TEORIA MASZYN MATEMATYCZNYCH

Komentarz do wykładu doc.dr Zdzisława Pawlaka
dla III-go roku Studium Zaocznego Matematyki UW

Sekcja Metod Numerycznych

Zeszyt 1

O R G A N I Z A C J A M A S Z Y N B E Z A D R E S O W Y C H

Rozdział I : O B L I C Z E N I A P R O S T E

Pojęcie o b l i c z e n i a jest chyba jednym z najbardziej podstawowych pojęć matematycznych, mimo to nie było ono do tej pory analizowane dostatecznie szczegółowo. Studia nad teorią aparatów matematycznych wymagają - zdaniem autora - uprzedniego dokładnego zbadania tego pojęcia, a co najmniej klasyfikacji różnego rodzaju obliczeń. W niniejszej książce podjęto prymitywną próbę zdefiniowania pewnej klasy obliczeń oraz zbadania niektórych ich własności w zakresie, w jakim jest to niezbędne do dalszych rozważań, dotyczących maszyn matematycznych.

Mając na uwadze konstrukcję maszyn, musimy spojrzeć na obliczenie jako na p r o c e s tworzenia z jednych obiektów - za pomocą określonych operacji - nowych obiektów. Przez obiekty rozumiem tutaj twory matematyczne takie jak: liczby, wektory, macierze itp, a ściśle - napisy, których używamy do ich oznaczenia. Tak więc obliczenie jest procesem przekształcania napisów w myśl określonych reguł. W maszynach elektronowych pojęcia matematyczne są przedstawione nie za pomocą napisów, lecz sygnałów elektrycznych; obliczenie możnaby więc interpretować jako proces przekształcania sygnałów elektrycznych. Zrozumiałe, że charakter procesu przekształcania napisów jest inny od procesu przekształcania sygnałów elektrycznych

i trudno badać proces obliczania z konieczną dla praktyki drobiazgowością bez sprecyzowania czy chodzi nam o przekształcenie napisów czy sygnałów. Równie kłopotliwe jest stosowanie konsekwentnego podziału obliczeń na "ręczne" i "elektronowe", dlatego będziemy się starali przede wszystkim tak formułować zagadnienia, by ich rozwiązanie nie zależało od założeń technicznych, zdając sobie sprawę, że konkretne zastosowania mogą znacznie odbiegać od podanych schematów. Wszędzie tam, gdzie takie uzależnienie się od strony technicznej będzie niemożliwe, podamy wyraźnie warunki techniczne dotyczące realizacji procesu obliczenia.

W dalszym ciągu wprowadzimy pojęcie p r o c e s u, które zależnie od interpretacji może oznaczać obliczenie na liczbach całkowitych, rzeczywistych, zespolonych, wektorach, macierzach, wartościach logicznych czy innych obiektach matematycznych. Pod pojęcie procesu podpada również dowodzenie twierdzeń w systemach sformalizowanych oraz szereg innych p o j ę ć ma - tematycznych. Tak ogólne ujęcie ma dla nas istotne znaczenie, pozwala bowiem na rozważania konstrukcji maszyn matematycznych w znacznym stopniu niezależnie od tego czy są one przeznaczone do obliczeń arytmetycznych, matematycznych, logicznych, macierzowych, czy też do dowodzenia twierdzeń matematycznych.

Definicje podane w dalszym ciągu mają więc postać na tyle ogólną, aby mogły stanowić ewentualnie punkt wyjścia do różnych zastosowań.

§ 1. P r o c e s y.

Procesem $\mathcal{A} = \langle A, O, R \rangle$ nazwiemy skończony zbiór A obiektów.

skończony zbiór O o p e r a c j i określonych w zbiorze A, oraz relację R porządkującą zbiór O.

Jeżeli zbiór O jest dobrze uporządkowany przez relację R, powiemy, że proces $\mathcal{A}$ jest s e k w e n c y j n y.

Jeżeli zbiór O jest częściowo [1] uporządkowany przez relację R, powiemy, że proces $\mathcal{A}$ jest jednoczesny[2].

W dalszym ciągu obiekty będziemy oznaczali małymi literami greckimi, operacje - dużymi literami greckimi.

Proces $\mathcal{A} = \langle A, O, R \rangle$ nazwiemy p r o s t y m jeżeli:

1. Dla każdej operacji $\Delta \in O$ istnieją $1(\Delta), p(\Delta), w(\Delta) \in A$.

---

1. Mówimy, że relacja R porządkuje dobrze skończony zbiór przedmiotów Z, jeżeli dla każdej pary elementów zbioru Z relacja R jest określona; np. relacja mniejszości $<$ dobrze porządkuje skończony zbiór liczb naturalnych $1, 2, \ldots, k$, gdyż dla każdej pary liczb $1 \leq a, b \leq k$, bądź $a < b$, bądź $b < a$.
   Mówimy, że relacja R porządkuje częściowo skończony zbiór przedmiotów Z, jeżeli relacja R jest określana nie dla wszystkich par przedmiotów należących do Z, np. relacja "być synem" porządkuje częściowo zbiór wszystkich ludzi, gdyż zachodzi tylko między rodzicami a ich synem.
2. Innymi słowy, jeżeli w procesie wykonywana jest w każdej chwili tylko jedna operacja mówimy, że proces jest sekwencyjny, natomiast jednocześnie jest wykonywanych kilka operacji - proces nazywamy jednoczesnym.

zwane odpowiednio l e w y m a r g u m e n t e m operacji Δ, p r a w y m a r g u m e n t e m operacji Δ, oraz w y n i k i e m operacji Δ[3].

2. Dla każdego obiektu $\alpha \in A$ istnieje co najmniej jedna jedna taka operacja $\Delta \in O$, że $\alpha = 1(\Delta)$ lub $\alpha = p(\Delta)$ lub $\alpha = w(\Delta)$.

3. Istnieje dokładnie jedno takie $\alpha \in A$, że dla żadnego $\Delta \in O$ nie zachodzi: $\alpha = 1(\Delta)$ ani $\alpha = p(\Delta)$. α nazwiemy w y n i k i e m k o ń c o w y m a Δ o p e r a c j ą k o ń c o w ą p r o c e s u $\mathcal{A}$.

Jeżeli $\alpha \in A$ oraz istnieją $\Delta, \Omega \in O$, takie że $\alpha = w(\Omega)$ oraz $\alpha = 1(\Delta)$ lub $\alpha = p(\Delta)$, to α nazwiemy w y n i k i e m c z ę ś c i o w y m procesu $\mathcal{A}$.

Jeżeli $\alpha \in A$ oraz nie ma takiego $\Delta \in O$, że $\alpha = w(\Delta)$, to α nazwiemy d a n ą p o c z ą t k o w ą procesu $\mathcal{A}$.[4]

Np. jeżeli A jest zbiorem liczba naturalnych, a O - zbiorem działań arytmetycznych określonych na liczbach naturalnych, to jest r a c h u n k i e m l i c z b n a t u r a l n y c h;

---

3. $\Delta \in O$ - należy czytać: Δ jest elementem zbioru O.

4. Przyjęliśmy w definicji procesu prostego, że wszystkie operacje są dwuargumentowe. Nic nie stoi na przeszkodzie, aby rozpatrywać operacje o dowolnej, skończonej ilości argumentów 0, 1, 2..k. /operacja zero-argumentowa będzie wtedy daną/. Wszystkie otrzymane w dalszym ciągu rezultaty bardzo łatwo rozszerzyć na procesy z operacjami wieloargumentowymi. Dla prostoty pozostaniemy w dalszym ciągu przy operacjach dwuargumentowych.

Np. jeżeli A jest zbiorem liczb naturalnych, a O - zbiorem działań arytmetycznych określonych na liczbach naturalnych, to jest r a c h u n k i e m l i c z b n a t u r a l n y c h; jeżeli A jest zbiorem macierzy, O - zbiorem działań macierzowych, to $\mathcal{A}$ jest r a c h u n k i e m m a c i e r z o w y m; jeżeli A jest zbiorem wartości logicznych, O - zbiorem działań logicznych to $\mathcal{A}$ jest r a c h u n k i e m l o g i c z n y m; jeżeli A jest zbiorem aksjomatów, O - zbiorem operacji wnioskowania / np. regułą odrywania i regułą podstawioną/, to jest d o w o d e m f o r m a l n y m[1].

Zbiór A można również interpretować jako zbiór konkretnych przedmiotów fizycznych, np. części samochodu, a zbiór O - jako zbiór operacji składania przedmiotów należących do A; proces $\mathcal{A}$ jest wtedy p r o d u k c j ą. Otrzymane w dalszym ciągu rezultaty można również w pewnym stopniu stosować do organizacji produkcji; jednakże zagadnienie to dość daleko odbiega od poruszanego przez nas tematu, nie będziemy się więc nim zajmować tutaj szczegółowo, a posłużymy się czasem dla ilustracji rozpatrywanych zagadnień. Maszynę matematyczną można bowiem również traktować jako pewnego rodzaju "fabrykę" realizującą proces produkcyjny, gdzie produktami końcowymi są napisy.

1. Przez dowód formalny rozumie się często w matematyce postępowanie odwrotne do opisanego, tj. przechodzenie od wniosków do przesłanek, a nie od przesłanek do wniosków jak podano tutaj. W dalszym ciągu pozostaniemy jednak przy pierwszym rozumieniu słowa dowód, tzn. przez dowód rozumiemy takie postępowanie, które prowadzi od przesłanek do wniosków. Takie postępowanie nazywane jest czasem wnioskowaniem.

Proces przedstawia więc konkretny rachunek, czy konkretny dowód matematyczny. Obiektami procesu w przypadku rachunku wykonanego na liczbach naturalnych - są wszystkie dane, wyniki częściowe oraz wynik końcowy rachunku, a operacjami - wszystkie działania arytmetyczne wykonane w trakcie obliczenia. W procesie dowodzenia konkretnego twierdzenia obiektami są wszystkie przesłanki dowodu wraz z tematami oraz samo twierdzenie, którego dowodzimy. Operacjami dowodu natomiast są wszystkie zastosowania w trakcie dowodu reguły dowodzenia.

Oczywiście w reakcie dowodzenia czy obliczenia jakaś operacja np. dodawanie, może być zastosowana wielokrotnie.

Ponieważ O jest zbiorem wszystkich operacji wykonanych w procesie, więc w O niektóre operacje mogą wystąpić wielokrotnie jak np. wspomniane dodawanie. Podobnie zbiór A, może zawierać kilka egzemplarzy takich samych obiektów, np. liczba 8 może być daną w rozpatrywanym procesie obliczeniowym, jak i jednocześnie jakimś wynikiem częściowym, czy też wynikiem końcowym obliczenia. Zbiór A zawiera wszystkie obiekty występujące w procesie, jakaś liczba może więc występować w zbiorze A wielokrotnie[1].

Relacja R mówi w jakiej kolejności są wykonywane działania procesu. Zagadnieniem kolejności wykonywania działań procesu zajmiemy się nieco dokładniej w jednym z następnych paragrafów.

Jeżeli w procesie $\mathcal{A}$ nie została wykonana jeszcze dana operacja, powiemy, że proces $\mathcal{A}$ jest w s t a n i e p o c z ą-

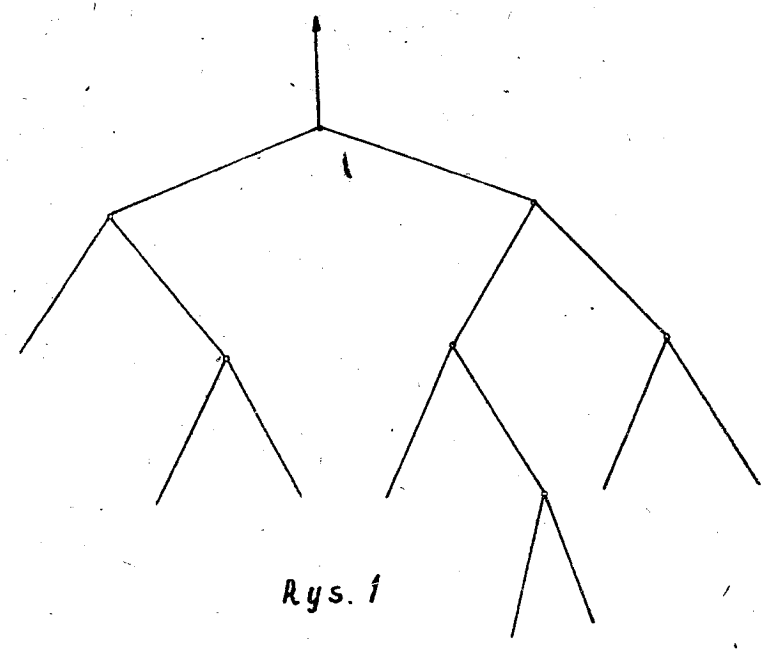
1. Pojęcie operacji ma tutaj więc nieco inny sens niż to się przyjmuje w matematyce.

t k o w y m; jeżeli w procesie $\mathcal{A}$ zostały wykonane wszystkie operacje, powiemy że proces $\mathcal{A}$ jest w s t a n i e k o ń c o w y m; jeżeli $\mathcal{A}$ jest procesem jednoczesnym i w $\mathcal{A}$ została wykonana i-ta operacja powiemy, że proces $\mathcal{A}$ jest w s t a n i e i.

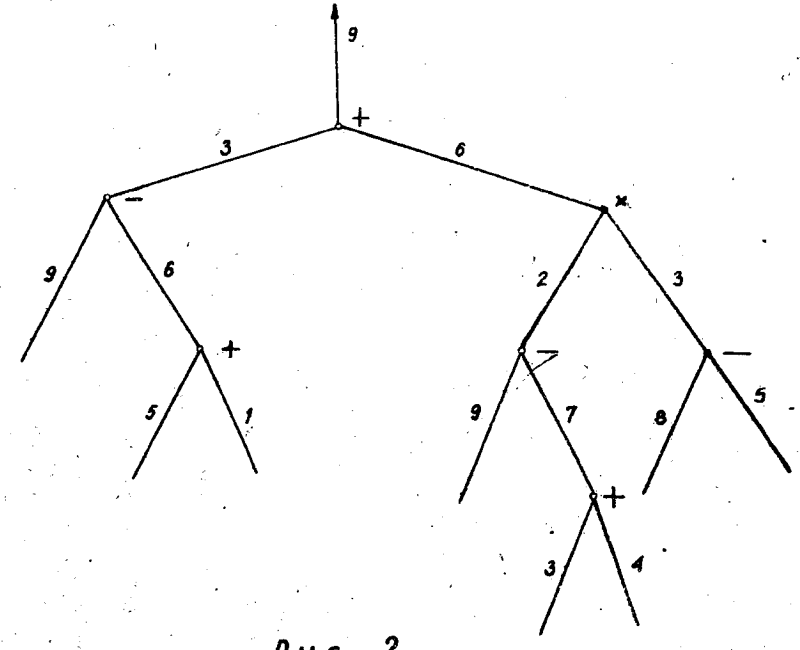
Inna jest maszyna realizująca rachunek logiczny niż maszyna służąca do wykonywania rachunków macierzowych, czy dowodzenia twierdzeń matematycznych. Tym niemniej maszyny te posiadają szereg cech wspólnych. W dalszym ciągu będziemy starali się badać takie własności maszyn matematycznych, które nie należą od rodzaju procesu realizowanego przez maszynę, a więc to co wspólne jest w maszynach, np. do rachowania i dowodzenia twiedzeń. Ewentualne zastosowanie otrzymanych wyników do budowy maszyn matematycznych, musi więc być związane z uwzględnieniem specyfiki procesu obliczeniowego oraz szeregu spraw technicznych, których w zasadzie nie będziemy tutaj poruszali. Będziemy się więc zajmować ogólnymi zasadami realizowania procesów. Zaczniemy od procesów prostych, a w dalszych rozdziałach pojęcie procesu nieco rozszerzamy.

§ 2. P r o c e s y i d r z e w a

Nie będziemy tutaj podawać dokładnej definicji drzewa, którą można znaleźć w dowolnym podręczniku teorii grafów, a poprzestaniemy na wyjaśnieniu tego pojęcia za pomocą przykładów. Przykład drzewa pokazany jest na rysunku 1. Odcinki przedstawiają gałęzie drzewa, punkty natomiast przedstawiają rozgałęzienia. Z formalnego punktu widzenia proces prosty można uważać za roz -



Rys. 1



Rys. 2

rzerzenie pojęcia drzewa. Rozgałęzienia / punkty/ odpowiadają operacjom, gałęzie /odcinki/ - obiektom. Przyjęliśmy taki sposób rysowania drzew, że oba argumenty są rysowane poniżej punktu, przedstawiającego operację wykonaną na tych argumentach; wynik natomiast jest rysowany w górę od odpowiadającej mu operacji. Z rysunku widać wyraźnie, że w procesie operacje nie mogą być wykonywane w zupełnie dowolnej kolejności. Jeżeli argumentem operacji jest wynik częściowy innej operacji, to może być ona wykonana dopiero po otrzymaniu wyniku częściowego z poprzedniej operacji, a więc kolejność wykonania obu operacji nie jest dowolna.

Z formalnego punktu widzenia proces prosty można uważać za rozszerzenie pojęcia drzewa. Różnica między procesem prostym a drzewem jest taka, że w procesie prostym zbiór rozgałęzień jest uporządkowany w przeciwieństwie do drzewa. Nie jest bowiem rzeczą obojętną w jakiej kolejności wykonujemy operacje w procesie, natomiast w drzewie porządek rozgałęzień nie ma znaczeni

Tak więc każdy proces prosty można graficznie przedstaw w postaci drzewa. Sposób ten ma szereg zalet, gdyż pozwala na łatwe uchwycenie całej struktury procesu jednym spojrzeniem oka

1. Aby nie używać dwu terminów " proces prosty " i "drzewo", moglibyśmy wprowadzić pojęcie drzewa uporządkowanego, tj. drzewa, w którym zbiór punktów jest uporządkowany. Pozostajemy jednak przy terminie " proces ", podkreśla on bowiem fakt pewnych zmian w czasie, które z terminem drzewa się nie kojarzą

Tak więc każdy proces prosty można graficznie przedstawić w postaci drzewa. Sposób ten ma szereg zalet, gdyż pozwala na łatwe uchwycenie całej struktury procesu jednym spojrzeniem oka /o ile proces nie jest zbyt duży/, a ponadto umożliwi nam łatwiejsze zrozumienie szeregu spraw dyskutowanych w dalszym ciągu.

Przykład przedstawienia procesu obliczenia za pomocą drzewa przedstawiony jest na rys. 2. Rysunek ten przedstawia proces obliczenia w stanie końcowym, tj. po wykonaniu wszystkich operacji. Narysowanie stanu początkowego tego procesu oraz jego stanów pośrednich nie przedstawia trudności. W procesie tym zbiorem A obiektów są liczby : $\{9,3,6,9,6,2,3,5,1,9,7,8,5,3,4\}$, zbiorem O operacji są działania arytmetyczne : $\{+,-,.,+,-,-,+\}$. Jak to już wspomnieliśmy poprzednio, zbiór A zawiera wszystkie obiekty występujące w procesie, a zbiór O - wszystkie operacje wykonane w procesie.

Procesy proste mają następującą elementarną własność :

J e ż e l i p r o c e s p r o s t y z a w i e r a n o p e r a c j i d w u a r g u m e n t o w a n y c h , t o i l o ś ć d a n y c h w t y m p r o c e s i e w y n o s i n + 1 .

Wynika to stąd, że jeżeli proces zawiera n operacji, to oczywiście zawiera również n wyników częściowych oraz 2n + 1 obiektów. Ponieważ suma danych i wyników częściowych równa jest ilości obiektów, więc ilość danych wynosi 2n +1-n-n+1.

§3. P o r z ą d e k p r o c e s ó w p r o s t y c h .

W paragrafie tym zajmować się będziemy procesami sekwencyjnymi, tj. procesami, w których zbiór operacji jest dobrze uporządkowany. Zagadnienie uporządkowania zbioru operacji sprowadza się do tego, w jakiej kolejności operacje te mają być wykonywane zakładając, że w każdej chwili wykonywana jest tylko jedna operacja.

Dla uproszczenia przyjmijmy, że w każdej operacji Δ procesu $\mathcal{A}$ przypiszemy liczbę naturalną $I(\Delta)$ w ten sposób, że jeżeli $\Delta < \Omega$, to $I(\Delta) < I(\Omega)$, gdzie $\Delta < \Omega$ oznacza, że operacja Δ jest wykonywana po operacji Ω, lub inaczej - operacja Ω, jest wykonywana przed operacją Δ. Tak więc operacja wykonywane później otrzymują mniejsze numery- natomiast operacje wykonywane wcześniej - numery większe [1].

Zagadnienie uporządkowania zbioru operacji sprowadza się więc do odpowiedniej numeracji rozgałęzień drzew.

Np. operacje w procesie przedstawionym na rys.2 możemy ponumerować jak to pokazane jest na rys.3. Operacje przedstawione na rysunku nie kropkami jak to miało miejcse poprzednio, a kółeczkami, w których podano numery działań. Tak więc przebieg obliczenia przedstawionego na rys.3 będzie wyglądał następująco :

7. 5+1 = 6

6. 3+4 = 7

5. 9-7 = 2

1 - Możnaby również numerować operacje w sposób odwrotny, tzn. operacjom wykonywanym wcześniej przypisać numery mniejsze - a operacjom wykonywanym później - numery większe, jednakże pierwszy sposób jest wygodniejszy.

4. 8-5 = 3

3. 2·3 = 6

2. 9-6 = 3

1. 3+6 = 9

Oczywiście można przyjąć również inny sposób numeracji działań. W dalszym ciągu będziemy rozpatrywać tylko cztery sposoby uporządkowania działań procesu, które oznaczymy $p, \bar{P}, w, \bar{W}$ i nazwiemy odpowiednio :

P i $\bar{P}$ - porządki p o p r z e c z n e

W i $\bar{W}$ - porządki w z d ł u ż n e ; ponadto wprowadzimy jeszcze następujące nazwy :

P i W - porządki n o r m a l n e

$\bar{P}$ i $\bar{W}$ - porządki d u a l n e

Zasady numerowania działań w każdym z wymienionych porządków pokazano na rys.: 4,5,6 i 7.

W przypadku numeracji poprzecznej drzewo dzielimy jak gdyby na piętra i działania numerujemy poczynając od piętra najwyższego, kolejno piętrami, tak że znajdujące się na niższych piętrach mają numery większe od dowolnych działań, znajdujących się na piętrach wyższych. Dla porządku normalnego P, na każdym " piętrze " działania są numerowane od strony prawej do lewej, dla porządku dualnego $\bar{P}$ - odwrotnie, tj. od strony lewej do prawej.

W numeracji wzdłużnej działania są numerowane jak gdyby wzdłuż drzewa. Szczegóły wynikają z rysunków : 6 i 7[1].

1- Numeracja rozgałęzień drzewa może być również podana w postaci rokurencyjnej, jednakże wymaga to określenia szeregu dodatkowych pojęć, z których w dalszym ciągu nie zrobilibyśmy użytku - ze ściślejszego określenia numeracji więc zrezygnowano.

Przebieg obliczenia podanego na rys.2 dla porządków $P, \bar{P}, W, \bar{W}$ będzie miał postać :

	Porządek P	Porządek W	Porządek $\bar{P}$	Porządek $\bar{W}$
7.	3+4 = 7	8-5 = 3	3+4 = 7	5+1 = 6
6.	5+1 = 6	3+4 = 7	8-5 = 3	9-6 = 3
5.	9-7 = 2	9-7 = 2	9-7 = 2	3+4 = 7
4.	8-5 = 3	2.3 = 6	5+1 = 6	9-7 = 2
3.	9-6 = 3	5+1 = 6	2.3 = 6	8-5 = 3
2.	2.3 = 6	9-6 = 3	9-6 = 3	2*3 = 6
1.	3+6 = 9	3+6 = 9	3+6 = 9	3+6 = 9

Jeżeli proces jest jednoczesny, przebieg obliczenia na rys.2 może mieć np. postać :

3+4 = 7, 8-5 = 3
5+1 = 6, 9-7 = 2
9-6 = 3, 2.3 = 6
3+6 = 9,

Przyjęliśmy tutaj, że wykonawane są jednocześnie dwie operacje. Operacje wykonywane jednocześnie są zapisane w jednym wierszu. Oczywiście przebieg obliczenia może być również inny od podanego.

§4. P r z y k ł a d y p r o c e s ó w p r o s t y c h .

W poprzednim paragrafie rozpatrywaliśmy przakład obliczenia określonego na liczbach naturalnych.
Podobnie możemy określić obliczenie na liczbach wymiernych, rzeczywistych, zespolonych czy innych.

Przykład obliczenia logicznego pokazany jest na rys.8. Cyfry 0 i 1 oznaczają wartości logiczne. Jedyna operacja występująca w tym obliczeniu - oznaczona strzałką - jest

nazywana implikacją i określona jest następująco :

L	P	W
1	1	1
1	0	0
0	1	1
0	0	1

Litery L,P,W oznaczają odpowiednio lewy i prawy argument oraz wynik działania. Przebieg tego obliczenia dla różnych porządków jest następujący :

6.	0→1 = 1	1→0 = 0	1→0 = 0	0→1 = 1
5.	0→0 = 1	0→0 = 1	0→0 = 1	1→1 = 1
4.	1→0 = 1	0→1 = 1	1→0 = 0	0→0 = 1
3.	1→1 = 1	1→0 = 0	0→1 = 1	1→0 = 0
2.	1→0 = 0	1→1 = 1	1→1 = 1	1→0 = 0
1.	1→0 = 0	1→0 = 0	1→0 = 0	1→0 = 0

Jak wspomnieliśmy, dowodzenie - ściślej wnioskowanie formalne - jest również procesem prostym. Obiektami są tutaj twierdzenia teorii maremetycznej, natomiast operacje to reguły wnioskowania, pozwalające w sposób formalny z twierdzeń już uznanych za prawdziwe otrzymywać nowe twierdzenia. Przykład wnioskowania pokazany jest na rys.9. W procesie tym występują dwie operacje : operacja podstawiania oraz operacja odrywania, które krótko omówimy. W obu operacjach argumentami są wyrażenia i wynikiem jest również wyrażenie. Operacja podstawiania polega na podstawianiu do wyrażenia, będącego prawym argumentem - na miejsce jakiejś litery - wyrażenie, które jest lewym argumentem tej operacji. Otrzymane

po podstawieniu wyrażenie jest wynikiem operacji. Operację podstawienia oznaczamy literą P ze wskaźnikiem u dołu wskazującym, na miejsce każdego symbolu w prawym argumencie dokonujemy podstawienia. Np. P_x oznacza podstawienie na miejsce litery x. Operacja pdrywania polega na odrywaniu od wyrażenia postaci $\alpha \to \beta$ wyrażenia α. Lewym argumentem tej operacji jest wyrażenie α, prawym - wyrażenie $\alpha \to \beta$ a wynikiem - wyrażenie β. Operację odrywania oznaczamy literą O. Dla uproszczenia zarówno dane jak i wyniki częściowe są oznaczone na rysunku 9 literami a,b,c,d,e,f,g,h,i,j. Danymi są następujące formuły :

a. $[p \to q] \to \{[q \to r] \to (p \to r)\}$

b. $[p \to (p' \to q)]$

c. $[(p' \to p) \to p]$

d. $(p' \to p)$

e. p

Postępując według rys.9 jako wynik końcowy otrzymamy formułę :

f. $\{(p \to p)\}$

przy czym wynikami częściowymi są następujące formuły :

g. $[p \to (p' \to p)]$

h. $[p \to (p' \to p)] \to \{[(p' \to p) \to r] \to (p \to r)\}$

i. $\{[(p' \to p) \to r] \to (p \to r)\}$

j. $\{[(p' \to p) \to p] \to (p \to p)\}$

Dla różnych porządków proces dowodzenia możemy przedstawić następująco :

5.	$e\,P_q\,b = g$	$d\,P_q\,e = h$	$d\,P_q\,a = h$	$e\,P_q\,b = g$
4.	$d\,P_q\,a = h$	$e\,P_q\,b = g$	$e\,P_q\,b = g$	$d\,P_q\,a = h$
3.	$g\,O\,h = i$	$g\,O\,h = i$	$g\,O\,h = i$	$g\,O\,h = i$
2.	$e\,P_r\,i = j$	$e\,P_r\,i = j$	$e\,P_r\,i = j$	$e\,P_r\,i = j$
1.	$c\,O\,j = f$	$c\,O\,j = f$	$c\,O\,j = f$	$c\,O\,j = f$

W omawianym przykładzie porządki P i $\bar{W}$ nie różnią się od siebie, podobnie jak i porządki $\bar{P}$ i W, ogólnie jednak porządki te są oczywiście różne [1].

