istnieje takie obliczenie P_s, że $\bar{b} = P_1/\bar{a}, q_0/$.

D e f i n i c j a 9:

I n s t r u k c j ą programu, o etykiecie q nazywamy $J/q/ = \langle \varphi/q/, H/q/\rangle$.

$[\ J/q/\ ]\ /\bar{a}/ = \langle\ g/\bar{a},q/, h/\bar{a},q/\rangle = p/\langle\ \bar{a},q\rangle\ /$ - mówimy /przy ustalonym q/, że ta produkcja odpowiada danej instrukcji J/q/, która jak widać, wskazuje, jaką funkcję elementarną należy wykonać oraz etykiety instrukcji następnych. Widać stąd, że program można przedstawić jako skończony ciąg instrukcji:

$$J/q_0/ = \langle\ \varphi\ /q_0/,\ H/q_0/\rangle\ ,$$
$$J/q_1/ = \langle\ \varphi\ /q_1/,\ H/q_1/\rangle\ ,$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$J/q_1/ = \langle\ \varphi\ /q_1/,\ H/q_1/\rangle\ .$$

D e f i n i c j a 10:

F u n k c j a o b l i c z a l n a przez program S: $f_S: A^* \dashrightarrow A^*$ - funkcja taka, że dla każdego słowa $\bar{a} \in A^*$ istnieje takie obliczenie P_S, że $f_S\ /\bar{a}/ = P_1/\bar{a}, q_0/$. /Tzn. wartość funkcji jest wynikiem obliczenia, czyli jej argumenty i wartości tworzą zbiór par spełniających program S/.

D e f i n i c j a 11:

Programy $S = \langle G, A^*, R, g, h\rangle$ i $S_1 = \langle G_1, A^*, R_1, g_1, h_1\rangle$ nazywamy r ó w n o w a ż n y m i /$S \sim S_1$/, jeśli dla każdej pary $\bar{a}$, $\bar{b}$ spełniających program S w pewnym obliczeniu P_S istnieje obliczenie P_{S1}, w którym $\bar{a}$, $\bar{b}$ spełniają S_1 / i na odwrót /. Oznacza to, że dla każdego $\bar{a} \in A^*$: $f_S/\bar{a}/ = f_{S1}/\bar{a}/$.

3. Rozdział II
ELEMENTARNE WŁASNOŚCI PROGRAMÓW

Odwzorowania h i g są funkcjami, a więc jeśli $\bar{a}_1 = \bar{a}_2$ i $q_1 = q_2$, to $h/\bar{a}_1, q_1/ = h/\bar{a}_2, q_2/$ i $g/\bar{a}_1, q_1/ = g/\bar{a}, q_2/$.

Wynika stąd

L e m a t 1

Dla każdej pary słów $\bar{a}$, $\bar{b} \in A^*$, spełniających program S, istnieje dokładnie jedno obliczenie $/p_0, p_1, \ldots, p_s/$ takie, że $\bar{a} = \bar{a}_0$ i $\bar{b} = \bar{a}_s$.

D o w ó d:

Przypuśćmy, że istnieją dwa różne obliczenia: $/p_0, \ldots, p_s/$ i $/p'_0, \ldots, p'_r/$ i w obu wynikiem jest $\bar{b}$, a słowem początkowym $\bar{a}$. Z definicji obliczenia musi być $p_0 = p'_0 = \langle \bar{a}, q_0 \rangle$. Niech i będzie najmniejszą liczbą taką, że $p_{i+1} \neq p'_{i+1}$.

Mamy więc:

$$p_i = \langle \bar{a}_i, q_{1i} \rangle = \langle \bar{a}'_i, q''_{1i} \rangle = p'_i$$

Ale w takim razie

$$q_{1(i+1)} = h/\bar{a}_i, q_{1i}/ = h/\bar{a}'_i, q''_{1i}/ = q'_{1(i+1)} ,$$

$$\bar{a}_{i+1} = g/\bar{a}_i, q_{1i}/ = g/\bar{a}'_i, q''_{1i}/ = \bar{a}'_{i+1} ,$$

czyli $p_{i+1} = p'_{i+1}$

doszliśmy więc do sprzeczności. Cbdo.

O k r e ś l e n i e:

Dane są programy $S_A = \langle G_A, A^*, R_A, \varphi_A, h_A \rangle$ oraz $S_B = \langle G_B, B^*, R_B, \varphi_B, h_B \rangle$. Odwzorowaniem S_A w S_B nazywamy układ odwzorowań /G,F,E/ takich, że G: $Q_A \longrightarrow Q_B$,

F: $A^* \longrightarrow B^*$,

E: $R_A \longrightarrow R_B$.

T w i e r d z e n i e 1

Dla każdego programu S istnieje równoważny mu program S', którego graf przepływu spełnia warunek: $\bigwedge_{q \in Q'} \overline{\overline{H'/q/}} \leq 2$.

D o w ó d:

Załóżmy na początek, że istnieje jeden taki wierzchołek $\hat{q}$, że $H/\hat{q}/ = \{q_{t1},\ldots,q_{tr}\}$ i $r > 2$. Instrukcję $J/\hat{q}/ = \langle \varphi/\hat{q}/, H/\hat{q}/ \rangle$ programu S zastępujemy ciągiem r-1 instrukcji następującej postaci:

$$J_1/q'_{to}/ = \langle \varphi'/q'_{to}/, \{\bar{q}_{t1}, q'_{t1}\}\rangle ,$$

$$J_2/q'_{t1}/ = \langle r_o , \{\bar{q}_{t2}, q'_{t2}\}\rangle ,$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$J_{r-2}/q'_{t(r-3)}/ = \langle r_o, \{\bar{q}_{t(r-1)}, \bar{q}_{tr}\}\rangle$$

Określamy przekształcenie S na S' takie, że

$$F/A^*/ = A^*, \qquad E/R/ = R' = R,$$

$$G/Q/ = \bar{Q} \cup \{q'_{to},\ldots,q'_{t(r-2)}\} = Q', \text{ gdzie } \bar{Q} = \{\bar{q} : q \in Q\},$$

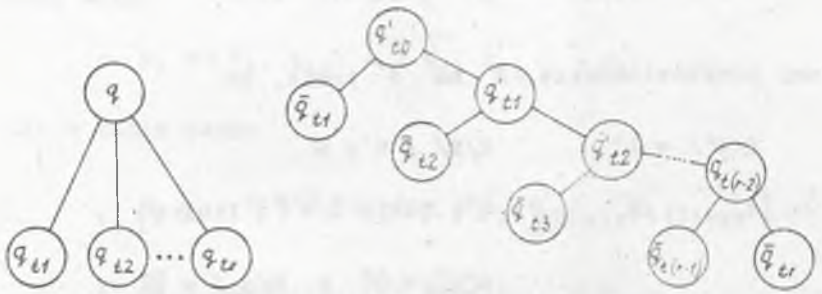
$$G/Q^+/ = \bar{Q}^+ \text{ i } G/q_o/ = \bar{q}_o .$$

$$G/q/ = \begin{cases} /q'_{to},\ldots,q'_{t(r-2)}/ & \text{jeśli } q = \hat{q}, \\ \bar{q} & \text{jeśli } q \neq \hat{q}. \end{cases}$$

$$\varphi'/q'/ = \begin{cases} \varphi/q/ & \text{jeśli } q' = G/q/ \text{ i } H/q/ \leq 2, \\ \varphi/\hat{q}/ & \text{jeśli } q' = q'_{to}, \\ r_o & \text{jeśli } q' = q'_{ti} \quad i = 1,\ldots,r-2, \end{cases}$$

$$h'/\bar{a},q'/=\begin{cases} q'_{to} \text{ jeśli istn. } q \text{ takie, że } q'=G/q/ \text{ i } h/\bar{a},q/=\hat{q} \\ \bar{q}_{ti} \text{ jeśli } q'=q'_{t(i-1)} \text{ i } h/\bar{a},\hat{q}/=q_{ti} \\ q'_{ti} \text{ jeśli } q'=q'_{t(i-1)} \text{ i } h/\bar{a},\hat{q}/=q_{tj} \text{ i } i<j\leq r \\ \bar{q}_{tr} \text{ jeśli } q'=q'_{t(r-2)} \text{ i } h/\bar{a},\hat{q}/=q_{tr} \\ G/h/\bar{a},q// \text{ dla } q'=G/q/ \text{ i } H/q/ \leq 2 \end{cases} \quad i=1,\ldots,r-1$$

W ten sposób $H/q'/$ też jest określone. Odpowiada to przekształceniu fragmentu grafu Q z wierzchołkiem $\hat{q}$ i jego następnikami $H/\hat{q}/$ w następujący sposób:



Pokażemy teraz, że dowolna para słów $\bar{a}$, $\bar{b}$ spełnia program S = spełnia program S'.

1° $\bar{a}$, $\bar{b}$ spełniają S w obliczeniu $p_0 = \langle \bar{a}, q_0 \rangle$ $\bar{a} = \bar{a}_0$

.....

$$p_i = \langle [\varphi /q_{1(i-1)}/] /\bar{a}_{i-1}/, h/\bar{a}_{i-1}, q_{1(i-1)}/ = \langle \bar{a}_i, \bar{q}_{1i} \rangle,$$

$$p_{i+1} = \langle [\varphi /q_{1i}/] /\bar{a}_i/, h/\bar{a}_i, q_{1i}/ \rangle = \langle \bar{a}_{i+1}, q_{1(i+1)} \rangle,$$

.....

$$p_s = \langle \bar{a}_s, q_s \rangle = \langle \bar{b}, q_s \rangle \quad \text{i} \quad q_s \in Q^+$$

Niech dla $q_{1i} \in Q$: $H/q_{1i}/ > 2$ /tzn. $q_{1i} = \hat{q}$/. Zastępujemy produkcję p_{i+1} ciągiem produkcji /zgodnych z nowo określonymi instrukcjami/. Niech przy tym $H/q_{1i}/ = \{q_{1i1},\dots,q_{1ir}\} = \{q_{t1},\dots,q_{tr}\}$ oraz $h/\bar{a}_i, q_{1i}/ = q_{1(i+1)} = q_{1ik} = q_{tk}$, gdzie $i \leqslant k \leqslant r$,

$$p_i = \langle \bar{a}_i, q_{1i} \rangle \ ,$$

$$p'_{io} = \langle [\varphi /q_{1i}/] /\bar{a}_i/, q'_{1i1} \rangle = \langle \bar{a}_{i+1}, q'_{1i1} \rangle \ ,$$

$$p'_{i1} = \langle \bar{a}_{i+1}, q'_{1i2} \rangle \ ,$$

$$\dots\dots\dots\dots\dots\dots$$

$$p'_{ik-1} = \langle \bar{a}_{i+1}, q'_{1ik} \rangle = \langle \bar{a}_{i+1}, q_{1(i+1)} \rangle \ ,$$

$$p_{i+2} = \langle [\varphi /q_{1(i+1)}/] /\bar{a}_{i+1}/, h/\bar{a}_{i+1}, q_{1(i+1)}/ \rangle \ .$$

Jak widać, nowe produkcje nie zmieniły słowa $\bar{a}_{i+1}$, a więc i tu wynik będzie $\bar{b}$, bo dalsze produkcje są nie zmienione. Z określenia S' widać, że otrzymaliśmy ciąg będący obliczeniem w S'.

2^o niech $/p'_o, p'_1,\dots,p'_s/$ będzie obliczeniem $\bar{a}$ w S' i $\bar{a}'_s = \bar{b}$. Niech i będzie najmniejszą liczbą taką, że $q'_{1i} \in Q' - \bar{Q}$ i $q'_{1i} \neq q'_{to}$. A le w takim razie $\varphi'/q'_{1(i+1)}/ = r_o$, bo tak poprzednio określiliśmy.
Niech $q'_{1(i+1)},\dots,q'_{1(i+k)} \in Q' - \bar{Q}$ i $\varphi'/q'_{1(i+j)}/ = r_o$, $j=1,\dots,k$, $1 \leqslant k \leqslant s$. Grupę produkcji $p'_i, p'_{i+1},\dots,p_{i+k+i}$ możemy więc zastąpić produkcją postaci $p_i = \langle [\varphi'/q'_{1(i-1)}/] /\bar{a}_{i-1}/, q_{1(i+k+1)} \rangle$, która jest produkcją w programie S. Ponadto $\bar{a}_i = \bar{a}_{i+k}$ - bo wyeliminowaliśmy produkcję z funkcjami tożsamościowymi /modyfikując jednocześnie pozostałe tak, by "przeskakiwały" te usunięte/. Pozostałe produkcje są takie same, a więc i tu $\bar{b}$ jest wynikiem obliczenia.

Założyliśmy początkowo, że tylko dla jednego wierzchołka $\hat{q} \in Q$: $H/\hat{q}/ > 2$. Widać teraz, że postępowanie takie możemy rozszerzyć - jest ono bowiem skończone /bo Q - zbiór skończony i w każdym obliczeniu ciąg produkcji jest skończony/.

Cbdo.

Będziemy teraz mogli instrukcje programu przedstawić w postaci:

$$J/q/ = \langle \varphi\ /q/, \{q', q''\}\rangle \qquad \text{gdzie} \quad \{q', q''\} = H/q/.$$

T w i e r d z e n i e 2

Dla każdego programu S określonego przez ciąg instrukcji

$$J/q_0/ = \langle \varphi\ /q_0/, \{q_0', q_0''\}\rangle ,$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$J/q_1/ = \langle \varphi\ /q_1/, \{q_1', q_1''\}\rangle ,$$

istnieje równoważny mu program $\bar{S}$, którego instrukcje spełniają warunek:

$$\bar{\varphi}\ /\bar{q}_i/ \neq r_0, \text{ to } \bar{q}_i' = \bar{q}_i'' = \bar{q}_{i+1} \qquad i = 0,\ldots,l-1.$$

D o w ó d: /uproszczony o tyle, że w przeciętnym programie nie koniecznie trzeba modyfikować wszystkie instrukcje/.

