Z. PAWLAK

ORGANIZACJA MASZYN BEZADRESOWYCH

WARSZAWA 1965
PAŃSTWOWE WYDAWNICTWO NAUKOWE

Okładkę projektował
STEFAN NARGIEŁŁO

DRUKARNIA UNIWERSYTETU im. ADAMA MICKIEWICZA W POZNANIU, ul. FREDRY 10

PRZEDMOWA

Tematem książki są nowe koncepcje organizacji maszyn matematycznych. Dla odróżnienia od maszyn konwencjonalnych omawiane tu maszyny nazwano *maszynami bezadresowymi*. Główną cechą charakterystyczną maszyn bezadresowych jest realizowanie procesów obliczeniowych bezpośrednio w zadanym języku formalnym bez konieczności uprzedniego zaprogramowania rozwiązania problemu, jak to ma miejsce w obecnie budowanych maszynach matematycznych.

Punktem wyjściowym rozważań dotyczących organizacji maszyn bezadresowych jest pojęcie procesu prostego określone w rozdziale I. Dwa dalsze rozdziały zawierają definicje i własności języków opisujących procesy proste. W dalszych rozdziałach podano różne sposoby realizacji procesów prostych, w zależności od przyjętego języka oraz środków technicznych. Rozdział ostatni poświęcony jest zastosowaniu otrzymanych rezultatów do automatycznego programowania adresowych maszyn cyfrowych.

Z podanych rozwiązań wynika, że maszyny bezadresowe powinny mieć znacznie większą efektywną szybkość liczenia oraz prostszą konstrukcję od maszyn budowanych na dotychczasowych zasadach. Prawdziwość powyższej hipotezy może być ostatecznie zweryfikowana jedynie przez praktykę. Tym niemniej wiele zalet maszyn bezadresowych jest bezspornych i Czytelnik znający zasady organizacji współczesnych maszyn matematycznych zauważy je z łatwością, jak np. automatyczne umieszczanie wyników częściowych lub zmniejszanie pojemności pamięci maszyny.

Materiał zawarty w książce nie wyczerpuje poruszanego tematu, a stanowi raczej punkt wyjścia do dalszych badań w tym kierunku, tym niemniej jest on wystarczający do podjęcia badawczych prac konstrukcyjnych nad maszynami bezadresowymi. Przedstawione koncepcje są szczególnie korzystne dla bardzo małych maszyn matematycznych oraz maszyn o ultrawielkiej szybkości liczenia.

Zawarty w książce materiał został opracowany przeze mnie w latach

1952-1962, częściowo na zlecenie Polskiej Akademii Nauk, z której stypendium korzystałem, w zasadzie niezależnie od wyników uzyskanych w tej dziedzinie za granicą. Pierwsze wyniki dotyczące organizacji maszyn bezadresowych zostały bowiem opublikowane za granicą około roku 1960 (patrz np. Kalmár [5], [6], [7], Barton [2], Wegner [34]) i uwzględnienie ich tutaj było moim zdaniem niecelowe. Zresztą badania za granicą w dziedzinie maszyn bezadresowych idą w nieco innym kierunku niż ten, który tutaj przedstawiono, i połączenie całego dostępnego materiału (dotyczącego tego problemu) w jedną konsekwentną całość byłoby w chwili obecnej sprawą dość trudną, jeżeli nie niemożliwą.

Niektóre rezultaty dotyczące organizacji maszyn bezadresowych mogą również znaleźć zastosowanie w innych dziedzinach jak np. ekonomia, językoznawstwo, prakseologia, organizacja produkcji itp. Dla wszystkich tych zastosowań szczególnie ważne jest ściślejsze określenie pojęcia procesu i zbadanie jego właściwości. W książce tej bowiem pojęcie to potraktowano dość powierzchownie, mając na uwadze zastosowanie go tylko do maszyn matematycznych. Zbliżone jest ono nieco do pojęcia procesu wprowadzonego przez Currego [3], oraz Wanga [33], jakkolwiek Wang dokładniejszej definicji pojęcia procesu nie podaje, natomiast je praktycznie stosuje w związku z maszynowym dowodzeniem twierdzeń.

*

Pragnę wyrazić podziękowanie profesorowi A. H. Taubowi, kierownikowi laboratorium maszyn cyfrowych uniwersytetu w Illinois, dyskusje z Nim bowiem pozwoliły mi na obranie właściwego kierunku badań. Cenną pomoc zawdzięczam również doktorowi Andrzejowi Ehrenfeuchtowi, z którym szczegółowo przedyskutowałem znaczną część podanego materiału. Wiele istotnych uwag zawdzięczam również docentowi Stefanowi Paszkowskiemu, doktorowi Andrzejowi Wakuliczowi, magistrowi Jackowi Blikle oraz magister Jadwidze Przymusińskiej. Wszystkim im chciałbym wyrazić swą wdzięczność.

ZDZISŁAW PAWLAK

Warszawa 1963

ROZDZIAŁ I

OBLICZENIA PROSTE

Pojęcie *obliczenia* 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 którym jest to niezbędne do dalszych rozważań, dotyczących maszyn matematycznych.

Mając na uwadze konstrukcję maszyny, musimy spojrzeć na obliczenie jako na *proces* wytworzenia z jednych obiektów — za pomocą określonych operacji — nowych obiektów. Przez *obiekty* rozumiem tutaj liczby, wektory, macierze itp. ściślej *symbole*, których używamy do ich oznaczenia. Tak więc obliczenie jest procesem przekształcania symboli w myśl określonych reguł. W maszynach elektronowych pojęcia matematyczne są przedstawiane nie za pomocą symboli, lecz sygnałów elektrycznych; obliczenie można by więc interpretować jako proces przekształcania sygnałów elektrycznych. Zrozumiałe, że charakter procesu przekształcania symboli jest inny od procesu przekształcania sygnałów elektrycznych i trudno badać proces obliczenia z konieczną dla praktyki drobiazgowością bez sprecyzowania czy chodzi nam o przekształcanie symboli, 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.

W dalszym ciągu wprowadzimy pojęcie *procesu*, które zależnie od interpretacji może oznaczać obliczenie na liczbach całkowitych, rzeczywistych, zespolonych, wektorach, macierzach, wartościach logicznych czy innych obiektach matematycznych. Tak ogólne ujęcie ma dla nas istotne znaczenie, pozwala bowiem na rozważania konstrukcji maszyn matema-

tycznych w znacznym stopniu niezależnie od tego czy są one przeznaczone do obliczeń numerycznych, logicznych, czy innych [28] i [31].

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

§ 1. PROCESY

Procesem $\mathfrak{A}=\langle A, O, R\rangle$ nazwiemy skończony zbiór A obiektów, skończony zbiór O operacji określonych w zbiorze A oraz relację R porządkującą [9] zbiór O. Jeżeli zbiór O jest dobrze uporządkowany przez relację R, to powiemy, że proces $\mathfrak{A}$ jest *sekwencyjny*. Jeżeli zbiór O jest częściowo uporządkowany przez relację R, to powiemy, że proces $\mathfrak{A}$ jest *jednoczesny* (1).

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

Proces $\mathfrak{A}=\langle A, O, R\rangle$ nazwiemy *prostym*, jeżeli:

1. Dla każdej operacji $\Delta\in O$(2) istnieją dokładnie trzy obiekty $l(\Delta)$, $p(\Delta)$, $w(\Delta)\in A$, zwane odpowiednio: *lewym* argumentem operacji Δ, *prawym* argumentem operacji Δ oraz *wynikiem* operacji Δ.

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

3. Istnieje dokładnie jedno takie $\alpha\in A$, że dla żadnego $\Delta\in O$ nie zachodzi: $\alpha=l(\Delta)$ ani $\alpha=p(\Delta)$. Wtedy α nazwiemy *wynikiem końcowym*, a Δ takie, że $\alpha=w(\Delta)$, *operacją końcową* procesu $\mathfrak{A}$.

4. Dla każdej operacji $\Delta\in O$ istnieje dokładnie jeden taki ciąg $\Delta_1,\alpha_1,\Delta_2,\alpha_2,\ldots,\Delta_n,\alpha_n$, że $\Delta_i\in O$, $\alpha_i\in A$, $\Delta_1=\Delta$, α_n jest wynikiem końcowym procesu $\mathfrak{A}$ oraz dla każdego i $(1\leqslant i\leqslant n)$ α_i jest wynikiem operacji Δ_i oraz α_{i-1} jest argumentem tej operacji.

Jeżeli $\alpha\in A$ oraz istnieją $\Delta, \Omega\in O$, takie że $\alpha=w(\Omega)$ oraz $\alpha=l(\Delta)$ lub $\alpha=p(\Delta)$, to α nazwiemy *wynikiem częściowym* procesu $\mathfrak{A}$.

(1) Innymi słowy, jeżeli w procesie wykonywane są operacje kolejno, to mówimy, że proces jest *sekwencyjny*, natomiast jeżeli operacje wykonywane są jednocześnie (niekoniecznie wszystkie) — proces nazywamy *jednoczesnym*.

(2) $\Delta\in O$ — należy czytać: Δ jest elementem zbioru O.