Na zakończenie tego paragrafu jeszcze jeden przykład procesu prostego, tym razem pozamatematycznego.

Przykładem procesu prostego może być splatanie sznurów. Najpierw z pojedyńczych nitek skręcamy cienkie sznurki, z tych sznurków skręcamy liny, z lin powrozy. Obiektami takiego procesu są wszystkie nitki dane na początku oraz wszystkie otrzymane pośrednio. Operacjami natomiast są czynności skręcania. Oczywiście w procesie takim może być wiele różnych splotów, np. skręcanie w lewo, w prawo, splatanie 3,4, czy 5 sznurów itp. Proces taki możemy również przedstawić w postaci drzewa, oznaczając w rozgałęzieniach rodzaj zastosowanego splotu, a gałęzie oznaczając symbolem

---

1 - W matematyce na ogół pojęcie dowodu formalnego definiuje się następująco :ciąg formuł $F_1,\ldots,F_k\ (k > 0)$ nazywamy dowodem formalnym formuły F_k z formuł wyjściowych $D_1,\ldots,D_l\ (l \geqslant 0)$, jeżeli każda formuła F_i jest jedną z formuł $D_1,\ldots,D_l$ lub aksjomatem, albo jest bezpośrednio otrzymana za pomocą ustalonych reguł wnioskowania z formuł ją poprzedzających. Patrz np. S.C. Kleene. Introduktion to Metamathematics. str. 82 /tłum. rosyjskie/.
Podobnie możemy zdefiniować obliczenie.
Dany jest ciąg liczbowy $D_1,\ldots,D_l$. Obliczeniem nazywamy ciąg liczb $L_1, L_2,\ldots,L_k\ (k > 0)$, taki, że każda liczba L_i jest jedną z liczb $D_1,\ldots,D_l$ lub jest otrzymywana bezpośrednio z liczb poprzednich $L_1,\ldots,L_{i-1}$, za pomocą jednej z ustalonych operacji arytmetycznych, np. dodawania, odejmowania, mnożenia, dzielenia.
Takie określenie jest jednak dla naszych celów niewygodne.

użytego do danego splotu sznurka.

§5. P r z y k ł a d y p r o c e s ó w n i e p r o s t y c h

Dla lepszego zrozumienia pojęcia procesu prostego podamy proste przykłady procesów, które nie są prostymi, w myśl podanej poprzednio definicji.

Rozpatrzmy np. dodawanie dwu liczb zapisanych w systemie pozycyjnym

$$\begin{array}{r} 9\ 8\ 4\ 5\ 7 \\ +\ 2\ 5\ 1\ 8\ 9 \\ \hline 1\ 2\ 3\ 6\ 4\ 6 \end{array}$$

Jeżeli w procesie jako argumenty przyjmiemy ciągi symboli 98457, 25189, to dodawanie jest procesem prostym, gdyż z dwu napisów, za pomocą odpowiedniej operacji otrzymamy nowy napis 123646.

Jeżeli jednak będziemy rozpatrywać wymieniony proces jako operację na pojedyńczych cyfrach, a nie jako operację na ciągach liczbowych symboli - to dodawanie nie jest procesem prostym. Danymi w tym procesie są cyfry 9,8,4,5,7, 2,5,1,8,9, i w wyniku otrzymamy cyfry 1,2,3,6,4,6. A więc jako rezultat nie jedną cyfrę a 6 cyfr. Ponadto w wyniku każdej operacji na cyfrach obu argumentów otrzymujemy dwie wielkości : cyfrę wyniku oraz cyfrę przeniesienia. A więc nie jest to zgodne z definicją procesu prostego, gdyż w procesie prostym otrzymujemy jako rezultat każdej operacji jeden obiekt.

Innym przykładem procesu, który nie jest prostym jest dowodzenie twierdzeń w systemie sformalizowanym w następującym przypadku. Dany jest zbiór aksjomatów oraz reguł

wnioskowania. Proces dowodzenia wszystkich możliwych twierdzeń, które można otrzymać z aksjomatów, stosując n razy dozwolone reguły wnioskowania - nie jest procesem prostym. W wyniku otrzymujemy bowiem nie jedno a wieje twierdzeń.

Obu wymienionych procesów nie da się przedstawić za pomocą drzewa.

A więc dowód jednego, konkretnego twierdzenia jest procesem prostym, natomiast proces dowodzenia skończonej ilości twierdzeń w systemie sformalizowanym nie jest już procesem prostym.

W dalszym ciągu tej książki będziemy zajmować się tylko procesami prostymi.

Rozdział II: PROGRAM PROCESÓW PROSTYCH.

W tym rozdziale zajmiemy się sposobami opisywania procesów prostych. Pewnego rodzaju opisem procesów są drzewa, króre wykorzastywaliśmy do przedstawienia procesów, jednakże w dalszym ciągu będziemy zajmowali się opisem liniowym, tj. opiesm, w którym proces jest przedstawiony za pomocą liniowego ciągu symboli. Liniowy opis procesu prostego będziemy nazywali p r o g r a m e m tego procesu. Programy będziemy oznaczali dużymi literami greckimi, np. Φ , Ψ , Γ , itp.

Powiemy, że program Φ jednoznacznie określa proces prosty $\mathcal{A}$, jeżeli :

1. Program Φ wyznacza porządek operacji w procesie $\mathcal{A}$, oraz
2. jednoznacznie określa argumenty każdej operacji.

Z formalnego punktu widzenia problem opisu procesów prostych sprowadza się do zagadnienia, w jaki sposób przedstawiać liniowo strukturę drzew. Można podać szereg różnych sposobów rozwiązania tego zadania. Zajmiemy się tylko takimi, które wydają się mieć pewne znaczenie dla maszyn matematycznych.Ponieważ program jest to ciąg symboli, możemy wprowadzić tu pojęcie j ę z y k a .

Językiem I będziemy nazywali skończony zbiór symboli A zwany alfabetem języka I - wraz ze skończonym zbiorem reguł R, pozwalających z symboli alfabetu tworzyć wyrażenia poprawne, zwane tutaj p r o g r a m a m i .

Zależnie od doboru alfabetu oraz reguł tworzenia pro-

gramów otrzymamy różne języki.

Tak więc program tego samego procesu w różnych językach może mieć różne postacie.

W dalszym ciągu rozpatrzymy kilka języków przydatnych do opisywania procesów prostych.

§1. J ę z y k p o d s t a w o w y / I_1 / .

Niech małe litery łacińskie oznaczają dane, symbol $*$ - wyniki częściowe, a duże litery łacińskie - operacje [1].

Konsekwentne odróżnienie nazw od tego co one oznaczają jest dość kłopotliwe i dlatego nie zawsze rozróżnienie to będziemy przestrzegali. W szczególności nazwy operacji będziemy utożsamiali z samą operacją, gdy nie będzie to prowadziło do nieporozumień. W przypadkach wątpliwych będziemy wyraźnie mówili, czy chodzi o operację czy o jej nazwę.

Ciąg

$$a_{2n+1}\ a_{2n}\ D_n\ a_{2n-1}\ a_{2n-2}\ D_{n-1} \ldots a_3\ a_2\ D_1\ a_1$$

nazwiemy n o r m a l n y m p r o g r a m e m [2] p o d s t a w o w y m procesu $\mathcal{A}$, jeżeli dla każdego i $(1 \leq 2 \leq n)$ $a_{2i+1}\ a_{2i}$ są nazwami lewego i prawego argumentu działania D_i, oraz $D_{i+1} > D_i$, gdzie $>$ oznacza relację porządkującą operacje procesu $\mathcal{A}$ w porządku normalnym, tj. porządku P lub W [3].

---

1 - Używając porównań gramatycznych, małe litery oraz symbol $*$ można by interpretować jako rzeczowniki naszego języka /nazwy obiektów/, a duże litery - jako czasowniki /nazwy czynności/.

2 - Zamiast terminu program, możnyby też używać pojęć: formuła procesu, algorytm procesu lub schemat procesu.

3 - Innymi słowy działania są napisane w kolejności według numeracji P lub W.

Ciąg

$a_1 D_1 a_2 a_3 \ldots . D_{n-1} a_{2n-2} a_{2n-1} D_n a_{2n} a_{2n+1}$

nazwiemy d u a l n y m p r o c e s e m p o d s t a w o w y m procesu $\mathcal{A}$, jeżeli dla każdego i $(1 \leqslant i \leqslant n)$ a_{2i+1}, a_{2i} są nawiasami lewego i prawego argumentu działania D_i, oraz $D_i \prec D_{i+1}$, gdzie $\prec$ oznacza relację porządkującą operacje procesu $\mathcal{A}$ w porządku dualnym, tj. $\bar{P}$ lub $\bar{W}$.

Dla przykładu rozpatrzymy formułę procesu przedstawionego na rys. 10 [1].

Programy tego procesu w języku podstawowym dla różnych dyskutowanych porządków będą miały postać :

afF bcC d*E ghG a*B **D **A* /p/

ghG efF d*E **D bcC a*B **A* /W/

*A**Ba* D** Cbc Ed* Ggh Fef /$\bar{P}$/

*A**D**Ggh Ed* Fef Ba*Cbe /$\bar{W}$/

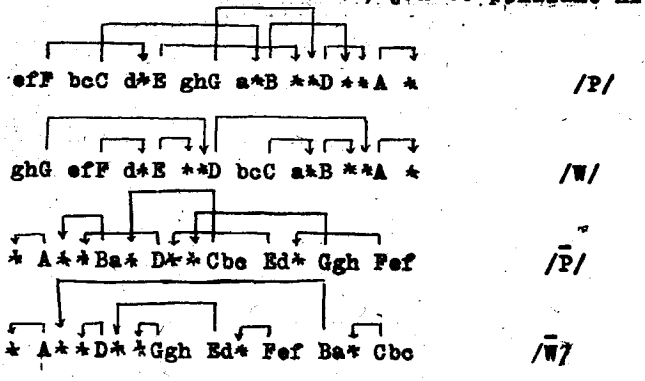
Litery w nawiasach z prawej strony oznaczają porządek procesu. Dla przejrzystości między trójkami symboli porobiono odstępy.

Dla porządków P i W programy należy czytać od strony lewej do prawej, dla porządków $\bar{P}$ i $\bar{W}$ - odwrotnie, tj. od strony prawej do lewej.

Aby podane programy były jednoznaczne musimy wiedzieć, które symbole wyników częściowych oznaczają wyniki każdej operacji.

Ustalenie tej zależności nie przedstawia trudności. Dla każdego porządku, można podać prosta reguły, ustalające zależność między symbolami działań a odpowiadającymi im symbolami wyników częściowych.

Zanim przejdziemy do dokładnego opisania tych reguł, symbol wyniku częściowego każdego działania ustalimy na podstawie drzewa - zaznaczając symbole wyników częściowych odpowiednich działań - strzałkami, jak to pokazano niżej:



Łatwo zauważyć, że narysowanie strzałek wskazujących powiązania między operacjami i wynikami częściowymi może być wykonane bez pomocy drzewa, sylko na podstawie analizy struktury formuły.

Zanim podamy regułę podporządkowującą każdemu symbolowi działania, symbol wyniku częściowego, określimy najpierw indukcyjnie pojęcie symbolu wolnego w programie podstawowym φ.

Niech φ będzie programem podstawowym procesu prostego z porządkiem P,

1. Jeżeli D_i jest symbolem działania o największym numerze przy numeracji P / tj. pierwszym od lewej symbolem działania w programie podstawowym /, a

---

1 - Jeszcze raz przypominamy, że symbole na rysunku nie są obiektami rozpatrywanego procesu a ich nazwami.

jest pierwszym od lewej symbolem wolnym dla symbolu operacji D_i ,symbolicznie $D_i \rightarrow \sigma$.

2. Jeżeli D_j jest dowolnym symbolem działania w programie Φ , to symbol wyniku częściowego jest wolny dla symbolu działania D_j, symbolicznie $D_j \rightarrow \sigma$, jeżeli nie istnieje w programie Φ taki symbol działania D_k, $i \leq k \leq j$, że $D_k \rightarrow \sigma$.

Dla programu Φ procesu s porządkiem W pojęcie symbolu wolnego jest zdefiniowane następująco :

1. Jeżeli D_i jest sybolem działania o najmniejszym numerze przy numeracji W / tj. pierwszym od prawej symbolem działania w programie podstawowym /, a σ jest pierwszym od prawej symbolem wyniku częściowego, to σ jest wolny dla D_i, symbolicznie $D_i \rightarrow \sigma$.
2. Jeżeli D_j jest dowolnym symbolem działania w programie Φ , to symbol wyniku częściowego σ jest wolny dla symbolu działania D_j, symbolicznie $D_j \rightarrow \sigma$, jeżeli nie istnieje w programie Φ takie symbol działania D_k, $j > k > i$, że $D_k \rightarrow \sigma$.

Podobne rsguły można podać dla porządków P i $\bar{W}$.

Obecnie możemy podać zasadę przypisującą każdemu symbolowi działania w programie podstawoamy - symbol odpowiadającego wyniku: częściowego.

Dla każdego porządku w programie podstawowym Φ wynik działania D_i jest oznaczony symbolem wolnym dla symbolu D_i.

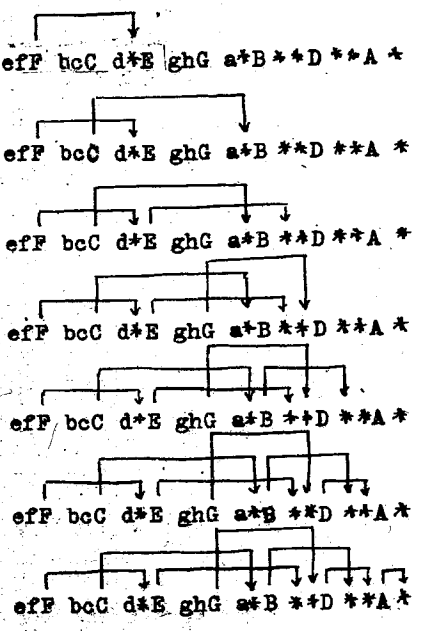
Pisząc krócej, dla obu porządków P i W zasadę stawiania

strzałek możemy sformułować następująco :

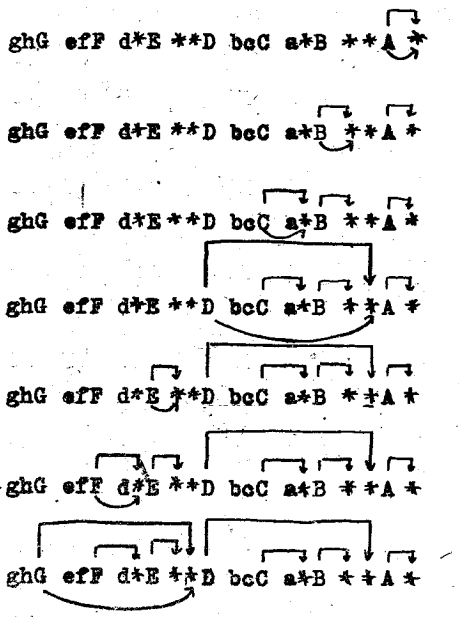
Każdy symbol operacji łączymy strzałką z najbliższym po prawej stronie symbolem wolnego wyniku częściowego : dla porządku P piszemy kolejno strzałki, poczynając od lewej strony formuły, dla porządku W - od strony prawej. Przez wolny symbol wyniku częściowego, rozumiemy symbol nie połączony jeszcze żadną strzałką.

Kolejność stawiania strzałek ilustruje przykład :

Porządek P



Porządek W



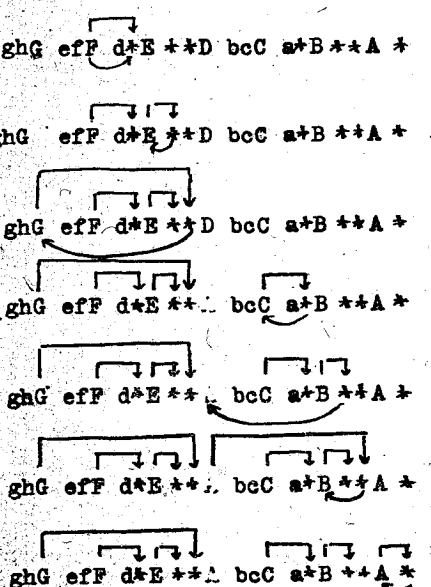
Dla porządku P stawianie strzałek rozpoczynamy od stro-
ny lewej do prawej, dla porządku W - odwrotnie, tj. od str
ny prawej do lewej. W tym ostatnim przypadku nie jest to c
sem wygodne, a więc tylko dla porządku W reguła reguła sta-
wianie strzałek możemy sformułować tak, aby stawianie strz
łek również wykonywać od strony lewej do prawej, podobnie
jak dla porządku P. Reguła stawiania strzałek będzie miała
wtedy postać:

W p r o g r a m i e p o d s t a w o w y m $\bar{\phi}$ p r o -
c e s u $\mathcal{A}$ z p o r z ą d k i e m W , k a ż d y s y m b
w y n i k u c z ę ś c i o w e g o ł ą c z y m y z n a
b l i ż s z y m w o l n y m s y m b o l e m

z i a ł a n i a z l e w e j s t r o n y : r y s o w a -
i w s t r z a ł e k z a c z y n a m y o d p i e r w s z e -
o s y m b o l u z l e w e j s t r o n y .

Zastosowanie ostatniej reguły ilustruje przykład :

Porządek W



Analogiczne reguły możemy podać dla porządków $\bar{P}$ i $\bar{W}$, uwzględniając tylko odwrotny kierunek pisania formuł.

Tak więc jeżeli znamy porządek procesu, program w języku podstawowym jest jednoznaczny.

Dla celów konstrukcji maszyn matematycznych, czasem może być wygodniejsze sformułowanie reguł wyrażających zależność między symbolami działań, a odpowiadającymi im symbolami wyników częściowych w innej postaci.

W programie podstawowym Φ procesu $\mathcal{A}$ z porządkiem P i-temu symbolowi działania odpowiada i-ty symbol wyniku częściowego. Działania i symbole wyników częściowych, w programie są ponumerowane kolejno od lewej do prawej bądź odwrotnie.

Przykład:

	e	f	F	b	c	C	d	*	E	g	h	G	a	*	B	*	*	D	*	*	A	*
Numer działania			1			2			3			4			5			6			7	
Numer wyniku częściowego								1						2		3	4		5	6		7

A więc wynik częściowy o numerze i jest rezultatem działania o numerze i [1].

Dla programów podstawowych procesów z porządkiem W, zasada ta jest nieco inna.

Niech σ oznacza dowolny symbol programu podstawowego Φ procesu $\mathcal{A}$ z porządkiem W - oraz niech σ' oznacza następny po σ symbol programu Φ.

Każdemu symbolowi działania D_i w programie Φ przypiszemy funkcję $F_i(\sigma)$, określoną następująco :

1. $F_i(D_i) = 1$
2. $F_i(\sigma') = F_i(\sigma)$, jeżeli σ jest symbolem danej.
3. $F_i(\sigma') = F_i(\sigma) + 1$, jeżeli σ jest symbolem działania.
4. $F_i(\sigma') = F_i(\sigma) - 1$, jeżeli σ jest symbolem wyniku częściowego.

Można wykazać, że :

Jeżeli Φ jest procesem podstawowym procesu $\mathcal{A}$ z porządkiem W, to wynik częściowy działania D_i jest w programie Φ oznaczony przez taki symbol σ dla którego $F_i(\sigma) = 0$.

Podobne reguły obowiązują dla porządków $\bar{P}$ i $\bar{W}$.

Zastosowanie podanej reguły ilustruje poniższa tabelka [1]:

	g	h	G	e	f	F	d	*	E	*	*	D	b	c	C	a	*	B	*	*	A	*
Nr. działania			1			2			3			4			5			6			7	
F_1			1	1	1	2	2	1	2	1	0											
F_2						1	1	0														
F_3									1	0												
F_4												1	1	1	2	2	1	2	1	0		
F_5															1	1	0					
F_6																		1	0			
F_7																					1	0

1 - Numeracja działań przyjęta w przykładzie jest odwrotna niż w poprzednio podanej definicji porządku P. Gdybyśmy działania ponumerowali od strony prawej do lewej, to otrzymalibyśmy numerację P. Czasem jednak będziemy numerować działania tak jak w przykładzie, gdyż jest on ze względu na kierunek czytania formuły wygodniejszy.

1 - Działania są tu ponumerowane nie w porządku W a odwrotnie, tj. w kolejności ich wykonywania.

W kolejnych wierszach podano wartości funkcji poszczególnych działań.

§2. J ę z y k b e z n a w i a s o w y / I_2 /.

Alfabet języka beznawiasowego składa się z małych oraz dużych liter łacińskich. Małe litery oznaczają dane, a duże litery - operacje. Przyjmiemy, że wyniki częściowe są oznaczane tymi samymi symbolami co odpowiadające im operacje [1]./patrz rys. 11/.

Ciąg

$$a_{2n+1}\ a_{2n}\ a_{2n-1}\ a_{2n-2}\ \ldots\ a_3 a_2 a_1$$

jest n o r m a l n y m p r o g r a m e m b e z n a w i a s o w y m w języku I_2, jeżeli dla każdego i $(1 \leqslant i \leqslant n)$ a_{2i+1}, a_{2i} są symbolami lewego i prawego argumentu działania numer i, przy normalnej numeracji działań, tj. numeracji P lub W.

Ciąg

$$a_1\ a_2\ a_3\ \ldots\ a_{2n-2}\ a_{2n-1}\ a_{2n}\ a_{2n+1}$$

jest d u a l n y m p r o g r a m e m b e z n a w i a s o w y m w języku I_2, jeżeli dla każdego

i $(1 \leqslant i \leqslant n)$ a_{2i}, a_{2i+1}

są symbolami lewego i prawego argumentu działania numer i,

---

1 - W języku tym duże litery nie mają jednego znaczenia; mogą oznaczać zarówno operację jak i obiekt, otrzymany w wyniku operacji. Taka niejednoznaczność prowadzi czasem do nieporozumień. Dla uniknięcia będziemy w przypadkach wątpliwych wyraźnie pisali, czy chodzi nam o operację czy o jej wynik.

przy dualnej numeracji działań, tj. $\bar{P}$ lub $\bar{W}$.

Program procesu przedstawionego na rys. 11 w języku I_2, zależnie od porządku, przedstawimy w jednej z postaci:

ef bc dF gh aC EG BD A /P/

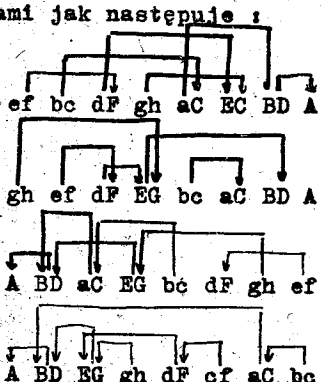
gh ef dF EG bc aC BD A /W/

A BD aC EG bc dF gh ef /$\bar{P}$/

A BD EG gh dF ef aC bc /$\bar{W}$/

Dla jednoczesnego odczytywania programów beznawiasowych musimy wiedzieć, który symbol działania jest skojarzony z każdą parą symboli argumentów.

Dla języka beznawiasowego możemy podać reguły określające sposób ich odczytania, podobnie jak to uczyniliśmy dla języka podstawowego. Nie będziemy ich tutaj podawać. Zainteresowany czytelnik znajdzie je z łatwością samodzielnie. Dla ułatwienia - w poprzednio podanych programach - zaznaczymy działania odpowiadające każdej parze argumentów strzałkami jak następuje :



§3. J ę z y k Ł u k a s i e w i c z a /I_3/.

Logik polski Jan Łukasiewicz wprowadził symbolikę sto-

sowaną czasem w logice matematycznej [1].

Język ten możemy również określić, wychodząc z pojęcia procesu. Przyjmiemy jako alfabet języka I_3 małe oraz duże litery łacińskie, oraz przyjmijmy jak w językach beznawiasowych, że wyniki częściowe są oznaczone odpowiadającymi im symbolami operacji.

Dla określenia języka I_3 przyjmiemy, że obiekty procesu są ponumerowane jak to pokazano na rys. 12 i 13 [2].

Ciąg

$$a_1\ a_2\ a_3\ \ldots\ldots a_{2n-2}\ a_{2n-1}\ a_{2n}\ a_{2n+1}$$

nazwiemy n o r m a l n y m p r o g r a m e m w języku Łukasiewicza, jeżeli dla każdego i

$$\left(1 \leq i \leq 2n+1\right)\ a_i$$

jest symbolem przedmiotu o numerze i, przy numeracji W.

Ciąg

$$a_{2n+1}\ a_{2n}\ a_{2n-1}\ a_{2n-2}\ldots\ldots a_3\ a_2\ a_1$$

nazwiemy d u a l n y m p r o g r a m e m w języku Łukasiewicza, jeżeli dla każdego i $\left(1 \leq i \leq 2n+1\right)\ a_i$ jest symbolem przedmiotu o numerze i, przy numeracji $\bar{W}$.