Każdą instrukcję programu S rozbijamy na dwie w $\bar{S}$:

$$J/q_i/ = \langle \varphi\ /q_i/, \{q_i', q_i''\}\rangle \cdots \rangle \begin{cases} J/\bar{q}_i/ = \langle \bar{\varphi}\ /\bar{q}_i/, \{\bar{q}_{i+1}, \bar{q}_{i+1}\}\rangle, \\ J/\bar{q}_{i+1}/ = \langle r_0\ , \{\bar{q}_{i+1}', \bar{q}_{i+1}''\}\rangle. \end{cases}$$

Wiąże się to z odwzorowaniem programu $S = \langle G, A^*, R, \varphi, h\rangle$ na program $\bar{S} = \langle \bar{G}, A^*, R, \bar{\varphi}, \bar{h}\rangle$ w ten sposób, że dla

$Q = \{q_0, \ldots, q_l\}$ mamy: $G/Q/ = \bar{Q} = \{\bar{q}_0, \bar{q}_1, \ldots, \bar{q}_{2l+1}\}$, gdzie

$G/q_0/ = \bar{q}_0$ i $G/Q^+/ = \bar{Q}^+$.

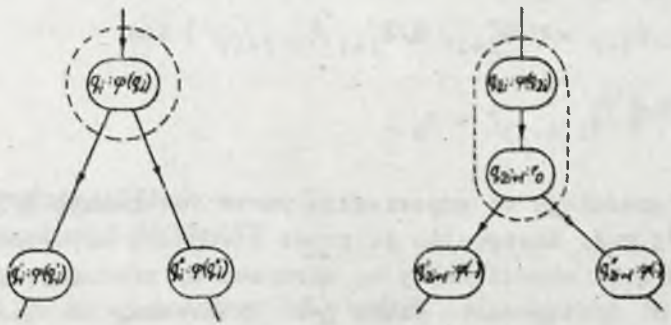
$$G/q_i/ = \{\bar{q}_{2i}, \bar{q}_{2i+1}\} \qquad i = 0, \ldots, l ,$$

$$\bar{\varphi}/\bar{q}_j/ = \begin{cases} \varphi/q_{j/2}/ & \text{dla } j = \text{parzystych,} \\ r_0 & \text{dla } j - \text{nieparzystych.} \end{cases}$$

$$\bar{h}/\bar{a}, \bar{q}_i/ = \begin{cases} \bar{q}_{i+1} & \text{jeśli } i \text{ parzyste,} \\ G_1/h/\ /\bar{a}, q_{(i-1)/2}/ & \text{jeśli } i \text{ nieparzyste,} \end{cases}$$

$i, j = 0, \ldots 2\,l+1$

Odpowiada to określonemu na rysunku przekształceniu grafu programu:



$\bar{a}, \bar{b}$ spełniają program $S = \bar{S}$ spełniają program $\bar{S}$.

1° Niech $/p_0, p_1, \ldots, p_s/$ będzie obliczeniem $\bar{a}$ w S i $\bar{b} = \bar{a}_s$. Ciąg $/\bar{p}_0, \bar{p}_1, \ldots, \bar{p}_{2s+1}/$ powstały z $/p_0, \ldots, p_s/$ przez zastąpienie każdej produkcji p_j $/i \leqslant j \leqslant s/$ odpowiadającej pewnej

instrukcji $J/q_i/$ $/0 \leq i \leq l/$ przez parę produkcji $\bar{p}_{2j}, \bar{p}_{2j+1}$ odpowiadających instrukcjom $J/\bar{q}_i/$, $J/\bar{q}_{i+1}/$ określonym j.w., jest obliczeniem $\bar{a}$ w S.

Niech $j=1$; $\bar{a}_1 = [\varphi\ /q_0/\ ]\ /\bar{a}_0/ = [\varphi\ /q_0/\ ]\ /\bar{a}/ = r_0/[\varphi/\bar{q}_0/]/\bar{a}/ = \bar{a}'_2$.

Niech $\bigwedge_{1 \leq i \leq j} \bar{a}_j = \bar{a}'_{2j}$; wtedy $\bar{a}_{j+1} = [\varphi\ /q_{1j}/\ ]/\bar{a}_j/ =$

$= r_0[\varphi\ /\bar{q}_{1(2j)}/\ ]\ /\bar{a}'_{2j}// = \bar{a}'_{2j+2}$.

A więc także $\bar{a}_s = \bar{a}'_{2s} = \bar{b}$ - czyli $\bar{a}$, $\bar{b}$ spełniają S .

2^o Niech $/\bar{p}_0, \bar{p}_1, \ldots, \bar{p}_r/$ będzie obliczeniem $\bar{a}$ w S i niech $\bar{a}_r = \bar{b}$. Niech i $/0 \leq i \leq r/$ będzie najmniejszą liczbą taką, że jednocześnie

$$\bar{p}_{i+1} = \langle [\bar{\varphi}\ /\bar{q}_{1i}/\ ]\ /\bar{a}'_i/,\ \bar{q}_{1(i+1)} \rangle =$$

$$= \langle\ \bar{a}'_{i+1},\ \bar{q}_{1(i+1)} \rangle \quad ,$$

i

$$\bar{p}_{i+2} = \langle\ \bar{a}'_{i+1},\ \bar{h}/\bar{a}'_{i+1}, \bar{q}_{1(i+1)}/ \rangle \text{ tzn.}$$

$$\bar{\varphi}\ /\bar{q}_{1(i+1)}/ = r_0 \ .$$

Jak widać, produkcje te odpowiadają parze instrukcji $J/\bar{q}_{1i}/$, $J/\bar{q}_{1(i+1)}//$j.w./. Zastępując je przez produkcję odpowiadającą produkcji $J/q_{1i}/$ określonej j.w. otrzymujemy produkcję w programie S. Postępowanie takie jest skończone, bo skończone jest obliczenie; ponieważ zachowane zostają wszystkie operacje na słowach i kolejność ich wykonywania, więc wynik się nie zmieni.

Cbdo.

W n i o s e k

Przez "dopisanie" między dwoma dowolnymi wierzchołkami q_{1i}, $q_{1(i+1)}$ grafu programu łuku /tzn. drogi bez cykli/ o wierzchołkach $q_{1i1},\ldots,\bar{q}_{1ik}$, gdzie rozszerzona funkcja

$$\bar{\varphi}/q/ = \begin{cases} \varphi\ /q/ & \text{gdy } q \in Q\ , \\ r_o & \text{gdy } q = \bar{q}_{1ij} \quad j = i,\ldots,k\ . \end{cases}$$

$$\bar{h}/\bar{a},q/ = \begin{cases} h/\bar{a},q/ & \text{gdy } q \in Q - \{q_1\}\ , \\ \bar{q}_{1i1} & \text{gdy } q = q_{1i}\ , \\ \bar{q}_{1i(j+1)} & \text{gdy } q = \bar{q}_{1ij} \quad j = 1,\ldots,k-1 \\ q_{1(i+1)} & \text{gdy } q = \bar{q}_{1ik}\ , \end{cases}$$

otrzymujemy program równoważny.
Odpowiada to w tablicy instrukcji dopisaniu w miejsce $J/q_1/ = \langle \varphi\ /q_1/, H/q_1/ \rangle$ ciągu instrukcji: $J/q_1/ = \langle \varphi\ /q_1/, \bar{q}_{11} \rangle$,

$J/\bar{q}_{11}/ = \langle r_o, \bar{q}_{12} \rangle$,

.....

$J/\bar{q}_{1k}/ = \langle r_o, H/q_1/ \rangle$.

Rozdział III
SKŁADANIE PROGRAMÓW

Dane są programy $S_1 = \langle G_1, A^*, R_1, \varphi_1, h_1 \rangle$,

$S_2 = \langle G_2, A^*, R_2, \varphi_2, h_2 \rangle$.

D e f i n i c j a 13:

Z ł o ż e n i e m $S_2 * S_1$ programów S_1, S_2 nazywamy program S określony następująco:

$$S = \langle G, A^*, R, \varphi, h \rangle \text{ , gdzie: } R = R_1 \cup R_2$$

$$Q = Q_1 \cup Q_2 - Q_1^+$$

przy czym $q_0 = q_{01}$ i $Q^+ = Q_2^+$, zaś funkcje φ i h będą

$$\varphi/q/ = \begin{cases} \varphi_1/q/ & \text{gdy } q \in Q_1 \text{ ,} \\ \varphi_2/q/ & \text{gdy } q \in Q_2 \text{ ,} \end{cases}$$

$$\text{oraz } h/\bar{a},q/ = \begin{cases} h_1/\bar{a},q/ & \text{gdy } q \in Q_1 - Q_1^+ \\ q_{02} & \text{gdy } q \in Q_1^+ \\ h_2/\bar{a},q/ & \text{gdy } q \in Q_2 \end{cases} \quad \text{dla każdego } \bar{a} \in A^*$$

W ten sposób określony graf G jest grafem przepływu - posiada dokładnie jeden wierzchołek początkowy, niepusty zbiór wierzchołków końcowych oraz dla dowolnego $q \in Q - \{q_0\}$ istnieje droga

$$D/q_0,q/ = \begin{cases} D_1/q_{01},q_1^+/, D_2/q_{02},q/ & \text{dla } q \in Q_2 \text{ ,} \\ D_1/q_{01}, q/ & \text{dla } q \in Q_1 \end{cases}$$

i dla każdego $q \in Q - Q^+$ istnieje $q^+ \in Q^+$ taki, że istnieje droga

$$D/q,q^+/ = \begin{cases} D_1/q,q_1^+/, D_2/q_{02},q_2^+/ & \text{dla } q \in Q_1 \text{ ,} \\ D_2/q,q_2^+/ & \text{dla } q \in Q_2 \text{ .} \end{cases}$$

Widać stąd, że działanie programu $S = S_2 * S_1$ odpowiada działaniu jednego programu po drugim, gdzie dla każdego obliczenia w S_1, wynik jest zarazem słowem początkowym dla obliczenia S_2. Złożenie programów określa tablica instrukcji składająca się z kolejno po sobie wypisanych instrukcji

programu S_1 /z odpowiednio zmodyfikowanymi następnikami wierzchołków końcowych/ a potem instrukcji S_2 .

L e m a t 2

Składanie programów jest łączne, ale nie przemienne /analogicznie, jak superponowanie funkcji/.

L e m a t 3

Jeśli $f_1: A^* \longrightarrow A^*$ i $f_2 : A^* \longrightarrow A^*$ są funkcjami obliczalnymi, odpowiednio, przez programy S_1 i S_2, to funkcja f_s obliczalna przez $S = S_2 * S_1$ jest superpozycją tamtych funkcji: $f_s/\bar{a}/ = f_2/f_1\ /\bar{a}//$ /jeśli $\bar{a}$ należy do dziedziny f_1 a $f_1/\bar{a}/$ należy do dziedziny f_2/.

T w i e r d z e n i e 3

Dane są programy S_1, S_1', S_2, S_2' takie, że $S_1 \sim S_1'$ i $S_2 \sim S_2'$. Składanie programów zachowuje równoważność: $S_2 * S_1 \sim S_2' * S_1'$.

D o w ó d

Oznaczamy $S = S_2 * S_1$ i $S' = S_2' * S_1'$.
Pokazać należy, że $\bar{a}$, $\bar{b} \in A^*$ spełniają $S_2 * S_1$ = spełniają $S_2' * S_1'$.
Niech $/p_0,\ldots,p_s/$ będzie obliczeniem $\bar{a}$ /z wynikiem $\bar{b}$/ w S. Zgodnie z określeniem drogi $D/q_0,q^+/$ w grafie złożenia programów istnieje taka liczba $0 \leqslant r \leqslant s$, że $p_0,\ldots,p_r$ jest odpowiednikiem obliczenia $/p_0^1,\ldots,p_r^1/$ słowa $\bar{a}$ w programie S_1 , gdzie wynikiem jest $\bar{a}_r$ i $H/q_{1r}/ = q_{02}$ i poprzednie produkcje są zależne jedynie od φ_1, Q_1, h_1. $/\bar{a}$, $\bar{a}_r/$ spełniają więc S_1, a tym samym - spełniają S_1' w pewnym obliczeniu $p_0',\ldots,p_k'$. Obliczenie $p_{r+1},\ldots,p_s$ jest tu więc obliczeniem $\bar{a}_r$ w programie S z wynikiem $\bar{b}$, istnieje więc odpowiednie obliczenie w S_2' dla słowa $\bar{a}$: $p_0'',\ldots,p_m''$ /z tymże wynikiem/.
Ostatecznie więc $\bar{a}$, $\bar{b}$ spełniają program S' w obliczeniu

$/p_0', \ldots, p_k'^*, p_0'', \ldots, p_m''/$ /gdzie $p_k'^*$ różni się od p_k' tym, że $h'^* /\bar{a}_k, q_{1k}/ = q'_{o2}$, a $h/\bar{a}_k, q_{1k}/$ - nieokreślony/.

Cbdo.

D e f i n i c j a 14

Mówimy, że program $\hat{S} = \langle \hat{G}, A^*, \hat{R}, \hat{\varphi}, \hat{h} \rangle$ jest z a w a r t y w programie $S = \langle G, A^*, R, \varphi, h \rangle$, gdy spełnione są warunki:

1° $\hat{Q} \subseteq Q$ /$\hat{Q}$ jest pod grafem grafu Q/,

2° $\bigwedge_{q \in Q} q \in \hat{Q} \Rightarrow \hat{\varphi}/q/ = \varphi/q/$,

3° $\bigwedge_{\bar{a} \in A^*} \bigwedge_{q \in \hat{Q}} h/\bar{a}, q/ = \hat{h}/\bar{a}, q/$ i $\hat{h}$ określone $\equiv$ gdy h określone, a więc $H/q/$ też określone,

4° $\hat{Q}$ jest grafem przepływu.

5° w $\hat{Q}$ nie istnieje wierzchołek $\hat{q}$ taki, że $\hat{q} \in H/q/$ i $q \in Q - \hat{Q}$ /odpowiada to pewnym ograniczeniom w niektórych istniejących językach programowania/.

Z definicji tej wynika, że $\hat{Q}^+ \subseteq Q^+$ i jeśli $q_0 \in \hat{Q}$, to $\hat{Q} = Q$.

T w i e r d z e n i e 4

Jeżeli program S_2 powstaje z programu S_1 zawierającego program $\hat{S}_1$ przez zastąpienie $\hat{S}_1$ programem $\hat{S}_2$ takim, że $\hat{S}_1 \sim \hat{S}_2$ i w taki sposób, że $\hat{q}_{o2} = \hat{q}_{o1}$, to $S_2 \sim S_1$.