Jeżeli $\alpha\in A$ oraz nie ma takiego $\Delta\in O$, że $\alpha=w(\Delta)$, to α nazwiemy *daną początkową* procesu $\mathfrak{A}$(1).

Np. jeżeli elementami zbioru A są liczby naturalne, a elementami zbioru O działania arytmetyczne określone w zbiorze liczb naturalnych, to $\mathfrak{A}$ jest *rachunkiem liczb naturalnych*. Jeżeli elementami zbioru A są macierze, a elementami zbioru O działania macierzowe, to $\mathfrak{A}$ jest *rachunkiem macierzowym*. Jeżeli elementami zbioru A są wartości logiczne, a elementami zbioru O — operacje logiczne, to $\mathfrak{A}$ jest *rachunkiem logicznym*. Jeżeli elementami zbioru A są aksjomaty jakiejś teorii sformalizowanej oraz ich konsekwencje, a elementami zbioru O są reguły wnioskowania tej teorii, to $\mathfrak{A}$ jest *dowodem formalnym* w tej teorii(2).

Zbiór A można również interpretować jako zbiór konkretnych przedmiotów fizycznych, np. części samochodu (cały samochód jest również częścią samochodu), a zbiór O — jako zbiór operacji składania przedmiotów należących do A; proces $\mathfrak{A}$ jest wtedy *produkcją*. 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 zajmowali tutaj szczegółowo, a posłużymy się nim tylko dla ilustracji rozpatrywanych zagadnień. Maszynę matematyczną można bowiem również traktować jako pewnego rodzaju „fabrykę" realizującą proces produkcyjny, gdzie produktami końcowymi są symbole.

Proces przedstawia więc konkretny rachunek, czy konkretny dowód matematyczny. Obiektami procesu w przypadku rachunku wykonanego na liczbach naturalnych — są wszystkie dane początkowe, wyniki częś-

(1) Przyjęliśmy w definicji procesu prostego, że wszystkie operacje są dwuargumentowe. Nic nie stoi na przeszkodzie, aby rozpatrywać operacje o dowolnej skończonej liczbie argumentów $0, 1, 2, \ldots, k$. Wszystkie otrzymane w dalszym ciągu rezultaty bardzo łatwo przenieść na procesy z operacjami wieloargumentowymi. Dla prostoty pozostaniemy przy operacjach dwuargumentowych.

(2) 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*.

ciowe oraz wynik końcowy rachunku, a operacjami — wszystkie działania arytmetyczne wykonane w czasie obliczenia. W procesie dowodzenia konkretnego twierdzenia obiektami są wszystkie przesłanki dowodu wraz z lematami oraz samo twierdzenie, którego dowodzimy. Operacjami dowodu natomiast są wszystkie zastosowane w czasie dowodu reguły dowodzenia.

Ponieważ zbiór O jest zbiorem wszystkich operacji wykonanych w procesie, więc w zbiorze 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 zarówno 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.

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

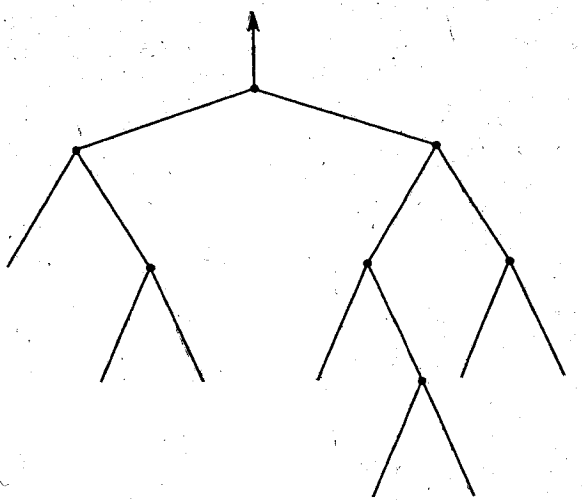
Jeżeli w procesie $\mathfrak{A}$ nie została wykonana jeszcze żadna operacja, to powiemy, że proces $\mathfrak{A}$ jest *w stanie początkowym*; jeżeli w procesie $\mathfrak{A}$ zostały wykonane wszystkie operacje, powiemy że proces $\mathfrak{A}$ jest *w stanie końcowym*; jeżeli $\mathfrak{A}$ jest procesem sekwencyjnym i w $\mathfrak{A}$ została wykonana i-ta operacja, to powiemy, że proces $\mathfrak{A}$ jest *w stanie i*.

Maszyna realizująca rachunek logiczny jest inna niż maszyna służąca do wykonywania rachunków macierzowych lub dowodzenia twierdzeń matematycznych. Tym niemniej maszyny te mają wiele cech wspólnych.

W dalszym ciągu będziemy starali się badać takie własności maszyn matematycznych, które nie zależą od rodzaju procesu realizowanego przez maszynę, a więc to co wspólne jest w maszynach, np. do rachowania i dowodzenia twierdzeń. Ewentualne zastosowanie otrzymanych wyników do budowy maszyn matematycznych, musi więc być związane z uwzględnieniem specyfiki procesu obliczeniowego oraz wielu 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 rozszerzymy.

§ 2. PROCESY I DRZEWA

Nie będziemy tutaj podawać dokładnej definicji *drzewa*, którą można znaleźć w dowolnym podręczniku teorii grafów np. [12], a poprzestaniemy na wyjaśnieniu tego pojęcia za pomocą przykładów. Przykład drzewa pokazany jest na rysunku 1. Odcinki stanowią gałęzie drzewa, punkty natomiast stanowią rozgałęzienia. 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.



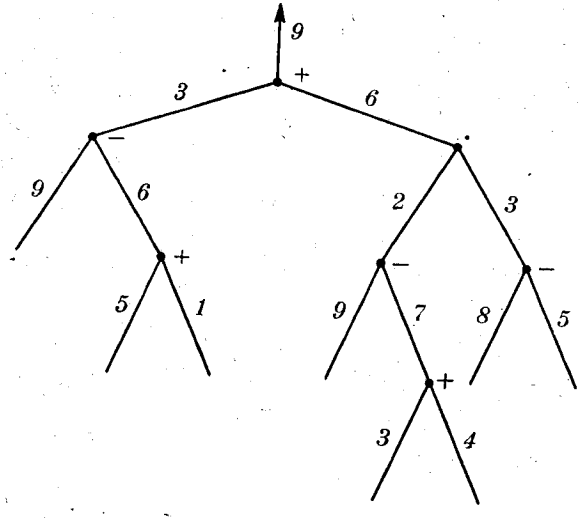
Rys. 1

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 operacji jest uporządkowany. Nie

jest bowiem rzeczą obojętną, w jakiej kolejności wykonujemy operacje w procesie, natomiast w drzewie porządek rozgałęzień nie ma znaczenia (1).

Tak więc każdy proces prosty można graficznie przedstawić w postaci drzewa. Sposób ten ma wiele zalet, gdyż pozwala na łatwe uchwycenie całej struktury procesu jednym spojrzeniem oka (jeżeli proces nie jest zbyt duży), a ponadto umożliwi łatwiejsze zrozumienie wielu spraw dyskutowanych w dalszym ciągu.

Przykład przedstawienia procesu obliczenia za pomocą drzewa pokazano na rysunku 2. Rysunek ten przedstawia proces obliczenia w stanie



Rys. 2

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}, a zbiorem O operacji są działania arytmentyczne: {+, −, ·, +, −, −, +}. Jak już to wspominaliśmy poprzednio, zbiór A

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

zawiera wszystkie obiekty występujące w procesie, a zbiór O — wszystkie operacje wykonane w procesie.

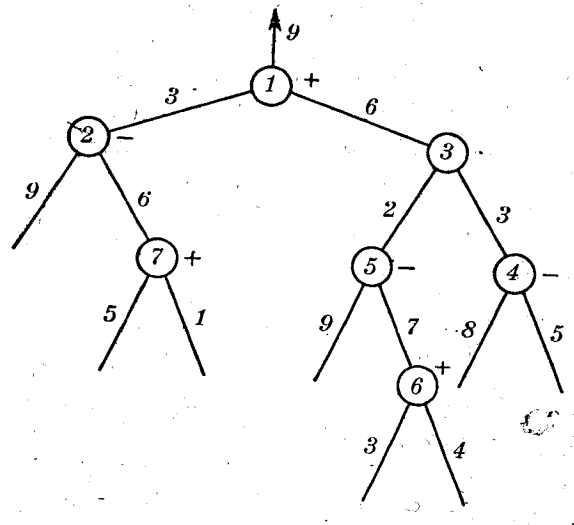
Procesy proste mają następującą elementarną własność:

Jeżeli proces prosty zawiera n operacji dwuargumentowych, to liczba danych początkowych w tym procesie wynosi $n+1$.

Wynika to stąd, że jeżeli proces zawiera n operacji, to oczywiście zawiera również n wyników operacji oraz $2n+1$ obiektów. Ponieważ suma danych początkowych i wyników operacji równa jest liczbie obiektów, więc liczba danych początkowych wynosi $2n+1-n=n+1$.

§ 3. PORZĄDEK PROCESÓW PROSTYCH

W paragrafie tym zajmować się będziemy procesami sekwencyjnymi, tj. procesami, w których wszystkie operacje są dobrze uporządkowane. Zagadnienie uporządkowania zbioru operacji sprowadza się do tego, w jakiej kolejności operacje te mają być wykonywane.