Przykład programu w języku Łukasiewicza procesu pokazanego na rys.11 - dla porządków W i $\bar{W}$ - ma postać :

A B a C b c D E d F e f G g h /W/

a b c C B d e f F e g h G D A /$\bar{W}$/

---

1 - Patrz Jan Łukasiewicz, Elementy logiki formalnej. PWN 1959.

2 - Zasady tej numeracji są identycznie jak numeracji działań w porządku W i $\bar{W}$, dlatego będziemy je oznaczali również symbolicznie W i $\bar{W}$ i będziemy nazywali numeracją wzdłużną.

Łatwo zauważyć, że w programie normalnym Łukasiewicza operacje są wpisane w porządku W i podobnie dla programu dualnego [1].

W języku Łukasiewicza z porządkiem W lewy argument każdego działania jest oznaczony symbolem sąsiadującym z prawej strony z symbolem działania, natomiast prawy argument znajdujemy na podstawie następującej reguły rekurencyjnej:

1. Jeżeli σ_i jest ostatnim symbolem działania w programie Łukasiewicza, to prawy argument działania σ_i jest oznaczony w programie symbolem σ_{i+2} [2].
2. Jeżeli σ_k jest symbolem działania w programie Łukasiewicza, to prawy argument działania σ_k jest oznaczony takim σ_r, że $k<r$, oraz σ_r nie jest symbolem lewego argumentu dla żadnego działania, oraz nie istnieje w programie żadne działanie σ_t, $t>p$, dla którego σ_r jest symbolem prawego argumentu.

Podobnie możemy określić lewe argumenty w programie Łukasiewicza procesu z porządkiem $\bar{W}$.

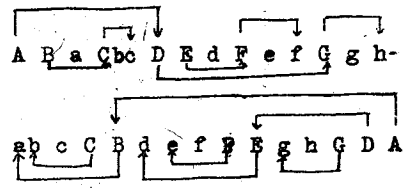
Zaznaczając szukane argumenty każdego działania strzał-

---

1 - Należy pamiętać, że dla określenia języka Łukasiewicza, numerowaliśmy nie operacje w procesie a obiekty. Oczywiście w ten sposób ustaliliśmy jednocześnie porządek operacji. Jednocześnie warto zauważyć, że w procesie z porządkiem W działania są wykonywane w porządku W, jednakże w programie nie są one zapisywane w tym porządku.

2 - Przypominamy, że symbole w programie łukasiewicza procesu z porządkiem W są numerowane kolejno od strony lewej do prawej tak, że σ_i oznacza symbol i-ty w programie, licząc od lewej strony programu.

kami - dla porządków W i $\bar{W}$ w poprzednio podanych przykładach programów otrzymamy:



Prawe argumenty działania i w procesie z porządkiem W możemy również określić za pomocą funkcji : $F_i'(\sigma)$, określonej następująco :

1. $F_i'(\sigma_i) = 0$,
2. $F_i'(\sigma') = F_i'(\sigma) + 1$, jeżeli σ jest symbolem działania
3. $F_i'(\sigma') = F_i'(\sigma) - 1$, jeżeli σ jest symbolem danej

σ_i - jest symbolem działania nr. i ; pozostałe oznaczenia jak poprzednio.

σ - jest symbolem prawego argumentu działania nr. i, jeżeli $F_i'(\sigma) = 0$ [1].

Przykład zastosowania funkcji $F_i(\sigma)$ do określenia prawych argumentów przedstawiony jest niżej :

	A	B	a	C	b	c	D	E	d	F	e	f	G	g	h
numer działania		2		3			4	5		6			7		
F_1	0	1	2	1	2	1	0								
F_2		0	1	0	1										
F_3				0	1	0									

c.d. tabelki-verte!

1 - Zakładamy, że σ nie jest symbolem działania D_i, bo wtedy również $F_i'(\sigma) = 0$.

F_4						0	1	2	1	2	1	0			
F_5							0	1	0						
F_6									0	1	0				
F_7												0	1	0	

Podobnie można określić lewe argumenty w języku Łukasiewicza w przypadku procesu z porządkiem $\bar{W}$.

§4. J ę z y k n a w i a s o w y /I_4/.

Znany, powszechnie w matematyce stosowany język nawiasowy można również wyprowadzić, wychodząc z pojęcia procesu. Wyjaśnimy to na przykładzie procesu pokazanego na rys. 14. Gałęzie drzewa, które odpowiadają danym nazwiemy gałęziami w o l n y m i , pozostałe gałęziami z w i ą z a n y m i . Alfabetem języka nawiasowego są następujące symbole : nawiasy)(, małe oraz duże litery alfabetu łacińskiego. Przyjmiemy, że dane są oznaczone małymi literami, wyniki częściowe - parą nawiasów, jak to pokazano na rys.14, a operacje dużymi literami.

Przyjmijmy, że obchodzimy drzewo wzdłuż linii kreskowanej i postępujemy według następującej reguły :

1. Jeżeli idziemy w dół gałęzi związanej, piszemy nawias otwarty.
2. Jeżeli idziemy w górę gałęzi związanej, piszemy nawias zamknięty.
3. Jeżeli idziemy w dół gałęzi wolnej, piszemy jej nazwę.
4. Jeżeli idziemy w górę gałęzi wolnej - nic nie piszemy.

5. Jeżeli zmienimy kierunek ruchu " z góry w dół "- zapisujemy mijany symbol działania.
6. W innych przypadkach mijanego symbolu działania nie zapisujemy.

Łatwo sprawdzić, że tak otzrymane wyrażenie jest formułą nawiasową rozpatrywanego procesu. Tak więc formuła nawiasowa jest również opisem struktury drzewa, tzn. procesu prostego.

Na podstawie rys.14 otrzymamy więc program :

$$((aB(bCc))\ A\ ((\ dE\ (eFf))\ D\ (gGh)))$$

W programie nawiasowym działania nie są uporządkowane w żadnym z porządków P, $\bar{P}$, W lub $\bar{W}$

Dla języka nawiasowego można podać jednak algorytmy, pozwalające wyznaczyć porządek P, $\bar{P}$, W lub $\bar{W}$ na podstawie analizy formalnej struktury programu. Zanim podamy te reguły, ponumerujemy działania w podanym przykładzie, na podstawie drzewa.

	((aB (bCc))	A	((dE (eFf))	D	(gGh)))		
P	3	6	1	5	7	2	4
$\bar{P}$	2	4	1	5	7	3	6
W	2	3	1	5	6	4	7
$\bar{W}$	6	7	1	4	5	2	3

P o r z ą d e k w z d ł u ż n y . Numery działań dla porządku W są jednoznacznie wyznaczane przez lewe nawiasy, natomiast numeracja $\bar{W}$ jest jednoznacznie wyznaczana nawiasami prawostronnymi.

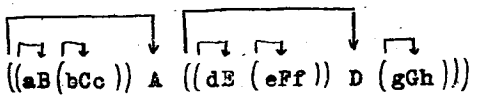
Wyjaśnimy to na przykładzie porządku W. Ponumerujemy nawiasy lewostronne liczbami naturalnymi 1,2,...:k, poczynając od pierwszego nawiasu z lewej strony. Każdemu nawiasowi lewostronnemu przypiszemy najbliższy wolny symbol z prawej strony. Symbol działania jest wolny dla nawiasu nr.i, jeżeli nie jest przypisany żadnemu nawiasowi lewostronnemu o numerze k, gdzie k > i.

Łatwo sprawdzić, że jeżeli działania ponumerujemy liczbami odpowiadających im nawiasów, to otrzymamy numerację W.

Postępując podobnie dla nawiasów prawostronnych, otrzymamy numerację $\bar{W}$.

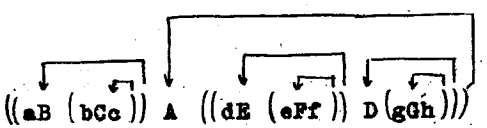
Przykład zastosowania tej zasady dla numeracji działań:

Porządek W



Nr. nawiasu	12	3		45	6		7
Nr. działania		2	3	1	5	6	4	7

Porządek $\bar{W}$



Nr. nawiasu			76			54		321
Nr. działania	6	7	1	4	5	2	3	

Zasady numeracji działań dla porządków wzdłużnych, możemy również zdefiniować w innej postaci. Zasadę tę podamy dla porządku W; dla porządku $\bar{W}$ jest ona podobna.

Przyjmijmy, że lewe nawiasy są ponumerowane jak poprzed-

nio.

Z każdym lewostronnym nawiasem nr. i skojarzymy funkcję $K_i(\sigma)$, określoną rekurencyjnie :

1. $K_i(\sigma) = 1$, jeżeli σ jest lewostronnym nawiasem o numerze i ;
2. $K_i(\sigma') = K_i(\sigma) + 1$, jeżeli σ' jest lewostronnym nawiasem, albo symbolem danej ;
3. $K_i(\sigma') = K_i(\sigma) - 1$, jeżeli σ' jest prawostronnym nawiasem, albo symbolem operacji.

Można wykazać, że $K_i(\sigma) = 1$, to σ jest symbolem działania o numerze i dla porządku W.

Poniższa tabelka podaje wartości funkcji K_i dla dyskutowanego przykładu.

Kolejne nawiasy	(	(	a	E	(	b	C	c	)	)	A	(	(	d	E	(	e	F	f	)	)	D	(	g	G	h	)	)	)
	1	2			3	4					4	5			6							7							
K_1	1	2	3	2	3	4	3	4	3	2	1																		
K_2		1	2	1																									
K_3					1	2	1																						
K_4											1	2	3	2	3	4	3	4	3	2	1								
K_5												1	2	1															
K_6															1	2	1												
K_7																						1	2	1					
Kolejne operacje				2			3				1				5			6				4		7					

P o r z ą d e k p o p r z e c z n y . Obecnie określimy funkcję numerującą działania w programie nawiasowym w porządku $\bar{P}$. Dla określenia porządku P można postąpić

podobnie.

Najpierw określimy funkcję $H(\sigma)$, podporządkowującą każdemu symbolowi formuły liczbę, zwaną rzędem tego symbolu.

σ_i jest i-tym symbolem rozpatrywanego programu.

1. $H(\sigma_1) = 1$.

2. $H(\sigma_{i+1}) = \begin{cases} H(\sigma_i) + 1, \text{ jeżeli } \sigma_{i+1} \text{ jest symbolem danej lub lewostronnym nawiasem,} \\ H(\sigma_i) - 1, \text{ jeżeli } \sigma_{i+1} \text{ jest symbolem działania lub prawostronnym nawiasem.} \end{cases}$

Niech σ_k^p będzie k-tym symbolem rzędu p programu Φ. Wprowadzimy funkcję $G(\sigma_k^p)$, której argumentami są symbole programu Φ, a wartościami liczby naturalne takie, że jeżeli argumentem jest symbol działania σ, to wartość funkcji $G(\sigma)$ jest równa numerowi działania σ dla porządku $\bar{P}$. Funkcja $G(\sigma)$ jest określona rekurencyjnie :

1. $G(\sigma_1^1) = G$

2. $G(\sigma_{k+1}^p) = \begin{cases} G(\sigma_k^p), \text{ jeżeli } \sigma_{k+1}^p \text{ jest nawiasem lub symbolem działania,} \\ G(\sigma_k^p) + 1, \text{ jeżeli } \sigma_{k+1}^p \text{ jest symbolem operacji.} \end{cases}$

3. $G(\sigma_{p+1}^1) = G(\sigma_p^*)$, gdzie σ_p^* oznacza ostatni symbol rzędu p, w programie.

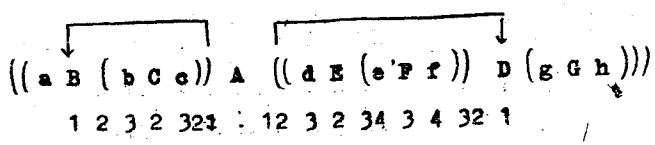
Wartości funkcji H i G dla rozpatrywanego przykładu podane są w tabelce.

Program	(	(	a	E	(	b	C	c	)	)	A	(	(	d	E	(	e	F	f	)	)	D	(	g	G	h	)	)	)
	1	2	3	2	3	4	3	4	3	2	1	2	3	4	3	4	5	4	5	4	3	2	3	4	3	4	3	2	1
	0	1	3	2	3	6	4	6	4	2	1	2	4	6	5	6	8	7	8	7	5	3	8	7	6	7	6	3	1

A r g u m e n t y d z i a ł a ń . Określiliśmy już algorytm numerowania działań w programie nawiasowym w dowolnym z dyskutowanych porzędków : P, $\bar{P}$, W i $\bar{W}$. Pozostało jeszcze określenie algorytmu, przypisującemu każdemu symbolowi działania symbole jego argumentów.

Jeżeli argumentami działań są dane, to symbole tych argumentów są napisane z lewej i prawej strony symbolu działania. Jeżeli natomiast jeden bądź dwa argumenty są wynikami częściowymi, to obok symbolu działania - w programie nawiasowym - są napisane odpowiednie nawiasy, które moglibyśmy przyjąć za symbole wyników częściowych. Musielibyśmy tylko wiedzieć w jaki sposób znaleźć symbole operacji, których są one wynikiem. Oczywiście działaniami tymi są, działania, odpowiadające nawiasom reprezentującym wynik częściowy ; a więc jeżeli nawias jest lewostronny, możemy odpowiadające mu działanie znaleźć za pomocą funkcji K_1, jeżeli jest natomiast - prawostronny, to odpowiadające mu działanie znajdziemy za pomocą funkcji wyznaczającej porządek $\bar{W}$, która jets określona podobnie jak funkcja K_1.

Np. argumentami działania A w dyskutowanym przykładzie są obiekty usyskane w wyniku działań B oraz D, jak to pokazano strzałkami w programie :



W języku nawiasowym program procesu prostego nie zależy od porządku procesu i jest jednakowy dla wszystkich porząd-

dków, w przeciwieństwie do poprzednich języków, w których program był zależny od porządku procesu.

Rozdział III: KLASYFIKACJA JĘZYKÓW.

W poprzednim rozdziale określiliśmy cztery języki przydatne do opisu procesów prostych, które w dalszym ciągu będziemy nazywać językami symbolicznymi, lub krótko - $\mathfrak{G}$ -językami. Dla dalszych rozważań będzie czasem wygodniej pisać programy w nieco innej postaci, aniżeli to przyjęliśmy w rozdziale II. W niniejszym rozdziale podamy jeszcze inne postacie programów dla poprzednio określonych języków.

§ 1. J ę z y k i p r z e d m i o t o w e .

Program określimy jako ciąg symboli oznaczających obiekty oraz operacje, który spełnia pewne warunki.

Jeżeli bowiem w programie - zamiast symboli oznaczających obiekty - występują same obiekty procesu, czy możemy w dalszym ciągu mówić o programie?

Np. czy wyrażenie $((3+5)\cdot 7)$ jest programem?

Oczywiście nie. Nie spełnia ono bowiem definicji programu. Aby zdać sobie wyraźnie sprawę z różnicy między programem a wyrażeniem, w którym zamiast symboli oznaczających obiekty procesu występują, same obiekty, wyobraźmy sobie, że rozważamy proces montażu samochodu. Obiektami tego procesu są elementy i podzespoły samochodu. A więc w naszym przypadku - w programie na miejscu symboli, oznaczających obiekty - należałoby postawić same obiekty, czyli poszczególne części samochodu.

Ponieważ w obliczeniu obiektami są również symbole, rozróżnianie obiektów od oznaczających je symboli może być czasem kłopotliwe, podobnie jak odróżnienie programu od wyra-

żenia, w którym zamiast symboli obiektów występują same obiekty.

W dalszym ciągu dla prostoty wyrażenia, otrzymane z programów - przez zastąpienie w nich symboli obiektów samymi obiektami - również będziemy nazywali programami. Dla odróżnienia od programów symbolicznych nazwiemy je programami p r z e d m i o t o w y m i , a odpowiednie języki - językami p r z e d m i o t o w y m i , lub krócej $\mathfrak{L}$ -językami. Oczywiście program przedmiotowy jest opisem jednego konkretnego procesu, w przeciwieństwie do programu symbolicznego, który opisuje całą klasę procesów.

W języku beznawiasowym oraz w języku Łukasiewicza mamy tylko dwa rodzaje symboli : symbole obiektów oraz symbole operacji / które jednocześnie oznaczają wyniki operacji/. Natomiast w językach : podstawowym oraz nawiasowym mamy trzy rodzaje symboli : symbole danych, symbole wyników częściowych [1], oraz symbole operacji.

A więc w odpowiednich językach przedmiotowych będziemy mieli również dwa albo trzy rodzaje symboli, z tym, że zamiast symboli danych będziemy mieli obiekty. W dalszym ciągu przyjmiemy, że obiektami rozważanych procesów są symbole 0,1,2,...

Np. program przedmiotowy w języku podstawowym procesu, przedstawionego na rys. 2 ma postać :

34+ 51+ 9*- 85- 9*-**.**** /P/

Dla pozostałych porządków $\mathfrak{L}$ -programy będą miały postać

---

[1] - Przyjęliśmy, że nawiasy są również symbolami wyników częściowych.

podobną.

$\tilde{\pi}$ -program przedstawia proces w stanie początkowym.

Analogicznie możemy przedstawić proces w dowolnym stanie.

$\tilde{\pi}$ -program procesu po wykonaniu operacji i będziemy oznaczać $\tilde{\pi}_i$. W szczególności $\tilde{\pi}_0$ oznacza $\tilde{\pi}$ -program procesu, w którym nie zrealizowano jeszcze żadnej operacji.

Np.

$\tilde{\pi}_3$ 34+ 51+ 97- 85- 96- 2*.**+* /P/

Liczby z gwiazdkami u góry są wynikami częściowymi.

Przyjmiemy, że w języku beznawiasowym oraz języku Łukasiewicza wyniki częściowe są zapisywane na miejscu odpowiedniego symbolu działania i wynik częściowy jest również oznaczony gwiazdką u góry. Np. :

$\tilde{\pi}_0$ + - 9 + 5 1 . - 9 + 3 4 - 8 5

$\tilde{\pi}_3$ + - 9 + 5 1 . 2 9 7 3 4 3 8 5

Odczytanie programu $\tilde{\pi}_3$ nie przedstawia trudności. Należy pamiętać, że w przykładzie wszystkie liczby są jednocyfrowe, a więc np. 5 1 oznacza dwie liczby 5 i 1, a nie liczbę " pięćdziesiąt jeden ".

Podobne określenia można wprowadzić do języka nawiasowego.

§2. J ę z y k i u p r o s z c z o n e .

Jeżeli w symbolicznym języku podstawowym przyjmiemy alfabet, składający się z symboli 1,0, oraz symboli działań, gdzie 1 oznacza dowolną daną, a 0 - dowolny wynik częściowy, to tak otrzymany język nazwiemy językiem u p r o s z cz o -

n y m , lub krótko μ -językiem, a programy w μ -języku, nazwiemy μ -programami.

Podobnie, jeżeli w języku beznawiasowym lub języku Łukasiewicza jako alfabet przyjmiemy symbol 1, oznaczający dowolną daną oraz symbol działania, to taki język nazwiemy również językiem uproszczonym.

Jeżeli w symbolicznym języku nawiasowym usuniemy symbole danych, to tak otrzymany język nazwiemy uproszczonym językiem nawiasowym.

Przykład programów w językach uproszczonych :

11F 11C 10E 11G 10B 00D 00A 0 /P/

A B 1 C 1 1 D E 1 F 1 1 G 1 1

((B (C)) A ((E (F)) D (G)))

Drugi -program napisany jest w języku Łukasiewicza.

Warto zwrócić uwagę, że w uproszczonym języku nawiasowym obowiązują - po nieznacznych modyfikacjach - te same zasady numeracji działań co i w symbolicznym języku nawiasowym. Jednocześnie obecnie nawiasy informują o tym, czy argumentami działania są dane czy wyniki częściowe. Jeżeli Δ jest symbolem działania, to rozmieszczenie nawiasów ma następujące znaczenie :

(Δ) - argumenty są danymi,

$(\Delta($ - lewy argument jest daną, a prawy wynikiem częściowym

$)\Delta)$ - lewy argument jest wynikiem częściowym, a prawy daną,

$)\Delta($ - oba argumenty są wynikami częściowymi.

Z niektórymi własnościami języków uproszczonych zapoznamy się w dalszych rozdziałach.

§3. Definicje formalne języków.

W dotychczasowych rozważaniach punktem wyjściowym do określenia języków było pojęcie procesu prostego. Program określiliśmy jako opis procesu prostego. Taką definicję języka możnaby nazwać sematyczną, gdyż odwoływała się ona do znaczenia.

Wszystkie rozpatrywane języki można również określić formalnie bez uciekania się do ich znaczenia, określając tylko jakie ciągi symboli uważamy za wyrażenia poprawne, tj. programy.

Wyjaśnimy to na przykładzie języków symbolicznych. Zaczniemy od podania indukcyjnej definicji programu w $\mathcal{G}$-języku podstawowym.

1. Jeżeli α i β są małymi literami łacińskimi, a Δ -dużą literą łacińską, to wyrażenie $\alpha\,\beta\,\Delta\,*$ jest programem.

2. Jeżeli ϕ i Ψ są programami, to wyrażenie $(\phi)\,[\Psi]$ jest również programem, gdzie (ϕ) oznacza wyrażenie otrzymane z programu ϕ, przez usunięcie w ϕ ostatniego symbolu, a $[\Psi]$ oznacza wyrażenie otrzymane z programu Ψ przez zastąpienie w nim dowolnego symbolu dalej - gwiazdką.

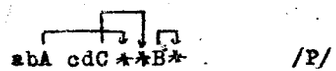
Np. jeżeli ϕ ma postać abA* oraz Ψ ma postać deB*, to (ϕ) jest abA, natomiast $[\Psi]$ = *eB*, a więc wyrażenie $(\phi)[\Psi]$ = = abA*cB* jest programem.

Z definicji indukcyjnej nie wynika jednak, czy jest to program procesu z porządkiem P czy W. A więc definicja indukcyjna języka jest nie jednoznaczna.

Np. formułę

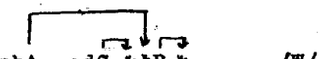
abA cdC **B*

możemy czytać jako



/P/

lub



/W/

Podobnie można zdefiniować indukcyjnie program w języku beznawiasowym oraz w języku Łukasiewicza.

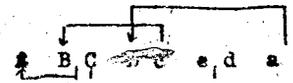
1. Każda mała litera jest programem.

2. Jeżeli α i β są programami, a Δ dużą literą, to $\Delta\,\alpha\,\beta$ jest programem.

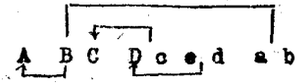
Np. jeżeli α jest Bab oraz β jest Cod, to ABabCcd jest programem. Nie wiemy jednak z definicji indukcyjnej, czy otrzymany program jest w języku Łukasiewicza czy też w języku beznawiasowym i w ostatnim przypadku nie mamy jego porządku. A więc i tutaj również program otrzymany za pomocą definicji indukcyjnej nie określa jednoznacznie procesu prostego.

Np. program ABCDcedab możemy odczytać na jeden ze sposobów.

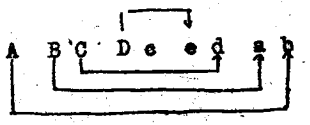
1. Język beznawiasowy, porządek P.



2. Język beznawiasowy, porządek W.



3. Język Łukasiewicza, porządek W



W każdym przypadku program opisuje inny proces.

Definiując indukcyjnie program nawiasowy, nie otrzymamy jednoznaczności, bowiem struktura programi nawiasowego nie zależy od porządku opisywanego procesu. Tak więc język nawiasowy możemy określić formalnie, nie tracąc przy tym jednoznaczności.

Rozdział IV: REALIZACJA JĘZYKÓW PRZEDMIOTOWYCH.

Obecnie zajmiemy się badaniem ogólnej struktury maszyny, realizującej procesy sekwencyjne według programu opisującego ten proces w dowolnym, ale dla danej maszyny ustalonym, języku przedmiotowym.

W dalszym ciągu zamiast pisać : maszyna realizuje proces $\mathcal{A}$, według programu Φ, będziemy pisali krótko : maszyna realizuje program Φ.

Jeżeli maszyna realizuje programy w języku przedmiotowym / $\mathcal{J}$ -języku/, to powiemy, że proces jest realizowany przez maszynę bezpośrednio, a maszynę nazwiemy $\mathcal{J}$ -maszyną.

$\mathcal{J}$ -maszynę realizującą przedmiotowy język podstawowy /I_1/ z porządkiem P - oznaczymy $\mathcal{J}(I_1(P))$. Podobne oznaczenie wprowadzimy dla innych języków i porządków. Np. $\mathcal{J}(I_4(W))$ - oznacza maszynę realizującą przedmiotowy język nawiasowy z porządkiem W.

W niniejszym rozdziale podamy ogólną strukturę $\mathcal{J}$ -maszyny dla różnych podanych w rozdziale II języków.

§1. O g ó l n y s c h e m a t $\mathcal{J}$ - m a s z y n y .

Uproszczony schemat ogólny maszyny realizującej bezpośrednio procesy proste pokazany jest na rys. 19.

Maszyna składa się z trzech zasadniczych elementów

1. O - operatora,
2. P - pamięci,
3. S - sterowania / nie pokazanego na rys./

Operator O realizuje wszystkie operacje występujące w procesie. Jeżeli proces jest rachunkiem liczb naturalnych, operator jest wtedy arytmometrem, wykonującym cztery podstawowe działania arytmetyczne : dodawanie, odejmowanie, mnożenie i dzielenie.[1]

Operator posiada trzy wejścia $\alpha_1, \alpha_2, \Delta$, oraz jedno wyjście α_3 .

α_1 - jest wejściem lewego argumentu

α_2 - jest wejściem prwego argumentu

Δ - jest wejściem symbolu działania

α_3 - jest wyjściem wyniku operacji.

O p e r a t o r może więc być - za pośrednictwem wejścia Δ - nastawiony na wykonanie odpowiedniej operacji. Argumenty operacji są wprowadzane z pamięci do operatora za pośrednictwem wejść α_1 i α_2 . Wynik operacji jest umieszczany w pamięci za pośrednictwem wyjścia α_3 .

P a m i ę ć maszyny możemy sobie wyobrazić jako pokratkowaną taśmę papieru. W każdej kratce może być zapisana liczba, /symbol działania lub inny symbol języka realizowanego przez maszynę.

Trzecim zasadniczym elementem maszyny sekwencyjnej jest

---

1 - Jeżeli proces jest obliczeniem logicznym, to operator wykonuje operacje logicznie; jeżeli proces jest dowodzeniem, to operator wykonuje operacje wnioskowania. W niniejszej książce nie będziemy się bliżej zajmowali budową wewnętrzną operatora, przyjmiemy tylko, że realizuje on potrzebne, wszystkie operacje.

s t e r o w a n i e , które analizuje formułę i na jej podstawie wyszukuje odpowiednie dane z pamięci, wprowadza je do operatora, nastawia operator na operację podaną w formule i po wykonaniu operacji wynik jej umieszcza odpowiednio w pamięci.