D o w ó d

Niech $\bar{a}$, $\bar{b}$ spełniają S_1. Istnieje więc w S_1 odpowiednie obliczenie $/p_0, p_1, \ldots, p_s/$. Rozpatrzymy dwie możliwości:

1° $q_{1s} \in (Q^+ - \hat{Q}_1^+)$ - i to oznacza, że w drodze $/q_0, q_{11}, \ldots, q_{1s}/$ żaden z wierzchołków nie należy do $\hat{Q}_1$; wtedy $\bar{a}$, $\bar{b}$ spełniają S_2 przez to samo obliczenie $/p_0, p_1, \ldots, p_s/$.

2° $q_{1s} \in \hat{Q}_1^+$ - a więc istnieje $0 \leq i \leq s$, że $q_{11} = \hat{q}_{o1}$ - i wtedy $/p_i, p_{i+1}, \ldots, p_s/$ jest w $\hat{S}_1$ pewnym obliczeniem o wyniku $\bar{b}$

dla słowa $\bar{a}_i$. Ale w takim razie istnieje w $\hat{S}_2$ obliczenie $p_0', p_1', \ldots, p_k'$ takie, że $/\bar{a}_i, \bar{b}/$ spełniają $\hat{S}_2$. Obliczenie $/p_1, \ldots, p_{i-1}, p_0', \ldots, p_k'/$ jest obliczeniem dla $\bar{a}$ w S_2 z wynikiem $\bar{b}$.

Cbdo.

Jak widać, jeśli $S = S_2 * S_1$, to S_2 jest zawarty w S.

D e f i n i c j a 15

P o d p r o g r a m e m programu S nazywamy taki program $\hat{S}$, że istnieją programy $S_1, \ldots, S_m$, że $S = S_m * S_{m-1} * \ldots * S_1$ i istnieje $1 \leqslant i \leqslant m$ takie, że program $\hat{S}$ jest zawarty w programie S_i.

W n i o s e k:

Przez zmianę podprogramów równoważnych otrzymujemy programy równoważne.

Rozdział IV
ODWZOROWANIE PROGRAMÓW

/Patrz określenie w rozdziale II/

T w i e r d z e n i e 5

Jeśli 1° G jest funkcją wzajemnie jednoznaczną i taką, że $G/q_{0A}/ = q_{0B}/$ oraz $G/Q_A^+/ = Q_B^+$, zaś funkcje przejścia są związane: $G/h_A/\bar{a}, q_A// = h_B/F/\bar{a}/, G/q_A//$ /to znaczy grafy G_A i G_B są izomorficzne/

2° funkcja E spełnia warunek:

$$\left[\bigwedge_{r \in R_A} \bigwedge_{\bar{a} \in A^*} \left[Er_A\right] /F\bar{a}/ = F/r_A/\bar{a}//\right] \text{i} \left[\bigwedge_{q \in Q_A} \bigwedge_{\bar{a} \in A^*} \left[E\varphi_A/q_A/\right] /F\bar{a}/ =\right.$$

$$\left. = F/\left[\varphi_A/q_A/\right] /\bar{a}// = \left[q_B/G/q_A//\right] /F\bar{a}/\right] ,$$

to jeśli para słów /$\bar{a}$, $\bar{b}$/ spełnia program S_A, to /F/$\bar{a}$/ , F/$\bar{b}$// - spełnia program S_B.

D o w ó d

Niech $p_0,\dots,p_s$ będzie obliczeniem dla /$\bar{a}$, $\bar{b}$/ w S_A ,

niech $p_0 = \langle \bar{a}, q_{0A} \rangle$,

niech $0 \leqslant i \leqslant s$: $p_i = \langle \bar{a}_i , q_{1iA} \rangle$.

Wtedy obliczenie /$\bar{p}_0,\bar{p}_1,\dots,\bar{p}_s$/ takie, że $\bar{p}_i = \langle F/\bar{a}_i/, G/q_{1iA}/\rangle$ jest obliczeniem dla F/$\bar{a}$/, F/$\bar{b}$/ w S_B.

Mamy bowiem: $\bar{p}_0 = \langle F/\bar{a}_0/, G/q_{0A}/\rangle = \langle F/\bar{a}_0/, q_{0B}\rangle$,

$\bar{p}_i = \langle F/\bar{a}_i/, G/q_{1iA}/\rangle$, to następna produkcja

$$\bar{p}_{i+1} = \langle \left[\varphi_B/G/q_{1iA}//\right] /F/\bar{a}_i//, h_B/F/\bar{a}_i/, G/q_{1iA}//\rangle =$$

$$= \langle F/\left[\varphi_A/q_{1iA}/\right] /\bar{a}_i//, G/h_A/\bar{a}_i, q_{1iA}//\rangle =$$

$$= \langle F/\bar{a}_{i+1}/, G/q_{1(i+1)A}/\rangle \ .$$

A więc także dla s: F/$\bar{a}_s$/ = F/$\bar{b}$/ jest wynikiem tego obliczenia.

Cbdo.

Stąd widać, że dla funkcji obliczalnych przez programy S_A, S_B zachodzi: $F(f_A/\bar{a}/) = f_B/F\bar{a}/$ dla każdego $\bar{a}$ z dziedziny f_A. Relacja określona przez trójkę odwzorowań /G,F,E/ między programami S_A a S_B jest zwrotna i przechodnia.

ZAKOŃCZENIE

Należałoby zapewne dokładniej i szerzej zbadać zwłaszcza opisane na końcu zależności między programami obliczającymi funkcje określone na różnych dziedzinach /z różnymi alfabetami/, przy grafach programów niekoniecznie izomorficznych/lecz zależnych od siebie w jakiś bardziej ogólny sposób/, co pociągałoby za sobą także uogólnienie związków między zbiorami funkcji elementarnych.

Nie jest wykluczone, że własności te mogłyby mieć pewne znaczenie przy translacjach i interpretacjach języków programowych, przy zmianach poziomów języków - przy czym zawsze należałoby uwzględniać niedopuszczalność pewnych struktur w niektórych językach, a więc konieczność obrania innej drogi tłumaczenia.

Literatura

[1] MALCEV A.: Algoritmy i rekursivnyje funkcii. Moskva 1965.
[2] PAWLAK Z.: Organizacja maszyn cyfrowych./Skrypt do wykładów/.

FORMALISATION OF THE NOTION OF THE PROGRAMM

Summary

The aim of this paper is to give mathematical model for programs. Equivalence of programs is considered and some elementary properties of program are showen. Further composition of programs and the notion of subprogram are introduced and some relations between programs are defined and investigated.

© Instytut Maszyn Matematycznych
Warszawa, ul. Krzywickiego 34.

Z.P.

ALGORYTMY, V, N° 10 /1969/, 45-50

ON THE SIMPLE FORM OF THE DEDUCTION SYSTEMS

by Antoni MAZURKIEWICZ

Received Oct. 18th,1968 r.

The aim of the paper is to show how a given deduction system can be reduced to a " simple " form, suitable for automatic processing. It may be of some interest to compare this simple form with the well known results in the domain of recursive function,namely with Kleene's normal form of recursively enumerable predicates [1]. The results of this paper may be applied to any first order theory.

1. By the d e d u c t i o n s y s t e m /or simply: system/ S over an alphabet V we mean a quadruple (V, W, A, R) where V is a finite set of symbols, $W \subset V^*$ is a set of well formed f o r m u l a s in S /shortly: wff's/, $A \subset W$ is a set of a x i o m s in S, and R is a set of r u l e s in S i.e. word predicates [2] over $W \times W \times \ldots \times W$.

A finite sequence of wff's:

$$w_0, w_1, \ldots, w_n$$

is called a p r o o f in S if for each i, $0 \leqslant i \leqslant n$, either w_i is an axiom in S or there exists a rule r in S and k-tuple of indices $(i_1, i_2, \ldots, i_k)$, $0 \leqslant i_j \leqslant i$, such that $r(w_{i_1}, w_{i_2}, \ldots, w_{i_k}, w_i)$ holds.

The wff w is to be a t h e o r e m of S (briefly : $\vdash_S w$) if it is the last wff of some proof in S.

The system S is said to be r e c u r s i v e if sets W, A, R are recursive. It is known that if S is recursive then $T(S)$ (the set of all theorems of S) is recursively enumerable [2] (but not necessarily recursive).

2. The system S is said to be s i m p l e if each rule in S is a binary word predicate. It is easy to show that for each w being a theorem in any simple system S there exists a proof with the following properties:

1. $w_0 \in A$,
2. for each positive $i \leqslant n$ there is an r in R such that $r(w_{i-1}, w_i)$,
3. $w_n = w$.

The system S is called a c o n j u n c t i o n system if there exists an element $\sigma \in V$, called a c o n j u n c t i o n s y m b o l, such that:

$$\vdash_S w_1 \text{ and } \vdash_S w_2 \text{ iff } \vdash_S w_1 \sigma w_2 . \quad /1/$$

3. Now we proceed to demonstrate the following:

T1. For each conjunction system S over V there exists a simple system S' over V with the same set of wff's as S such that:

i. If $\vdash_{S'} w$, then $\vdash_{S} w$

ii. If $\vdash_{S} w$, then either $\vdash_{S'} w$ or there exists a wff v such that $\vdash_{S'} v$, and $\vdash_{S'} v \sigma w$, where σ is the conjunction symbol of S.

iii. If S is recursive then S' and $T(S')$ are recursive.

D e m. Let $S' = (V, W', A', R')$ where $V = V, W' = W$. If $(a_0, a_1, \ldots, a_n, \ldots)$ is an arbitrary ordering /lexicographic, for example/ of axioms in S, then $A' = (a'_0, a'_1, \ldots, a'_n, \ldots)$, where

$$a'_0 = a_0 , \qquad /2/$$
$$a'_n = a'_{n-1} \sigma a_n , \quad n = 1,2,\ldots$$

Hence, if A is recursive, then so is A'. To each rule $r \in R$ corresponds a rule $r' \in R'$ such that:

$r'(v, w)$ iff there exists a sequence of wff's $w_1, w_2, \ldots, w_n$ such that
$v = w_1 \sigma w_2 \sigma \ldots \sigma w_{n-1}$,
$w = v \sigma w_n$, and
$r(w_1, w_2, \ldots, w_n)$ holds. /3/

Besides the above rules R' contains the "permutation" rules:

$r''(v, w)$ iff $v = p \sigma q$ and $w = q \sigma p$ for some p, q, /4/

and the "extension" rules:

$r'''(v, w)$ iff there exist p, q, s such that $v = p \sigma q$, $w = p \sigma s$ and either /5/ $r'(q, s)$ or $r''(q, s)$.

Clearly, if W and R are recursive, then R' is recursive. It follows that if S is recursive, then so is S'. Obviously S' is simple.

Now, let $\vdash_{S'} w$. It is easy to see that if w is an axiom of S', then it is a theorem of S (The system S is a conjunction system). By induction, assume that $\vdash_S v$ and let w be such that either $r'(v,w)$, or $r''(v, w)$, or $r'''(v, w)$. If $r'(v, w)$, then there exist w', and $w_1, w_2, \ldots, w_n$ such that $v = w_1 \sigma w_2 \sigma \ldots \sigma w_n$, $w' = v \sigma w'$, and $r(w_1, w_2, \ldots, w_n)$ holds for some r in R. Since $\vdash_S v$ then $\vdash_S w_1$ because S is a conjunction system. Thus $\vdash_S w'$ from the definition of $\vdash_S$, and $\vdash_S v \sigma w'$ because S is a conjunction system. That is, $\vdash_S w$. If $r''(v, w)$, then $v = p \sigma q$, and $w = q \sigma p$ for some p, q. That is, $\vdash_S w$. Similarly if $r'''(v, w)$, and $\vdash_S v$, then $\vdash_S w$. Hence in all cases we have $\vdash_S w$ which proves (i).

Conversely, if $\vdash_S w$ then there exists a proof

$$w_0, w_1, \ldots, w_n$$

in S, $\vdash_S w_i$ for each positive $i \leqslant n$, and from /1/

$$\vdash_S w_0 \sigma w_1 \sigma \ldots \sigma w_n .$$

If $n = 0$, then w is an axiom of S, say, a_i. Then $a_i' = a_{i-1}' \sigma a_i$ /or $a_i' = a_i$ if $i = 0$/ is the axiom of S',

hence a theorem of S' and a_i' fulfills /ii/. By induction, assume $\vdash_{S'} u \sigma w_0 \sigma w_1 \sigma \ldots \sigma w_{n-1}$ for some n. Then by rule /3/, and /5/

$$\vdash_{S'} u \sigma w_0 \sigma w_1 \sigma \ldots \sigma w_{n-1} \sigma w_n .$$

Denoting $u \sigma w_0 \sigma w_1 \sigma \ldots \sigma w_{n-1}$ by v we obtain /ii/.

To demonstrate /iii/ we can observe that for each w the set of wff's, which optionally can occur in the proof of w in S', contains words not longer than the length of w. Because V is finite, and S' is recursive, then there exists an effective procedure to check for arbitrary w whether w is or is not a theorem of S'. Hence, $T(S')$ is recursive.

4. The deduction system S' is essentially poorer than S i.e. $T(S') \subset T(S)$ and $T(S') \neq T(S)$. However, we can obtain another simple system equivalent to S by adjoining to R' one additional rule. We have the following:

T2. For each conjuction system S over V there exists a simple conjunction system S^* over V /with the same set of wff's/ such that:

i. $\vdash_S w$ iff $\vdash_{S^*} w$,

ii. if S is recursive then so is S^* .

D e m. At first we construct a system S' as above. Next, we define the system S^* as S' with one additional rule:

$r^*(w, v)$ iff there exists a wff u such that

$$w = u \sigma v$$

/6/

Obviously, S^* is recursive if S is . Now , by T1 if $\vdash_S w$, then either $\vdash_{S'} w$ (hence $\vdash_{S^*} w$) or $\vdash_{S'} u \sigma w$ for some u , and by new rule /6/ $\vdash_S w$. Conversely, if $\vdash_{S^*} w$, then omitting in the proof all applications of the rule /6/ we obtain $\vdash_{S'} u \sigma w$, therefore, by T1 we have $\vdash_S u \sigma w$, and by /1/ $\vdash_S w$.

5. We have demonstrated somewhat more than T2 ascertains. Namely, that rule /6/ need not be applied more than once, as the last step of the proof. Speaking about the rules of S' as being the "effective" ones /for they assure the recursiveness of $T(S')$ /and about the rule r^* as being the "non-effective" one /for it may cause a lack of recursiveness of $T(S^*)$ /the following can be added to the thesis of T2:

iii. Each proof in S^* requires several applications of the "effective" rules, and at most one application of the "non-effective" rule.