Rys. 3

Dla uproszczenia przyjmiemy, że każdej operacji Δ procesu $\mathfrak{A}$ przypiszemy liczbę naturalną $I(\Delta)$ w ten sposób, że jeżeli $\Delta \succ \Omega$, to $I(\Delta) < I(\Omega)$, gdzie $\Delta \succ \Omega$ oznacza, że operacja Δ jest wykonywana po operacji Ω, lub

inaczej — operacja Ω jest wykonywana przed operacją Δ. Tak więc operacje 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 rysunku 2 możemy ponumerować tak, jak to pokazane jest na rysunku 3. Operacje przedstawiono na rysunku nie kropkami jak to miało miejsce poprzednio, a kółeczkami, w których podano numery działań. Tak więc przebieg obliczenia przedstawionego na rysunku 3 będzie wyglądał następująco:

7. $5+1=6$,
6. $3+4=7$,
5. $9-7=2$,
4. $8-5=3$,
3. $2 \cdot 3=6$,
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, $\overline{W}$ i nazwiemy następująco:

P i $\bar{P}$ — porządki *poprzeczne,*

W i $\overline{W}$ — porządki *wzdłużne:*

ponadto wprowadzimy jeszcze następujące nazwy:

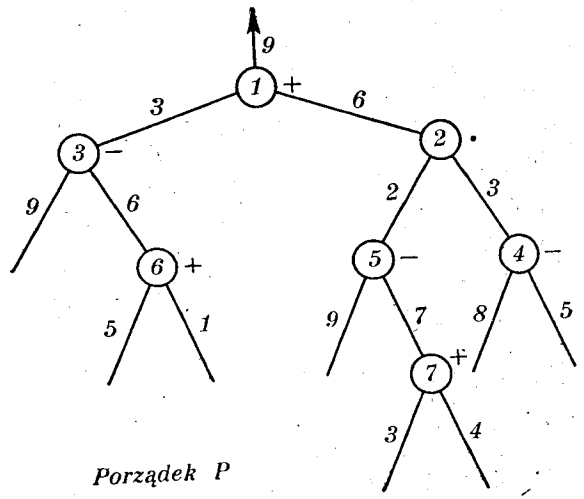
P i W — porządki *normalne,*

$\bar{P}$ i $\overline{W}$ — porządki *dualne.*

Zasady numerowania działań w każdym z wymienionych porządków pokazane są na rysunkach: 4, 5, 6, 7.

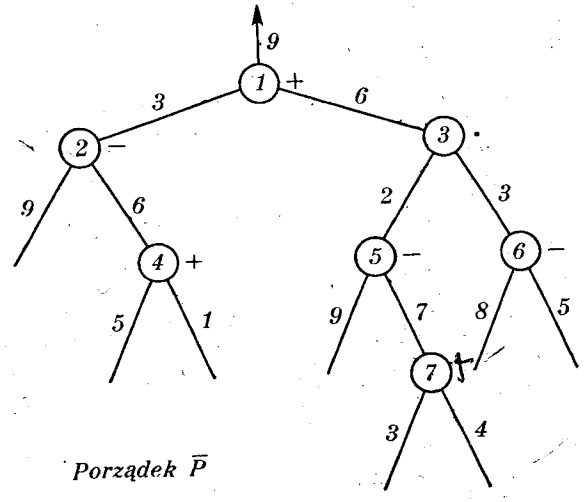
W przypadku numeracji poprzecznej drzewo dzielimy jakgdyby na „piętra" i działania numerujemy poczynając od piętra najwyższego, ko-

(1) Można by również numerować operacje w sposób odwrotny, tzn. operacjom wykonywanym wcześniej przypisać numery mniejsze, a operacjom wykonywanym później — numery większe, jednakże pierwszy sposób jest wygodniejszy.



Porządek P

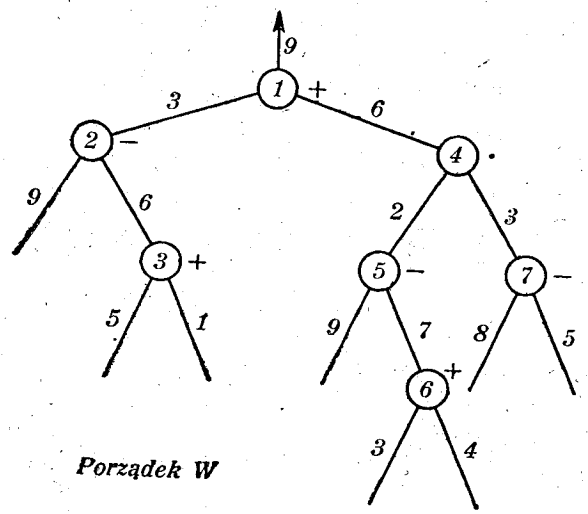
Rys. 4



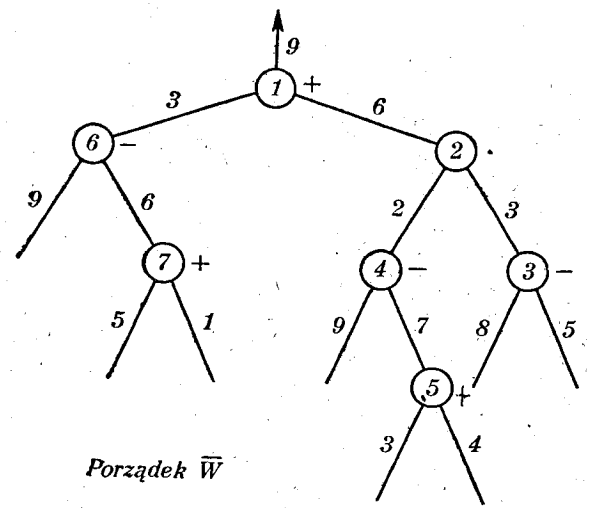
Porządek $\bar{P}$

Rys. 5

lejno piętrami, tak że działania 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 nu-



Rys. 6



Rys. 7.

merowane od strony prawej do lewej; dla porządku dualnego $\overline{P}$ — odwrotnie, tj. od strony lewej do prawej.

W numeracji wzdłużnej działania są numerowane jakgdyby wzdłuż drzewa. Szczegóły wynikają z rysunków: 6 i 7(1).

Przebieg obliczenia podanego na rysunku 2 dla porządków P, $\overline{P}$, W i $\overline{W}$ będzie miał postać:

	Porządek P	Porządek W	Porządek $\overline{P}$	Porządek $\overline{W}$
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.	$8-5=3$,	$2\cdot 3=6$,	$5+1=6$,	$9-7=2$,
3.	$9-6=3$,	$5+1=6$,	$2\cdot 3=6$,	$8-5=3$,
2.	$2\cdot 3=6$,	$9-6=3$,	$9-6=3$,	$2\cdot 3=6$,
1.	$3+6=9$,	$3+6=9$,	$3+6=9$,	$3+6=9$.

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

$3+4=7$, $8-5=3$,
$5+1=6$, $9-7=2$,
$9-6=3$, $2\cdot 3=6$.
$3+6=9$,

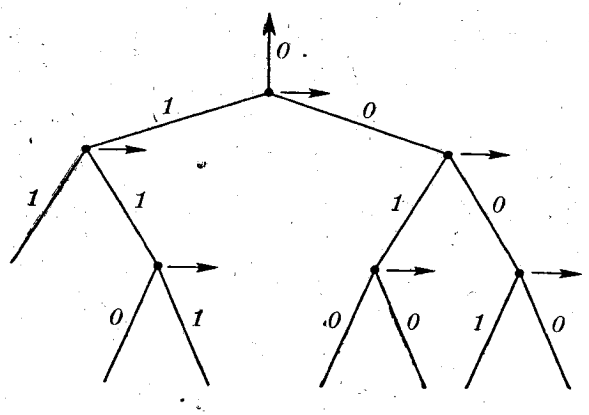
Przyjęliśmy tutaj, że wykonywane są jednocześnie dwie operacje. Operacje wykonywane jednocześnie są napisane w jednym wierszu. Oczywiście przebieg obliczenia może być również inny od podanego.

§ 4. PRZYKŁADY PROCESÓW PROSTYCH

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

Przykład obliczenia logicznego pokazany jest na rysunku 8. Cyfry 0 i 1 oznaczają wartości logiczne. Jedyna operacja występująca w tym obliczeniu — oznaczona strzałką — jest nazywana *implikacją* i określona jest

(1) Numeracja rozgałęzień drzewa może być również podana w postaci rekurencyjnej, jednakże wymaga to określenia wielu dodatkowych pojęć, z których w dalszym ciągu nie zrobilibyśmy użytku — ze ściślejszego określenia numeracji zrezygnowaliśmy.