Jeżeli proces realizowany przez maszynę jest w stanie i powiemy, że maszyna jest w stanie i.

Wszystkie czynności maszyny, związane z wykonaniem jednej operacji procesu nazwiemy c y k l e m p r a c y m a s z y n y . Działanie maszyny będziemy opisywali przez podanie jej cyklu pracy, oraz stanu początkowego.

Cykl pracy π -maszyny składa się z czterech kroków :

1. Odszukanie w pamięci kolejnego symbolu operacji i nastawienie operatora na odpowiednią operację.
2. Odszukanie w pamięci argumentów nastawionej operacji oraz przesłanie ich do operatora.
3. Wykonanie operacji.
4. Odszukanie w pamięci miejsca, gdzie ma być umieszczony wynik operacji oraz zapisanie w nim otrzymanego wyniku.

W dalszym ciągu nie będziemy zajmować się punktem 3, dlatego będziemy go w cyklu pracy pomijać.

§2. R e a l i z a c j a j ę z y k a p o d s t a w o w e g o z p o r z ą d k i e m p o p. r z e c z n y m .

Opiszemy działanie maszyny tylko dla porządku P; dla porządku $\bar{P}$ - działanie maszyny jest podobne.

Na rys.16 narysowana jest maszyna w stanie początkowym, dla procesu przedstawionego na rys.2.

Stan początkowy.

1. W pamięci maszyny wpisany jest π -program w języku podstawowym.
2. Wejścia operatora O $\alpha_1, \alpha_2, \triangle$ są nastawione na odczytywanie pierwszych trzech / licząc od lewej strony / komórek pamięci.
3. Wyjście α_3 operatora O jest nastawione na zapis w pierwszej komórce, w której wpisany jest symbol wyniku częściowego [1].

Cykl pracy.

1. Wejścia $\alpha_1, \alpha_2, \triangle$ są przesuwane o trzy komórki w prawo.
2. Operator O jest nastawiony na odczytanie działania. Argumenty są przesyłane do operatora.
3. Wyjście α_3 operatora O jest przesuwane do najbliższej z prawej strony niezapisanej kratki pamięci, gdzie zapisujemy wynik działania [2].

---

1 - W tym przypadku moglibyśmy nie wpisywać w programie symbolu wyniku częściowego, gdyż wolna kratka w pamięci wskazuje, że w niej należy zapisać wynik operacji.

2 - W opisie cyklu pracy nie podano sposobu zakończenia procesu.
Przyjmijmy, że w rozpatrywanych językach oprócz symboli operacji jest symbol końca programu / np. przecinek lub kropka / .
Jeżeli sterowanie odczyta symbol końca programu, maszyna przerywa działanie. Sprawą tą jednak nie będziemy się bliżej zajmowali.

Kolejne stany procesu są następujące :

π_0 - 3 4 + 5 1 + 9 * - 8 5 - 9 * - * * . * * + *

π_1 - 3 4 + 5 1 + 9 7 - 8 5 - 9 * - * * . * * + *

π_2 - 3 4 + 5 1 + 9 7 - 8 5 - 9 6 - * * . * * + *

π_3 - 3 4 + 5 1 + 9 7 - 8 5 - 9 6 - 2 * . * * + *

π_4 - 3 4 + 5 1 + 9 7 - 8 5 - 9 6 - 2 3 . * * + *

π_5 - 3 4 + 5 1 + 9 7 - 8 5 - 9 6 - 2 3 . 3 * + *

π_6 - 3 4 + 5 1 + 9 7 - 8 5 - 9 6 - 2 3 . 3 6 + *

π_7 - 3 4 + 5 1 + 9 7 - 8 5 - 9 6 - 2 3 . 3 6 + 9 [1]

Stan maszyny $\pi(I_1(P))$ dla stanu π_7 pokazany jest na rys.

Dla potostałych stanów procesu maszyny można narysować podobnie.

Przedstawiona koncepcja maszyny wymaga analizowania symboli we wszystkich miejscach pamięci, celem znalezienia komórki, w której ma być umieszczony wynik częściowy.

---

1 - W maszynie gwiazdki przy wynikach częściowych nie są pisane tak, że po wykonaniu obliczenia nie wiemy, które liczby są danymi, a które wynikami częściowymi. W programach użyliśmy gwiazdek dla łatwiejszego czytania.

§3. Realizacja języka podstawowego z porządkiem wzdłużnym.

Ogólny schemat oraz zasada działania maszyny $\mathfrak{T}(I_1(W))$ / oraz maszyny $\mathfrak{T}(I_1(\bar{W}))$ / jest identyczna- jak dla języka podstawowego z porządkiem P. Jedynie w inny sposób następuje odszukanie miejsca, gdzie ma być umieszczony wynik częściowy. Dlatego opiszemy tylko tan fragment cyklu pracy., który dotyczy umieszczania wyniku w pamięci.

Szukanie miejsca, w którym ma być umieszczony wynik częściowy można zrealizować na podstawie funkcji $F_1(\sigma)$/Rozdział II,§1/.

Sterowanie posiada więc dodatkowe urządzenie, które realizuje funkcję $F_1(\sigma)$. Urządzeniem tym jest licznik L, do którego można dodawać 0,1 lub -1. Celem znalezienia funkcji, w której ma być umieszczony wynik działania Δ, urządzenie sterujące " bada " zawartoś- wszystkich komórek pamięci, poczynając od komórki, w której znajduje się symbol Δ i zależnie od odczytywanego symbolu, zgodnie ze wzorem /Rozdział II,§1 / dodaje do licznika 0, 1 lub -1. Komórka, dla której licznik L przyjmuje wartość 0 , jest miejscem gdzie należy wpisać wynik częściowy.

A więc również i dla porządku W jest analizowanie wszystkich miejsc pamięci.

Kolejne stany obliczenia dla porządku W są następujące:

$\mathfrak{T}_0$ - 8 5 - 3 4 + 9 * - * * . 5 1 + 9 * - * * *

$\mathfrak{T}_1$ - 8 5 - 3 4 + 9 * - * 3 . 5 1 + 9 * - * * *

$\mathfrak{T}_2$ - 8 5 - 3 4 + 9 7 - * 3 . 5 1 + 9 * - * * + *

$\mathfrak{T}_3$ - 8 5 - 3 4 + 9 7 - 2 3 . 5 1 + 9 * - * * + *

$\mathfrak{T}_4$ - 8 5 - 3 4 + 9 7 - 2 3 . 5 1 + 9 * - * 6 + *

$\mathfrak{T}_5$ - 8 5 - 3 4 + 9 7 - 2 3 . 5 1 + 9 6 - * 6 + *

$\mathfrak{T}_6$ - 8 5 - 3 4 + 9 7 - 2 3 . 5 1 + 9 6 - 3 6 + *

$\mathfrak{T}_7$ - 8 5 - 3 4 + 9 7 - 2 3 . 5 1 + 9 6 - 3 6 + 9

Działanie $\mathfrak{T}$ -maszyny dla języka beznawiasowego jest podobne, dlatego w dalszym ciągu języka tego nie będziemy rozważać.

§4. Realizacja języka Łukasiewicza.

Rozważymy tylko pracę maszyny dla porządku $\bar{W}$. Dla porządku W zasada działania jest identyczna, należy tylko uwzględnić odwrotny kierunek pisania programu.

Schemat ogólny maszyny jest taki sam jak dla języka podstawowego, inaczej odbywa się tylko szukanie symbolu działania, szukanie argumentów oraz umieszczanie wyniku częściowego.

Działanie maszyny jest następujące :

Stan początkowy.

1. W pamięci maszyny znajduje się $\overline{\pi}$ -program w języku Łukasiewicza.
2. Niech symbol pierwszej operacji, która ma być wykonana znajduje się w kratce numer i [1].

   Patrz rysunek 18.

   Wejście Δ jest nastawione na odczyt zawartości kratki i+1 i wejście α_2 -na odczyt kratki, oraz wejście α_1 - na odczyt kratki i+2.
3. Wyjście α_3 jest nastawione na zapis wyniku częściowego w kratce i [2].

Cykl pracy.

1. Wejście Δ jest przesuwane do następnej kolejki kratki pamięci, w której znajduje się następny symbol działania. Wejście α_2 jest przesuwane do następnej - sąsiedniej kratki z lewej strony nowego symbolu działania. Wejście α_1 na podstawie obliczenia wartości funkcji $F_1(b)$ /patrz rozdział II,§3/ jest nastawione na odczyt lewego argumentu [3].

1.- Dla łatwiejszego opisu stanów maszyny oraz cyklu pracy przyjmijmy, że kratki są kolejno ponumerowane. Przyjęcie takiej numeracji w niczym nie zmienia zasady pracy maszyny.

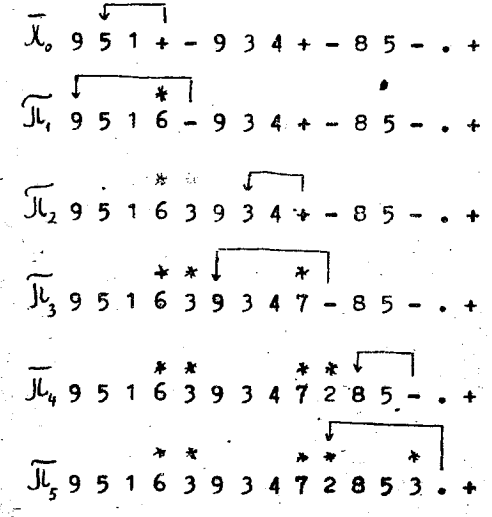
2.- Przyjmiemy, że wyniki częściowe są zapisywane w pamięci na miejsce odpowiadających im symboli działań.

3 - Dla uproszczenia przyjmijmy, że nastawienie wejść operatora na odpowiednie kratki pamięci jest równoważne z nastawieniem operatora na odpowiednie działanie oraz przesłanie do niego aggumentów działania. I podobnie będziemy uważali, że nastawienie wyjścia α_3 na odpowiednią kratkę pamięci jest równoważne z zapisaniem w niej wyniku częściowego.

2. Wyjście α_3 operatora O jest nastawione na zapis w kratce, w której znajduje się aktualny symbol działania.

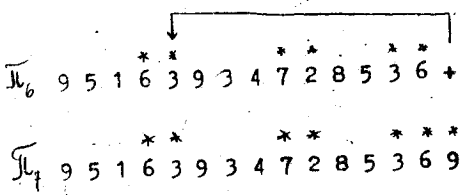
Obliczanie funkcji $F_i'(b)$ jest podobne do obliczania funkcji $F_i(b)$, określającej miejsce umieszczenia wyniku częściowego w języku podstawowym z porządkiem W. A więc maszyna realizująca język Łukasiewicza posiada licznik L, zwany obecnie licznikiem lewego argumentu, którego działanie jest identyczne jak w maszynie poprzedniej, z tą tylko różnicą, że zawartość początkowa licznika jest O, a nie jak poprzednio 1[1].

Kolejne stany procesu są następujące :



---

1 - po częściowym zrealizowaniu obliczenia w pamięci na miejscu symboli działań znajdują się liczby. Aby było możliwe obliczenie funkcji F_i konieczne jest posiadanie informacji o tym, że w danym miejscu pamięci znajdował się symbol działania. Fakt ten zaznaczamy gwiazdką, jak to pokazano na rysunku 18.



Wyniki częściowe są zaznaczone gwiazdkami. Strzałki wskazują lewe argumenty każdego działania.

§5. Realizacja języka Łukasiewicza za pomocą redukcji programu.

Maszynę pracującą w języku Łukasiewicza można również zrealizować na innej zasadzie, którą nazwiemy obliczeniem przez redukcję programu. Niech ϕ będzie programem Łukasiewicza procesu z porządkiem W. Powiemy, że program ϕ jest zredukowany, jeżeli w programie ϕ pierwszy z prawej strony symbol działania oraz odpowiadające mu symbole argumentów zastąpimy jednym symbolem, np. gwiazdką * . Tak otrzymany program możemy znowu zredukować itd. Oczywiście po wykonaniu n kroków redukcyjnych, gdzie n jest liością działań w programie, otrzymamy program składający się z jednego symbolu .
Niech ϕ^i oznacza program i-krotnie zredukowany. W szczególności ϕ^0 przedstawia program niezredukowany. Przykład redukowania programu podany jest niżej.

ϕ^0 A B a C b c d

ϕ^1 A B a * d

ϕ^2 A * d

ϕ^3 *

Redukcja π -programu polega na wpisywaniu na miejsce symbolu operacji jej wyniku oraz wymazywania obu argumentów operacji.

Maszyna działająca na zasadzie redukowania programu, w każdym cyklu dokonuje redukcji programu oraz wykonuje działania [1] w programie zredukowanym.

Schemat takiej maszyny pokazany jest na rysunku

Aby przeczytać program od strony lewej do prawej przyjęto porządek W.

Pamięć maszyny składa się z trzech części A, B, P. Część B składa się z trzech komórek a,b,c. Program wraz z danymi może być przesuwany w lewo, w prawo wzdłuż pamięci. Wejścia operatora są zawsze nastawione na odczytywanie i zapis komórek P, jak to pokazano na rysunku 20. Działanie maszyny jest następujące :

1 - Dla porządku W proces redukcji przebiega identycznie z tym tylko, że redukowane jest zawsze pierwsze działanie z lewej strony.

Stan początkowy.

1. Program Φ jest umieszczony w pamięci P w ten sposób, że symbol pierwszego wykonywanego działania znajduje się w komórce "c" pamięci B.

Cykl pracy.

1. Operator odczytuje symbol działania z komórki c pamięci B, oraz argumenty z komórek a i b pamięci B. Wynik działania jest wpisywany do komórki c pamięci B.
2. Po wykonaniu operacji formuła jest redukowana.

Redukcja programu odbywa się następująco :

1. Zawartość komórek a i b jest kasowana.
2. Część programu znajdująca się w pamięci A jest przesuwana o dwie komórki w prawo tak, że w komórkach a i b znajdują się - po przesunięciu- zawartości dwóch pierwszych komórek z prawej strony pamięci A.
3. Cały program jest przesuwany w pamięci tak długo w lewo, aż w komórce c pamięci B znajdzie się pierwszy symbol działania programu zredukowanego.

Kolejne stany obliczenia w opisanej metodzie mają następującą postać :

$\overline{\pi}_0$ 9 5 1 + - 9 3 4 + - 8 5 - . +

$\overline{\pi}_1$ 9 6 - 9 3 4 + - 8 5 - . +

$\overline{\pi}_2$ 3 9 3 4 + - 8 5 - . +

$\overline{\pi}_3$ 3 9 7 - 8 5 - . +

$\overline{\pi}_4$ 3 2 8 5 - . +

$\overline{\pi}_5$ 3 2 3 . +

$\overline{\pi}_6$ 3 6 +

$\overline{\pi}_7$ 9

§6. Realizacja języka nawiasowego za pomocą redukcji programu.

Działanie $\overline{\pi}$-maszyny realizującej $\overline{\pi}$-język nawiasowy jest podobne do działania maszyn poprzednich. W pamięci maszyny jest zapisany $\overline{\pi}$-program realizowanego procesu. Urządzenie sterujące, zależnie od przyjętego porządku, oblicza kolejność działań, według wzorów podanych w rozdziale II, §4.

Po znalezieniu właściwego symbolu działania urządzenie sterujące odnajduje oba jego argumenty.
Wynik każdej operacji może być wpisywany na miejsce symbolu wykonywanego działania.

Dokładne przestudiowanie tego schematu nie przedstawia trudności.

Maszyna realizująca $\overline{\pi}$-język nawiasowy może również działać na zasadzie redukcji programu.

Redukcja $\overline{\pi}$-programu nawiasowego przebiegać może według porządku P, $\overline{P}$, W, lub $\overline{W}$. Porządków poprzecznych nie będziemy rozpatrywać. Określimy redukcje programu w porządku W. Redukcja dla porządku $\overline{W}$ przebiega podobnie.

Niech Φ^0 będzie programem nawiasowym niezredukowanym. Powiemy, że program Φ^0 jest zredukowany jednokrotnie, symbolicznie Φ^1, jeżeli w programie Φ^0 symbol działania o największym numerze przy numeracji W - wraz z odpowiadającymi mu symbolami argumentów oraz nawiasami - jest zastąpiony symbolem $*$.

Program Φ^1 możemy znowu zredukować itd. Φ^i oznacza program i-krotnie zredukowany. Oczywiście jeżeli program zawiera n symboli działań, to $\Phi^n = *$.

Redukowanie programu nawiasowego w porządku W jest następujące:

Φ^0 (((a C b) B c) A (d E (e F f)))
Φ^1 (((a C b) B c) A (d E *))
Φ^2 (((a C b) B c) A *)
Φ^3 ((* B c) A *)
Φ^4 (* A *)
Φ^5 *

W przykładzie dla ułatwienia, redukowany fragment programu zaznaczono klamrą.

Redukcja π -programu nawiasowego polega na wpisywaniu na miejsce symbolu działania, oraz jego argumentów i nawiasów - wyniku tego działania.

Schemat maszyny $\pi(I_4(W))$, realizującej π -język nawiasowy pokazano na rys.21.

Pamięć maszyny jest podzielona na 3 części A, B, C. Pamięć B posiada pięć komórek a,b,c,d,e. Wejścia i wyjścia

operatora O są na stałe przyłączone do odpowiednich komórek pamięci B, jak to pokazano na rys.20.

Stan początkowy.

1. Przed rozpoczęciem liczenia program jest ustawiony w pamięci w ten sposób, że nawias lewostronny o największym numerze znajduje się w komórce a pamięci B.

Cykl pracy.

1. Zapisanie wyniku operacji w komórce a.
2. Wymazywanie zawartości komórek b,c,d,e.
3. Przesunięcie części formuły, znajdującej się w pamięci C o cztery miejsca w lewo tak, że zawartość komórek b, c, d, e, znajduje się odpowiednio w komórkach b,c,d,e.
4. Przesunięcie całego programu w prawo, aż kolejny nawias lewostronny znajdzie się w komórce a.

Kolejne stany maszyny przy obliczaniu rozpatrywanego przykładu są następujące:

π_0 ((9 - 5 + 1)) + ((9 - (3 + 4)) . (8 - 5)))
π_1 ((9 - 5 + 1)) + ((9 - (3 + 4)) . 3*))
π_2 ((9 - 5 + 1)) + ((9 - 7 .)
π_3 ((9 - 5 + 1)) + (2* . 3*))
π_4 ((9 - 5 + 1)) + 6*)
π_5 ((9 - 6*) + 6*)
π_6 (3* + 6*)
π_7 9*

Dla przejrzystości redukowane wyrażenia zaznaczono strzałkami, oraz wyniki operacji - gwiazdkami.

Oczywiście symbole $*$, $\llcorner\!\lrcorner$ nie należą do $\mathfrak{A}$ -języka.

§7. Uwagi ogólne o bezpośredniej realizacji procesów sekwencyjnych.

Przedstawione w tym rozdziale zasady realizacji maszyn nie mają znaczenia praktycznego z dwu następujących powodów.

1. W praktyce liczby przedstawione są nie za pomocą jednego symbolu, a ciągu symboli /cyfr/, natomiast działania są przedstawiane za pomocą jednego symbolu. W konsekwencji powoduje to szereg trudności technicznych, polegających na niewłaściwym wykorzystaniu pamięci.

2. Drugą zasadniczą wadą dyskutowanych schematów maszyn jest to, że w realizacji języków przedmiotowych konieczne jest wielokrotne analizowanie zawartości komórek pamięci, dla obliczania funkcji określających, gdzie ma być umieszczony wynik częściowy, czy też znalezienie argumentów itp. Ogólnie biorąc czynność ta jest technicznie wysoce czasochłonna i dlatego praktyczne zastosowanie takiej metody jest nieopłacalne.

Dyskusja realizacji bezpośredniej procesów miała na celu zapoznanie czytelnika z pewnymi własnościami języków, które będą przydatne w rozdziałach następnych.

Rozdział V: REALIZACJA JEZYKÓW UPROSZCZONYCH ZA POMOCA MASZYNY Z ODDZIELNA PAMIECIA WYNIKÓW CZEŚCIOWYCH.

Obecnie rozpatrzymy drugą grupę maszyn realizujących procesy proste. Maszyny należące do tej grupy charakteryzują się tym, że realizują one proces obliczeniowy według uproszczonego programu /μ-programu/, znajdującego się w maszynie, oraz posiadają oddzielną pamięć wyników częściowych.

Można podać różne zasady realizowania języków uproszczonych. W tej książce przedyskutujemy dwie z nich, które wydają się szczególnie interesujące ze względów praktycznych.

Pierwszą klasę maszyn oznaczamy przez $\mathcal{M}_1$, drugą przez $\mathcal{M}_2$, a maszyny należące do odpowiednich klas będziemy nazywali maszynami $\mathcal{M}_1$-maszynami lub $\mathcal{M}_2$-maszynami.

W niniejszym rozdziale zostanie opisana klasa $\mathcal{M}_1$, w rozdziale następnym - klasa $\mathcal{M}_2$.

§1. Pojęcie rozkazu.

Zanim opiszemy schemat μ-maszyn określimy pojęcie r o z k a z u programu. Niech ϕ będzie programem uproszczonym i niech D_1 będzie symbolem działania nr.i w programie ϕ. $l(D_1)$ oraz $p(D_1)$ oznaczają symbole lewego i prawego argumentu programu D_1.

i - tym rozkazem programu ϕ, symbolicznie $R_i(\phi)$/lub krótko-R_i/ - nazwiemy trójkę symboli $\{l(D_i), p(D_i), (D_i)\}$.

Oczywiście w μ-programie zawierającym n symboli operacji istnieje n rozkazów.

W języku podstawowym rozkazami są kolejne trójki symboli w programie. Np. w $\mathfrak{B}$ -programie:

a b A c d B $*$+C$*$ /p/

rozkazami są:

R_3 a b A

R_2 C d B

R_1 * * C

W μ -programie rozkazy te będą miały postać:

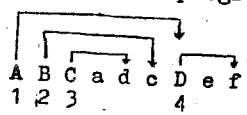
R_3 1 1 A

R_2 1 1 B

R_1 0 0 C.

W językach beznawiasowym oraz Łukasiewicza, symbole rozkazu R_1 nie są napisane obok siebie.

W języku Łukasiewicza procesu z porządkiem W, symbol działania oraz symbol lewego argumentu są napisane w programie obok siebie natomiast symbol prawego argumentu znajduje się w innym miejscu programu. Np. w $\mathcal{L}$ -programie Łukasiewicza



Mamy następujące rozkazy:

R_4 D e f

R_3 C a d

R_2 B C c

R_1 A B D

W języku-μ rozkazy te będą miały postać:

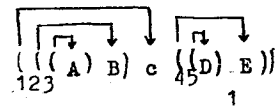
R_4 D 1 1

R_3 C 1 1

R_2 B C 1

R_1 A B D

W μ-języku nawiasowym symbole wszystkich rozkazów są w programie napisane obok siebie, np. w programie



mamy następujące rozkazy:

R_5 -(D)

R_4 -)E)

R_3 -(A)

R_2 -)B)

R_1 -)C(

Rozkazy zostały napisane w porządku W. Oczywiście można je napisać również w innym porządku.

Rozkaz opisuje więc wykonanie jednej operacji w maszynie.

W jednych językach symbole każdego rozkazu są napisane w programie obok siebie, w innych - symbole rozkazu muszą być w programie odszukane.

§2. O g ó l n y s c h e m a t μ_1 - m a s z y n y .

Ogólny schemat μ_1-maszyny jest pokazany na rys. 22. Maszyna składa się z:

1. O - operatora,
2. P - pamięci programu,
3. D - pamięci danych,
4. C - pamięci wyników częściowych,
5. S - sterowania.

Rola i działanie operatora jest identyczne jak w maszynach opisanych w poprzednim rozdziale.

Rola poszczególnych pamięci jest oczywista.

Urządzenie sterujące S analizuje program w pamięci P w kolejności: $p, \bar{P}, w, \bar{W}$. Wynik każdej operacji jest umieszczony

---

1 - Nawiasy z lewej i prawej strony każdego symbolu działania uważamy za symbole argumentów /patrz Rozdział III, §2/.

według odpowiedniego algorytmu w pamięci wyników częściowych C. Argumenty każdego działania, zależnie od tego czy są danymi czy wynikami częściwymi, są pobierane z pamięci D lub C.

Jeżeli maszyna pracuje w języku, w którym działania w programie są zapisane w kolejności p, $\bar{P}$, w, $\bar{W}$, sterowanie wykonuje kolejno operacje programu, w przypadku przeciwnym, porządek operacji jest obliczany na podstawie podanych poprzednio funkcji numerujących operacje, lub na zasadzie redukowania programu.

Wszystkie dane są zapisane w pamięci D w takim porządku w jakim występują one w programie.

Umieszczanie i pobieranie wyników częściowych z pamięci C odbywa się na zasadzie odpowiednich algorytmów. Można wykazać, że algorytm pobierania i zapisywania wyników częściowych nie zależy od zastosowanego języka do opisu procesu prostego, a tylko od porządku wykonawania działań.[1]

W następnym paragrafie zapoznamy się dokładniej z pamięcią wyników częściowych dla porządków poprzecznych i wzdłużnych.

Działanie μ_1-maszyny jest następujące:

Stan początkowy

1. W pamięci znajduje się μ-program realizowanego procesu.
2. Sterowanie jest nastawione na odczytynie pierwszego rozkazu programu.
3. Wejścia α_1, α_2 operatora są nastawione na odczyt obu argumentów działania z pamięci D.
4. Wyjście α_3 operatora O jest nastawione na zapis w pierwszej komórce pamięci C.

1 - Jest to istotna cecha maszyn klasy u_1.

Cykl pracy.

1. Wykonanie rozkazu odczytanego przez urządzenie sterujące S.
2. Odczytanie przez urządzenie sterujące S następnego rozkazu.

Wykonanie rozkazu polega na:

1. Odczytaniu pierwszego argumentu.
2. Odczytaniu drugiego argumentu.
3. Zapisaniu wyniku operacji.[1]

Dla porządków P i W pierwszym odczytywanym argumentem jest lewy argument, dla porządków $\bar{P}$ i $\bar{W}$ - prawy argument.

Działanie μ_1-maszyny dla różnych języków opiszemy szczegółowiej w dalszych paragrafach. W następnym paragrafie rozpatrzymy dokładniej działanie pamięci wyników częściowych C.

§3. P a m i ę ć w y n i k ó w c z ę ś c i o w y c h.

W niniejszym paragrafie określimy dwa rodzaje pamięci wyników częściowych C_1 i C_2, które nazwiemy odpowiednio pamięcią l i n i o w ą i pamięcią r e w e r s y j n ą.