A close similarity of this result to the Kleene normal form of recursively enumerable predicates can be easily seen.

References

[1] KLEENE S.C.: Introduction to Metamathematics. Van Nostrand, Amsterdam 1952.

[2] DAVIS M.: Computability and Unsolvability. McGraw-Hill, New York, 1958.

A.M.

ALGORYTMY, V, N° 10 /1969/, 51-57

WYZNACZANIE METODĄ MONTE CARLO ROZKŁADU PRAWDOPODOBIEŃSTWA ROZSTĘPU W PEWNYM CIĄGU ZMIENNYCH LOSOWYCH

Witold KUPŚĆ

Pracę złożono 28.12.67 r.

Dla ciągu X_k zmiennych losowych określonych jako $X_0=0$, $X_k=X_{k-1}+\Delta_k$ $(k=1,2,\ldots n)$ gdzie Δ_k są niezależnymi zmiennymi losowymi o rozkładzie normalnym $N(0,\sigma)$, określono rozstęp jako $D_n = \max_{0\le k\le n} X_k - \min_{0\le k\le n} X_k$ i wyznaczono metodą Monte Carlo wartości dystrybuanty rozstępu $Pr(D_n < d)$ dla $n = 1(1)7, 9(5)29$ i $d=0\ (0.5)16$.

1. SFORMUŁOWANIE ZADANIA

Przy rozwiązywaniu pewnego zagadnienia technicznego powstało następujące zadanie:
Dany jest ciąg $\{X_k\}$ zmiennych losowych określony jako

$$X_0 = 0 , \qquad /1/$$

$$X_k = X_{k-1} + \Delta_k , \quad k = 1,2,\ldots,n ,$$

gdzie Δ_k są niezależnymi zmiennymi losowymi o rozkładzie normalnym $N(0, \sigma)$.

Określa się zmienną losową D_n jako

$$D_n = \max_{0\le k\le n} X_k - \min_{0\le k\le n} X_k \qquad /2/$$

i należy wyznaczyć jej rozkład prawdopodobieństwa:

$$\Pr\left(D_n < d\right) = F\left(d\right) . \qquad /3/$$

Zmienną losową D_n można nazwać rozstępem ciągu n zależnych zmiennych losowych $\{X_k\}$. Jeśli istnieje funkcja gęstości łącznego rozkładu zmiennych $\{X_k\}$, którą zapisać można w postaci $f\left(x_1,x_2,\ldots,x_n\right)$, to dystrybuanta rozstępu (3) jest określona wzorem (por. np. [1])

$$\Pr\left\{D_n < d\right\} = \sum_{k=1}^{n}\left\{\int_{-d}^{0}\left[\int_{x_k}^{x_k+d}\cdots\int_{x_k}^{x_k+d} f\left(x_1,x_2,\ldots,x_n\right)\prod_{\substack{j=1\\ j\neq k}}^{n} dx_j\right] dx_k + \int_{0}^{d}\left[\int_{x_k}^{d}\cdots\int_{x_k}^{d} f\left(x_1,x_2,\ldots,x_n\right)\prod_{\substack{j=1\\ j\neq k}}^{n} dx_j\right] dx_k\right\} \qquad /4/$$

Funkcja gęstości łącznego rozkładu zmiennych X_k jest w przypadku rozpatrywanego zadania n-wymiarową funkcją rozkładu normalnego

$$f\left(x_1,x_2,\ldots,x_n\right) = \frac{1}{\left(2\pi\right)^{\frac{n}{2}}\ \sigma^{n}} \exp\left\{-\frac{1}{2\sigma^2}\sum_{k=1}^{n}\left(x_k-x_{k-1}\right)^2\right\} . \qquad /5/$$

Obliczenie wartości dystrybuanty sprowadza się do zcałkowania wyrażenia (4) przy funkcji $f\left(x_1,x_2,\ldots x_n\right)$ określonej wzorem (5).

2. SPOSÓB OBLICZANIA PRAWDOPODOBIEŃSTW

Efektywne całkowanie wyrażenia (4) jest trudne już dla $n > 1$, gdyż wymaga obliczania całek wielokrotnych z wielowymiarowego rozkładu normalnego dla zmiennych losowych skorelowanych ze sobą. Dlatego dla oszacowania wartości dystrybuant $F_n(d)$ posłużono się metodą Monte Carlo w następujący sposób.

Dokonuje się N realizacji $\{x_{kl}\}$ ciągu $\{x_k\}$ zmiennych losowych określonych przez wyrażenie (1) i dla każdej z nich wyznacza się wartość

$$d_l = \max_{0 \le k \le n} x_{kl} - \min_{0 \le k \le n} x_{kl} \quad (l = 1,2,\ldots,N \quad k = 0,2,..n). \qquad /6/$$

Oznaczając przez A zdarzenie polegające na przyjęciu przez d_l wartości mniejszej niż zadane d, otrzymuje się oszacowanie $\hat{P}(A)$ prawdopodobieństwa $P(A)$ czyli wartości nieznanej dystrybuanty $F(d)$ jako

$$\hat{P}(A) = \hat{F}(d) = \frac{M}{N}, \qquad /7/$$

gdzie M - liczba realizacji zdarzenia A w ciągu N doświadczeń. Gdy realizacje N ciągów $\{x_k\}$ są niezależne, wtedy zgodnie z prawem wielkich liczb Bernouillego dla dowolnego $\varepsilon > 0$, zachodzi:

$$\lim_{N \to \infty} \Pr\left\{ \left| \frac{M}{N} - F(d) \right| < \varepsilon \right\} = 1. \qquad /8/$$

Dla dużych N wielkość M/N można więc przyjąć za dobre oszacowanie $F(d)$. Dla dużych N znany jest również rozkład estymatora $\hat{F}(d)$. Jest to rozkład normalny o wartości średniej $E\{\hat{F}(d)\} = F(d)$ i odchyleniu standartowym $\sigma_{\hat{F}} = \sqrt{F(d)[1 - F(d)]/N}$. Znajomość rozkładu pozwala zbudować przedział ufności dla $F(d)$ w postaci

$$\Pr\left\{\hat{F}(d) - t_\alpha \sigma_{\hat{F}} \leq F(d) \leq \hat{F}(d) + t_\alpha \sigma_{\hat{F}}\right\} = 1 - \alpha . \quad /9/$$

Najczęściej prawdziwa wartość σ_F jest nieznana, gdyż zależy od $F(d)$. W takich przypadkach przedział ufności określony jest wzorem:

$$\Pr\left\{\frac{N}{N+t_\alpha^2}\left[\hat{F}(d) + \frac{t_\alpha^2}{2N} - t_\alpha\sqrt{\frac{\hat{F}(d)[1-\hat{F}(d)]}{N} + \left(\frac{t_\alpha}{2N}\right)^2}\right] \leq F(d) \leq \right.$$

$$\left. \leq \frac{N}{N+t_\alpha^2}\left[\hat{F}(d) + \frac{t_\alpha^2}{2N} + t_\alpha\sqrt{\frac{\hat{F}(d)[1-\hat{F}(d)]}{N} + \left(\frac{t_\alpha}{2N}\right)^2}\right]\right\} = 1 - \alpha . \quad /10/$$

Wzory (9) i (10) podają przedziały ufności dla ustalonych argumentów d dystrybuanty $F(d)$. Można jednak również otrzymać oszacowanie dla całego przedziału, w którym dystrybuanta jest określona. W tym celu należy skorzystać z rozkładu statystyki Kołmogorowa $R_N = \sup |\hat{F}_N(d) - F(d)|$ i wyznaczyć granice przedziału ufności przy pomocy wzoru (por. [2]):

$$\Pr\left\{\hat{F}_N(d) - r_\alpha \leq F(d) \leq \hat{F}_N(d) + r_\alpha\right\} = 1 - \alpha , \quad /11/$$

gdzie r_α - α% wartość zmiennej losowej R_N. Tak wyznaczony przedział nie zależy od parametrów nieznanej dystrybuanty $F(d)$, jest natomiast zwykle szerszy od przedziałów określonych wzorami (9) i (10).

W obu wzorach (9) i (10) t_α oznacza taką wartość zmiennej o rozkładzie normalnym $N(0,1)$, że $P(|T| < t_\alpha) = 1 - \alpha$.

3. WYNIKI OBLICZEŃ

Obliczenia przeprowadzono na maszynie cyfrowej ZAM 2 dla $n = 1,2,3,4,5,6,7,9,14,19,24$ i 29 przy $N = 1000$, korzystając

z programowego generatora liczb pseudolosowych o rozkładzie normalnym. W obliczeniach przyjęto zestandaryzowaną wartość $\sigma = 1$. Tabela 1 podaje otrzymane wartości dystrybuant empirycznych $F_n(d)$ oraz otrzymane oszacowania wartości średnich rozstępu $\bar{D}_n$ i odchyleń standartowych $\hat{\sigma}_{D_n}$. Wyniki obliczeń dla $n = 1$ można porównać ze stablicowanymi wartościami rozkładu modułu zmiennej losowej normalnej $N(0,1)$, gdyż

$$Pr(D_1 < d) = \frac{2}{\sqrt{2\pi}} \int_0^d \exp(-x^2/2)\,dx . \qquad /11/$$

Tabela 1. Dystrybuanta empiryczna $\hat{F}_n(d)$

d	$\hat{F}_n(d)$					
n	1	2	3	4	5	6
0.0	.000	.000	.000	.000	.000	.000
0.5	.389	.120	.032	.011	.001	.001
1.0	.684	.367	.184	.078	.042	.017
1.5	.856	.635	.421	.272	.165	.090
2.0	.943	.819	.624	.477	.346	.229
2.5	.988	.910	.791	.660	.535	.422
3.0	.997	.963	.889	.793	.688	.603
3.5	1.000	.980	.949	.897	.793	.723
4.0		.993	.971	.934	.869	.823
4.5		.999	.989	.966	.926	.900
5.0		1.000	.994	.984	.960	.949
5.5			.996	.992	.976	.968
6.0			.998	.995	.985	.977
6.5			1.000	.997	.989	.986
7.0				.999	.994	.989
7.5				1.000	.995	.994
8.0					.998	.994
8.5					1.000	.997
9.0						.998
9.5						.999
10.0						1.000
$\bar{D}$	0.821	1.357	1.833	1.917	2.619	2.921
$\hat{\sigma}_{\bar{D}}$	0.615	0.801	0.946	1.053	1.221	1.253

Tabela 1. (c.d)

d	$\hat{F}_n(d)$					
n	7	9	14	19	24	29
0.0	.000	.000	.000	.000	.000	.000
0.5	.000	.000	.000	.000	.000	.000
1.0	.008	.000	.000	.000	.000	.000
1.5	.057	.020	.001	.000	.000	.000
2.0	.156	.084	.011	.000	.000	.000
2.5	.342	.199	.044	.010	.000	.000
3.0	.523	.357	.121	.045	.010	.003
3.5	.646	.494	.222	.097	.025	.017
4.0	.743	.613	.342	.176	.073	.040
4.5	.832	.727	.475	.274	.145	.081
5.0	..889	.813	.577	.395	.261	.154
5.5	.939	.883	.693	.513	.356	.234
6.0	.963	.918	.765	.616	.449	.318
6.5	.974	.944	.829	.679	.524	.406
7.0	.985	.966	.863	.739	.614	.496
7.5	.990	.975	.905	.796	.691	.568
8.0	.994	.985	.929	.841	.743	.629
8.5	.996	.992	.948	.873	.792	.710
9.0	.998	.996	.964	.899	.845	.769
9.5	.998	.998	.973	.924	.880	.813
10.0	1.000	.999	.986	.944	.918	.851
10.5		.999	.996	.960	.940	.881
11.0		1.000	.997	.975	.952	.903
11.5			.997	.984	.966	.920
12.0			.998	.989	.974	.946
12.5			.998	.993	.980	.960
13.0			.999	.996	.986	.968
13.5			1.000	.997	.991	.976
14.0				.998	.991	.979
14.5				.998	.995	.984
15.0				.998	.997	.986
15.5				.999	.997	.990
16.0				.999	.997	.992
D	3.332	3.779	4.929	5.785	6.716	7.460
$\hat{\delta}_o$	1.306	1.520	1.841	2.133	2.172	2.524

Korzystając ze wzoru (9) wyznaczono 95 % przedziały ufności dla dystrybuanty $F(d)$ i zestawiono wyniki w tabeli 2.

Tabela 2. Granice przedziałów ufności dla $F(d)$

d	F (d)	$\hat{F}(d) - t_\alpha \sigma_{\hat{F}}$	$\hat{F}(d) + t_\alpha \sigma_{\hat{F}}$
0.5	0.3829	0.3589	0.4191
1.0	0.6827	0.6552	0.7128
1.5	0.8664	0.8348	0.8772
2.0	0.9545	0.9301	0.9559
2.5	0.9876	0.9811	0.9949
3.0	0.9973	0.9937	0.9998

Literatura .

[1] WOŁODIN B.G., GANIN M.P. i inni: Problemy rachunku prawdopodobieństwa. Warszawa, PWN 1966 (str. 115) .

[2] KENDALL M.G., STUART A.: The advanced theory of statistics. London 1960, vol. II, p. 457.

EVALUATION BY MONTE CARLO METHOD THE DISTRIBUTION OF RANGE OF A SEQUENCE OF RANDOM VARIABLES

Summary

Let X_k be a sequence of random variables defined by

$$X_0 = 0 , \; X_k = X_{k-1} + \Delta_k \qquad (k = 1, 2, \ldots , n) ,$$

where Δ_k are uncorrelated random variables distributed normally $N(0, \sigma)$. Define the range of n-variables by $D_n = \max_{0 \le k \le n} X_k - \min_{0 \le k \le n} X_k$.

Values $Pr(D_n < d)$ are calculated by Monte Carlo method for $n = 1(1)7, \; 9(5)29$ and $d = 0.(0.5)16$.

© Instytut Maszyn Matematycznych
Warszawa, ul. Krzywickiego 34

R.Ł.

ALGORYTMY, V, N° 10 /1969/, 59-75

ALGORYTM LEWOSTRONNEJ REDUKCJI SŁÓW

Jan MAŁUSZYŃSKI

Pracę złożono 18.10.1968 r.

Artykuł zawiera algorytm rozkładu syntaktycznego słów dla języków bezkontekstowych generowanych przez niecykliczne i nieskracające gramatyki. W celu opisu algorytmu wprowadzono pojęcie drzewa lewostronnej redukcji słowa terminalnego.