Rys. 8

następująco:

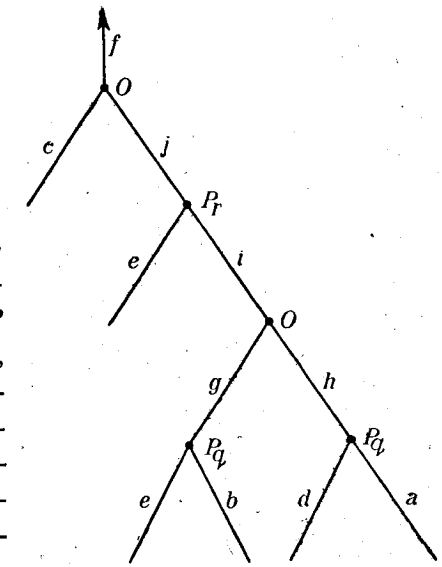
l	p	w
1	1	1
1	0	0
0	1	1
0	0	1

Litery l, p, w oznaczają odpowiednio lewy i prawy argument oraz wynik działania. Przebieg tego obliczenia dla różnych porządków jest następujący:

	P	$\overline{P}$	W	$\overline{W}$
6.	$0\to1=1$,	$1\to0=0$,	$1\to0=0$,	$0\to1=1$,
5.	$0\to0=1$,	$0\to0=1$,	$0\to0=1$,	$1\to1=1$,
4.	$1\to0=0$,	$0\to1=1$,	$1\to0=0$,	$0\to0=1$,
3.	$1\to1=1$,	$1\to0=0$,	$0\to1=1$,	$1\to0=0$,
2.	$1\to0=0$,	$1\to1=1$,	$1\to1=1$,	$1\to0=0$,
1.	$1\to0=0$,	$1\to0=0$,	$1\to0=0$,	$1\to0=0$.

Jak wspomnieliśmy, dowodzenie — ściślej wnioskowanie formalne — jest również procesem prostym. Obiektami są tutaj twierdzenia teorii matematycznej, natomiast operacje — to reguły wnioskowania, pozwalające w sposób formalny z twierdzeń już uznanych za prawdziwe otrzymywać

nowe twierdzenia. Przykład wnioskowania pokazany jest na rysunku 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 podstawieniu do wyrażenia, będącego prawym argumentem — na miejsce dowolnej, ale ustalonej, litery — wyrażenia, które jest lewym argumentem tej operacji. Otrzymane po podstawieniu wyrażenie jest wynikiem operacji. Operację podstawiania oznaczymy literą P ze wskaźnikiem u dołu wskazującym, na miejsce którego symbolu w prawym argumencie dokonujemy podstawienia. Np. P_x oznacza podstawienie na miejsce litery x. Operacja odrywania 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 oznaczymy 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:



Rys. 9

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 rysunku 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:

	P	$\overline{P}$	W	$\overline{W}$
5.	$e\ P_q\ b=g,$	$d\ P_q\ a=h,$	$d\ P_q\ a=h,$	$e\ P_q\ b=g,$
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 $\overline{W}$ nie różnią się od siebie, podobnie jak i porządki $\overline{P}$ i W. Ogólnie jednak porządki te są oczywiście różne(1).

Na zakończenie tego paragrafu jeszcze jeden przykład procesu prostego, tym razem nie z matematyki.

Przykładem procesu prostego może być splatanie sznurów. Najpierw z pojedynczych nitek skręcamy cienkie sznurki, z tych sznurków skręcamy liny, z lin powrozy. Obiektami takiego procesu są wszystkie nitki dane na początku oraz wszystkie otrzymane z nich sznury pośrednie. Operacjami natomiast są czynności skręcania. Oczywiście w procesie takim może być wiele różnych splotów, np. skręcanie w lewo, w prawo, splatanie 3, 4,

(1) W matematyce na ogół dowód formalny definiuje się następująco: ciąg formuł $F_1, F_2, \ldots, F_k$ $(k>0)$ nazywamy *dowodem formalnym* formuły F_k z formuł wyjściowych $D_1, D_2, \ldots, D_l$ $(l\geqslant 0)$, jeżeli każda formuła F_i jest jedną z formuł $D_1, D_2, \ldots, D_l$ lub aksjomatem, albo jest bezpośrednio otrzymana za pomocą ustalonych reguł wnioskowania z formuł ją poprzedzających (patrz [10]).

Podobnie możemy zdefiniować obliczenie. Dany jest ciąg liczb $D_1, D_2, \ldots, D_l$. *Obliczeniem* nazywamy ciąg liczb $L_1, L_2, \ldots, L_k$ $(k>0)$ taki, że każda liczba L_i jest jedną z liczb $D_1, D_2, \ldots, D_l$ lub jest otrzymana bezpośrednio z liczb poprzednich $L_1, L_2, \ldots, L_{i-1}$ za pomocą jednej z ustalonych operacji arytmetycznych, np. dodawania, odejmowania, mnożenia i dzielenia. Takie określenie jest jednak dla naszych celów niewygodne.

czy 5 sznurów itp. Proces taki możemy również przedstawić w postaci drzewa, zaznaczając w rozgałęzieniach rodzaj zastosowanego splotu, a gałęzie oznaczając symbolem użytego do danego splotu sznurka.

§ 5. PRZYKŁADY PROCESÓW NIEPROSTYCH

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

Rozpatrzmy np. dodawanie dwu liczb zapisanych w systemie pozycyjnym

$$\begin{array}{r} 98457 \\ + \\ 25189 \\ \hline 123646 \end{array}$$

Jeżeli w procesie tym 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 pojedynczych cyfrach, a nie jako operacje na ciągach 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 otrzymaliśmy 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 wiele twierdzeń.

Dowód jednego konkretnego twierdzenia jest procesem prostym, natomiast proces dowodzenia skończonej liczby twierdzeń w systemie sformalizowanym nie jest już procesem prostym.

W dalszym ciągu będziemy zajmowali się tylko procesami prostymi.

ROZDZIAŁ II

PROGRAMY PROCESÓW PROSTYCH

W tym rozdziale zajmiemy się sposobami opisywania procesów prostych. Pewnego rodzaju opisem procesów są drzewa, które używaliśmy już do przedstawienia procesów, jednakże w dalszym ciągu będziemy zajmowali się opisem liniowym, tj. opisem, w którym proces jest przedstawiony za pomocą liniowego ciągu symboli. Liniowy opis procesu prostego będziemy nazywali *programem*(1) tego procesu. Programy będziemy oznaczali dużymi literami greckimi, np. Φ, Ψ, Γ itp.

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

1. Program Φ wyznacza porządek operacji w procesie $\mathfrak{A}$

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 wiele 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 *języka*.

Językiem J będziemy nazywali skończony zbiór symboli A, zwany *alfabetem* języka J — wraz ze skończonym zbiorem reguł R, pozwalających z symboli alfabetu tworzyć wyrażenia poprawne, zwane tutaj *programami*. Zależnie od doboru alfabetu oraz reguł tworzenia programów można otrzymać 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 [17], [19], [21], [22], [23], [25], [28] i [30].

(1) Zamiast terminu *program*, można by też używać pojęć: *formuła procesu*, *algorytm procesu* lub *schemat procesu*.

§ 1. JĘZYK PODSTAWOWY (J_1)

Niech małe litery łacińskie oznaczają dane, symbol $*$ — wyniki częściowe, a duże litery łacińskie — operacje(1).

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

Ciąg

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

nazwiemy *normalnym programem podstawowym* procesu $\mathfrak{A}$, jeżeli dla każdego i $(1 \leqslant i \leqslant n)$ a_{2i+1}, a_{2i} są nazwami lewego i prawego argumentu działania D_i oraz $D_{i+1} \prec D_i$, gdzie $\prec$ oznacza relację porządkującą operacje procesu $\mathfrak{A}$ w porządku normalnym, tj. porządku P lub W (2).

Ciąg

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

nazwiemy *dualnym programem podstawowym* procesu $\mathfrak{A}$, jeżeli dla każdego i $(1 \leqslant i \leqslant n)$ a_{2i}, a_{2i+1} są nazwami lewego i prawego argumentu działania D_i oraz $D_i \succ D_{i+1}$, gdzie $\succ$ oznacza relację porządkującą operacje procesu $\mathfrak{A}$ w porządku dualnym, tj. $\overline{P}$ lub $\overline{W}$.

Dla przykładu rozpatrzmy formułę procesu przedstawionego na rysunku 10 (3).

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

(P) *efF bcC d*E ghG a*B **D **A *,*

(W) *ghG efF d*E **D bcC a*B **A *,*

$(\overline{P})$ ** A** Ba* D** Cbc Ed* Ggh Fef,*

$(\overline{W})$ ** A** D** Ggh Ed* Fef Ba* Cbc.*

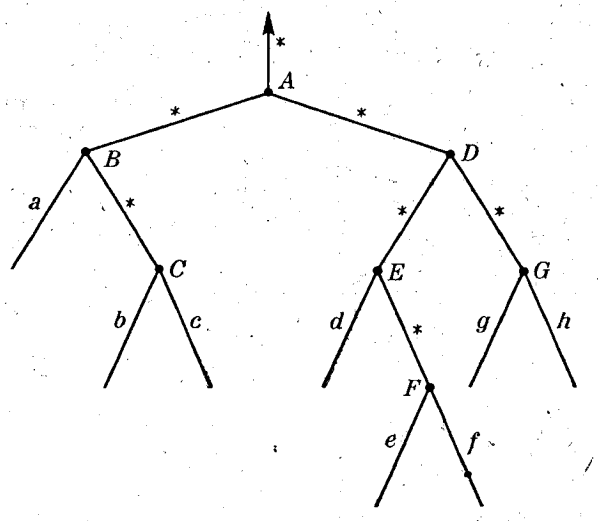
(1) Używając porównań gramatycznych, małe litery oraz symbol $*$ można by interpretować jako rzeczowniki naszego języka (nazwy obiektów), a duże litery — jako czasowniki (nazwy czynności).