Najpierw określimy pamięć liniową C_1.

Niech C_1 posiada komórki $\{c_1, c_2, \ldots\ldots c_k\}$.

Pamięć C_1 realizuje następujące operacje:

1. Zapis.
2. Odczyt.
3. Przygotowanie zapisu.
4. Przygotowanie odczytu.

Dla uproszczenia przyjmiemy, że w każdej komórce może być zapisany jeden symbol języka /liczbę n cyfrową w zapisie pozycyjnym traktujemy jako jeden symbol./.

1 - Przypominamy, że umówiliśmy się nie podawać jeszcze jednego kroku, a mianowicie- wykonania operacji przez operator O.

W pamięci w każdej chwili może być przygotowana tylko jedna komórka do odczytu, oraz jedna komórka - do zapisu.

W pamięci można zapisać tylko komórkę przygotowaną do zapisu.

W pamięci można odczytać tylko komórkę przygotowaną do odczytu.

Niech c_i będzie komórką przygotowaną do odczytu, c_j - komórką przygotowaną do zapisu w pamięci C_1.

W dalszym ciągu przyjmiemy następujące oznaczenia operacji rozliczeniowych w pamięci C_1:

$(\vec{C_1})$ - odczytaj zawartość komórki c i przygotuj do odczytu komórkę c_{i+1}.

$(\overleftarrow{C_1})$ - odczytaj zawartość komórki c_i i przygotuj do odczytu komórkę c_{i-1}.

$[\overrightarrow{C_1}]$ - zapisz w komórce c_j i przygotuj do zapisu komórkę c_{j+1}.

$[\overleftarrow{C_1}]$ - zapisz w komórce c_j i przygotuj do zapisu komórkę c_{j-1}.

Przy zapisywaniu do pamięci, poprzednia zawartość komórki zostaje skasowana.

Jeżeli pamięć P realizuje operacje $(\vec{C_1})$ lub $[\vec{C_1}]$ /albo $(\overleftarrow{C_1})$ lub $[\overleftarrow{C_1}]$/, to powiemy, że C_1 jest pamięcią liniową.[1]

W szczególnym przypadku pamięć liniowa może realizować tylko operację $(\vec{C_1})$ lub tylko operację $[\vec{C_1}]$.

Jeżeli w pamięci C_1, komórka c_s jest przygotowana do odczytu, a komórka c_t jest przygotowana do zapisu, to powiemy, że pamięć C_1 jest w stanie $\{s,t\}$.

1 - Przyjmiemy, że operacje zapisu i odczytu $(\vec{C_1})$, $[\vec{C_1}]$ nie są wykonywane w pamięci liniowej C_1 jednocześnie a kolejno.

Jeżeli maszyna wykonała tą operację powiemy, że pamięć maszyny jest w stanie $S_1(C_1)$. W szczególności $S_0(C_1)$ oznacza stan pamięci C_1 przed rozpoczęciem liczenia.

Warunkiem koniecznym na to, aby $\mathcal{M}_1$-maszyna realizowała proces w porządku poprzecznym /tj. P lub $\bar{P}$ / jest, aby pamięć wyników częściowych była pamięcią liniową oraz aby $S_0(C_1) = \{i,i\}$, gdzie $1 \leq i \leq k$.

Własność ta wynika bezpośrednio z definicji procesu prostego z porządkiem poprzecznym.

A więc w pamięci liniowej wyniki każdej operacji są umieszczane w kolejnych miejscach $c_1, c_{i+1}, \ldots, c_{i+n}$ pamięci C_1, a argumenty są odczytywane kolejno, poczynając od miejsca c_1[1].

Działanie pamięci liniowej możemy sobie wobrazić następująco: przyjmijmy, że wynik każdej operacji jest zapisany na oddzielnej kartce. Wszystkie kartki z wynikami są układane kolejno jedna po drugiej. Odczytywanie natomiast następuje od kartki położonej najniżej i polega na czytaniu kolejnych kartek od dołu.

Ponieważ wykorzystane wyniki częściowe są z pamięci C_1 nie wymazywane; jeżeli proces zawiera n operacji, w maszynie potrzebna jest pamięć C_1, posiadająca n komórek[2].

1 - Numeracja komórek pamięci jest tutaj nieistotna i służy tylko do opisu działania pamięci. Aby uniknąć numeracji komórek, moglibyśmy działanie pamięci liniowej wyrazić w ten sposób: wynik każdej operacji wpisujemy w najbliższe wolne /tj. nie zapisane / miejsce pamięci ; wyniki są odczytywane w kolejności zapisu. Do zapisania nowego wyniku powinniśmy zaznaczyć tylko, która komórka była zapisana ostatnio. Podobnie przy odczycie: do odczytywania nowego argumentu z pamięci C_1, konieczna jest zaznaczona komórka, która była odczytywana ostatnio.

2 - Ilość komórek pamięci będziemy nazywali jej pojemnością.

Gdybyśmy przyjęli, że po wykorzystaniu każdy wynik częściowy jest z pamięci wymazywany, to można wykazać, że w pamięci liniowej znajduje się jednocześnie co najwyżej $E\left(\frac{n+1}{2}\right)$[1] wyników częściowych[2].

Obecnie rozparzymy drugi rodzaj pamięci wyników częściowych, a mianowicie pamięć rewersyjną lub krótko pamięć C_2.

Pamięć rewersyjna realizuje następujące operacje:

1. Zapis.
2. Odczyt.
3. Przygotowanie / do zapisu lub odczytu /.

W przeciwieństwie do pamięci liniowej - w pamięci rewersyjnej mamy tylko jeden rodzaj przygotowania, niezależny od tego, czy ma być w pamięci wykonana operacja zapisu, czy też odczytu, który będziemy oznaczali strzałką pojedyńczą $\rightarrow$. Wprowadzimy jeszcze dodatkowe oznaczenie $[\vec{c}_2]$ oraz $[\overleftarrow{c}_2]$, które oznaczają odpowiednio:

$[\vec{c}_2]$ - przygotuj do zapisu komórkę c_{j+1}, oraz zapisz w komórce c_{j+1}.

$[\overleftarrow{c}_2]$ - przygotuj do zapisu komórkę c_{j-1} oraz zapisz w komórce c_{j-1}.

Pozostałe oznaczenia jak poprzednio.

Pamięć C_2 nazwiemy rewersyjną, jeżeli pamięć C_2 realizuje operacje: $(\overleftarrow{c}_2)$ i $[\vec{c}_2]$ /albo/ $(\vec{c}_2)$ i $[\overleftarrow{c}_2]$/[3]

1 - $E(x)$ oznacza część całkowitą z x.
2 - Zakładając, że wykorzystane wyniki częściowe są wymazane, oraz, że po zapisaniu ostatniej komórki ponownie zapisujemy pierwszą komórkę; do obliczenia procesu zawierającego n operacji potrzebna jest pamięć liniowa o pojemności $E\left(\frac{n+1}{2}\right)$
3 - Pamięć rewersyjną realizuje obie operacje $(\overleftarrow{c}_2)$ $[\vec{c}_2]$. Operacje mogą być wykonywane kolejno. Jednoczesne wykonanie operacji jest nie dozwolone.

Można wykazać, że:

Warunkiem koniecznym, aby $\mathcal{M}_1$-maszyna realizowała proces w porządku wzdłużnym / tj. W lub $\overline{W}$ / jest, aby pamięć wyników częściowych była pamięcią rewersyjną.

Można wykazać, że jeżeli proces jest realizowany w porządku wzdłużnym, to dla porządku W lub $\overline{W}$ wystarczy pamięć wyników częściowych o pojemności $\lfloor \log_2 n \rfloor$, gdzie n jest ilością operacji w procesie[2].

Pamięć rewersyjna działa więc następująco:

1. Pierwszy wynik częściowy jest wpisany do komórki c_1.
2. Każdy następny wynik częściowy jest wpisany do najbliższej wolnej komórki.
3. Odczytywana jest zawsze zawartość ostatniej zapisanej komórki[2].
4. Po odczytaniu zawartość komórki jest wymazywana.

Gdybyśmy wyniki częściowe zapisywali na oddzielnych kartkach i układali je kolejno na stosie, to odczytanie argumentu polegałoby na odczytaniu i wyjęciu ze stosu ostatniej kartki. Pamięć rewersyjna działa więc jak gdyby odwrotnie do pamięci liniowej.

Warto zauważyć, że działanie pamięci wyników częściowych nie należy od języka, w którym maszyna pracuje, a tylko od porządku w jakim jest proces realizowany.

Czy odczytany argument z pamięci wyników jest lewym czy prawym argumentem, mówi o tym formuła realizowanego procesu.

Ogólnie, dla procesów z porządkiem normalnym odwrotnie, tj.

1 - Należy to rozumieć w ten sposób, że jeżeli dobierzemy dla realizacji procesu odpowiedni porządek / W lub $\overline{W}$ /, to w pamięci C będzie co najwyżej zajętych jednocześnie $\lfloor \log_2 n \rfloor$ komórek. Dla żadnego porządku wzdłużnego jednak ilość zajętych jednocześnie miejsc w pamięci C nie może przekroczyć $\lfloor \log_2 n \rfloor$.
2 - Należy zwrócić uwagę, że chodzi tu o ostatnią zapisaną, a nie zapisywaną komórkę.

odczytywany jest najpierw prawy argument.

§4. Realizacja uproszczonego języka podstawowego z porządkiem poprzecznym / maszyny $\mathcal{M}_1(I_1(P))$ /.

$\mathcal{M}_1$ -maszynę realizującą uproszczony język podstawowy z porządkiem P oznaczamy $\mathcal{M}_1(I_1(P))$.

Rozpatrzymy obecnie szczegółowo działanie maszyny $\mathcal{M}_1(I_1(P))$. Dla porządku P działanie jest analogiczne.

Wszystkie pamięci maszyny $\mathcal{M}_1(I_1(P))$ są pamięciami liniowymi. W języku podstawowym rozkazy są umieszczane w programie w kolejności ich wykonywania.

Zatem odczytywanie programu przez urządzenie sterujące polega na analizowaniu kolejnych trójek symboli w pamięci P. Zależnie od odczytywanego rozkazu następuje pobranie argumentu z pamięci D albo C_1.

W pamięci D magazynowane są wszystkie dane w takim samym porządku, w jakim występują one w I_1 P programie.

Wykonanie rozkazu można opisać następującą tabelką :

$l(\alpha_1')$	$p(\alpha_2')$	1	2	3
0	0	$(\overrightarrow{C_1})$	$(\overrightarrow{C_1})$	$[\overrightarrow{C_1}]$
0	1	$(\overrightarrow{C_1})$	$(\overrightarrow{D})$	$[\overrightarrow{C_1}]$
1	0	$(\overrightarrow{D})$	$(\overrightarrow{C_1})$	$[\overrightarrow{C_1}]$
1	1	$(\overrightarrow{D})$	$(\overrightarrow{D})$	$[\overrightarrow{C_1}]$

$l(\alpha_1'), p(\alpha_2')$ są symbolami lewego i prawego argumentu w odczytanym rozkazie. Liczby 1,2,3 oznaczają kolejne kroki w cyklu pracy.

Realizacja procesu przedstawionego na rys.2 w maszynie $\mathcal{M}_1(I_1(P))$ przebiega następująco :

Rozkaz	Pamięć D								Pamięć C_1						
11+	$\bar{3}$	4	5'	1	9	8	5	9	7'						
11+	3	4	$\bar{5}$	$\bar{1}$	9'	8	5	9	7'	6					
10-	3	4	5	1	$\bar{9}$	8'	5	9	$\bar{7}$	$\bar{6}$'	2				
11-	3	4	5	1	9	$\bar{8}$	$\bar{5}$	9'	7	6'	2	3			
10-	3	4	5	1	9	8	5	$\bar{9}$'	7	$\bar{6}$	2'	3	3		
00+	3	4	5	1	9	8	5	9	7	6	$\bar{2}$	3	3'	6	
00+	3	4	5	1	9	8	5	9	7	6	2	3	$\bar{3}$	$\bar{6}$	9

Apostrofy przy liczbach oznaczają stany pamięci, po wykonaniu odpowiedniego rozkazu[1]. Dla ułatwienia , argumenty wykonanych działań oznaczono kreskami u góry.

§5. Realizacja uproszczonego języka podstawowego z porządkiem wzdłużnym / maszyna $\mathcal{M}_1 I_1(W)$ /

W maszynie $\mathcal{M}(I_1(W))$ pamięci są następujące :

D - pamięć liniowa

P - pamięć liniowa

C_2 - pamięć rewersyjna

1 - Tzn. liczby przygotowane do zapisu / ściślej komórki pamięci przygotowane do odczytu; w których znajdują się zaznaczone liczby./ Komórek przygotowanych do zapisu w pamięci C_1 nie oznaczono dla prostoty.

Tablica cyklu pracy ma postać :

1 (α_1')	p (α_2')	1	2	3
0	0	$(\overleftarrow{C_2})$	$(\overleftarrow{C_2})$	$[\overrightarrow{C_2}]$
0	1	$(\overleftarrow{C_2})$	$(\overrightarrow{D})$	$[\overrightarrow{C_2}]$
1	0	$(\overrightarrow{D})$	$(\overleftarrow{C_2})$	$[\overleftarrow{C_2}]$
1	1	$(\overrightarrow{D})$	$(\overrightarrow{D})$	$[\overleftarrow{C_2}]$

Przebieg procesu obliczenia ma postać następującą :

Rozkaz	Pamięć D								Pamięć C_2
1 1 -	$\bar{8}$	$\bar{5}$	$3'$	4	9	5	1	9	$3'$ $\bar{\ }$
1 1 +	8	5	$\bar{3}$	$\bar{4}$	$9'$	5	1	9	3 $7'$
1 0 -	8	5	3	4	$\bar{9}$	$5'$	1	9	3 $\bar{7}'$
0 0 .	8	5	3	4	9	$5'$	1	9	$\bar{3}$ $\bar{2}'$
1 1 +	8	5	3	4	9	$\bar{5}$	$\bar{1}$	$9'$	6 $6'$
1 0 -	8	5	3	4	9	5	1	$\bar{9}$	6 $\bar{6}'$
0 0 +	8	5	3	4	9	5	1	9	6 $\bar{3}$
									9
0	8	5	3	4	9	5	1	9	9

Na podstawie definicji pamięci rewersyjnej oraz tabelki cyklu prycy, prześledzenie przebiegu procesu nie przedstawia trudności. Oznaczenia jak w maszynie poprzedniej. W pamięci C_2 w górnych wierszach podano stan przed wykonaniem rozkazu, a w dalszych - dolnych - po jego wykonaniu. W rozkazach 00. oraz 00+ ostatnia liczba w pamięci C_2 jest lewym, a

przedostatnia - prawym argumentem.

§6. Realizcja uproszczonego języka Łukasiewicza.

Ponieważ język Łukasiewicza opisuje procesy z porządkiem wzdłużnym, więc pamięć wyników częściowych w dyskutowanej maszynie jest pamięcią rewersyjną.

Pamięć programu P zawiera program obliczenia w języku Łukasiewicza.

Dane są umieszczone w pamięci danych w takim samym porządku, w jakim występują symbole danych w programie Łukasiewicza.

Umieszczanie wyników częściowych oraz pobieranie argumentów z pamięci C_2 odbywa się identycznie jak to opisano w paragrafach poprzednich, nie będziemy więc tego powtarzać. Zajmiemy się natomiast nieco dokładniej opisem pobierania danych z pamięci D, gdyż jest ono inne niż to, które rozważaliśmy do tej pory. Dla przykładu rozpatrzymy maszynę $\mathfrak{M}_1(I_3(\overline{w}))$.

Na rys. 23 jest przedstawiona maszyna $\mathfrak{M}_1(I_3(\overline{w}))$ w stanie początkowym. W pamięci P jest program Łukasiewicza procesu przedstawionego na rys. 2.

Dla prostoty wejścia α_1', α_2' operatora O są narysowane bezpośrednio do pamięci D, a nie za pośrednictwem sterowania S, jak to ma miejsce w rzeczywistości / porównaj schemat na rys. 22 /.

α_1' oraz α_2' oznaczają symbole lewego i prawego argumentu.

Symbole danych są kolejno ponumerowane liczbami 1,2,...8, zwanymi w dalszym ciągu adresami danych.

Pamięć danych D składa się z komórek $d_1, d_2, \ldots\ldots d_8$.

W komórce d_1 znajduje się argument oznaczony w programie numerem i.[1].

Odszukanie kolejnych rozkazów jest oczywiste.

Wykonanie rozkazu opisuje poniższa tabelka :

$l(\alpha_1')$	(α_2')	1	2	3
0	0	$(\overleftarrow{c_2})$	$(\overleftarrow{c_2})$	$[\overrightarrow{c_2}]$
0	1_i	$(\overleftarrow{c_2})$	(d_i)	$[\overrightarrow{c_2}]$
1_i	0	(d_i)	$(\overleftarrow{c_2})$	$[\overrightarrow{c_2}]$
1_i	1_j	(d_i)	(d_j)	$[\overrightarrow{c_2}]$

Zero w tabelce oznacza, że lewy bądź prawy argument jest wynikiem częściowym. Symbole danych napisano z adresami u dołu. (d_1)-oznacza odczytanie i-tej komórki pamięci D.

Realizacja procesu przebiegać będzie następująco :

Rozkaz	Pamięć D								Pamięć C_2			
	1	2	3	4	5	6	7	8				
$1_2\ 1_3$ +	9	$\bar{5}$	$\bar{1}$	9	3	4	8	5	6			
1_1 + −	$\bar{9}$	5	1	9	3	4	8	5	$\frac{6}{3}$			
$1_5\ 1_6$ +	9	5	1	9	$\bar{3}$	$\bar{4}$	8	5	3	7		
1_4 + −	9	5	1	$\bar{9}$	3	4	8	5	3	$\frac{\bar{7}}{2}$		
$1_7\ 1_8$ −	9	5	1	9	3	4	$\bar{8}$	$\bar{5}$	3	2	3	
− − .	9	5	1	9	3	4	8	5	3	$\frac{2}{6}$	$\bar{3}$	
− . +	9	5	1	9	3	4	8	5	$\frac{3}{9}$	6		

§7. Obliczanie adresów danych w uproszczonym języku Łukasiewicza za pomocą pamięci rewersyjnej.

1 - Ponumerowanie symboli danych w programie jest wykonywane również przez maszynę, tak, że w programie adresów nie piszemy. Sprawą tą zajmiemy się w następnym paragrafie.

W maszynie opisanej w poprzednim paragrafie odczytywanie rozkazów programu było dość niewygodne. Ponad to nie podano w jaki sposób obliczane są numery danych.

W niniejszym paragrafie podamy prostą metodę, która pozwala na wyszukanie każdej danej potrzebnej do obliczenia - z pamięci D.

Ogólny schemat maszyny pozostaje bez zmiany.

Wyniki częściowe są umieszczane i odczytywane identycznie, jak poprzednio - w pamięci rewersyjnej C_2.

Dane są również umieszczone w pamięci D identycznie jak poprzednio.

W inny sposób odbywa się tylko analiza programu i obliczanie adresów danych.

Obliczanie adresów danych argumentów odbywa się za pomocą układu przedstawionego na rys. 24. Układ ten składa się z :

C_2' - pamięci rewersyjnej adresów, zwanej w dalszym ciągu skorowidzem adresów.

L - licznika danych.

P' - pamięci liniowej programów.[1]

Uproszczona zasada działania maszyny ze skorowidzem adresów jest następująca:

Sterowanie analizuje kolejno komórki pamięci P'

Jeżeli w odczytanej komórce pamięci P' jest symbol danej, do licznika danych L jest dodawana jedynka i zawartość licznika jest umieszczana w pamięci C_2'.

Jeżeli w odczytywanej komórce pamięci P' jest symbol działania, operator jest nastawiony na odczytane działanie,

1 - W poprzednio omówionej maszynie pamięć programów nie była pamięcią liniową.

oraz pobrane są argumenty według adresów odczytanych ze skorowidza [1].

Po wykonaniu operacji i zapisaniu jej wyniku w pamięci C_2 - w skorowidzu C_2 jest napisane zero.

Jeżeli adresem argumentu jest zero - argument jest odczytywany z pamięci wyników częściowych C_2'.

Działanie skorowidza ilustruje następująca tabelka :

Program	Adres danej	Skorowidz C_2' przed wykonaniem działania				po wykonaniu działania		
1	1	1						
1	2	1	2					
1	3	1	2	3				
+		1	$\bar{2}$	$\bar{3}$		1	0	
-		$\bar{1}$	$\bar{0}$			0		
1	4	0	4					
1	5	0	4	5				
1	6	0	4	5	6			
+		0	4	$\bar{5}$	$\bar{6}$	0	4	0
-		0	$\bar{4}$	$\bar{0}$		0	0	
1	7	0	0	7				
1	8	0	0	7	8			
-		0	0	$\bar{7}$	$\bar{8}$	0	0	0
•		0	0	$\bar{0}$		0	0	
+		$\bar{0}$	$\bar{0}$			0		

W każdym wierszu tablicy podano stan skorowidza. Jeżeli symbolem odczytanym z programu jest symbol działania - podano

1- To znaczy , że argumenty są pobierane według dwu ostatnich adresów w skorowidzu.

stany skorowidza przed i po wykonaniu działania[1].

Adresy z kreskami u góry oznaczają argumenty każdego działania. Jeżeli adres jest równy zeru , argument jest odczytywany z pamięci C_2, według algorytmu pracy pamięci rewersyjnej.

Dla łatwiejszego odczytywania w tablicy podano również adresy wszystkich danych oraz strzałkami zaznaczono lewe srgumenty każdego działania - o ile są one danymi.

W maszynie ze skorowidzem adresów czytanie danych jest znwcznie prostsze, aniżeli w maszynie, w której adresy obliczane są na innej zasadzie.

§8. Realizacja uproszczonego języka Łukasiewicza za pomocą redukcji programu.

Działanie maszyny realizującej program Łukasiewicza za pomocą redukcji jest w zasadzie podobne do działania maszyny opisanej w poprzednim paragrafie, z tą jedynie różnicą, że odszukiwanie kolejnych rozkazów odbywa się na zasadzie redukcji. A więc pamięć P jest zbudowana podobnie jak pamięć maszyny przedstawionej na rys. 20 ; zamiast danych są tutaj zapisywane w pamięci symbole danych.

Przebieg redukowania programu jest identyczny jak to opisano w paragrafie 5,rozdziału IV.

§9. Realizacja uproszczonego języka nawiasowego.

Zależnie od przyjętego porządku obliczenia pamięć wyników

1 - Zwracamy uwagę , że komórka ,w której zapisamo zero, nie jest równoważna komórce pustej.

Częściowych może być pamięcią liniową bądź rewersyjną. Porządek działań oraz symbole argumentów każdego działania można obliczyć na podstawie wzorów podanych w rozdziale II. Bliżej tym się zajmować nie będziemy.

Symbole danych można również kolejno ponumerować w programie i w tej samej kolejności umieścić dane w pamięci D. Pobieranie danych będzie więc identyczne, jak w maszynie realizującej język Łukasiewicza.

W wypadku zrealizowania obliczenia w kolejności wzdłużnej łatwo jest otrzymać kolejne rozkazy na zasadzie redukcji formuły.

Jeżeli w uproszczonym programie nawiasowym procesu przedstawionego na rys. 2 , symbole danych kolejno ponumerujemy, otrzymamy:

$$((-\ (\cdot)) + ((-\ (+))\ \cdot\ (-)))$$

adresy danych

Należy zwrócić uwagę , że ten sam nawias względem jednego działania gra rolę symbolu wyniku częściowego, względem innego natomiast - rolę symbolu danej. Np. $(-\ (+$ nawias 2 mówi, że prawym argumentem działania jest wynik częściowy, oraz ,że lewym argumentem działania + jest dana.

A więc kolejne rozkazy w porządku W, w podanym programie mają postać:

$$\begin{array}{l} 7\,(-_8 \\ 5\,(-_6 \\ 4\,(-(\\ )\cdot(\\ 2\,(+_3 \\ 1\,(-(\\ )+(\end{array}$$

Nawiasy z adresami oznaczają dane , nawiasy bez adresów

- wyniki częściowe.

Podana zasada realizacji języka nawiasowego jest technicznie dość niewygodna. W następnym paragrafie zapoznamy się z praktyczniejszą realizacją języka nawiasowego.

§10. Obliczanie adresów danych w języku nawiasowym za pomocą pamięci rewersyjnej.

Podobnie jak to uczyniliśmy w języku Łukasiewicza - w języku nawiasowym możemy również zastosować pamięć rewersyjną do obliczenia adresów danych. A więc schemat całej maszyny pozostaje w zasadzie bez zmiany, ale dochodzą tylko dodatkowe urządzenia, służące do obliczania adresów danych, jak to pokazano na rys. 25. Rola pamięci P_1' C_2' oraz licznika L jest identyczna jak w maszynie ,realizującej język Łukasiewicza , natomiast pamięć C_2 jest również pamięcią rewersyjną i służy do magazynowania symboli operacji. Pamięć C_2nazwiemy skorowidzem adresów, natomiast pamięć C_2' skorowidzem operacji.

Dla łatwiejszego opisu działania maszyny zmodyfikujemy uproszczony język w ten sposób, że na miejsce każdego symbolu danej, będziemy pisać dowolny inny symbol, np. 1. A więc poprzednio rozpatrywany program przepiszemy w postaci:

$$((1 - (1 + 1)) + ((1 - (1 + 1)) \cdot (1 - 1)))$$

Działanie maszyny jest następujące : program znajduje się w pamięci liniowej P ; sterowanie analizuje kolejno symbole programu i zależnie od odczytywanego symbolu wykonuje następujące czynności :

	Symbol odczytany	Czynności
1.	(	1. Odczytanie następnego symbolu.
2.	1	1. Dodanie do licznika L jedynki i zapisanie liczby, znajdującej się w L, w pamięci C_2'. 2. Odczytanie pamięci P'.
3.	Symbol działań	1. Zapisanie odczytanego symbolu działania w pamięci C_2. 2. Odczytanie pamięci P'.
4.	)	1. Odczytanie symbolu działania w pamięci C_2. 2. Nastawienie operatora na odczytanie działania. 3. Odczytanie dwu ostatnich adresów z pamięci C_2' i wpisanie na miejsce przedostatniego adresu 0^1. 4. Pobranie argumentów operacji w/g odczytanych adresów.. 5. Wykonanie operacji. 6. Zapisanie wyniku operacji w pamięci C_2. 7. Odczytanie pamięci P'.

Adres zero oznacza $(\overleftarrow{C_2})$.

Działanie maszyny ilustruje tabelka :

1 - Przypominamy , że w pamięci rewersyjnej po odczytaniu, zawartość komórek jest wymazywana, przy czym zera nie utożsamiamy z komórką pustą.