WSTĘP

Problem rozkładu syntaktycznego słów języków bezkontekstowych ma duże znaczenie praktyczne i był wielokrotnie rozważany przez różnych autorów /m.in. [1],[3],[6],[7],[9],[10] /. Ogólny przegląd metod stosowanych przy rozwiązywaniu tego zagadnienia znaleźć można w pracach [2] oraz [5]. Z analizą syntaktyczną wiąże się ściśle problem rozpoznawania słów polegający na badaniu przynależności słowa do zadanego języka. Każdy algorytm rozkładu syntaktycznego można zmodyfikować tak, aby mógł służyć do rozpoznawania, jednak nie na odwrót.

Treścią tej pracy jest algorytm rozkładu syntaktycznego słów dla języków bezkontekstowych generowanych przez pewną klasę gramatyk. Oprócz opisu podana jest również realizacja algorytmu w postaci programu w języku EOL II. Zastosowana metoda rozkładu syntaktycznego jest zbliżona do algorytmu NDS [5].

1. PODSTAWOWE POJĘCIA I DEFINICJE

Alfabetem nazywamy skończony niepusty zbiór symboli zwanych literami. Słowem σ nad alfabetem V nazywamy dowolny skończony ciąg elementów $\sigma = s_1 s_2 s_3 \ldots s_k$, z których każdy należy do V. /W dalszej treści oznaczać będziemy słowa zawsze za pomocą małych liter alfabetu greckiego/. Liczbę elementów słowa σ nazywamy długością słowa i oznaczamy $|\sigma|$. Słowo składające się z zero liter nazywamy słowem pustym i oznaczamy ε .

Zbiór wszystkich słów nad alfabetem V /wraz z ε / oznaczamy V^* .

D e f i n i c j a 1

Gramatyka bezkontekstowa G jest to czwórka $G = (V,T,P,B)$ przy czym:

1. V jest alfabetem,
2. $T \subseteq V$,
3. $P \subseteq (V-T) \times V^*$ jest zbiorem skończonym,
4. $B \subseteq (V-T)$ nazywa się bazą.

Elementy zbioru T nazywamy symbolami terminalnymi. Elementy zbioru P nazywamy produkcjami i zapisujemy je w postaci $u \to \xi$. Jeżeli $\eta, \xi \in V^*$ i istnieją słowa $\sigma_1, \sigma_2 \in V^*$ oraz produkcja $u \to \tau$ takie, że $\eta = \sigma_1 u \sigma_2$ i $\xi = \sigma_1 \tau \sigma_2$ to mówimy, że ξ jest bezpośrednią konsekwencją η , a η bezpośrednią redukcją ξ i zapisujemy to w następujący sposób $\eta \Rightarrow \xi$. Jeżeli $\eta, \xi \in V^*$ i istnieje ciąg $\eta_0, \eta_1, \ldots, \eta_r$ taki, że

$\eta_0 = \eta$, $\eta_r = \xi$ oraz dla każdego i , $0 \leq i < r$ $\eta_i \Rightarrow \eta_{i+1}$ to piszemy $\eta \overset{*}{\Rightarrow} \xi$ i ciąg $\eta_0, \eta_1, \ldots, \eta_r$ nazywamy wywodem słowa ξ ze słowa η , a ciąg $\eta_r, \ldots, \eta_1, \eta_0$ redukcją słowa ξ do słowa η , przy czym jeżeli $\eta \in B$ i $\xi \in T$ to wywód taki nazywamy wywodem podstawowym.

D e f i n i c j a 2

Język bezkontekstowy $L(G)$ generowany przez gramatykę bezkontekstową $G = (V,T,P,B)$ jest to podzbiór zbioru T^* taki, że $\alpha \in L(G)$ wtedy i tylko wtedy, gdy istnieje $b \in B$, że $b \overset{*}{\Rightarrow} \alpha$ i $\alpha \in T^*$.

P r z y k ł a d 1

Dana jest gramatyka $G = (V,T,P,B)$; $V = \{x,y,a,b,c\}$; $T = \{a,b,c\}$; $B = \{x\}$; $P = \{x \to ay,\ x \to yb,\ x \to b,\ y \to xcy,\ y \to a\}$

Słowo bcbcab należy do $L(G)$ ponieważ istnieje wywód podstawowy $x \Rightarrow yb \Rightarrow xcyb \Rightarrow xcxcyb \Rightarrow bcxcyb \Rightarrow bcxcab \Rightarrow bcbcab$.

Wywodem prawostronnym nazywamy taki wywód $\eta_0, \eta_1, \ldots, \eta_r$, że dla każdego $0 \leq i < r$ jeżeli $\eta_i = \omega_i u_i \xi_i$ oraz $\eta_{i+1} = \omega_i \tau_i \xi_i$, przy czym $u_i \Rightarrow \tau_i$, to $\xi_i \in T^*$. Można wykazać zmieniając kolejność stosowanych produkcji, że jeżeli dla pewnego słowa istnieje wywód podstawowy, to dla tego słowa istnieje również prawostronny wywód podstawowy. Gramatyka bezkontekstowa G nazywa się niejednoznaczną jeżeli dla pewnego słowa z $L(G)$ istnieją co najmniej dwa różne prawostronne wywody podstawowe.

Gramatyka, która zawiera produkcje postaci: $z_i \to z_{i+1}$ $0 \leq i < n$ $z_i \in (V-T)$, przy czym $z_0 = z_n$, nazywa się gramatyką cykliczną.

Gramatyka bezkontekstowa G taka, że dla każdego $x \in (V-T)$ $(x \to \varepsilon) \notin P$ nazywa się gramatyką nieskracającą /ang. ε -free/. Posiada ona następującą własność: jeżeli $\alpha, \beta \in V^*$ i $\alpha \overset{*}{\Rightarrow} \beta$, to

$|\alpha| \leq |\beta|$. W [4] wykazano, że dla każdej gramatyki bezkontekstowej $G = (V,T,P,B)$ o jednoelementowej bazie istnieje nieskracająca gramatyka $G' = (V,T,P',B)$ o jednoelementowej bazie taka, że $L(G') = L(G) - \{\varepsilon\}$. Zbiór P' można wyznaczyć za pomocą prostego algorytmu. Twierdzenie to jest prawdziwe również przy założeniu, że baza jest zbiorem wieloelementowym.

2. OPIS ALGORYTMU ROZKŁADU SYNTAKTYCZNEGO

Sformułujemy teraz problem będący przedmiotem naszych rozważań.

Dana jest niecykliczna, nieskracająca gramatyka $G=(V,T,P,B)$ oraz słowo $\alpha \in T^*$. Zbadać, czy $\alpha \in L(G)$ i jeżeli $\alpha \in L(G)$ podać wszystkie prawostronne wywody podstawowe słowa α.

Niech $P = \{u_j \to \xi_j \mid j = 1,2,\ldots N\}$. Określamy $N+1$ elementowy alfabet $Q = \{q_j \mid j = 1,2,\ldots N+1\}$. Dla każdego $r \in (V^* \times T^* \times Q^*)$ określamy zbiory $S_j(r)$ $j = 1,2,\ldots N+1$, oraz $S(r)$. $r' \in S_j(r)$ $(j = 1,2,\ldots N)$ wtedy i tylko wtedy, gdy $r' \in (V^* \times T^* \times Q^*)$ oraz $r = (\sigma \xi_j, \tau, \rho)$ i $r' = (\sigma u_j, \tau, \rho q_j)$. $r' \in S_{N+1}(r)$ wtedy i tylko wtedy, gdy $r' \in (V^* \times T^* \times Q^*)$ oraz $r = (\sigma, s\tau, \varrho)$ i $r' = (\sigma s, \tau, \varrho q_{N+1})$. Każdy zbiór $S_j(r)$ jest zbiorem jednoelementowym lub zbiorem pustym. $r' \in S(r)$ wtedy i tylko wtedy, gdy istnieje $j \leq N+1$ takie, że $r' \in S_j(r)$.

Dla dowolnego słowa $\alpha \in T^*$ definiujemy indukcyjnie zbiór $R(\alpha) \subset (V^* \times T^* \times Q^*)$:

1. $(\varepsilon, \alpha, \varepsilon) \in R(\alpha)$,
2. jeżeli $r \in R(\alpha)$ i $r' \in S(r)$ to $r' \in R(\alpha)$,
3. żadne inne elementy nie należą do zbioru $R(\alpha)$.

D e f i n i c j a 3

Drzewem lewostronnej redukcji słowa $\alpha \in T^*$ w zadanej gramatyce bezkontekstowej $G = (V,T,P,B)$ przy zadanym alfabecie Q nazywamy parę $(R(\alpha), \Gamma)$ przy czym

1. $\Gamma \subset R(\alpha) \times R(\alpha)$,

2. jeżeli $r_1 \in R(\alpha)$ i $r_2 \in R(\alpha)$ to $(r_1, r_2) \in \Gamma$ wtedy i tylko wtedy, gdy $r_2 \in S(r_1)$:

Każdy element $r \in R(\alpha)$ nazywa się wierzchołkiem drzewa, każda para $(r_1, r_2) \in \Gamma$ łukiem.

L e m a t 1

Dla dowolnej niecyklicznej i nieskręcającej gramatyki $G = (V,T,P,B)$ drzewo lewostronnej redukcji każdego słowa $\alpha \in T^*$ jest skończone.

D o w ó d

Dla każdego $r \in R(\alpha)$ $S(r)$ jest zbiorem skończonym. Wystarczy wykazać, że nie istnieje nieskończony ciąg $r_1, r_2, \ldots$ taki, że dla każdego $i = 1,2,\ldots$ $r_{i+1} \in S(r_i)$ gdzie $r_i = (\sigma_i, \tau_i, \rho_i)$. Przypuśćmy, że ciąg taki istnieje. Ponieważ $|\alpha|$ jest liczbą skończoną, a G jest gramatyką nieskracającą, więc dla każdego i $|\sigma_i \tau_i| \leqslant |\alpha|$ i istnieją wskaźniki i_0 oraz $i_1 \geqslant i_0$ takie, że dla każdego $i > i_0$ $\tau_{i+1} = \tau_i$ oraz dla każdego $i > i_1$ $|\sigma_{i+1}| = |\sigma_i|$. Wynika stąd, że dla każdego $i > i_1$ $\sigma_{i+1} = \sigma_0 v_{i+1}$, $\sigma_i = \sigma_0 v_i$. Alfabet V jest skończony i wobec tego w ciągu nieskończonym $v_{i_1}, v_{i_1+1}, \ldots$ muszą występować powtórzenia. Jest to sprzeczne z założeniem o niecykliczności gramatyki G. Cbdo.

L e m a t 2

Dla każdego prawostronnego wywodu podstawowego $B \Rightarrow \eta_0 \Rightarrow \eta_1 \Rightarrow \ldots \Rightarrow \eta_k = \alpha \in T^*$ w dowolnej niecyklicznej i nieskręcającej gramatyce $G = (V,T,P,B)$ i każdego drzewa lewostronnej redukcji słowa α w tej gramatyce, istnieje taki ciąg $\{r_i\}$ $r_i = (\sigma_i, \tau_i, \rho_i) \in R(\alpha)$, że $0 \leqslant i \leqslant k + |\alpha|$ oraz dla każdego i $r_{i+1} \in S(r_i)$ i $\sigma_i \tau_i = \eta_{k-i+|\alpha|-|\tau_i|}$.

D o w ó d

Zadany wywód określa redukcję: $\eta_k = \lambda_k \xi_k \delta_k \overset{h_k}{\underset{h_1}{\Leftarrow}} \eta_{k-1} =$

$= \lambda_k u_k \delta_k = \lambda_{k-1} \xi_{k-1} \delta_{k-1} \overset{h_{k-1}}{\Leftarrow} \eta_{k-2} = \ldots \overset{h_2}{\Leftarrow} \eta_1 = \lambda_2 u_2 \delta_2 = \lambda_1 \xi_1 \delta_1 \overset{h_1}{\Leftarrow} \eta_0 =$

$\lambda_1 u_1 \delta_1 = u_1 \in B$, przy czym dla każdego $0 < j \leq k$ $\lambda_j, \xi_j \in V^*$, $\delta_j \in T^*$, $u_j \in (V-T)$ oraz h_j jest numerem produkcji $u_j \to \xi_j$.

Konstruujemy ciąg $\{r_i\}$ o własnościach podanych w tezie:

1. $r_0 = (\varepsilon, \alpha, \varepsilon)$, $\sigma_0 \tau_0 = \eta_{k-0+|\alpha|-|\alpha|} = \eta_k$; z założenia $\eta_k = \alpha$.

2. Dla każdego $0 \leq i < |\alpha| - |\delta_k|$ $r_{i+1} = S_{N+1}(r_i) \in S(r_i)$. Sprawdzamy, że $\sigma_i \tau_i = \eta_{k-i+|\alpha|-|\tau_i|} = \eta_{k-i+1} = \eta_k$ bo $|\alpha| - |\tau_i| = 1$; z drugiej strony $\sigma_i \tau_i = \eta_k$ z definicji $S_{N+1}(r_{i-1})$.

3. Przypuśćmy, że dla pewnego $l > 0$ $r_l = (\sigma_l, \tau_l, \rho_l) = (\lambda_w \xi_w, \delta_w, \rho_l)$ i $w = k-l+|\alpha|-|\tau_l|$. Wówczas przyjmujemy $r_{l+1} = (\sigma_{l+1}, \tau_{l+1}, \rho_{l+1}) = (\lambda_w u_w, \delta_w, \rho_l q_{h_w}) = S_{h_w}(r_l) \in S(r_l)$. Sprawdzamy, że $w-1 = k-l+|\alpha|-|\tau_l| - 1 = k-(l+1)-|\alpha|+|\tau_{l+1}|$; z drugiej strony $\sigma_{l+1} \tau_{l+1} = \lambda_w u_w \tau_w = \eta_{w-1}$ i żądana własność zachodzi.

4. Oznaczmy $m = |\delta_w| - |\delta_{w-1}| > 0$. Dla każdego $l+1 \leq i < l+m+1$ przyjmujemy $r_{i+1} = S_{N+1}(r_i) \in S(r_i)$. Sprawdzamy, że $w-i = k-i+|\alpha|-|\tau_i| - 1 = k-(i-1)+|\alpha|-|\tau_i|-1-i = k-l+|\alpha|-|\tau_i|-1$; z drugiej strony $\sigma_i \tau_i = \eta_{w-1}$ i żądana własność zachodzi.