(2) Innymi słowy działania są napisane w kolejności według numeracji P lub W.

(3) Jeszcze raz przypominamy, że symbole na rysunku nie są obiektami rozpatrywanego procesu, lecz ich nazwami.

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

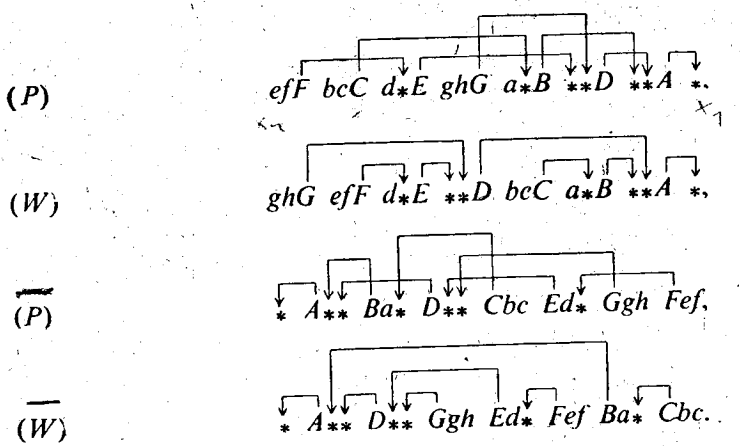
Dla porządków P i W programy należy czytać od strony lewej do prawej, dla porządków $\overline{P}$ i $\overline{W}$ — odwrotnie, tj. od strony prawej do lewej.



Rys. 10

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 proste reguły, ustalające zależność między symbolami działań a odpowiadającymi im symbolami wyników częściowych. Zanim przejdziemy do dokładniejszego opisania tych reguł, symbol wyniku częściowego każdego działania ustalimy na podstawie drzewa — zaznaczając symbole wyników częściowych odpowiednich działań — strzałkami, jak to pokazano na str. 25.

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

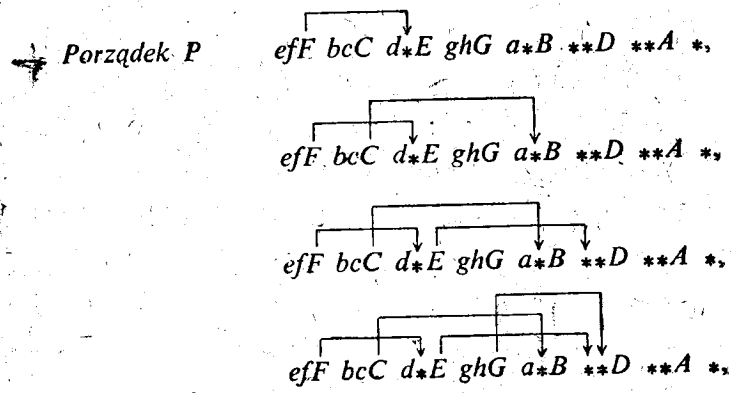


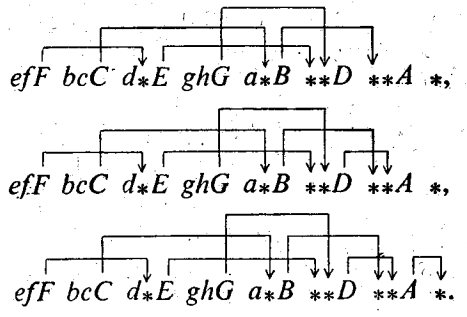
Dla porządków P i W zasada stawiania strzałek jest następująca:

Każdy symbol działania łączymy strzałką z najbliższym po jego prawej stronie wolnym symbolem wyniku częściowego: (symbol wyniku częściowego jest *wolny*, jeżeli nie został jeszcze połączony strzałką z żadnym symbolem działania). *Dla porządku P stawianie strzałek zaczynamy od pierwszego z lewej strony symbolu działania, dla porządku W od pierwszego z prawej strony symbolu działania.*

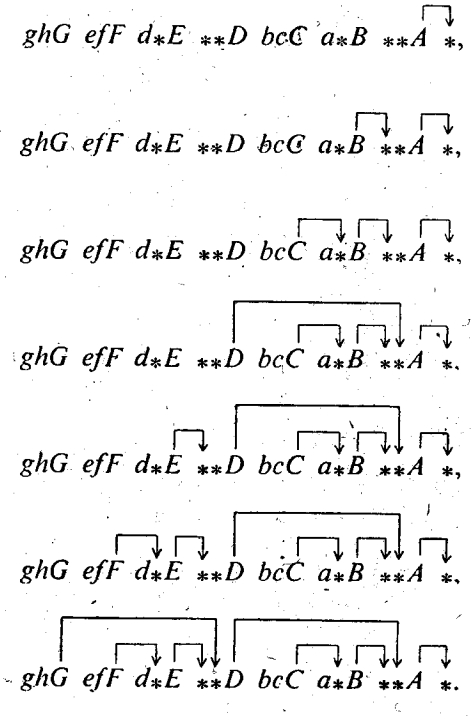
Podobną zasadę można podać dla porządków $\overline{P}$ i $\overline{W}$.

Kolejność stawiania strzałek ilustruje przykład:





Porządek W



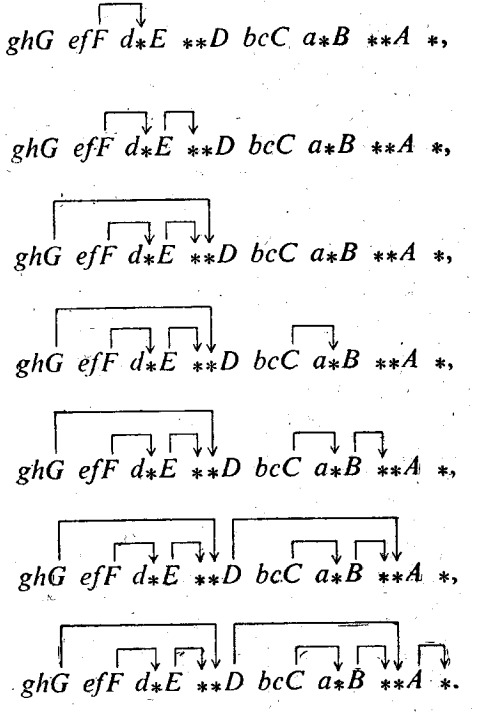
Dla porządku P stawianie strzałek rozpoczynamy od strony lewej do prawej, dla porządku W — odwrotnie, tj. od strony prawej do lewej. W tym ostatnim przypadku nie jest to czasem wygodne, a więc dla porządku W

regułę stawiania strzałek możemy sformułować tak, aby stawianie strzałek również wykonywać od strony lewej do prawej, podobnie jak dla porządku P. Reguła stawiania strzałek będzie miała wtedy postać:

W programie podstawowym Φ procesu $\mathfrak{A}$ z porządkiem W, każdy symbol wyniku częściowego łączymy z najbliższym wolnym symbolem działania z lewej strony; rysowanie strzałek zaczynamy od pierwszego symbolu wyniku częściowego z lewej strony.

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 $\mathfrak{A}$ z porządkiem $\mathfrak{P}$ i-temu symbolowi działania odpowiada i-ty symbol wyniku częściowego. Działania i symbole wyników częściowych są w programie 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 $\mathfrak{A}$ z porządkiem W oraz niech σ' oznacza następny po σ symbol programu Φ, a σ_i niech oznacza i-ty symbol działania w programie Φ.

Każdemu symbolowi działania σ_i w programie Φ przypiszemy funkcję $F_i(\sigma)$, określoną na zbiorze składającym się z σ_i oraz symboli leżących na prawo od σ_i.

1. $F_i(\sigma_i)=1$.
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 (2).

(1) Numeracja działań przyjęta w przykładzie jest odwrotna niż w podanej poprzednio 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 to ze względu na kierunek czytania formuły wygodniejsze.

(2) Działania są tu ponumerowane nie w porządku W, a odwrotnie, tj. w kolejności ich wykonywania.

Można wykazać, że:

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

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

Zastosowanie podanej reguły ilustruje poniższa tabelka (1):

	g	h	G	e	f	F	d	$*$	E	$*$	$*$	D	b	c	C	a	$*$	B	$*$	$*$	A	$*$
Numer 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

W kolejnych wierszach podane są wartości funkcji F_i dla poszczególnych działań.

§ 2. JĘZYK BEZNAWIASOWY (J_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 (2) (patrz rys. 11).

Ciąg

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

(1) Wykonanie tabelki kończymy na ostatnim zerze.