Program	Adres danej	Skorowidz adresów przed wykonaniem operacji				Skorowidz operacji S_o		
(								
(								
1	1	1						
-		1				-		
(		1				-		
1	2	1	2			-		
+		1	2			-	+	
1	3	1	$\bar{2}$	$\bar{3}$		-	+	
)		$\bar{1}$	$\bar{0}$			-		
)		0						
+		0				+		
(		0				+		
(		0				+		
1	4	0	4			+		
-		0	4			+	-	
(		0	4			+	-	
1	5	0	4	5		+	-	
+		0	4	5		+	-	+
1	6	0	4	$\bar{5}$	$\bar{6}$	+	-	+
)		0	4	$\bar{0}$		+		
)		0	0			+		
*		0	0			+	*	
(		0	0			+	*	
1	7	0	0	7		+	*	
-		0	0	7		+	*	-
1	8	0	0	$\bar{7}$	$\bar{8}$	+	*	-
)		0	$\bar{0}$	$\bar{0}$		+	*	
)		0	0			+		

Stan skorowidzów jest podany po wykonaniu odpowiednich działań. Pamięć wyników częściowych jest pamięcią rewrsyjną C_2. Dane są w pamięci D, w komórkach o odpowiadających im adresach. Kreski nad adresami mają znaczenie identyczne jak poprzednio.

jak poprzednio [1].

Wróćmy obecnie do języka uproszczonego, bez specjalnych symboli, oznaczających dane.

Zauważmy, że w takim języku obowiązuje następująca reguła rozstrzygania kiedy nawias jest symbolem danej :

1. Nawias (- jest symbolem danej, jeżeli po nim następuje w programie symbol operacji np. (+ .
2. Nawias) - jest symbolem danej, jeżeli przed nim w programie jest symbol operacji +) .

Teraz już nie trudno podać zasadę działania skorowidza adresów i skorowidza operacji. Zainteresowany czytelnik rozwięże to zadanie z łatwością samodzielnie.

§11. U w a g i o g ó l n e o $\mathcal{M}_1$ - m a s z y n a c h.

Najciekawszym uproszczonym językiem - z technicznego punktu widzenia - jest język podstawowy l_1. Analiza programu, pobieranie danych jest tu bardzo proste i polega na badaniu kolejnach komórek pamięci. Szczególnie interesujące są porządki wzdłużne, gdzie potrzeba niewielkiej pamięci wyników częściowych.

Język Łukasiewicza i nawiasowy jest mniej wygodny do realizacji technicznej, ze względu na niewygodne rozmieszczenie symboli rozkazów w programie, oraz ze względu na konieczność obliczania numeracji komórek, w których znajdują się akzualnie potrzebne argumenty.

Wyszukiwanie rozkazów oraz obliczanie numerów danych

1- Zauważmy, że nawiasy lewostronne nie odgrywają tu żadnej roli. Mogą więc w programie nie występować. Języki nawiasowe, zawierające tylko lewostronne bądź prawostronne nawiasy, zostały wprowadzone przez prof. Kalmara.

jest szczególnie niewygodne w języku nawiasowym.

Język Łukasiewicza jest pod tym względem nieco wygodniejszy.

Szczególnie ciekawa jest realizacja maszyny ze skorowidzem adresów ; wymaga ona bowiem kolejnego analizowania symboli programu.

Realizacja języka nawiasowego ze skorowidzem adresów jest tutaj dość wygodna technicznie. Najprostszym jednak do realizacji technicznej jest niewątpliwie język podstawowy z porządkiem W lub $\bar{W}$.

Języka beznawiasowego tutaj nie dyskutowaliśmy. Język beznawiasowy jest jak gdyby językiem pośrednim między językiem podstawowym oraz językiem Łukasiewicza. Zainteresowany czytelnik zbada z łatwością samodzielnie, działanie $\mathcal{M}_1$-maszyny, która realizuje język beznawiasowy.

Rozdział VI : REALIZACJA JĘZYKÓW UPROSZCZONYCH ZA POMOCA MASZYNY Z PAMIĘCIA ROBOCZA.

W rozdziale tym podamy drugi sposób realizowania języków uproszczonych. Jest on szczególnie wygodny dla języków Łukasiewicza oraz nawiasowego. Nasze rozważania ograniczymy więc tylko do tych dwóch języków. Maszyny należące do omawianej klasy oznaczamy przez $\mathcal{U}_2$ wraz z podaniem języka realizowanego przez maszynę. Np. $\mathcal{U}_2(I_3(W))$ oznacza maszynę z pamięcią roboczą, realizującą język Łukasiewicza w porządku W.

§1. O g ó l n y s c h e m a t m a s z y n y .

Ogólny schemat jest identyczny jak schemat maszyny z pamięcią wyników częściowych /rys.22/, z tą jedynie różnicą, że działanie maszyny jest obecnie inne.

Rozważanie nasze ograniczymy w dalszym ciągu tylko do porządków wzdłużnych.

$\mathcal{U}_2$ -maszyna składa się z :

1. O - operatora
2. D - liniowej pamięci danych
3. R - rewersyjnej pamięci roboczej
4. S - sterowania
5. P - liniowej pamięci programu

W maszynach dyskutowanych w poprzednim rozdziale C_2 była pamięcią wyników częściowych, obecnie natomiast R gra nieco inną rolę, chociaż zasada pamięci R pozostała bez zmiany. Pamięć robocza zawiera - w trakcie liczenia - zarówno dane jak i wyniki częściowe, natomiast pamięć wyników częściowych mogła zawierać tylko liczby otrzymywane w trakcie wykonawania operacji.

W rozdziale tym będzie wygodniej przyjąć nieco inne pojęcie rozkazu aniżeli to uczyniliśmy poprzednio. Mianowicie rozkazem będziemy w niniejszym rozdziale nazywali k a ż d y s y m b o l p r o g r a m u .

$\mathcal{U}_2$-maszyna analizuje kolejne rozkazy programu i postępuje zależnie od odczytanego rozkazu. Działanie maszyny opiszemy, podając sposób realizowania wszystkich możliwych rozkazów.

§2. R e a l i z a c j a u p r o s z c z o n e g o j ę z y k a Ł u k a s i e w i c z a .

Dla języka Łukasiewicza działanie maszyny jest następujące :

	Odczytany rozkaz	Czynność $\mathcal{U}_2$ -maszyny
1.	Symbol danej.	1. Odczytaj pamięć D. Odczytaną liczbę zapisz w pamięci R. 2. Odczytaj następny rozkaz.
2.	Symbol operacji.	1. Wykonaj odczytaną operację na dwu ostatnich liczbach, w pamięci roboczej R. 2. Wynik operacji zapisz na miejscu przedostatniej liczby w pamięci roboczej R. 3. Odczytaj następny rozkaz.

Jak widzimy, zasada działania maszyny jest bardzo prosta. Tablicę tę możnaby zapisać krócej następująco :

Rozkaz	Czynność $\mathcal{M}_2$-maszyny
Symbol danej	$(\vec{D}) \Rightarrow [\vec{R}]$, $(\vec{P})$
Symbol operacji	$(\vec{R})\ \Delta\ (R)\ [\vec{P}]$, $(\vec{P})$

Symbol $\Rightarrow$ oznacza przesyłanie.

Działanie maszyny dla rozpatrywanego przykładu obliczeniowego ma następujący przebieg :

	Pamięć D.								Pamięć R. przed				po		
	9	5	1	9	3	4	8	5							
1	x								9						
1		x							9	5	1				
1									9	5					
+									9	5	1		9	6	
=									9	6			3		
1				x					3	9					
1					x				3	9	3				
1						x			3	9	3	4			
÷									3	9	3	4	3	9	7
−									3	6	7		3	2	
1							x		3	2	8				
1								x	3	2	8	5			
=									3	2	8	5	3	2	3
•									3	2	3		3	6	
+									3	6			9		

Krzyżyki w tabelce wskazują, którą dana jest przepisywana z pamięci danych D do pamięci roboczej R.

Tabelka jest bardzo prosta i nie wymaga dodatkowych wyjaśnień.

§3. Realizacja uproszczonego języka nawiasowego.

Działanie maszyny rozpatrzymy dla języka zmodyfikowanego, tj. zawierającego specjalne symbole dla danych.

Maszyna $\mathcal{M}_2(I_4(W))$ /lub $\mathcal{M}_2(I_4(\bar{W}))$ / zawiera skorowidz operacji S_o nie pokazany na rysunku, który działa tak, jak to podano w poprzednim rozdziale.

Znaczenie poszczególnych rozkazów jest jak następuje :

Rozkaz	Czynność maszyny
(	1. Odczytaj następny rozkaz z P'.
1	1. Odczytaj kolejno daną z pamięci D i zapisz ją w pamięci roboczej R. 2. Odczytaj następny rozkaz z P'.
Symbol działania	1. Zapisz odczytany symbol działania w skorowidzu operacji S_o. 2. Odczytaj następny rozkaz z P'.
)	1. Wykonaj ostatnią operację zapisaną w skorowidzu operacji S_o na dwu ostatnich liczbach, zapisanych w pamięci roboczej. 2. Wynik wykonanej operacji zapisz w pamięci roboczej R / na miejsce przedostatniego argumentu /. 3. Odczytaj następny rozkaz z P'.

Zapisując podane zasady symboliczne otrzymamy :

Rozkaz	Czynności $\mathcal{M}_2$-maszyny
(	$(\vec{P'})$
1	$(\vec{D}) \Rightarrow [\vec{R}]$ $(\vec{P'})$

Symbol operacji	$\Delta \Rrightarrow [\overrightarrow{S_o}]$
)	$(\overleftarrow{S_o})\ (\overleftarrow{R})\ (\overleftarrow{R}) \Rrightarrow [\overrightarrow{R}]\ (\overrightarrow{P'})$

Działanie U_2-maszyny realizującej uproszczony język nawiasowy - dla dyskutowanego przykładu - przebiega następująco :

Program	Pamięć danych D 9	5	1	9	3	4	8	5	Pamięć robocza R				Skorowidz operacji S_o		
(															
(															
1	x								9					.	
−									9				−		
(									9	5			−		
1		x							9	5	$\bar{1}$		−		
+									9	$\bar{5}$	$\bar{1}$		−	+	
1			x						$\bar{9}$	$\bar{6}$			−	+	
)									9				−		
)									3						
+									3				+		
(									3				+		
(									3				+		
1				x					3	9			+		
−									3	9			+	−	
(									3	9			+	−	
1					x				3	9			+	−	
+									3	9	3		+	−	+
1						x			3	9	$\bar{3}$	$\bar{4}$	+	−	+
)									3	$\bar{3}$9	$\bar{7}$		+	−	
)									3	2			+		
·									3	2			+	·	
(									3	2			+	·	
1							x		3	2	8		+	·	−

−									3	2	8		+	·	−
1								x	3	2	$\bar{8}$	$\bar{5}$	+	·	−
)									3	$\bar{2}$	$\bar{3}$		+	·	
)									$\bar{3}$	$\bar{6}$			+		
)									9						

Widzimy, że tutaj również są zbędne lewe nawiasy ; nie powodują one bowiem żadnych czynności w maszynie.

Można również, jak to uczyniliśmy w poprzednim rozdziale, zastosować tutaj język uproszczony, nie zawierający specjalnych symboli danych, jednakże podane rozwiązanie wydaje się do zastosowań praktycznych wygodniejsze.

§4. U w a g i o g ó l n e o m a s z y n a c h z p a m i ę c i ą r o b o c z ą .

Opisana grupa maszyn ma szereg zalet technicznych. Najważniejszą z nich jest to, że symbole programu są tu czytane kolejno, a więc nie jest konieczne wielokrotne czytanie programu. Pamięć danych D jest pamięcią liniową, a więc dane są również czytane kolejno. Obie te własności znacznie upraszczają konstrukcję maszyny, kosztem oczywiście czasu działania. Ta sama dana jest bowiem najpierw odczytywana z pamięci D, a następnie zapisywana w pamięci R i przechowywana w pamięci R tak długo, aż będzie potrzebna do obliczenia. Następnie znów jest ona odczytywana i zapisywana w operatorze. A więc uproszczenie analizy programu zostało okupione wielokrotnym zapisem i odczytem danych tak, aby we właściwym czasie znajdowały się one na odpowiednim miejscu.

Tym niemniej do wielu zastosowań takie rozwiązanie wydaje się celowe. Zresztą schemat ten można nieco zmodyfikować, tak, aby ilość odczytów i zapisów została zredukowana do niezbędnego minimum. Nie będziemy tego bliżej dyskutowali, gdyż sprawa ta wiąże się z działaniem arytmometru, o którym - założyliśmy - nie będziemy bliżej dyskutować.

Dla języka podstawowego oraz beznawiasowego, realizacja z pamięcią roboczą jest również możliwa, ale mniej wygodna niż realizacja z pamięcią wyników częściowych tak, że tutaj jej nie podajemy.

Rozdział VII : REALIZACJA JĘZYKÓW UPROSZCZONYCH ZA POMOCĄ MASZYNY Z ADRESOWĄ PAMIĘCIĄ ROBOCZĄ.

W rozdziale tym przedstawimy jeszcze jedno rozwiązanie maszyny z pamięcią roboczą szczególnie przydatne do języka Łukasiewicza oraz języka nawiasowego. Zasadę tę można również stosować do innych języków, jednakże nie posiada ona takich zalet jak w przypadku języków Łukasiewicza i nawiasowego. Dlatego przedstawiamy tutaj tylko dwa wymienione ostatnio języki.

§1. O g ó l n y s c h e m a t m a s z y n y .

Uproszczony schemat maszyny jest przedstawiony na rys.26.

Maszyna składa się z :

1. O - operatora
2. S - sterowania
3. R - adresowej pamięci roboczej
4. P' - liniowej pamięci programu.

Działanie operatora O oraz pamięci P jest identyczne jak w rozdziałach poprzednich.

Pamięć R zawiera w stanie początkowym wszystkie dane rozmieszczone odpowiednio w pamięci.

Urządzenie sterujące analizuje program w pamięci P', oblicza adresy obu argumentów każdego działania i adres każdego wyniku częściowego, oraz przesyła argumenty z pamięci R do operatora O i umieszcza wynik operacji w pamięci R - pod obliczonym adresem.

§2. R e a l i z a c j a j ę z y k a Ł u k a s i e w i c z a.

Przyjmijmy, że wszystkie dane oraz wyniki częściowe są

dowolnie ponumerowane tak jednak, aby różne obiekty miały różne numery. Najwygodniej ponumerować je kolejno tak, jak występują ich symbole w programie. Np. przyjmiemy następującą numerację :

Program	1	1	1	+	-	1	1	1	+	-	1	1	-	.
Adres	1	2	3	4	5	6	7	8	9	10	11	12	13	14

W stanie początkowym dane są umieszczane pod odpowiadającymi im adresami w pamięci R.

Obliczanie adresów argumentów oraz wyników częściowych najwygodniej zrealizować za pomocą skorowidza adresów. Działanie skorowidza jest identyczne jak skorowidza przedstawionego na rys.24.

Maszyna działa następująco :

Rozkaz	Czynność maszyny
1	1. Zapisz adres odczytanego symbolu w skorowidzu. 2. Zwiększ liczbę w liczniku L o jeden. 3. Odczytaj następny rozkaz.
Symbol operacji	1. Odczytaj dwa ostatnie adresy ze skorowidza S_a . 2. Odczytaj z pamięci R komórki o adresach odczytywanych ze skorowidza. 3. Wykonaj odczytanie operacji. 4. Zapisz wynik operacji w pamięci R pod adresem wskazanym przez licznik L. 5. Zapisz zawartość licznika L w skorowidzu adresów. 6. Dodaj do licznika L jeden. 7. Odczytaj następny rozkaz.

Przypominamy, że odczytywane adresy są ze skorowidza wymazywane tak, że adres wyniku jest wpisywany na miejsce przedostatniego adresu.

Zasadę pracy ilustruje przykład :

Program	Adres	Pamięć robocza R															Skorowidz adresów S_a			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15				
1	1	9	5	1			9	3	4			8	5				1			
1	2	9	5	1			9	3	4			8	5				1	2		
1	3	9	5	1			9	3	4			8	5				1	2	3	
+	4	9	5	1	6		9	3	4			8	5				1	4		
-	5	9	5	1	6	3	9	3	4			8	5				5			
1	6	9	5	1	6	3	9	3	4			8	5				5	6		
1	7	9	5	1	6	3	9	3	4			8	5				5	6	7	
1	8	9	5	1	6	3	9	3	4			8	5				5	6	7	8
+	9	9	5	1	6	3	9	3	4	7		8	5				5	6	9	
-	10	9	5	1	6	3	9	3	4	7	2	8	5				5	10		
1	11	9	5	1	6	3	9	3	4	7	2	8	5				5	10	11	
1	12	9	5	1	6	3	9	3	4	7	2	8	5				5	10	11	12
+	13	9	5	1	6	3	9	3	4	7	2	8	5	3			5	10	13	
•	14	9	5	1	6	3	9	3	4	7	2	8	5	3	6		5	14		
+	15	9	5	1	6	3	9	3	4	7	2	8	5	3	6	9	15			

Widzimy, że pamięć robocza jest w trakcie realizowania programu kolejno zapełniana tak, że po zakończeniu obliczenia wszystkie komórki są zajęte. Adresy oraz argumenty każdej operacji są zaznaczone kreskami u góry.

§3. Realizacja języka nawiasowego.

W języku nawiasowym numeracje danych i wyników częściowych przeprowadzimy w sposób, jak pokazano na przykładzie :

Program $((1 - (1 + 1)) + ((1 - (1 + 1)) \cdot (1 - 1)))$

Adresy 3 4 5 6 7 8 9 10 11 12 13 14 15

A więc numerujemy kolejno symbole danych oraz nawiasy prawostronne.

Sterowanie maszyny realizującej język nawiasowy posiada skorowidz adresów S_A oraz skorowidz operacji S_o. / patrz rys.25./ oraz licznik adresów L.

Maszyna działa następująco:

Rozkaz	Znaczenie/czynność maszyn/
(	1. Odczytaj następny rozkaz.
1	1. Zawartość licznika L zapisz w skorowidzu adresów S_A. 2. Dodaj do licznika adresów L jeden. 3. Odczytaj następny rozkaz.
Symbol operacji	1. Zapisz odczytany symbol operacji w skorowidzu operacji S_o.
)	1. Odczytaj ostatni symbol operacji ze skorowidza operacji S_o. 2. Odczytaj dwa ostatnie adresy ze skorowidza adresów. 3. Odczytaj argumenty z pamięci R według adresów odczytanych ze skorowidza adresów.

4. Wykonaj odczytaną operację.
5. Zapisz wynik operacji w pamięci R pod adresem wskazanym przez licznik adresów.
6. Zapisz zawartość licznika adresów w skorowidzu adresów.
7. Odczytaj następny rozkaz.

Tablica ilustrująca wykonanie kolejnych rozkazów programu jest podobna jak dla języka Łukasiewicza, dlatego nie będziemy jej powtarzać.

Rozdział VIII : REALIZACJA JĘZYKÓW SYMBOLICZNYCH.

W językach symbolicznych różne dane są oznaczone różnymi symbolami. Ponieważ symbole danych można ponumerować, zamiast o symbolach, możemy mówić o ich numerach. Numery te będziemy nazywali adresami. Dla człowieka wygodniejsze jest posługiwanie się symbolami niż adresami, z punktu widzenia konstrukcji maszyny natomiast wygodniejsze jest używanie adresów.

W dalszym ciągu - w celu uproszczenia terminologii - wprowadzimy pojęcie adresu symbolicznego oraz adresu liczbowego. Np. litery x,y w programie $(x+y)$ - są adresami symbolicznymi. Natomiast jeżeli literze x przypiszemy numer $A(x)=5$, a literze y - numer $A(y)=10$, to program $(x+y)$ możemy zapisać w postaci $(5+10)$. Adresy zapisano tu tłustym drukiem, dla odróżnienia od liczb, na których wykonywane są operacje arytmetyczne.

W pozostałej części tego rozdziału będziemy używać pojęcia adresu symbolicznego - jednakże należy pamiętać, że w maszynie faktycznie na miejscu adresów symbolicznych występują w programie adresy liczbowe.

Pojęcie adresu ma tutaj nieco inny sens, aniżeli podaliśmy to w poprzednim rozdziale. Obecnie adres to symbol danej. Poprzednio adresem był numer symbolu danej - nie występujący w programie a obliczany przez maszynę; dane były przy tym numerowane kolejno. Obecnie sposób numeracji danych jest całkowicie dowolny.

Istotna różnica między obadwoma pojęciami adresów polega na tym, że poprzednio adresy porządkowały zbiór danych, obecnie natomiast warunek ten nie jest spełniony. Dane mogą być

ponumerowane w zupełnie dowolny sposób, byle różne dane miały różne numery.

Taka koncepcja adresu narzuca od razu określone rozwiązanie maszyny.

Najwygodniejszym wydaje się tu schemat maszyny z oddzielną pamięcią wyników częściowych.

Ogólny schemat maszyny, realizującej dowolny język symboliczny jest taki sam jak na rys.22.

Działanie $\mathcal{B}$-maszyny różni się od działania $\mathcal{M}_1$-maszyny jedynie sposobem odczytywania danych w pamięci D.

Pozostałe elementy działają bez zmiany, tj. pamięć wyników częściowych - zależnie od przyjętego porządku - jest pamięcią liniową lub rewersyjną, a analiza $\mathcal{B}$ - programu przez urządzenie sterujące może się odbywać na jednej z podanych poprzednio zasadsch.

W pamięci danych D wszystkie komórki są oznaczone dowolnymi symbolami danych, np. kolejnymi literami alfabetu łacińskiego[1].

Każda komórka jest oznaczona inną literą alfabetu, przy czym porządek oznaczenia nie gra tutaj roli.

Jeżeli argumenty działania mają adresy, np. z i y, to argumenty tego działania znajdują się w pamięci D, w komórkach oznaczonych x i y.

Wyniki częściowe są zapisywane i odczytywane identycznie jak w $\mathcal{M}$-maszynie.

1- Jeżeli komórek w pamięci jest więcej niż liter alfabetu, to do oznaczenia komórek mogą być użyte kombinacje, dwu, trzech, czy więcej liter.

§1. P o b i e r a n i e d a n y c h .

Działanie $\mathcal{G}$ -maszyny dla języka podstawowego jest oczywiste.

W przypadku zastosowania języka Łukasiewicza powstaje problem odszukiwania jednego z argumentów każdego działania. Najprostsze wydaje się tu również zastosowanie skorowidza adresów. Z tym, że w obecnym przypadku adresy danych nie są obliczane za pomocą licznika L /rys.24/, a w skorowidzu umieszczane są bezpośrednio adresy z programu. Ilustruje to poniższa tabelka :

Program	Skorowidz adresów			
a	a			
b	a	b		
c	a	$\bar{b}$	$\bar{c}$	
+	$\bar{a}$	$\bar{0}$		
-	0			
d	0	d		
e	0	d	e	
f	0	d	$\bar{e}$	$\bar{f}$
+	0	$\bar{d}$	$\bar{0}$	
-	0	0		
g	0	0	g	
h	0	0	$\bar{g}$	$\bar{h}$
)	0	$\bar{0}$	$\bar{0}$	
.	$\bar{0}$	$\bar{0}$		
+	0			

Podobnie jak poprzednio po wykonaniu działania oba adresy argumentów są ze skorowidza usuwane, a na miejsce przedostatniego argumentu wpisywane jest zero, które tutaj nie jest adresem, lecz wskazuje, że odpowiedni wynik znajduje się w pamięci wyników częściowych.

Dla przejrzystości adresy każdej operacji są oznaczone

kreskami u góry.

Podobnie działa maszyna dla symbolicznego języka nawiasowego. Skorowidz adresów i operacji zapewniają najprostszą realizację techniczną.

Realizacja programu nawiasowego ma wtedy postać następującą :

Program	Skorowidz adresów S_a				Skorowidz operacji S_O		
(							
(							
a	a						
-	a				-		
(	a				-		
b	a	b			-		
+	a	b			-		
c	a	$\bar{b}$	$\bar{c}$		-	+	
)	$\bar{a}$	$\bar{0}$			-	+	
)	0						
+	0				+		
(	0				+		
(	$\bar{0}$				+		
d	$\bar{0}$	d			+		
-	0	$\bar{d}$			+	-	
(	0	$\bar{d}$			+	-	
e	0	$\bar{d}$	e		+	-	
+	0	d	e		+	-	+
f	0	$\bar{d}$	$\bar{e}$	$\bar{f}$	+	-	+
)	0	$\bar{d}$	$\bar{0}$		+	-	
)	0	0			+		
.	0	0			.		
(	0	0			.		
g	0	0	g		.		

-	0	0	g		.	-	
h	0	0	g	h	.	-	
)	0	0	0		.		
)	0	0					
)	0		/				

Oznaczenia w tablicy są jak poprzednio.

Zrozumienie działania $\mathcal{G}$-maszyny, realizującej język nawiasowy nie przedstawia trudności, należy tylko pamiętać, że litery w programie oznaczają adresy w pamięci danych D.

§2. A d r e s y x i z m i e n n e .

Działanie $\mathcal{G}$-maszyny można więc przyrównać do postępowania rachmistrza, który realizuje program obliczenia według następującego schematu :

$$(a + b) \cdot c$$

$$a = 5$$

$$b = 7$$

$$c = 8$$

Najpierw czyta program i następnie odszukuje litery, oznaczające dane - na liście, znajdującej się pod programem - i znalezione liczby nastawia w arytmometrze.

Możliwy jest również inny schemat obliczenia, a mianowicie najpierw wszystkie dane są zapisywane na miejsce odpowiednich symboli, a następnie wykonywany jest rachunek według tak otrzymanego $\mathcal{T}$-programu. Ta druga metoda jest dla celów maszynowych znacznie mniej przydatna i nie będziemy jej rozważać.

Na marginesie obu schematów postępowania warto zwrócić uwagę na następujący fakt. W matematyce symbole danych nazywane są zmiennymi, co ma odzwierciadlać fakt, że na miejsce

odpowiedniej litery można wpisać dowolną liczbę.

Natomiast w języku nawiasowym możliwa jest również inna interpretacja liter występujących w formule, a mianowicie - jako adresów.

Interpretacja ta zależy od sposobu realizowania procesu obliczeniowego ; w pierwszym przypadku mówimy o adresie, w drugim - o zmiennej.

Zarówno adresy jak i zmienne są nazwami komórek pamięci. Różnica między nimi jest jedynie taka, że adres jest zapisany w komórce, w której nie jest nazwą, natomiast zmienna jest nazwą komórki, w której jest zapisana. Adres jest bowiem zapisany w pamięci programu, natomiast odpowiadająca mu komórka znajduje się w pamięci D i w pamięci P programu nie podstawiamy na miejsce adresu żadnych danych.