Ponieważ $r_{|\alpha|-|\delta_k|}$ i r_{l+m+1} mają takie własności jak r_l, więc konstrukcja ciągu $\{r_i\}$ została zakończona. Z określenia ciągu wynika, że istnieje dokładnie $|\alpha|$ takich wyrazów r_i , że $r_i = S_{N+1}(r_i)$ a więc $0 \leq i \leq k+|\alpha|$ cbdo.

L e m a t 3

Jeżeli dla pewnego drzewa lewostronnej redukcji słowa $\alpha \in T^*$ w niecyklicznej i nieskracającej gramatyce $G=(V,T,P,B)$ istnieje $r \in R(\alpha)$ postaci $r=(b, \varepsilon, \varrho)$ $b \in B$, to istnieje prawostronny wywód podstawowy $b = \eta_0 \Rightarrow \eta_1 \Rightarrow \ldots \Rightarrow \eta_k = \alpha$.

D o w ó d

Jeżeli $r \in R(\alpha)$ to istnieje ciąg $\{r_i\}$, $r_i = (\sigma_i, \tau_i, \varrho_i)$ $0 \leqslant i \leqslant n$ taki, że $r_0 = (\varepsilon, \alpha, \varepsilon)$, $r_n = r$ oraz dla każdego i $r_{i+1} \in S(r_i)$. $r_{i+1} \in S(r_i)$ wtedy i tylko wtedy, gdy istnieje liczba naturalna $h \leqslant N+1$ taka, że $r_{i+1} = S_h(r_i)$. Jeżeli $r_{i+1} = S_{N+1}(r_i)$ to $\sigma_{i+1}\tau_{i+1} = \sigma_i \tau_i$ w przeciwnym przypadku $\sigma_{i+1}\tau_{i+1} \Rightarrow \sigma_i \tau_i$. Utwórzmy ciąg słów $\{\sigma_i \tau_i\}$ $0 \leqslant i \leqslant n$ i wybierzmy z niego wszystkie wyrazy $\sigma_{i_j}\tau_{i_j}$ takie, że $\sigma_{i_j}\tau_{i_j} \neq \sigma_{i_j-1}\tau_{i_j-1}$ $i \leqslant j \leqslant k$ i dla każdego j $i_{j+1} > i_j$. Wynika stąd, że $\sigma_{i_k}\tau_{i_k} = \sigma_n\tau_n = b$. Otrzymaliśmy w ten sposób wywód podstawowy:

$b = \sigma_{i_k}\tau_{i_k} = \eta_0 \Rightarrow \sigma_{i_k-1}\tau_{i_k-1} = \eta_1 \Rightarrow \ldots \Rightarrow \sigma_{i_1}\tau_{i_1} = \eta_{k-1} \Rightarrow \sigma_0\tau_0 =$

$= \eta_k = \alpha$. Z definicji drzewa lewostronnej redukcji wynika, że wywód ten jest prawostronny. Cbdo.

Przypuśćmy, że na zbiorze Q określona została relacja całkowitego porządku. Określimy relację całkowitego porządku na zbiorze Q^* w następujący sposób:

1. Dla każdego słowa $\varrho \in Q^*$ $\varrho > \varepsilon$ jeżeli $|\varrho| > 0$ i $\varrho = \varepsilon$ jeżeli $|\varrho| = 0$.

2. Jeżeli $\varrho, \varphi \in Q^*$ $\varrho = s_1 s_2 \ldots s_l$, $\varphi = t_1 t_2 \ldots t_m$ to;

   a. Jeżeli $l=m$ i dla każdego $0 < j \leqslant l$ $s_j = t_j$ to $\varrho = \varphi$.

   b. Jeżeli istnieje najmniejszy wskaźnik $j_0 \leqslant \min(l,m)$

taki, że $s_{j_0} \neq t_{j_0}$ to $\varphi > \psi$ gdy $s_{j_0} > t_{j_0}$ i $\varphi < \psi$ gdy $s_{j_0} < t_{j_0}$.

c. Jeżeli $l \neq m$ i wskaźnik j_0 nie istnieje to $\varphi > \psi$, gdy $l > m$ i $\varphi < \psi$ gdy $l < m$.

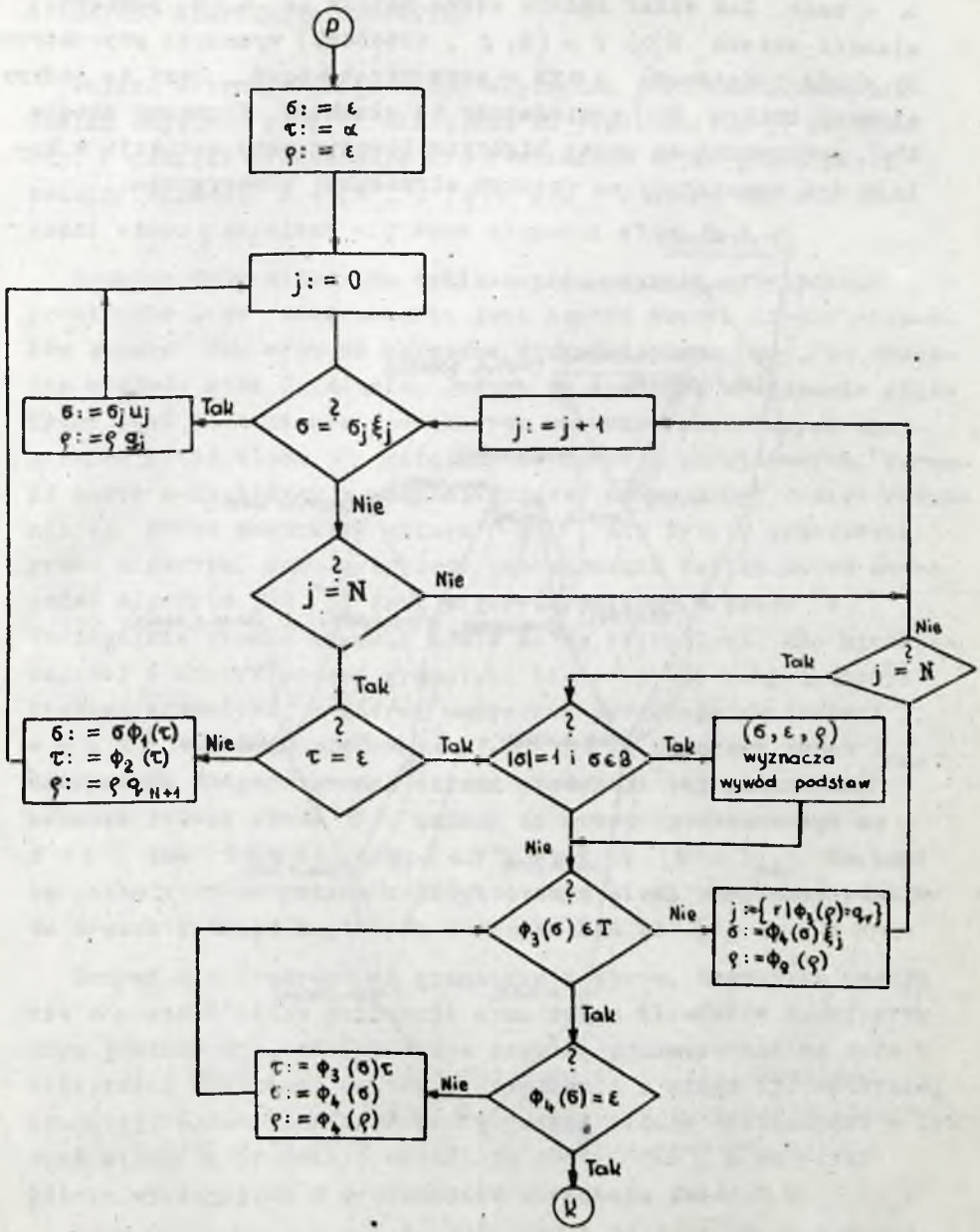
Określimy relację całkowitego porządku na zbiorze $R(\alpha)$. Jeżeli $r_i, r_j \in R(\alpha)$ $r_i = (\sigma_i, \tau_i, \varrho_i)$, $r_j = (\sigma_j, \tau_j, \varrho_j)$ to $r_i > r_j$, gdy $\varrho_i > \varrho_j$, $r_i < r_j$, gdy $\varrho_i < \varrho_j$, i $r_i = r_j$, gdy $\varrho_i = \varrho_j$. /Z definicji zbioru $R(\alpha)$ wynika, że jeżeli $\varrho_i = \varrho_j$ to $\sigma_i = \sigma_j$ i $\tau_i = \tau_j$/.

Algorytm rozkładu syntaktycznego słowa $\alpha \in T^*$ w niecyklicznej i nieskracającej gramatyce $G = (V,T,P,B)$ przy zadanym uporządkowanym alfabecie Q polega na kolejnym generowaniu w rosnącym porządku wszystkich elementów zbioru $R(\alpha)$. Elementy postaci $(b, \varepsilon, \varrho)$ $b \in B$, $\varrho \in Q^*$ wyznaczają jednoznacznie prawostronne wywody podstawowe słowa w gramatyce G /patrz dowód lematu 3/. Elementy te są wyróżniane w procesie generacji. Tak określony algorytm będziemy nazywali algorytmem lewostronnej redukcji słowa α w gramatyce G.

Rys. 1 przedstawia sieć działań dla maszynowej realizacji podanego algorytmu przy założeniu, że zadana jest pewna numeracja zbioru produkcji, a relacja porządku na zbiorze Q określona jest w następujący sposób: dla każdego $j \neq 1$ $q_j > q_1$ wtedy i tylko wtedy, gdy $j > 1$.

P r z y k ł a d 2

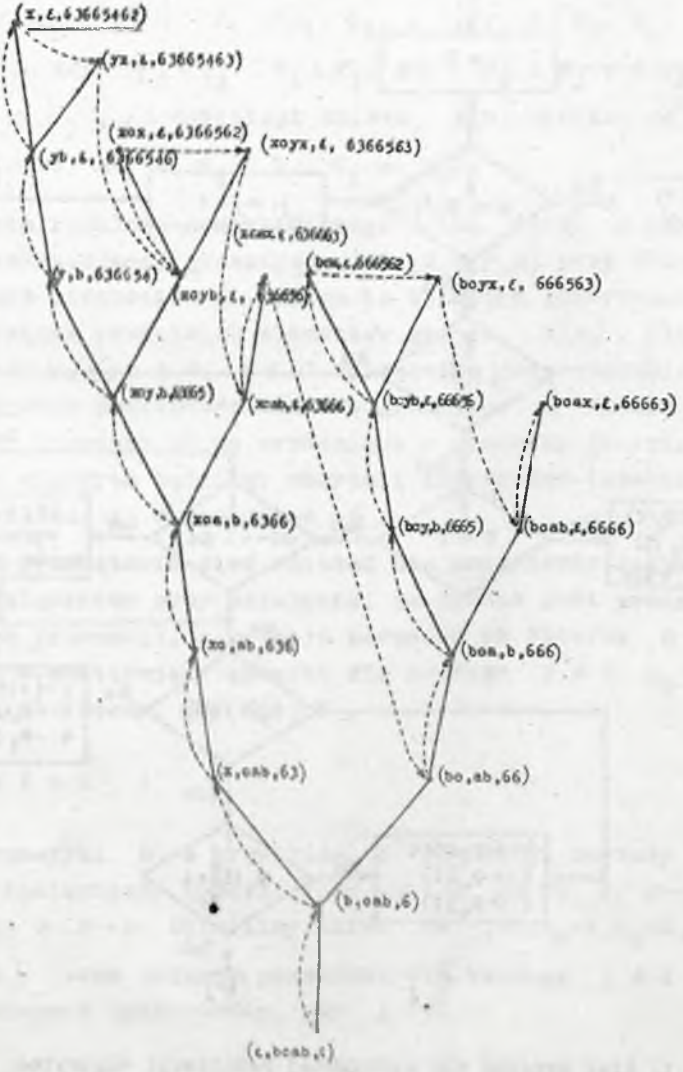
Dla gramatyki G z przykładu 1 produkcje zostały ponumerowane w następujący sposób: 1. $x \to ay$, 2. $x \to yb$, 3. $x \to b$, 4. $y \to xcy$, 5. $y \to a$. Określimy zbiór $Q = \{q_1=1, q_2=2, q_3=3, q_4=4, q_5=5, q_6=6\}$ oraz relację porządku: dla każdego $j \neq 1$ $q_j > q_1$ wtedy i tylko wtedy, gdy $j > 1$.



Rys. 1. Sieć działań dla maszynowej realizacji algorytmu lewostronnej redukcji. Dla każdego $\sigma = s_1 s_2 \ldots s_r \in (V^* - \{\varepsilon\})$

$$\varphi_1(\sigma) \stackrel{df}{=} s_1,\ \varphi_2(\sigma) \stackrel{df}{=} s_2 \ldots s_r,\ \varphi_3(\sigma) \stackrel{df}{=} s_r,\ \varphi_4(\sigma) \stackrel{df}{=} s_1 s_2 \ldots s_{r-1}$$

Rys. 2 przedstawia drzewo lewostronnej redukcji słowa α = bcab. Jak widać zadane słowo należy do L(G) ponieważ element zbioru $R(\alpha)$ r = $(x, \varepsilon, 63665462)$ wyznacza prawostronny wywód podstawowy $x \Rightarrow yb \Rightarrow xoyb \Rightarrow xcab \Rightarrow bcab$. Jest to jedyny element zbioru $R(\alpha)$ posiadający tę własność. Elementy zbioru $R(\alpha)$ generowane są przez algorytm lewostronnej redukcji w kolejności oznaczonej na rysunku strzałkami przerywanymi.



Rys. 2. Drzewo lewostronnej redukcji słowa bcab /przykład 2/

3. METODY ULEPSZENIA ALGORYTMU

Zaletą sformułowanego wyżej algorytmu jest stosunkowo niewielka objętość pamięci niezbędna do jego realizacji maszynowej. W pamięci przechowuje się w zasadzie tylko gramatykę i kolejny element $r = (\sigma, \tau, \varrho) \in R(\alpha)$. W miarę wzrostu długości słowa ϱ zmiejsza się suma długości słów σ i τ.