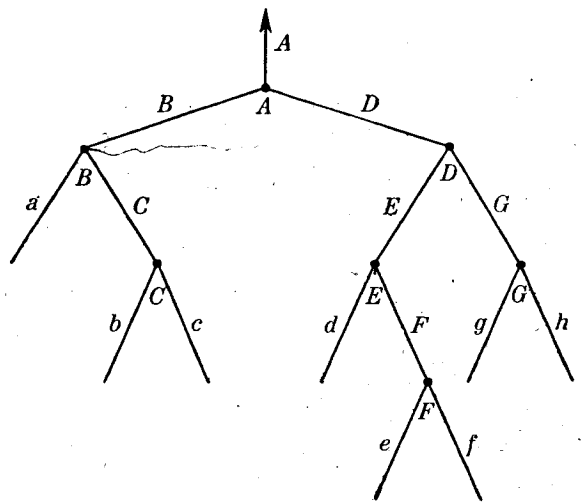
(2) Taka niejednoznaczność prowadzi czasem do nieporozumień. Dla ich uniknięcia będziemy w przypadkach wątpliwych wyraźnie pisali, czy chodzi nam o operację czy o jej wynik.

jest *normalnym programem beznawiasowym* w języku J_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 *dualnym programem beznawiasowym* w języku J_2, jeżeli dla każdego i $(1 \leqslant i \leqslant n)$ a_{2i}, a_{2i+1} są symbolami lewego i prawego argumentu działania numer i, przy dualnej numeracji działań, tj. $\overline{P}$ lub $\overline{W}$.



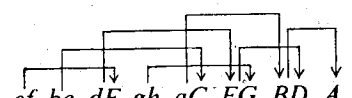
Rys. 11

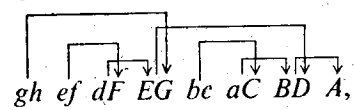
Program procesu przedstawionego na rysunku 11 w języku J_2, zależnie od porządku, przedstawiamy w jednej z postaci:

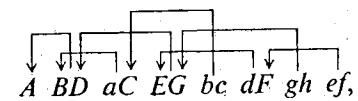
(P) *ef bc dF gh aC EG BD A,*
(W) *gh ef dF EG bc aC BD A,*
$(\overline{P})$ *A BD aC EG bc dF gh ef,*
$(\overline{W})$ *A BD EG gh dF ef aC bc.*

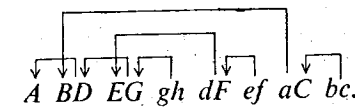
Dla jednoznacznego odczytania programów beznawiasowych musimy wiedzieć, który symbol działania jest skojarzony z każdą parą symboli argumentów.

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

(P) 

(W) 

$(\overline{P})$ 

$(\overline{W})$ 

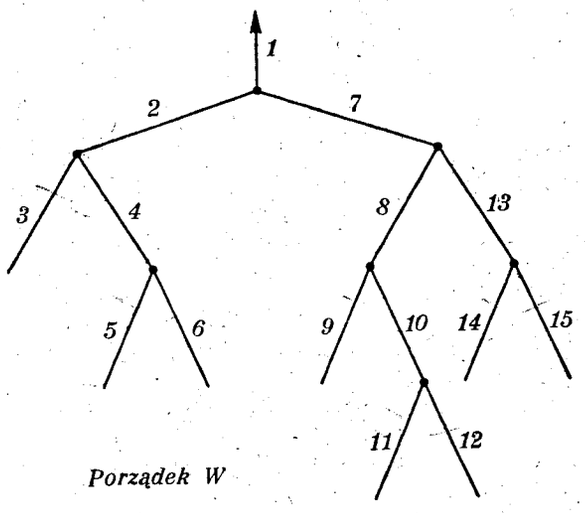
§ 3. JĘZYK ŁUKASIEWICZA (J_3)

Logik polski Jan Łukasiewicz wprowadził symbolikę stosowaną czasem w logice matematycznej [11].

Język ten możemy również określić, wychodząc z pojęcia procesu. Przyjmiemy jako alfabet języka J_3 małe oraz duże litery łacińskie oraz przyjmijmy jak w językach beznawiasowych, że wyniki częściowe są oznaczane odpowiadającymi im symbolami operacji.

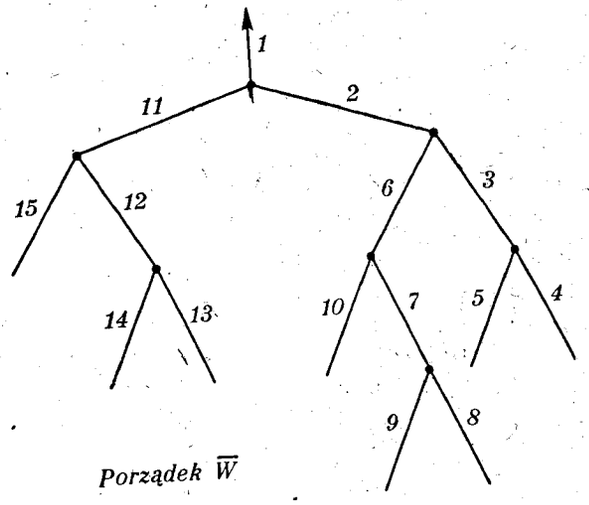
Dla określenia języka J_3 przyjmiemy, że obiekty procesu są ponumerowane tak, jak to pokazano na rysunkach 12 i 13(1).

(1) Zasady tej numeracji są identyczne jak numeracji działań w porządku W i $\overline{W}$, dlatego będziemy je oznaczali również symbolicznie W i $\overline{W}$ i będziemy nazywali *numeracją wzdłużną*.



Porządek W

Rys. 12



Porządek $\overline{W}$

Rys. 13

Ciąg

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

nazwiemy *normalnym programem* w języku Łukasiewicza, jeżeli dla każdego i $(1 \leqslant i \leqslant 2n+1)$ a_i jest symbolem obiektu o numerze i, przy numeracji W.

Ciąg

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

nazwiemy *dualnym programem* w języku Łukasiewicza, jeżeli dla każdego i $(1 \leqslant i \leqslant 2n+1)$ a_i jest symbolem obiektu o numerze i, przy numeracji $\overline{W}$.

Przykład programu w języku Łukasiewicza procesu pokazanego na rysunku 11 — dla porządków W i $\overline{W}$ — ma postać

(W) $\quad A\ B\ a\ C\ b\ c\ D\ E\ d\ F\ e\ f\ G\ g\ h$

$(\overline{W})$ $\quad a\ b\ c\ C\ B\ d\ e\ f\ F\ E\ g\ h\ G\ D\ A.$

Łatwo zauważyć, że w programie normalnym Łukasiewicza operacje są wpisane w porządku W, a w programie dualnym Łukasiewicza — w porządku $\overline{W}$ (1).

W języku Łukasiewicza procesu z porządkiem W lewy argument każdego działania jest oznaczony sąsiednim symbolem z prawej strony 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 najbliższym z prawej symbolem σ_r takim, że $k<r$, i σ_r nie jest symbolem lewego argumentu dla żadnego działania, oraz nie istnieje w programie takie działanie σ_t, $k<t<r$, dla którego σ_r jest symbolem prawego argumentu.

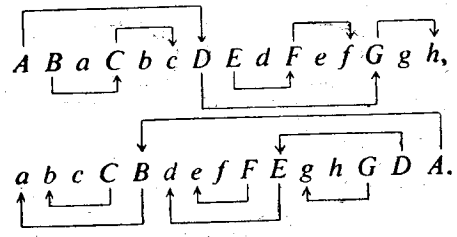
Podobnie możemy określić lewe argumenty w programie Łukasiewicza procesu z porządkiem $\overline{W}$.

Zaznaczając szukane argumenty każdego działania strzałkami — dla

(1) Należy pamiętać, że dla określenia języka Łukasiewicza numerowaliśmy nie operacje w procesie, a obiekty. Oczywiście w ten sposób ustaliliśmy jednocześnie porządek operacji.

(2) Przypominamy, że symbole w programie Łukasiewicza procesu z porządkiem W są numerowane kolejno od strony lewej do prawej.

porządków W i $\overline{W}$ 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).

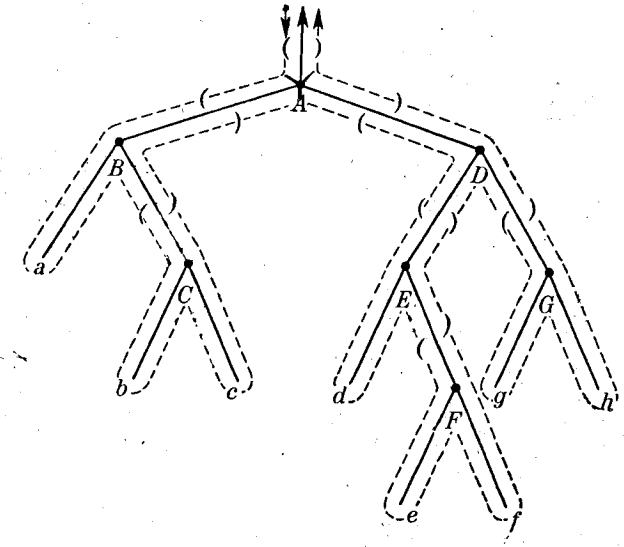
Przykład zastosowania funkcji $F'_i(\sigma)$ do określenia prawych argumentów przedstawiony jest niżej:

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

(1) Zakładamy, że σ nie jest symbolem działania σ_i, bo wtedy również $F'_i(\sigma)=0$.