Gdybyśmy w $\mathcal{T}$-maszynie zamiast $\mathcal{K}$-programu wpisali najpierw program symboliczny, a następnie na miejsce symboli danych napisali odpowiadające im dane, to zmienne grałyby rolę nazw komórek pamięci, w których zostałyby te znaki napisane.

§3. R e a l i z a c j a r ó ż n y c h p r o c e s ó w .

W dotychczasowych rozważaniach nie podawaliśmy w jaki sposób program jest zapisywany w pamięci programów P i przyjmowaliśmy, że maszyna realizuje tylko jeden program - zapisany w pamięci P.

Obecnie podamy krótki opis maszyny realizującej kolejno procesy według różnych programów[1].

---

1 - Gdybyśmy przyjęli, że każdy nowy program jest zapisywany przez obliczającego - bezpośrednio w pamięci programów, to konstrukcja maszyny niczym by się nie różniła od konstrukcji maszyn dyskutowanych poprzednio. Jednakże chcemy, aby zmiana programu obliczenia następowała również automatycznie, podobnie jak podstawianie danych czy parametrów.

Jeżeli $\phi_1, \phi_2 \ldots \phi_n$ są odpowiednimi programami procesów $\alpha_1, \alpha_2 \ldots \alpha_n$ to ciąg

$$\phi_1, \phi_2, \ldots \phi_n$$

nazwiemy ω -programem. ω -programy gdzie wmy pisali krótko

$\phi_i\ \omega_a$

Ogólny schemat maszyny realizującej ω -procesy w zasadzie nie różni się od schematów maszyn podanych poprzednio.

W stanie początkowym, w pamięci D wejściowej zanjduje się ω -program, tzn. umieszczone są w niej programy kolejnych procesów.

Cykl pracy ω -maszyny jest następujący :

1. Przepisanie programu z pamięci wejściowej do pamięci programów P.

2. Zrealizowanie programu znajdującego się w pamięci P.

3. Zapisanie wyniku końcowego obliczenia w pamięci wyjściowej.

Oczywiście zgodnie z definicją każdy program może być liczony wielokrotnie dla różnych danych i parametrów. Punkt 2 cyklu pracy przebiega według opisu podanego w poprzednich paragrafach. Dla zrealizowania podanego cyklu pracy urządzenie sterujące maszyny musi rozróżniać, czy z pamięci wejściowej jest odczytywany do pamięci P program, dane, czy w przypadku języka uproszczonego - parametry. Wymaga to oznaczenia w pamięci wejściowej różnymi symbolami programów, danych i parametrów. Na podstawie tych symboli urządzenie sterujące realizuje odpowiedni punkt cyklu pracy.

Maszyna może również posiadać oddzielną pamięć dla danych

parametrów oraz programów.

Możliwe są również inne rozwiązania ω -maszyny, zależnie od innych założeń wyjściowych.

W dotychczasowych rozważaniach przyjmowaliśmy, że w stanie początkowym dane znajdowały się w pamięci D, program znajdował się w pamięci P i program był realizowany jednokrotnie. W praktyce ważny jest przypadek, kiedy ten sam program jest realizowany wielokrotnie dla różnych danych. W takim przypadku nie mamy już do czynienia z procesem prostym, w sensie definicji podanej w rozdziale I.

Proces prosty określany w pozdziale I nazwiemy p r o c e s e m j e d n o k r o t n y m lub krótko ρ -procesem, a proces, polegający na wielokrotnym realizowaniu programu procesu prostego dla różnych danych nazwiemy p r o c e s e m w i e l o k r o t n y m lub krótko λ -procesem.

W niniejszym rozdziale podamy ogólne zasady realizowania procesów wieloktotnych.

Zanim przejdziemy do opisu realizacji procesów wielokrotnych rozrzerzymy pojęcie języka tak, aby za jego pomocą można było również opisywać procesy wielokrotne.

§1. J ę z y k i p r o c e s ó w w i e l o k r o t n y c h.

Języki procesów jednokrotnych nazwiemy ρ -językami, a odpowiadające im programy - ρ -programami. Niech Φ będzie ρ -programem i niech $x_1, x_2, \ldots x_n$ oznaczają zbiory danych, dla których ma być kolejno zrealizowany program Φ.

Wyrażenie

$$\Phi\, \lambda\, x_1, x_2, \ldots x_n$$

lub krócej

$$\Phi\, \lambda_n\, x_i$$

gdzie $n \geq 1$ nazwiemy p r o g r a m e m w i e l o k r o t n y m lub λ -programem.

W szczególnym przypadku $\Phi\, \lambda_1 x_i$ oznacza jednokrotne wykonanie programu Φ.

Φ może być programem uproszczonym lub symbolicznym. Dla Π -programów podana definicja nie obowiązuje. Np. wyrażenie

$$((a + b) \cdot c)\, \lambda\, 5, 7, 8, 3, 2, 1$$

jest λ -programem, oznaczającym, że należy najpierw obliczyć program $((a + b) \cdot c)$ dla danych a=6, b=7, c=8, oraz następnie wykonać ten sam program dla danych a=3, b=2, c=1. Oczywiście program ten można również zapicać w postaci :

$$((a + b) \cdot c)\, \lambda_2$$

a = 5
b = 7
c = 8
a = 3
b = 2
c = 1

W szczególności λ -program

$$((a + b) \cdot c)\, \lambda_1$$

a = 5
b = 7
c = 8

oznacza jednokrotne wykonanie programu $((a + b) \cdot c)$.

Dla uproszczonego języka nawiasowego powższy -program będzie miał postać

$$((+) \cdot)\, \lambda_2\, 5, 7, 8, 3, 2, 1.$$

Podobnie możemy przedstawić λ -program, jeżeli ϕ jest programem podstawowym, beznawiasowym, czy Łukasiewicza[1].

Wyrażenie $\phi\lambda_k$ będziemy nazywali λ - p r o g r a m e m u p r o s z c z o n y m program $\phi\lambda_k x_i$ np. $((a+b)\cdot c)\lambda_2$

§2. R e a l i z a c j a p r o c e s ó w w i e l o k r o t n y c h.

Można podać kilka różnych zasad realizacji procesów wielokrotnych. Tutaj zajmiemy się tylko jedną z nich, a mianowicie przypadkiem kiedy ϕ jest ρ -programem[2]. Dla języków symbolicznych zasada działania jest podobna.

Maszynę realizującą λ -program nazwiemy λ -maszyną.

λ -maszyna może działać na dowolnej dyskutowanej poprzednio zasadzie. Ogólny schemat λ -maszyny nie ulega w zasadzie zmianie. Pamięć D zawiera obecnie kolejno wszystkie dane $x_1, x_2, \ldots x_n$; pamięć P zawiera natomiast uproszczony λ -program, realizowanego procesu, tj. wyrażenia $\phi\lambda_k$.

λ -maszyna wyposażona jest dodatkowo w liniową pamięć W, w której zapisywane są kolejne wyniki końcowe każdego obliczenia.

Działanie maszyny jest następujące : po obliczeniu programu ϕ jedną z podanych poprzednio metod, wynik obliczenia jest zapisany w pamięci W i sterowanie ponownie analizuje od początku program ϕ , pobierając z pamięci D dalsze dane.

---

1 - λ -programy przypominają tzw. λ -system Churcha. Patrz np.

2 - Należy odróżnić ρ -program uproszczony oraz λ -program uproszczony. Np. $(+\ .)$ jest ρ -programem uproszczonym, a $((a+b)\cdot c)$ jest λ -programem uproszczonym.

Po n krotnym powtórzeniu ϕ -programu, λ -maszyna kończy działanie. Ilość powtórzeń programu ϕ dana jest w λ -programie. Zamiast ilości powtórzeń zadanej w programie, można również koniec obliczenia zrealizować przez umieszczenie na końcu danych - w pamięci danych - specjalnego symbolu, oznaczającego koniec liczenia. Pojawienie się tego symbolu jako danej, powoduje zatrzymanie maszyny.

W przypadku maszyny z adersową pamięcią roboczą λ -maszyna posiada dodatkową pamięć danych D. Dane z pamięci D są umieszczane według obliczanych adresów w pamięci roboczej R. W ten sposób ogólny schemat λ -maszyny nie zależy od tego, czy maszyna posiada pamięć wyników częściowych, czy też pamięć roboczą.

Pamięć danych D w λ -maszynie będziemy nazywali też w e j ś c i e m λ -maszyny, a pamięć W - w y j ś c i e m λ -maszyny.

§3. P a r a m e t r y i s t a ł e .

Przy wielokrotnym wykonaniu λ -programu, niektóre dane nie ulegają zmianie w każdym obliczeniu ; dane takie nazywamy p a r a m e t r a m i . Jeżeli dane posiadają tę samą wartość dla wszystkich obliczeń - mówimy o s t a ł y c h. W dalszym ciągu nie będziemy odróżniać parametrów od stałych; jedno i drugie nazwiemy parametrami.

Jeżeli stosujemy język symboliczny, parametry nie wymagają specjalnego traktowania, gdyż zmiana parametrów obliczenia sprowadza się do wpisania nowych parametrów w pamięci D. Jeżeli parametry nie ulegają zmianie, po prostu nie zmieniamy ich tak długo w pamięci D, jak długo są one potrzebne.

Jeżeli maszyna pracuje w języku uproszczonym, na oznaczenie parametrów wprowadzamy specjalny symbol, np. 2. 0 - oznacza więc wyniki częściowe. 1 - dane. 2 - parametry. Można wprowadzić oczywiście również inne oznaczenia, np. c, d, p - oznaczające odpowiednie wyniki częściowe, dane oraz parametry. μ -maszyna natomiast musi zawiedać apecjalną pamięć parametrów, z której parametry są w miarę potrzeby pobierane.

W języku symboilcznym możemy dokonywać w zasadzie zmiany niezależnie dowolnego parametru ; w językach uproszczonych natomiast, możliwe jest tylko zmienianie grupowo parametrów, tzn. zmienianie wszystkich parametrów, gdyż zmiana jednego parametru jest niewygodna.

Tak więc λ -program, w którym zmienne są dane oraz parametry ma postać

$$\phi\ \lambda P_1, X_1, X_2, \ldots X_k, P_2, X_{k+1}, X_{k+2}, \ldots X_l, P_j, X_{l+1}, \ldots X_n$$

lub krócej $\phi \lambda_n^p$

gdzie P_1, P_2,...P_p są kolejno zbiorami parametrów procesu, a x_1, x_2,.....x_n kolejnymi zbiorami danych.

$\phi \lambda_n^p$ oznacza, że program ϕ ma być obliczony dla p - zbiorów parametrów P_1, P_2, ... P_p oraz dla n - zbiorów danych x_1, x_2,.....x_n.

Wszystkie parametry oraz dane są umieszczane w pamięci D, w identycznym porządku, w jakim występują w λ -programie.

Dla odróżnienia, które liczby przedstawiają dane, a które parametry, zbiór danych oraz zbiór parametrów muszą być poprzedzone w pamięci D specjalnymi symbolami sygnalizującymi

czy po nich następują dane czy parametry.

Np.

$$(((a+b)\ .\ c)/f)\lambda_6^2 .\ p,\ 3,\ 2,\ d,\ 4,\ 5,\ 7,\ 8,\ 2,\ 1,\ p,\ 2,\ 3.$$
$$d,\ 3,\ 7,\ 8,\ 9,\ 4,\ 5.$$

oznacza, że w programie $(((a+b)\ .c)/f)$ a i b są parametrami, a c i f danymi, oraz, że najpierw wykonywane jest obliczenie przy ustalonych a i b /a = 3, b = 2/ dla następujących par danych : c = 4, f = 5, c = 7, f = 8, c = 2, f = 1. Następnie zmienione są oba parametry na a = 2 i b = 3 i program dla tych parametrów jest wykonywany dla następujących danych : c = 3, f = 3, c = 8, f = 9, c = 4, f = 5. Zmiana parametrów polega na zapisaniu do pamięci parametrów z pamięci wejściowej nowych wartości parametrów.

Dla ułatwienia zrozumienia zmiany parametrów, w przykładzie podano język symboliczny, jednakże podana metoda zmiany parametrów jest odpowiednia do języków uproszczonych.

Rozdział X: SKŁADANIE POD-PROGRAMU.

Pisanie i czytanie zbyt długich programów jest niewygodne. W praktyce stosowane są różne metody, służące do czytelniejszego przedstawiania długich wyrażeń matematycznych.

Np. a = A + B

gdzie A = e f - g

oraz E = x y + z

Podany przykład jest bardzo prosty i jego rozbicie na prostsze części w przykładzie tym, może się wydawać nieuzasadnione. W bardziej skomplikowanych przypadkach metoda taka jest wygodna i często stosowana.

W niniejszym rozdziale zajmiemy się różnymi metodami pisania zbyt długich programów w postaci wygodnej do manipulowania i konstrukcjami konstrukcyjnymi tego rodzaju zapisu.

Sprawa wygody zapisu programu nie zawsze jest jednakowo ważna. Np. jeżeli maszyna jest przeznaczona do obliczania ciągle tego samego problemu, sprawa wygody zapisu programu nie ma specjalnego znaczenia, gdyż program jest opracowywany jednorazowo i sprawa prostoty układania programu nie ma tutaj specjalnego znaczenia praktycznego, jak i ekonomicznego.

---

1 - Nie znaczy to, że za pomocą tej maszyny można rozwiązywać tylko jeden problem. Chodzi nam tylko o to, że zmiana problemu rozwiązaniowego przez daną maszynę następuje bardzo rzadko albo wcale. Natomiast na innym egzemplarzu identycznej maszyny może bać rozwiązywany inny problem, ale również jeden. Inaczej mówiąc obsługa maszyny nie musi często opracowywać nowych programów.

Jeżeli jednak maszyna jest przeznaczona do rozwiązywania bardzo często różnych zadań, a więc często muszą być opracowywane nowe programy, sprawa wygody opwacowywania programów ma zasadnicze znaczenie.

Kilka takich prostych metod rozważymy w dalszym ciągu, nie wyczerpując oczywiście tematu całkowicie.

§1. P o d p r o g r a m y i p r o g r a m y z ł o ż o n e.

W niniejszym rozdziale będziemy rozpatrywali tylko procesy proste jednokrotne, jakkolwiek wszystkie rezultaty są również ważne dla λ -procesów.

Podprocesem procesu $\mathcal{A}$ nazwiemy taki proces prosty $\mathcal{A}'$, że :

1. Każda operacja, należąca do $\mathcal{A}'$ należy również do $\mathcal{A}$.
2. Każdy obiekt, należący do $\mathcal{A}$ należy również do $\mathcal{A}$.
3. Jeżeli $\mathcal{L}$ jest lewym /lub prawym/ argumentem operacji w $\mathcal{A}'$, to $\mathcal{L}$ jest również lewym /lub prawym/ argumentem operacji Δ w $\mathcal{A}$.

Program podprocesu $\mathcal{A}'$ będziemy nazywali p o d p r o g r a m e m procesu $\mathcal{A}$. Program procesu $\mathcal{A}$, składającego się z podprogramów nazwiemy p r o g r a m e m z ł o ż o n y m procesu $\mathcal{A}$[1].

Dokładniejsze definicje programów złożonych podamy w dalszych paragrafach.

---

1 - Należy pamiętać, że program złożony opisuje proces prosty. Nazwa ta więc odnosi się tylko do struktury programu a nie procesu.

§2. O g ó l n y s c h e m a t m a s z y n y r e a l i z u j ą c e j p r o g r a m y z ł o ż o n e .

Maszyna realizująca programy złożone posiada dodatkową pamięć wyników pośrednich W, w której są magazynowane wyniki końcowe każdego podprogramu.

Program złożony znajduje się w pamięci wejściowej. Kolejne podprogramy są przepisywane w pamięci wejściowej do pamięci programów.

Najwygodniej jest, gdy dane i parametry każdego programu są umieszczane w pamięci wejściowej bezpośrednio po odpowiadającym im programie.

Wygodny jest również schemat z oddzilną pamięcią wejściową danych i podprogramów. Sprawy te zresztą nie mają większego znaczenia dla dalszych rozważań, a mogą być dokładniej dyskutowane przy podaniu szczegółowych założeń technicznych maszyny.

Zależnie od rodzaju zastosowanej pamięci wyników pośrednich K, mogą być stosowane różne sposoby składania podprogramów.

W następnych paragrafach opiszemy najważniejsze przypadki składania podprogramów. Rozważania nasze ograniczymy w zasadzie do podstawowego języka uproszczonego. Dla innych języków składanie podprogramów jest podobne.

§3. R e a l i z a c j a p r o g r a m ó w z ł o ż o n y c h z a p o m o c ą m a s z y n y z l i n i o w ą p a m i ę c i ą w y n i k ó w p o ś r e d n i c h.

Niech $\mathcal{A}'$ będzie podprocesem procesu $\mathcal{A}$. Jeżeli istnieje

taka operacja w podprocesie $\mathcal{A}'$, że argument α operacji Δ jest wynikiem częściowym operacji Ω, nie należącej do $\mathcal{A}'$, a należącej do $\mathcal{A}$, to α nazwiemy w y n i k i e m p o ś r e d n i m oraz oznaczymy w podprogramie symbolem #. Dane i wyniki częściowe będziemy oznaczać jak poprzednio 1,0.

Niech $\mathcal{A}', \mathcal{A}'', \ldots \mathcal{A}^{k}$ będą podprocesami procesu $\mathcal{A}$, niech $\phi_1, \phi_2 \ldots \phi_k$ będą podprogramami odpowiednich podprocesów.

Jeżeli wynik końcowy podprogramu ϕ_j jest wynikiem pośrednim w podprogramie ϕ_i zapiszemy $\phi_j < \phi_i \ (j<i)$, oraz $\mathcal{A}^j < \mathcal{A}^i$, gdzie $\mathcal{A}^j < \mathcal{A}^i$ z podprocesami odpowiadającymi podprogramom ϕ_j, ϕ_i.

Ciąg $\phi_1, \phi_2, \ldots \phi_n$ nazwiemy programem złożonym procesu $\mathcal{A}$, jeżeli dla każdego i $(1 \leq i \leq n)$ istnieje co najmniej jedno takie $n \geq j > i$, że $\phi_i < \phi_j$.

Niech ϕ_i zawiera t symboli wyników pośrednich.

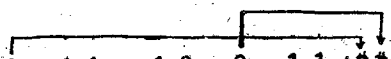
Powiemy, że program złożony jest u p o r z ą d k o w a n y p o p r z e c z n i e, lub inaczej, że podprogramy są uporządkowane w k o l e j n o ś c i P, jeżeli dla każdego i wyniki końcowe podprogramów $\phi_{k-1}, \phi_{k-2}, \ldots \phi_{k-t}$ są kolejnymi wynikami pośrednimi w ϕ_i, oraz k + t + c = i, gdzie c > 0.

Przykład

ϕ 1. 1 1 + 1 1 - 0 0 c
ϕ 2. 1 1 * 1 0 0
ϕ 3. 1 1 + # # / 0 0 - 0

Wynik pierwszego podprogramu ϕ_1 jest lewym argumentem drugiego działania w podprogramie ϕ_2, oraz wynik podprogramu ϕ_2 jest prawym argumentem drugiego działania w podprogramie ϕ_3.

Pisząc podprogramy w jednej linii i zaznaczając symbole wyników pośrednich i odpowiadające im podprogramy - strzałkami, otrzymamy :



1 1 + 1 1 - 0 0 ; 0, 1 1 . 1 0 - 0, 1 1 +#*/ 0 0 - 0

Warto zwrócić uwagę, że nie ważne jest tutaj w jakim porządku napisane są poszczególne podprogramy.

Zrozumiałe dlaczego użyliśmy tutaj terminu - porządek poprzeczny. Jest to bowiem uogólnienie pojęcia porządku poprzecznego, które użyliśmy do określenia języków w rozdziale II.

Dla łatwiejszego uchwycenia podobieństwa zauważmy, że program złożony możemy przedstawić w postaci drzewa w sposób następujący :

Podprogramy możemy interpretować jako punkty, gałęzie wchodzące do punktów - jako wyniki pośrednie /danych oraz wyników częściowych na drzewie nie przedstawiamy/. Wtedy możemy zastosować zasady numeracji rozgałęzień drzew identycznie, jak w rozdziale II. Porządek P będzie wtedy odpowiadał numeracji poprzecznej.

Podany przykład programu możemy przedstawić w postaci drzewa, jak to pokazano na rys. 27.

Pisanie podprogramów w porządku P jest bardzo proste. Zaczynamy od ostatniego podprogramu i wpisujemy wszystkie podprogramy, odpowiadające symbolom wyników pośrednich, poczynając od strony prawej. Następnie powtarzamy tę czynność dla przedostatniego programu itd., aż do końca.

Ponieważ ten sposób pisania podprogramów jest wygodny, możnaby podprogramy pisać w kolejności od ostatniego do pierwszego, pamiętając, że ich wykonanie następuje w odwrotnym kierunku. Tak więc poprzedni przykład możemy zapisać :

$\overline{\phi}_3$ 1 1 + # */ 0 0 - 0

$\overline{\phi}_2$ 1 1 . 1 0 - 0

$\overline{\phi}_1$ 1 1 + 1 1 - 0 0 . 0

Łatwo sprawdzić, że pamięć wyników pośrednich w maszynie realizującej podprogramy złożone w porządku P, winna być pamięcią liniową.

Wynik końcowy każdego podprogramu jest zapisywany w pamięci liniowej wyników pośrednich. Jeżeli argumentem wykonywanej operacji jest wynik pośredni, jest on odczytywany z pamięci wyników pośrednich.

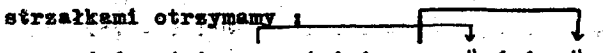
Dla innych języków uproszczonych zasada działania maszyny jest identyczne. W języku Łukasiewicza program złożony, w porządku P, będzie miał np. postać :

$\overline{\phi}_3$ # 1 1 + # .

$\overline{\phi}_2$ 1 1 1 . +

$\overline{\phi}_1$ 1 1 - 1 1 . +

lub pisząc program w jednej linii i zaznaczając podstawianie strzałkami otrzymamy :



1 1 - 1 1 . +, 1 1 1 . +, # 1 1 +# .

§4. Realizacja programów złożonych za pomocą maszyny z rewersyjną pamięcią wyników pośrednich.

Program złożony $\overline{\phi}_1 \overline{\phi}_2 \ldots \overline{\phi}_3$ nazywamy uporządko-

w a n y m w z d ł u ż n i e , jeżeli dla każdego i, Φ_{i+1} jest podprogramem najbliższego wolnego wyniku pośredniego w Φ_i. Przez wolny wynik pośredni rozumiemy wynik nie związany z żadnym podprogramem[1].

Np.

Φ_1 1 1 + 1 0 - 0

Φ_2 1 1 . 1 1 + 0 0/ 0

Φ_3 1 #2 + 1 0 - 0

Φ_4 #3 1 + #1 1 - 0 0 . 0

Liczbami przy symbolach wyników pośrednich zaznaczono odpowiadające im podprogramy. Oczywiście numery te w podprogramach nie występują. Tutaj napisano je tylko dla ułatwienia.

Pisząc podany program w jednej linii i zaznaczając podprogram każdego wyniku pośredniego strzałkami, otrzymamy:

1 1 + 1 0 - 0, 1 1 . 1 1 + 0 0/0 1 #+ 1 0 - 0, #1+#1 - 00.

Porządek liczenia poszczególnych podprogramów nie ma oczywiście znaczenia.

Drzewo tego programu pokazano na rys.28.

Maszyna realizująca programy złożone w porządku W, musi oczywiście jako pamięć wyników pośrednich zawierać pamięć rewersyjną.

Podobnie można postąpić dla innych języków uproszczonych.

---

1 - Definicja ta może być również podana na podstawie numeracji rozgałęzień drzewa.

§5. R e a l i z a c j a p r o g r a m ó w z ł o ż o n y c h z a p o m o c ą m a s z y n y z a d r e s o w ą p a m i ę c i ą w y n i k ó w p o ś r e d n i c h.

Poprzednie dwie metody składania podprogramów polegały na dobrym uporządkowaniu wszystkich podprogramów, co nie zawsze jest wygodne, chociaż większego kłopotu nie sprawia. Ponad to, jeżeli jakiś podprogram był w programie wykorzystany wielokrotnie, miał on być tyle razy powtarzany w programie złożonym.

Czasem jednak wygodniejsza jest inna metoda składania podprogramów, którą tu krótko przedstawimy.

Program złożony $\Phi_1, \Phi_2, \ldots \Phi_n$ nazywamy częściowo uporządkowanym, jeżeli dla każdego i istnieje takie $j < i$, że $\Phi_j < \Phi_i$.

Przyjmiemy w dalszym ciągu, że wynik końcowy każdego podprogramu Φ_i jest oznaczony w podprogramie Φ_i różnymi symbolami, np. 11 + 01 - a, oraz. że w odpowiednich podprogramach, odpowiadające im wyniki pośrednie są oznaczone tymi samymi symbolami, np. 1a + 01 - b.

Realizacja maszyny wykonującej podprogramy częściowo uporządkowane nie przedstawia trudności. Pamięć wyników pośrednich musi być wtedy pamięcią adresową. Symbole wyników pośrednich są wtedy interpretowane jako adresy. Program 1a.01 =b oznacza, że prawym argumentem jest wynik pośredni, znajdujący się w pamięci wyników pośrednich pod adresem a, i że wynik tego programu jest umieszczony w pamięci wyników pośrednich, pod adresem b.

Tak więc program częściowo uporządkowany może mieć np. postać :

Φ_1 1 1 + 1 0/ x
Φ_2 1 1 . 1 1- 0 0 . y
Φ_3 1 1 - x 0 . z
Φ_4 z 1 . x y + 0 0/ u

W tym przypadku każdy podprogram występuje w programie złożonym tylko raz, niezależnie od tego ile razy jest używany w programie złożonym.[1]

W przykładzie, np. wynik podprogramu Φ_1 jest użyty w podprogramie Φ_3 oraz Φ_4.

W przypadku języka Łukasiewicza czy nawiasowego, adres wyniku końcowego podprogramu może być podany w konwencjonalnej formie, np.