Poważną wadą algorytmu wykluczającą w wielu przypadkach praktyczne jego zastosowanie jest szybki wzrost liczby elementów zbioru $R(\alpha)$ wraz ze wzrostem długości słowa α, co znacznie wydłuża czas działania. Jednym ze sposobów ulepszenia algorytmu jest znalezienie dodatkowych warunków koniecznych spełnianych przez słowa $\sigma\tau$ należące do wywodów podstawowych. Warunki takie pozwoliłyby z góry eliminować pewne drogi drzewa redukcji tj. pewne podzbiory zbioru $R(\alpha)$ nie byłyby generowane przez algorytm. Jako przykłady zastosowania takich metod można podać algorytm SDS [5] orat algorytm opisany w pracy [9]. Szczególnie prosto warunki takie można sformułować dla nieskracającej i niecyklicznej gramatyki liniowej tj. takiej niecyklicznej gramatyki, w której wszystkie produkcje są postaci $w \to \alpha x \beta, \alpha \in T^*, \beta \in T^*$ $w \in (V-T)$, $x \in V$. Oznaczmy przez l_{max} maksymalną długość prawej strony produkcji tej gramatyki. Wówczas jeżeli słowo $\sigma\tau$ należy do wywodu podstawowego to $\sigma = \varepsilon$ lub $\sigma = \eta w \xi$, $\eta, \xi \in T^*$, $w \in V$ i $|\xi| < l_{max}$. Warunki te zostały wykorzystane w przytoczonym niżej programie rozkładu syntaktycznego napisanym w języku EOL II [8].

Danymi dla programu są gramatyka i słowo. Gramatykę zadaje się w postaci ciągu produkcji oraz ciągu elementów bazy, przy czym zakłada się, że produkcje zostały ponumerowane od zera w kolejności odwrotnej do ich występowania w ciągu tj. ostatniej produkcji nadano numer zero. Wszystkie litery występujące w lewych stronach produkcji określają zbiór $(V-T)$, a wszystkie litery występujące w produkcjach określają zbiór V.

Alfabet Q określony został w następujący sposób: dla każdego $0 \leq i \leq N+1$ $q_i = i$. Po wczytaniu gramatyki wyznaczana

jest maksymalna długość prawej strony produkcji l_{max}, zbiory V i T /przy czym zbiór T jest wypisywany przez maszynę/ oraz bada się czy gramatyka jest liniowa.

W przypadku gramatyki liniowej, w trakcie realizacji algorytmu lewostronnej redukcji po utworzeniu każdego nowego elementu $r \in R(\alpha)$ badane są sformułowane wyżej warunki konieczne na słowo $\sigma\tau$. Jeżeli nie są one spełnione przechodzi się do tworzenia następnej drogi drzewa lewostronnej redukcji. W obydwu przypadkach maszyna wypisuje wszystkie słowa σ odpowiadające osiągniętym wierzchołkom drzewa redukcji dla których $S(r)$ jest zbiorem pustym. Dla wierzchołków postaci $(b, \varepsilon, \varphi)$, $b \in B$ wypisywane jest dodatkowo słowo φ z pominięciem liter q_{N+1}. W programie zastosowano sposób pozwalający nie umieszczać liter q_{N+1} w słowie φ. Daje to pewną oszczędność miejsc pamięci.

W praktycznych zastosowaniach często można z góry założyć jednoznaczność gramatyki. Pozwala to zakończyć tworzenie drzewa redukcji po znalezieniu pierwszego elementu $r \in R(\alpha)$ postaci $(b, \varepsilon, \varphi)$ $b \in B$. O ile element taki nie istnieje /zadane słowo nie należy do $L(G)$/ analiza obejmuje całe drzewo lewostronnej redukcji.

Załóżmy, że istnieje szukany element $r \in R(\alpha)$ i gramatyka jest jednoznaczna. Wychodząc z początkowego wierzchołka drzewa lewostronnej redukcji posuwamy się w czasie realizacji algorytmu w ten sposób, że w każdym osiągniętym wierzchołku drzewa dokonujemy wyboru jednego z wychodzących łuków o ile łuki takie istnieją. Przy ustalonej numeracji produkcji wybór ten zdeterminowany jest przez przyjętą relację porządku na zbiorze Q.

Widać stąd, że dla niektórych słów gramatyk jednoznacznych można bardzo znacznie skrócić czas rozkładu syntaktycznego przez odpowiednie określenie relacji porządku na zbiorze Q lub co jest równoznaczne przez odpowiednią numerację produkcji gramatyki.

Autor pragnie wyrazić podziękowanie dr Antoniemu Mazurkiewiczowi za dużą pomoc okazaną w czasie rozwiązywania problemu.

Literatura

[1] ABBOT R.: Right-to-left Parsing Using a Predictive Grammar. Math. Linguistics and Automatic Translation. Sec VIII Rep Nr NSF-13. Harvard Comput. Lab. Cambridge Apr. 1964.

[2] FELDMAN I., GRIES D.: Translator Writing Systems. Comm. of the ACM vol. 11 nr 2 Feb. 1968.

[3] FLOYD R.: Syntactic Analysis and Operator Precedence. J. ACM 10. 1963.

[4] GINSBURG S.: The Mathematical Theory of Context-Free Languages. McGraw-Hill Book Company 1966.

[5] GRIFFITHS T.V., PETRICK S.R.: On the Relative Efficiencies of Context-Free Grammar Recognizers. Comm. of the ACM vol. 8 May 1965.

[6] INGERMAN P.: A Syntax-Oriented Translator. Academic Press 1966.

[7] KNUTH D.: On the Translation of Languages from Left to Right. Information and Control vol. 8 nr 6 1965.

[8] ŁUKASZEWICZ L.: Report on the Symbol Manipulation Language EOL II. Warszawa 1966 skrypt IMM.

[9] MAZURKIEWICZ A.: O problemie czytania dla języków formalnych. Algorytmy nr 9 1968.

[10] YOUNGER D.: Recognition and Parsing of Context-Free Languages in Time n^3. Information and Control vol. 10 nr 2 1967.

LEFT-TO-RIGHT BOTTOM-UP PARSING ALGORITHM

Summary

A version of left-to-right, bottom-up parsing algorithm for context-free languages generated by cyclically non-null, ε-free grammars is presented. The features of the algorithm are discussed with the use of left-to-right, bottom-up parsing tree.

© Instytut Maszyn Matematycznych
Warszawa, ul. Krzywickiego 34

A.M.

```
SECTION PARSING

/*                  CZYTANIE GRAMATYKI                  */

WA:CLEAR I1,L;       /*CZYTANIE LEWYCH STRON PRODUKCJI */
SET 'WA*L',Q1
READ I1,A4,RB
SET '**',A2

WB:CLEAR I1,L;       /* CZYTANIE PRAWYCH STRON PRODUKCJI */
SET 'WB *L',Q1
READ I1,A2,1
EQ I1,';'
GOPL WD
EQ I1,L
GOPL WB
GOTO WA
WD:CLEAR I1,L
SET 'WD*L',Q1
READ I1,A16,1
EQ I1,';'
GOMI WD

/*                  BADANIE GRAMATYKI                   */

WE:SET '**',A2
SET 'L',A6
MOVE B2,A12
SET 0,A15
SET '**',A5

LA:EQ A12,'**';      /* SZUKANIE MAX DLUGOSCI PRODUKCJI */
GOMI LB
MOVE B12,A13,'**'
GOMI LB3
FIND B13,K14
GT B14,B15
GOPL LC
CLEAR A14
GOTO LD
LC:CLEAR A15
MOVE A14,A15
```

```
LD:CLEAR A13,T4;    /* BADANIE LINIOWOSCI GRAMATYKI */
CLEAR A13,1
CLEAR A13,T4
GOMI LB
CLEAR A13
SET 'N',A6;          /* GRAMATYKA NIELINIOWA */

LB:EQ B12,T4;       /* TWORZENIE LISTY TERMINALNYCH */
GOMI LB1
LB0:CLEAR A12,1
GOTO LA

LB1:EQ B12,T5
GOPL LB0
MOVE A12,A5,1
GOTO LA
LB3:WRITE B5,Q1
SET '*L',Q1

/*                  ROZKLAD SYNTAKTYCZNY SLOWA                  */

B:CLEAR I1,L;    /* CZYTANIE SLOWA */
READ I1,A7,1
EQ B7,T5
GOMI DR2
EQ I1,';'
GOMI B
GOTO KON

KON:MOVE A7,A8
PORO:MOVE A8,A7,1;     /* DOPISYWANIE KOLEJNEJ LITERY */
GOMI K
POR1:EQ B6,'L'
GOMI PO11
FIND B7,K14,T4;  /*BADANIE WARUNKOW KONIECZNYCH DLA GR.LIN*/
GOMI PO11
GT B15,B14
GOPL PO11
PO12:CLEAR A13
CLEAR A14
GOTO DRP;    /* ZASTOSOWANO NIEWLASCIWA PRODUKCJE */
```

```
PO11:MOVE A2,A1,1;   /* SZUKANIE PRODUKCJI */
MOVE A2,A1,B7
GOMI POM
EQ B1,'**'
GOMI PO11
MOVE B2,A9,'**'
FIND B9,K10
MOVE B7,A11,N10
GOMI AM
AP:COMPRESS A9
COMPRESS A11
EQ A9,B11
GOPL AK
AM:CLEAR A9
CLEAR A11
CLEAR A10
GOTO PO11
POM:MOVE A1,A2
GOTO PORO

AK:EQ B6,'L'
GOMI AK1
MOVE B7,B13;   /* BADANIE WARUNKOW DLA GRAM. LINIOWEJ */
CLEAR A13,N10
CLEAR A13,T4
GOMI AK1
CLEAR A13
GOTO AM

AK1:CLEAR A11;    /* TWORZENIE BEZPOSREDNIEJ REDUKCJI */
CLEAR A7,N10


CLEAR A10
MOVE B1,A2
FIND B1,K10
MOVE A10,A20;     /* TWORZENIE STOSU NUMEROW */
GW:SET 0,A10
GW1:CLEAR A1,1
CLEAR A1,'**'
GOMI F
ADD A10,1
GOTO GW1
F:MOVE A4,A3,N10
MOVE B4,A7,1
MOVE A3,A4
MOVE A10,A21
GOTO POR1
```

```
K:FIND B7,K10;     /* KONIEC SLOWA */
MOVE B7,A7,N10
WRITE A7,Q1,N10;   /* WYPISYWANIE WYNIKOW POSREDNICH */
SET '*L',Q1
EQ A10,1
EQ B7,T16
GOPL DRO;          /* SLOWO NALEZY DO JEZYKA */
CLEAR A10
GOTO DRP

DRUK:DRO:MOVE B21,B10
DR3:MOVE A10,A22,1
GOMI DRP
WORD A22;          /* WYPISYWANIE STOSU NUMEROW */
WRITE A22,Q1
SET '*L',Q1
GOTO DR3

DR2:STOP

DRP:MOVE A1,A2;    /* TWORZENIE BEZPOSREDNIEJ KONSEKWENCJI */
DRPO:MOVE A7,A8,1
GOMI DR2;          /* ZBADANO WSZYSTKIE REDUKCJE */
EQ B8,T5
GOMI DRP1
GOTO DRPO
DRP1:MOVE A2,A1,N20
CLEAR A20,1
CLEAR A21,1
CLEAR A8,1
MOVE B2,B7,'**'
GOTO POR1
ENDS
KONP PARSING
```

ALGORYTMY, V, N° 10 /1969/, 77-88

PRELIMINARY ANALYSIS OF ALGOL-60 SOURCE PROGRAMS FOR ZAM COMPUTERS

by Jowita KONCEWICZ

Received March 15th,1968

The paper deals with the conversion of ALGOL-60 programs into sequences of compiler intermediate language words. The method of source text syntactic analysis and transformation by an automaton with 6 states is given. The describes method has been used in the design of first pass of two ALGOL-60 compilers for ZAM computers.

In preparation of an ALGOL-60 compiler for ZAM 21-ALFA computer,its preliminary part has been treated separately. This part, called first pass of the compiler, will perform the following tasks:

- reading the source program written in ALGOL hardware representation,
- identifying basic syntactic units of reference language[1]
- translation of the above units into symbols of so-called

[1] As the paper "Revised Report..." [1] does not introduce the term "basic syntactic unit of reference language", identifiers, numbers, strings, and basic symbols, excluding letters and digits, will be understood as the above mentioned term.

intermediate language with a uniform structure convenient for further compiler work.

The idea was to make the compiler dependent neither upon the actual hardware representation, nor upon the type of input equipment. On the other hand, shortened and uniform text of a source program is essential for multipass compiler work. It should be noticed that every pass transfers the text handled from the drum to the internal storage, and next, after having processed it, sends it back to the drum. A shortened and uniform text allows to save storage and simplifies the program design which transforms the symbols given.

It also seemed useful to reach a flexible organization of the program realizing the above mentioned problems. This flexibility would cause an easy exchange of some small parts (e.g. some constants in the program), without disturbing the general structure. The above changes can be made in hardware representation as well as in machine representation of separate intermediate language symbols (the latter very often occur during debugging and preliminary compiler exploitation). The required flexibility is obtained due to:

- the expression of the basic part of the pass 1 program in the form of a transition table of a small finite automaton,
- application of table-look-up technique to the majority of symbol identification and coding of intermediate language symbols,
- performing preparatory and supplementing operations of the automaton action by means of auxiliary subrutines.

1. Description of hardware representation alphabet

For all hardware representation texts the following six categories are distinguished:

1. identifiers,
2. delimiters,

3. key words,
4. numbers,
5. strings,
6. comments.

Texts are made of the following teletype code characters:
letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z;
numbers: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;
delimiter symbols: +, −, *, /, =, ,, ., ;, :, ., ', ◊, (,), [,];
allocation symbols: space, new line.

Because of the lack of teletype characters corresponding to the remaining delimiters in ALGOL, and because of the impossibility to underline the texts, the necessity arose to introduce the notion of key words, i.e. such texts which are not allowed to be used as identifiers. Hence if a delimiter symbol does not appear, after an identifier, a key word, or a number, a space, or a new line should follow. However, inside the identifiers and key words, except GO TO, a space (new line) is not permitted. In the remaining cases the symbols space and new line are of no importance.

Identifiers can be of any length. Strings can be of any length too, but the characters space and new line - inside the strings - are unnoticeable during source program reading. Strings cannot include any other strings. Comments can be used according to the definition given in [1] and they do not affect the object program.