§ 4. JĘZYK NAWIASOWY (J_4)

Znany, powszechnie w matematyce stosowany język nawiasowy można również wyprowadzić wychodząc z pojęcia procesu [31]. Wyjaśnimy to na przykładzie procesu pokazanego na rysunku 14. Gałęzie drzewa, które odpowiadają danym nazwiemy gałęziami *wolnymi*, pozostałe gałęziami *związanymi*. 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 rysunku 14, a operacje dużymi literami.



Rys. 14

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 otrzymane wyrażenie jest formułą nawiasową rozpatrywanego procesu. Tak więc formuła nawiasowa jest również opisem struktury drzewa, tzn. procesu prostego.

Na podstawie rysunku 14 otrzymamy więc program

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

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

Dla języka nawiasowego można podać jednak algorytmy, pozwalające wyznaczyć porządek P, $\overline{P}$, W lub $\overline{W}$ 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
$\overline{P}$	2	4	1	5	7	3	6
W	2	3	1	5	6	4	7
$\overline{W}$	6	7	1	4	5	2	3

Porządek wzdłużny. Numery działań dla porządku W są jednoznacznie wyznaczone przez nawiasy lewostronne, natomiast numeracja $\overline{W}$ jest jednoznacznie wyznaczona nawiasami prawostronnymi.

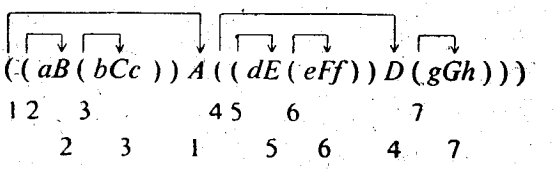
Wyjaśnimy to na przykładzie. Ponumerujmy nawiasy lewostronne kolejnymi liczbami naturalnymi 1, 2, ..., k, poczynając od pierwszego nawiasu z lewej strony. Każdemu nawiasowi lewostronnemu przypiszemy najbliższy wolny symbol z prawej strony.

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

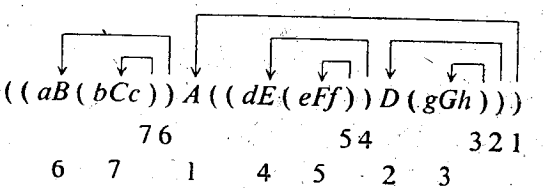
Postępując podobnie dla nawiasów prawostronnych, otrzymamy numerację $\overline{W}$.

Przykład zastosowania tej zasady dla numeracji działań:

Porządek W



Porządek $\overline{W}$



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

Przyjmijmy, że nawiasy lewostronne są ponumerowane jak poprzednio. Z każdym lewostronnym nawiasem nr i skojarzymy funkcję $K_i(\sigma)$, określoną rekurencyjnie:

1. $K_i(\sigma)=1$, jeżeli σ jest nawiasem lewostronnym o numerze i.

2. $K_i(\sigma')=K_i(\sigma)+1$, jeżeli σ' jest nawiasem lewostronnym albo symbolem danej.

3. $K_i(\sigma')=K_i(\sigma)-1$, jeżeli σ' jest nawiasem prawostronnym albo symbolem operacji.

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

Porządek poprzeczny. Obecnie określimy funkcję numerującą działania w programie nawiasowym w porządku $\overline{P}$. Dla określenia porządku P można postąpić podobnie.

Najpierw określimy funkcję $H(\sigma_i)$, przyporządkowującą każdemu symbolowi formuły liczbę, zwaną *rzędem* tego symbolu; σ_i jest i-tym symbolem rozpatrywanego programu.

Tabelka podaje wartości funkcji K_i dla dyskutowanego przykładu.

	(	(	a	B	(	b	C	c	)	)	A	(	(	d	E	(	e	F	f	)	)	D	(	g	G	h	)	)	)
Numer nawiasu	1	2			3						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					
Numer operacji				2			3				1				5			6					4			7			

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

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

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 $\overline{P}$. Funkcja $G(\sigma)$ jest określona rekurencyjnie:

1. $G(\sigma_1^1)=0$,

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

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

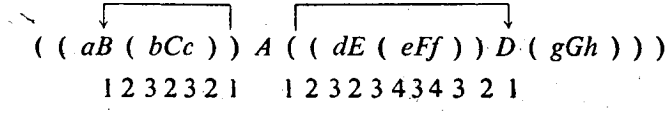
Wartości funkcji H i G dla rozpatrywanego przykładu podane są w tabelce.

	(	(	a	B	(	b	C	c	)	)	A	(	(	d	E	(	e	F	f	)	)	D	(	g	G	h	)	)	)
$H(\sigma_i)$	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
$G(\sigma_i)$	0	1	3	2	3	6	4	6	4	2	1	2	4	6	5	6	7	7	7	7	5	3	5	7	6	7	6	3	1

Argumenty działań. Określiliśmy już algorytm numerowania działań w programie nawiasowym w dowolnym z dyskutowanych porządków: P, $\overline{P}$, W i $\overline{W}$. Pozostało jeszcze określenie algorytmu, przypisującego 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_i, jeżeli jest natomiast — prawostronny, to odpowiadające mu działanie znajdziemy za pomocą funkcji wyznaczającej porządek $\overline{W}$, która jest określona podobnie jak funkcja K_i.

Np. argumentami działania A w dyskutowanym przykładzie są obiekty uzyskane 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 porządków, 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: *σ-językami*. Dla dalszych rozważań będzie czasem wygodniej pisać programy w nieco innej postaci niż w rozdziale II. W niniejszym rozdziale podamy jeszcze inne postacie programów dla poprzednio określonych języków.

§ 1. JĘZYKI PRZEDMIOTOWE

Program określiliśmy jako ciąg symboli oznaczających obiekty oraz operacje, które spełnia pewne warunki. Jeżeli jednak w programie — zamiast symboli oznaczających obiekty — występują same obiekty procesu, to 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.

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 przedmiotowymi*, a odpowiednie języki — *językami przedmiotowymi*, lub krócej: *π-językami*.

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 rysunku 2, ma postać

$$(P) \qquad 34+\ 51+\ 9*-\ 85-\ 9*-\ **\cdot\ **+\ *\ (2).$$

Dla pozostałych porządków π-programy będą miały postać podobną. π-program przedstawia proces w stanie początkowym. Analogicznie możemy przedstawić proces w dowolnym stanie. π-program procesu po wykonaniu i operacji będziemy oznaczać π_i. W szczególności π_0 oznacza π-program procesu, w którym nie zrealizowano jeszcze żadnej operacji.

Np.

$$(P) \qquad \pi_3 \quad 34+\ 51+\ \overset{*}{9}7-\ 85-\ \overset{*}{9}6-\ \overset{*}{2}*\cdot\ **+\ *$$

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. w języku Łukasiewicza:

$$\pi_0 \quad +-9+51\cdot\ -9+34-85,$$

$$\pi_3 \quad +-9+51\cdot\overset{*}{2}\overset{*}{9}7\overset{*}{3}43\overset{*}{8}5.$$

Odczytanie programu π_3 nie przedstawia trudności. Należy pamiętać, że w przykładzie wszystkie liczby są jednocyfrowe, a więc np. 51 oznacza dwie liczby 5 i 1, a nie liczbę pięćdziesiąt jeden.

Podobne określenia można wprowadzić do języka nawiasowego.

---

(1) Przyjęliśmy, że nawiasy są również symbolami wyników częściowych.

(2) W przykładzie wszystkie liczby są jednocyfrowe, a więc np. 34 oznacza dwie liczby 3 i 4.

§ 2. JĘZYKI UPROSZCZONE

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 uproszczonym*, 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:

(P) $\quad 11F\ 11C\ 10E\ 11G\ 10B\ 00D\ 00A\ 0$

$\quad A\ B\ 1\ C\ 1\ 1\ D\ E\ 1\ F\ 1\ 1\ G\ 1\ 1$

$\quad (\,(\,B\,(\,C\,)\,)\,A\,(\,(\,E\,(\,F\,)\,)\,D\,(G)))$

Drugi μ-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) — oba 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 można by nazwać semantyczną, 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 σ-języku podstawowym.

1. Jeżeli α i β oznaczają małe litery łacińskie, a Δ — dużą literę 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 danej gwiazdką.

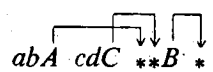
Np. jeżeli Φ ma postać $abA*$ oraz Ψ ma postać $deB*$, to (Φ) jest abA, natomiast $[\Psi]=*eB*$, a więc wyrażenie $(\Phi)\,[\Psi]=abA*eB*$ jest programem.

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

$$abA\ cdC\ **B\ *$$

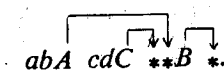
możemy czytać jako

(P)



lub

(W)



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

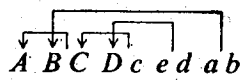
1. Każda mała litera jest programem.