Φ_1 $((a + b) . c) = z$
Φ_2 $(a + z) = u$
Φ_3 $(z + u) = y$

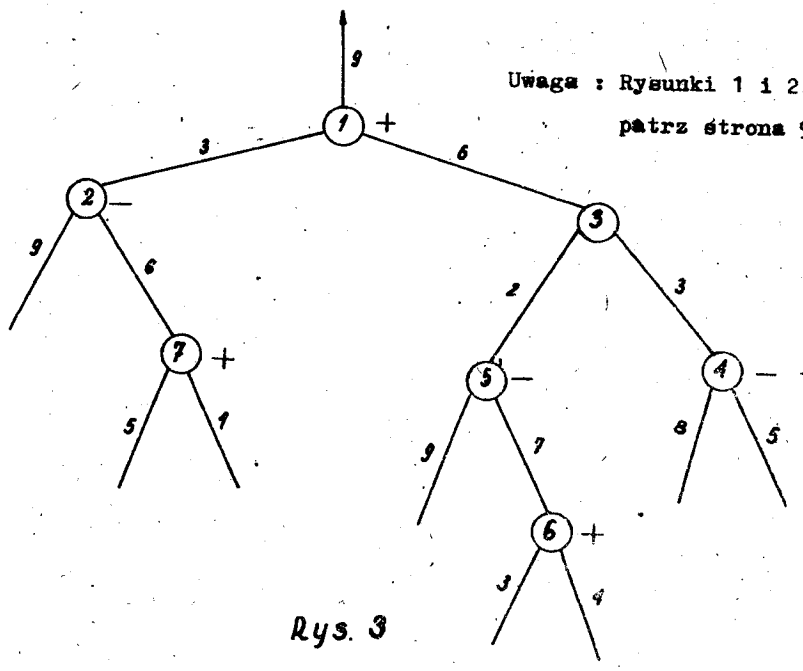
Znak równości jest tutaj zastosowany do znaczenia, że wynik końcowy jest przesłany pod określony adres.

Ogólnie biorąc, realizacja programów za pomocą maszyny z adresową pamięcią wyników pośrednich, jest wygodniejsza niż schematy, podane w dwu poprzednich paragrafach.

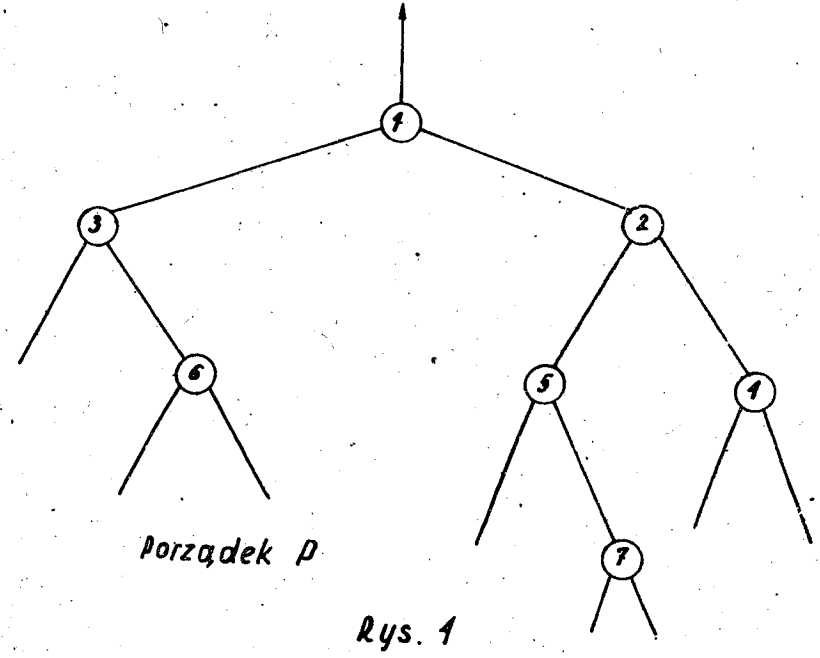
---

1 - Założyliśmy tutaj, że powtarzające się podprogramy są obliczane dla jednakowych danych oraz parametrów.

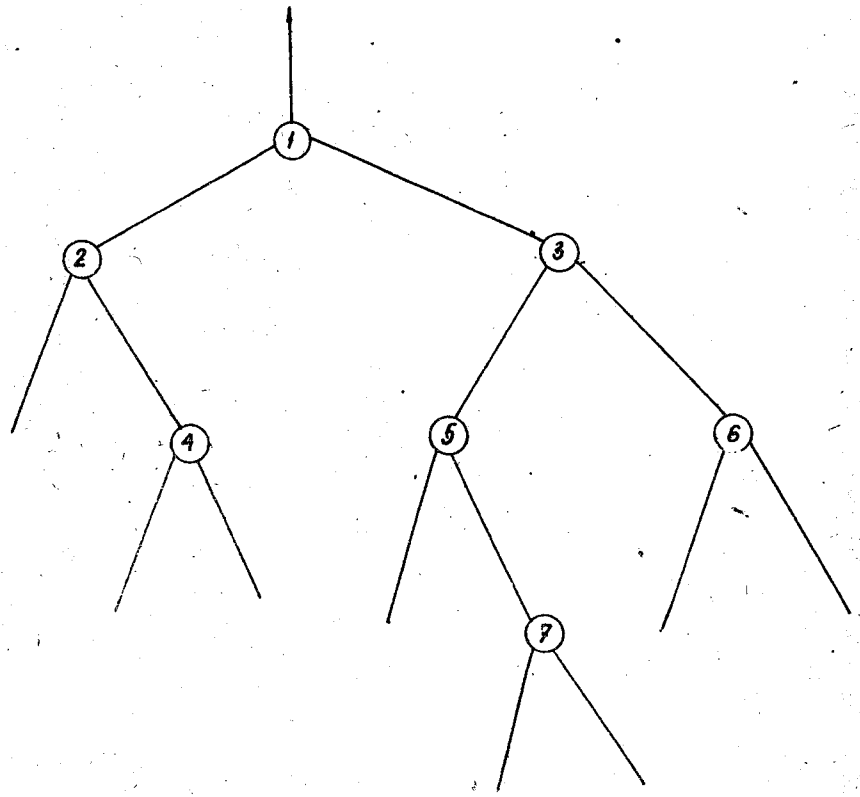
Uwaga : Rysunki 1 i 2, - patrz strona 9.



Rys. 3

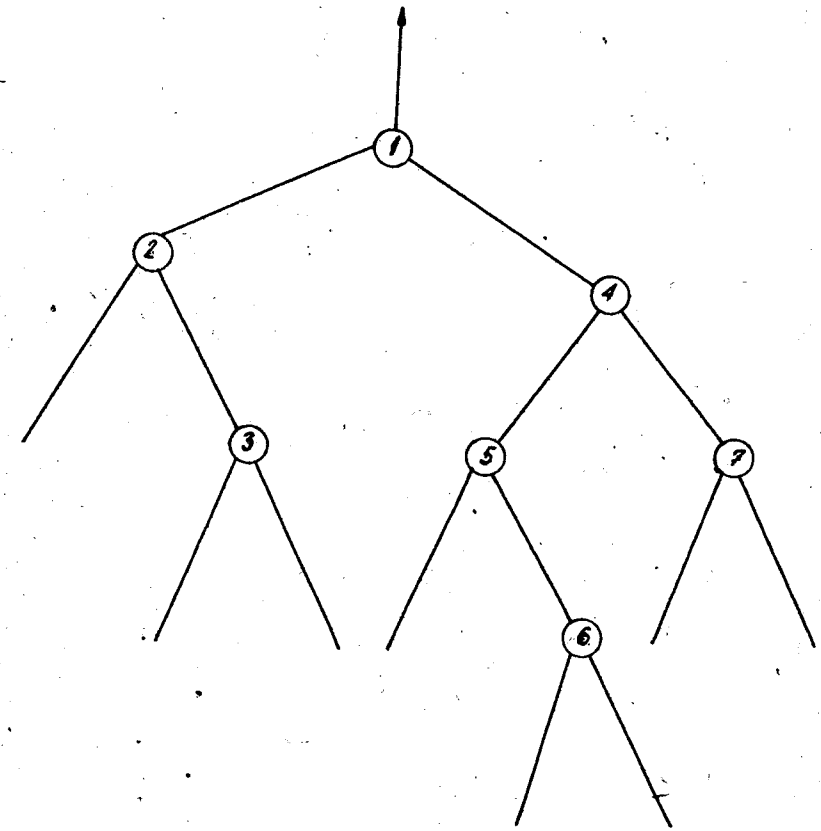


Rys. 1



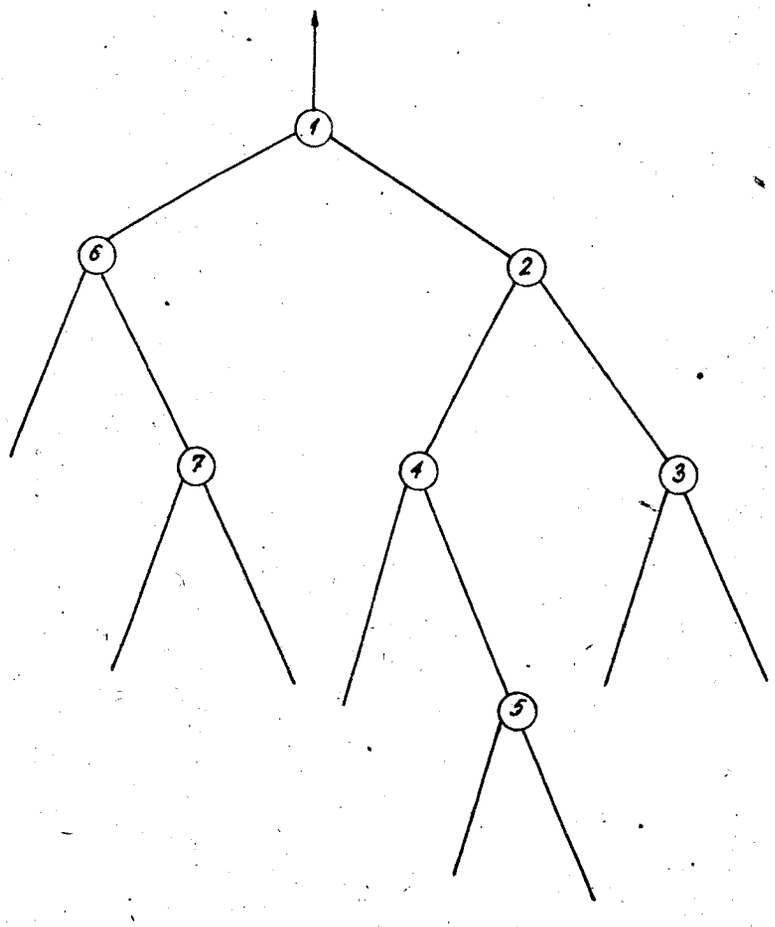
Porządek $\overline{P}$

Rys. 5



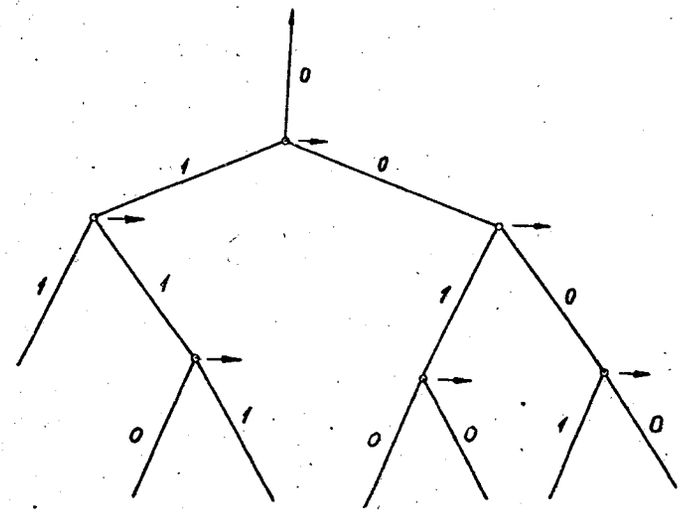
Porządek W

Rys. 6

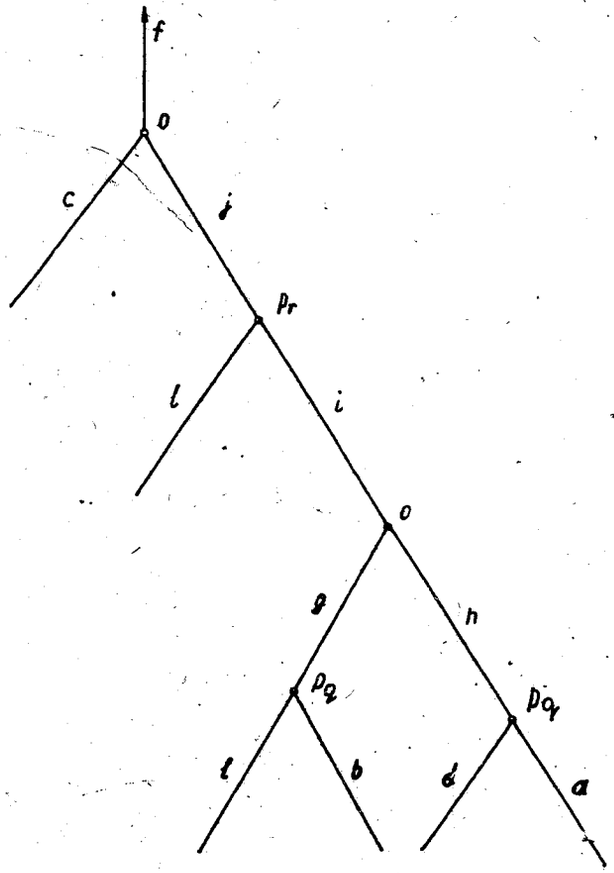


Porządek $\overline{W}$

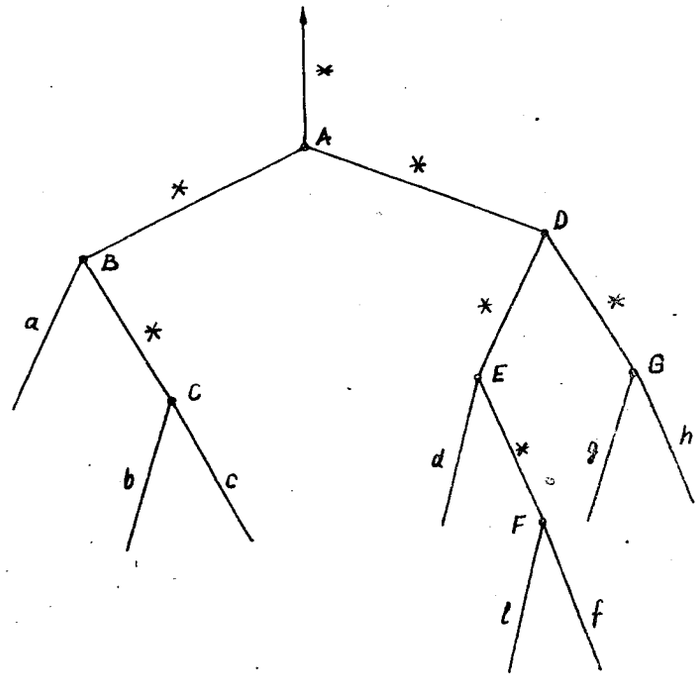
Rys. 7



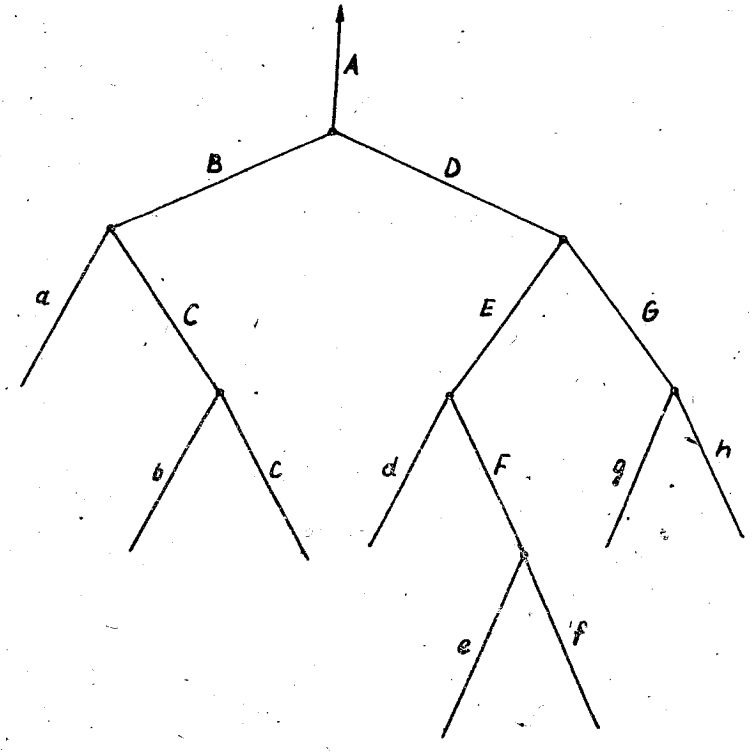
Rys. 8



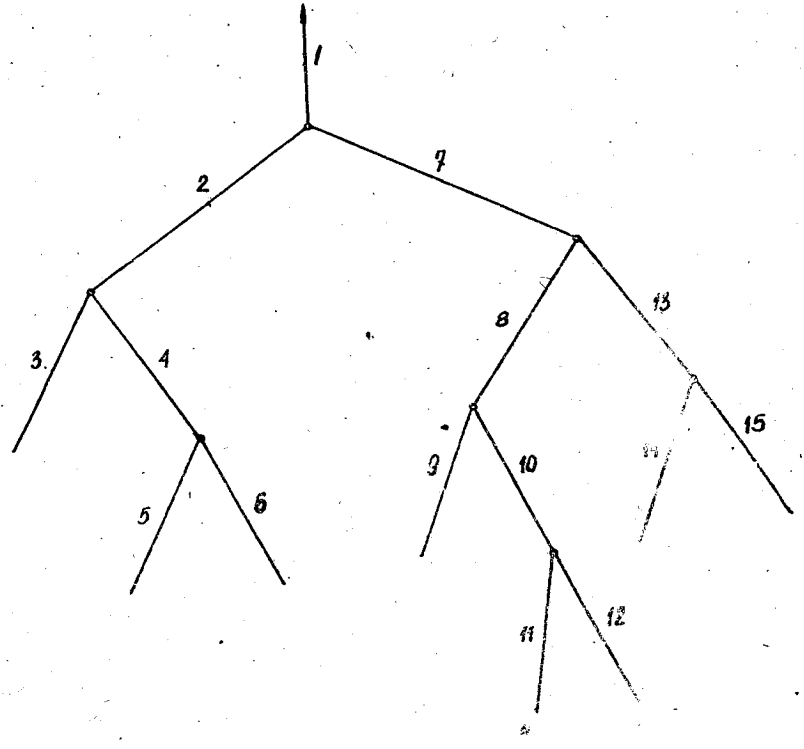
Rys. 9



Rys. 10

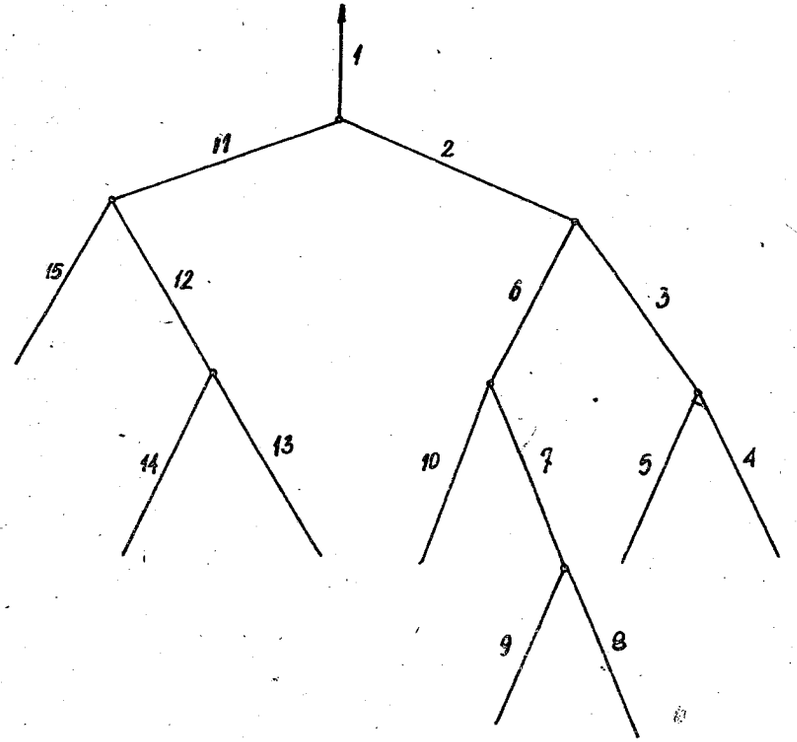


Rys 11



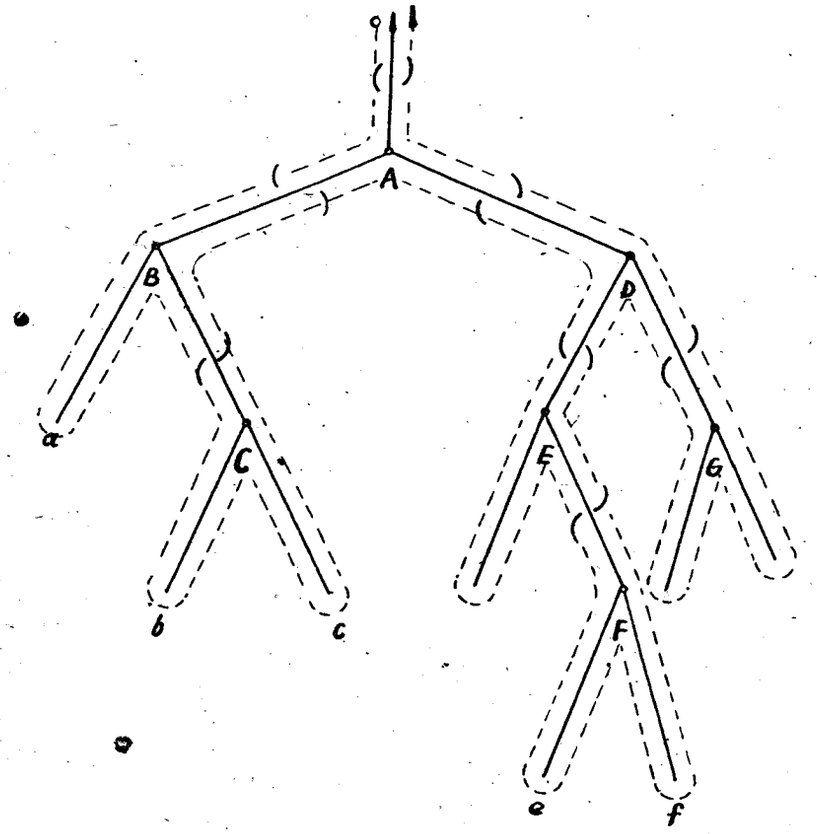
Porządek W

Rys. 12



Porządek $\overline{W}$

Rys. 13



Rys. 14

0
α_1
α_2
Δ
α_3
P

Rys. 15

0
α_1
α_2
Δ
α_3
3 4 + 5 1 + 9 * − 8 5 − 9 * − * * . * * + *
P

Rys. 16

Rys. 17

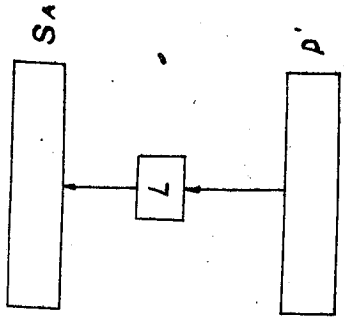
Rys. 18

Фиг. 19

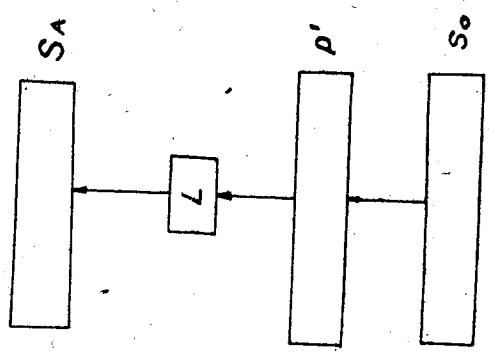
Фиг. 20

Rys. 21

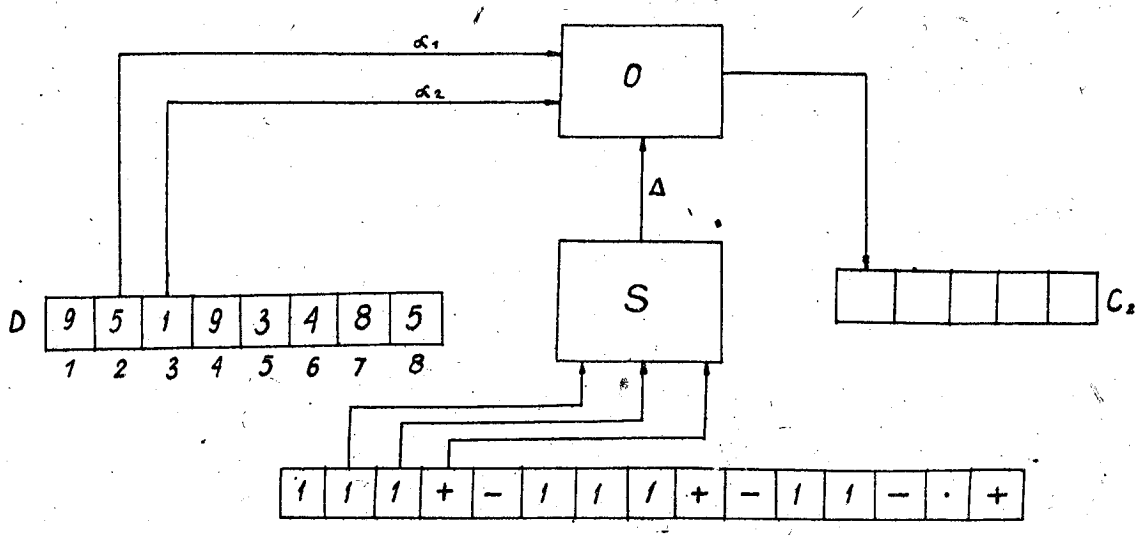
Rys. 22



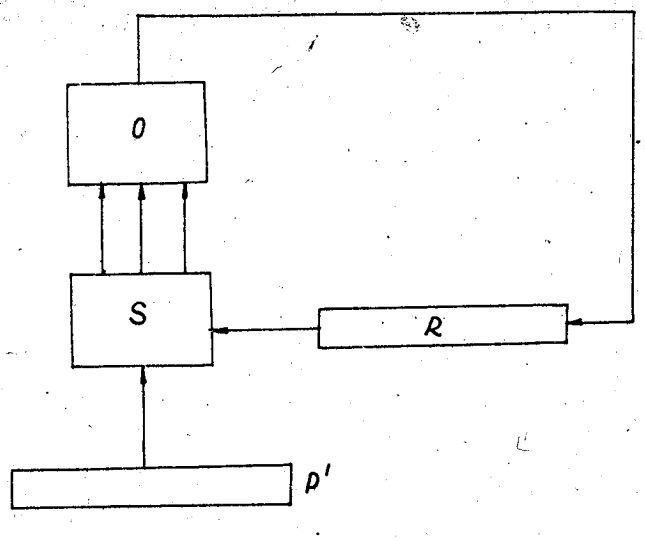
Rys. 24



Rys. 25



Rys. 23



Rys. 26