Table 1 gives the correspondence between the reference ALGOL language and the delimiters and key words of a hardware representation in ZAM computers.

2. Description of intermediate language alphabet

All symbols of intermediate language alphabet are of the same length equal to that of the computer word (24 bits) and are divided into the following five categories:

1. identifiers,
2. delimiters,
3. numbers,
4. strings,
5. errors.

The symbols of the first four categories represent suitable basic syntactic units of the language, but the "error" symbols are used to replace incorrectly made texts of the source program[1]. The make-up of all the symbols is the same: the first 15 bits of the word contain the code of the given symbol category, the remaining 9 bits include the code that distinguishes the symbol within its category.

The initial texts of identifiers and strings are stored in tables, since they may be of any length in hardware language representation. The corresponding symbols of the intermediate language consist of a code determining the symbol category and its relative address in an adequate table.

As in the ZAM-ALGOL implementation real and integer numbers are in the floating-point representation comprising two machine words, all numbers are stored in a common number table. Therefore, number symbols of the intermediate language, made identically as symbol identifiers or strings correspond to numbers in hardware language representation.

Thus, tables of identifiers, strings, and numbers that store their initial form, are a supplement to the source program in the intermediate language.

3. Problem realization

The following three categories of texts are distinguished

[1] Such a convention was accepted in connection with the principle that an occurence, of error does not interrupt the compiling process. Information about all the errors met is written by the compiler after the end of source program translation into an object program with a symbolic address, i.e. after four passes of compiler.

while the pass 1 program procedes to read a teletype symbol sequence:

1. sequence of letters or digits beginning with a letter and ending with a symbol of delimiter or symbol of allocation,
2. sequence of digits beginning with a digit, the symbol "." or the symbol "◊", ending with the symbols of delimiter, allocation or letter,
3. symbol of delimiter or of alignment format.

For every category of texts there is a separate device that reads a given text and translates it in to an intermediate language symbol. The symbols, when translated, are sent to the input of the program which realizes the operation of the so-called automaton A, controling the whole pass 1. The automaton A checks the source text correctness basing on the context made of the adjoining basic units of the reference language. It decides on writing symbols of the intermediate language, continuing the source program reading and translating, or finishing pass 1 operation.

General outline of action of pass 1 program is the following:

1. Read the symbol of the teletype code and put it into the auxiliary store MAG1.
2. If the symbol is not a letter, a digit, "◊" or "," , then go to point 4.
3. Read subsequent text symbols and store them in MAG1 until meeting the terminal symbol of this category; load the terminal symbol in the store MAG2.
4. Translate the text in MAG1 in to a symbol of the intermediate language.
5. Perform one cycle of A automaton action.
6. If in MAG2 is an alignment format symbol, then go to point 1.
7. Move the content of MAG2 to MAG1, load in MAG2 the aligment format symbol and go to point 4.

All the operations mentioned in items 1-4 and 6-7 of the general outline of action are supplementary for automaton A. Operations 1-4 translate the texts in hardware ALGOL representation in to machine representation of reference language. The latter representation constitutes the input alphabet of A. Operations of items 6-7 are result of the fact that the texts of the first and second categories are read together with their terminal symbol. Therefore, they are generally translated in to two basic units of the reference language, but automaton A reads the input alphabet successively one symbol at a time.

4. Table-look-up technique for translating hardware representation into intermediate representation

There are five following tables:

1. key words,
2. delimiters,
3. identifiers,
4. numbers,
5. strings.

The first two tables contain intermediate language codes of delimiters. They constitute a constant part of pass 1 program. The last three tables are formed by the pass 1 program. They contain respectively:

- identifiers loaded by four characters of the input alphabet in one storage word,
- numbers in floating point binary representation (two storage words),
- strings with their delimiter symbols, loaded by three input alphabet symbols in one storage word.

An appropriate table does not have to be consulted to see whether it contains numbers and strings before loading them, because they will appear explicite in the object program, and as many times as they were used in the source program. The above tables are made solely to get a uniform and concise form

of intermediate language. However, this principle cannot be applied to identifiers for corresponding values are loaded outside the object program according to a proper declaration, and the object program operations refer to their addresses. This causes the necessity of one-to-one translation of identifiers being identical in the source text - into intermediate language codes, thus it requires the table to be searched every time an identifier is being met.

In order to reduce the time for identifier table searching, a special technique of table organization was applied. The latter is a modified version of the technique used for similar tasks in ALFA compiler of M-20 computers [3].

It consists in assigning to every identifier the integer number n which is the value of the so-called allocation function for identifiers in a given storage area before checking the table. Then, the n-th location of the storage is checked instead of searching the whole table. Allocation function is defined as follows. The table contains $512 = 2^9$ storage locations. The sequence of storage words, including an identifier, in the form of four characters loaded in one word, is divided into 9 bit segments, and integer numbers in these segments are summed modulo 2^9. The obtained number $0 \leqslant n \leqslant 511$ is the expected identifier address in the table.

However, as the so computed assignment is not synonymous, it might occur that the n-th place in the table would be already occupied by another identifier. Then the following storage words are being consecutively investigated until the mentioned identifier or the first vacant location is being met. Then the identifier is loaded into it. The operation of table searching is acting so as if the beginning of the table was stocked to its end, i.e. the location with the relative address 0 is next to the storage location with the relative address 511.

The length of identifiers is different. One storage word contains four alphanumerical characters, therefore, besides the above discussed table, another one is made for identifiers,

whose length exceeds four characters. The latter identifiers in this table are stored successively.

The identifier address in the above discussed table together with the information on the identifier length is in this case loaded in the address assigned to the given identifier by the allocation function.

5. Description of the automaton A

The work of automaton A consists in repeating single cycles, each consisting of the following three operations:

1. read the input alphabet symbol,
2. write the output alphabet symbol,
3. change the state.

The input alphabet A comprises symbols: identifier, number, key word, delimiter, and the alignment format symbol,

The output alphabet A comprises symbols:

K - intermediate language code of the basic unit of the reference language,
B - blank,
E - error.

Nonterminal states A are marked by digits 0, 1, 2, 3, 4, 5. They are determined in Table 2, which defines the outputs and transitions of the automaton A. At the begining of the pass 1 action the automaton A is in the initial state (0).

Terminal states A are called in abbreviated form: ID, NB, EN, PR, AS, CO, and ST.

Terminal states denote a transfer to certain operations defined in Table 3. These operations have been distinguished for the reduce of the size of the outputs and transitions table of A. After proper subroutines performed the above operations, the state of the automaton A is changed (cf. Table 3), except for the case when the operation of pass 1 program is finished, which can occur in state EN. Such a connection of automaton

with subroutines which supplement it significantly facilitates the automaton construction and optimizes the pass 1 program.

6. Final notes

The described way of reading the source program is implemented in ALGOL-60 compiler, operating on ZAM 21-ALFA computer. Now it has been shown it can be applied in total in ALGOL-60 compiler for ZAM 41 computers, despite essential differences in both compilers design. It proves that the methods applied to realize the discussed problems ensure their independence of hardware representation as well as of the entire translation process.

Table 1. Correspondence between ALGOL reference language and ZAM hardware representation

Reference language	ZAM hardware representation	Reference language	ZAM hardware representation
+	+	:	:
-	-	;	;
x	*	:=	:=
/	/		
↑	DIV	step	STEP
<	POWER	until	UNTIL
>	LESS	while	WHILE
=	NOTLESS	comment	COMMENT
≤	EQUAL or =	(	(
≥	NOTGREATER	)	)
≠	GREATER	[	[
≡	NOTEQUAL	]	]
⊃	EQUIV	,	,
∨	IMPL	.	.
∧	OR	begin	BEGIN
¬	AND	end	END
go to	NOT	own	OWN
if	GOTO or GO TO	Boolean	BOOLEAN
then	IF	integer	INTEGER
else	THEN	real	REAL
for	ELSE	array	ARRAY
do	FOR	switch	SWITCH
,	DO	procedure	PROCEDURE
.	,	string	STRING
10	.	label	LABEL
	◊	value	VALUE

Table 2. Table of output and transition of automaton A .

State	0		1		2		3		4		5	
Symbol	Symb	st	Symb	st	Symb	st	Symb	st	Symb	st	Symb	st
Identifier	K	ID	K	3	K	3	K	3	B	4	B	PR
Number	E	0	K	3	K	3	K	3	B	4	E	5
BEGIN	K	NB	K	NB	K	NB	K	NB	B	4	K	NB
END	E	0	K	EN	K	EN	K	EN	K	EN	K	EN
ELSE	E	0	K	3	K	3	K	3	K	3	K	3
COMMENT	E	0	E	1	E	CO	E	3	E	4	E	5
key words	E	0	K	3	K	3	K	3	B	4	K	3
;	E	0	K	2	K	2	K	2	K	2	K	2
:	E	0	E	1	E	2	K	1	B	4	K	3
=	E	0	B	AS	E	2	K	3	B	4	K	3
)	E	0	E	1	E	2	K	5	B	4	K	5
'	E	0	E	1	E	2	B	ST	B	4	E	5
Space or new line	B	0	B	1	B	2	B	3	B	4	B	5
Other delimiters	E	0	K	3	K	3	K	3	B	4	K	3

Table 3. Subroutines supplementing the action of automaton A

Terminal state A	Operation	State A after operation
ID	if ":" is the next symbol, write K, if not, write E	0
NB	add 1 to the counter of BEGIN symbols	2
EN	subtract 1 from the counter of BEGIN symbols; if it equals 0 finish the pass 1 operation	4
PR	read the symbol sequence until the context ": (" is met and replace the written code ")" by the code ","	3
AS	replace the written code ":" by the code ":="	3
CO	read the symbol sequence until ";" is met	2
ST	read the symbol string until "'" is met, put the string in the table of strings and write the code "string"	3

References

[1] NAUR P. - ed : Revised Report on the Algorithmic Language ALGOL 60. Regnecentralen, Copenhagen, 1962.

[2] NAUR P.: The Design of the GIER ALGOL Compiler. BIT 3, 1963, pp. 124-140, 145-166.

[3] Al´fa - sistema avtomatizacii programmirovanija. Sbornik statej pod redakcijej A.P. Ersova, RIO SO AN SSSR, Novosibirsk, 1965.

[4] RANDELL B., RUSSELL L.J.: ALGOL 60 Implementation. Academic Press, London, 1964.

Warszawa, ul. Krzywickiego 34

A.W.

ALGORYTMY, V, N° 10 /1969/, 89-92

THE OPTIMUM SELECTION OF VARIOUS TYPES OF INTERNAL STORAGE

by Bartłomiej GŁOWACKI

Received October 15th,1968

Given a computer internal storage comprising a series of various storage types. For such a storage its cycle time distribution is so determined, that with a fixed cost of the internal storage, the mean cycle time is minimum.

In the process of computation the frequency of using internal storage locations is not the same for all of them. Small quantity of data is used with great frequency; medium quantity, with medium frequency; a very large quantity, with a low one. It is evident that the access time to the often used locations significantly influences the mean storage cycle, whereas the access time to seldom used locations exerts the minimum influence.

On the other hand, the speed of storage varies in the inverse proportion to the capacity, when the cost-per-bit is constant.

Both the above mentioned facts are taken into account while designing modern computers equipped with a hierarchic internal storage comprising a series of storage of various

capacities, cycles, and costs. These facts contribute to substantial performance/cost advantages if storage hierarchies can be effectively used. The choice of various computer storage types is left to the system designer. The present paper gives an exact solution of this problem.

Let the internal storage be α_i of storage types determined by

$$\alpha_i : \{k_i , t_i , s_i\} \qquad i = 1,2, \ldots , n ,$$

where

k_i - cost ,

t_i - cycle time ,

s_i - capacity .

We take into consideration that internal storage is a one level storage /uniformely addressed independently of the type/. We assume also that if

$$t_i < t_j ,$$

then the probability of carrying out an operation on α_i is greater than the same probability for α_j .
Let us assume that $p(x)$ is the probability of operation execution on the first x storage locations, then the probability of a request on α_i storage is determined as follows

$$P_i = p\left(\sum_{k=1}^{i} S_k\right) - p\left(\sum_{k=1}^{i} S_k\right) ,$$

and the mean cycle time

$$t_m = \sum_{i=1}^{n} t_i P_i . \qquad /1/$$

If the cost of α_i storage is admitted as

$$k_i = a_i \frac{s_i}{t_i} ,$$

where a_i is the constant, then the total cost of the internal storage equals

$$K = \sum_{i=1}^{n} k_i = \sum_{i=1}^{n} a_i \frac{s_i}{t_i} . \qquad /2/$$

Let us pose the problem.

> With a fixed total cost of internal storage find such a distribution of cycle times of various storage types so that the mean cycle time is the minimum.

We shall solve the problem using Lagrange's method. The problem is reduced to find the minimum function t_m under the condition $K = K_o$ = constant.

We introduce the indefinite multiplier m and make a function of n+1 variables.

$$\Phi /t_1, t_2, \ldots , t_n, m/ = t_m - m /K-K_o/ .$$

The necessary condition of the existence for the Φ function extremum reduces the problem to the following set of n+i equations

$$\frac{\partial}{\partial t_i} \left\{ \sum_{i=1}^{n} t_i P_i - m \left[\sum_{i=1}^{n} \left(\frac{a_i s_i}{t_i} \right) - K_o \right] \right\} = 0 ,$$

$$\sum_{i=1}^{n} \frac{a_i s_i}{t_i} - K_o = 0 .$$

While solving the set of equations we obtain

$$t_i = \sqrt{\frac{a_i s_i}{K_o^2 p_i}} \cdot \sum_{k=1}^{n} \sqrt{a_k s_k p_k} \quad . \qquad /3/$$

Substituting these results in the equation /1/ we obtain the minimum cycle time of internal storage

$$t_{min} = \frac{\left(\sum_{k=1}^{n} a_k s_k p_k \right)^2}{K_o} \qquad /4/$$

The probability distribution $p(x)$ being accepted on the basic of [1], or for approximate computations the nonhomogeneous Poisson's distribution, the required cycle times of separate storage types can be determined from formula /3/ if copacities are known. The minimum cycle time can be determined from formula /4/.

References:

[1] SMITH J.L.: Multiprogramming under Page on Demand Strategy. Com. ACM, vol. 10/10/, 1967. pp. 636-645.

© Instytut Maszyn Matematycznych
Warszawa, ul. Krzywickiego 34
A.M.

Cena zł 60,—

Druk „Znak" Zakł. 3. Zam. 282. 500.