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

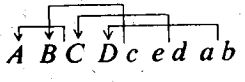
Np. jeżeli α jest *Bab* oraz β jest *Ccd*, to *ABabCcd* jest programem. Nie wiemy jednak z definicji indukcyjnej, czy otrzymany program jest

w języku Łukasiewicza czy też w języku beznawiasowym i w ostatnim przypadku nie znamy 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.

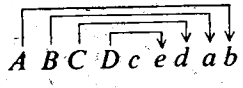
1. Język beznawiasowy, porządek $\overline{P}$



2. Język beznawiasowy, porządek $\overline{W}$



3. Język Łukasiewicza, porządek W



W każdym przypadku program opisuje inny proces.

Definiując indukcyjnie program nawiasowy, nie otrzymamy niejednoznaczności, bowiem struktura programu nawiasowego nie zależy od porządku opisywanego procesu.

ROZDZIAŁ IV

REALIZACJA JĘZYKÓW PRZEDMIOTOWYCH

Obecnie zajmiemy się badaniem ogólnej struktury maszyny, realizującej procesy sekwencyjne według programu opisującego ten proces w dowolnym języku przedmiotowym, ale dla danej maszyny ustalonym [14].

W dalszym ciągu zamiast pisać: maszyna realizuje proces $\mathfrak{A}$ według programu Φ będziemy pisali krótko: maszyna realizuje program Φ. Jeżeli maszyna realizuje programy w języku przedmiotowym (π-języku), to powiemy, że proces jest realizowany przez maszynę bezpośrednio, a maszynę nazwiemy *π-maszyną*. π-maszynę realizującą język podstawowy (J_1) z porządkiem P oznaczymy przez $\pi(J_1(P))$. Podobne oznaczenie wprowadzimy dla innych języków i porządków. Np. $\pi(J_4(W))$ oznacza maszynę realizującą przedmiotowy język nawiasowy z porządkiem W.

W niniejszym rozdziale podamy ogólną strukturę π-maszyny dla różnych języków podanych w rozdziale II.

§ 1. OGÓLNY SCHEMAT π-MASZYNY

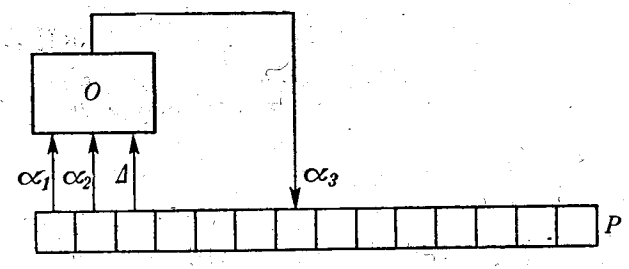
Uproszczony schemat ogólny maszyny realizującej bezpośrednio procesy proste pokazany jest na rysunku 15.

Maszyna składa się z trzech zasadniczych elementów:

1. O — operatora,
2. P — pamięci,
3. S — sterowania (nie pokazanego na rysunku).

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

(1) Jeżeli proces jest obliczeniem logicznym, to operator wykonuje operacje logiczne. W niniejszej książce nie będziemy się bliżej zajmowali budową wewnętrzną operatora, przyjmiemy tylko, że realizuje on wszystkie potrzebne operacje.



Rys. 15

Operator posiada trzy wejścia α_1, α_2, Δ oraz jedno wyjście α_3.

α_1 — jest wejściem lewego argumentu,
α_2 — jest wejściem prawego argumentu,
Δ — jest wejściem symbolu działania,
α_3 — jest wyjściem wyniku operacji.

Operator 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.

Pamięć 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 sterowanie, 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, to powiemy, że maszyna jest *w stanie i.*

Wszystkie czynności maszyny, związane z wykonaniem jednej operacji procesu nazwiemy *cyklem pracy maszyny*. 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.

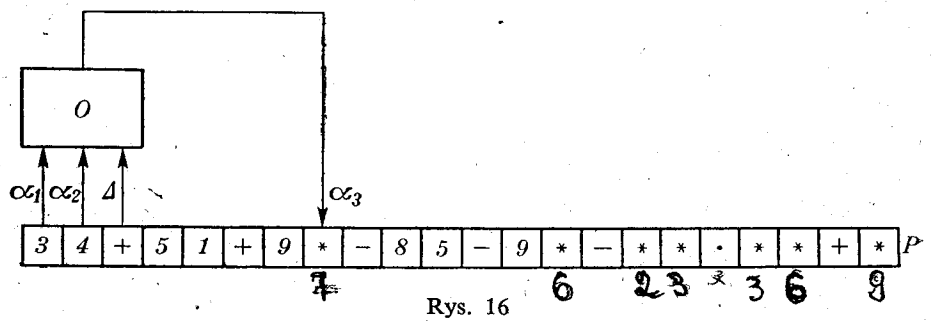
4. Odszukanie w pamięci miejsca, gdzie ma być umieszczony wynik operacji oraz zapisanie w nim otrzymanego wyniku.

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

§ 2. REALIZACJA JĘZYKA PODSTAWOWEGO Z PORZĄDKIEM POPRZECZNYM

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

Na rysunku 16 pokazana jest maszyna w stanie początkowym, dla procesu przedstawionego na rysunku 2.



Rys. 16

Stan początkowy maszyny.

1. W pamięci maszyny wpisany jest π-program w języku podstawowym.

2. Wejścia α_1, α_2, Δ operatora O są nastawione na odczytywanie pierwszych trzech (licząc od lewej strony) komórek pamięci.

3. Wyjście α_3 operatora O jest nastawione na zapisanie w pierwszej komórce, w której wpisany jest symbol wyniku częściowego(1).

(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.

Cykl pracy maszyny.

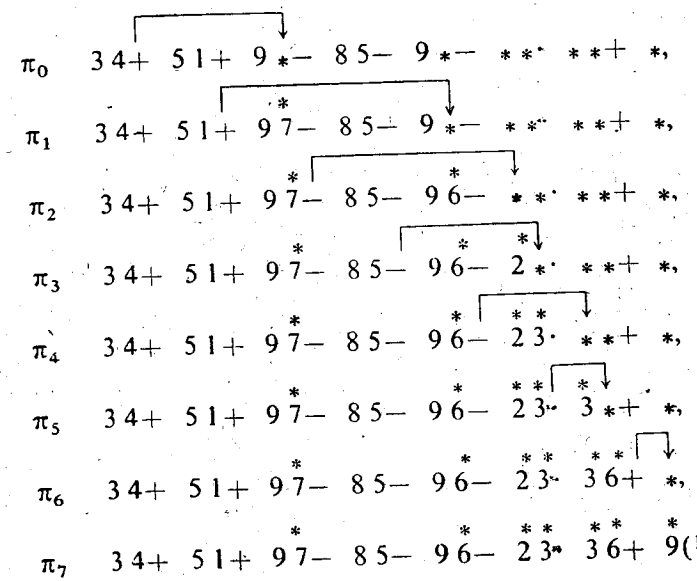
1. Wejścia α_1, α_2, Δ są przesuwane o trzy komórki w prawo.

2. Operator O jest nastawiony na odczytanie działania. Argumenty są przesłane do operatora.

3. Wyjście α_3 operatora O jest przesuwane do najbliższej z prawej strony niezapisanej kartki pamięci, gdzie zapisywany jest wynik działania.

W opisie cyklu pracy maszyny nie podano sposobu zakończenia procesu. Przyjmiemy, 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.

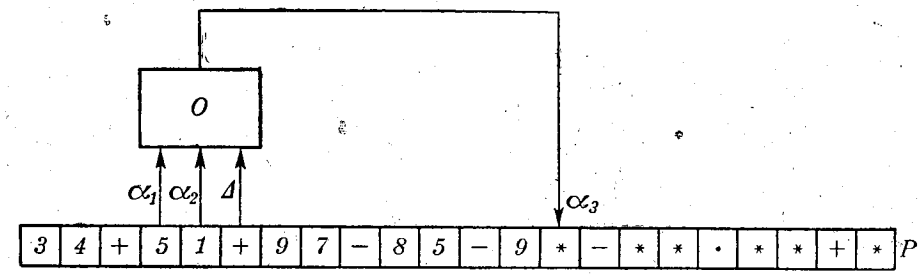
Kolejne stany procesu są następujące:



Stan maszyny $\pi(J_1(P))$ dla procesu w stanie π_1 pokazany jest na rysunku 17. Dla pozostałych stanów procesu stan maszyny można narysować podobnie.

(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.

Przedstawiona koncepcja maszyny wymaga analizowania symboli w kolejnych miejscach pamięci, w celu znalezienia komórki, w której ma być umieszczony wynik częściowy.



Rys. 17

§ 3. REALIZACJA JĘZYKA PODSTAWOWEGO Z PORZĄDKIEM WZDŁUŻNYM

Ogólny schemat oraz zasada działania maszyny $\pi(J_1(W))$ oraz maszyny $\pi(J_1(W))$ 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 ten fragment cyklu pracy, który dotyczy umieszczenia wyniku w pamięci.

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

Sterowanie ma więc dodatkowe urządzenie, które realizuje funkcję $F_i(\sigma)$. Urządzeniem tym jest licznik L, do którego można dodawać 0,1 lub −1. W celu znalezienia komórki, w którym ma być umieszczony wynik działania Δ, urządzenie sterujące „bada" zawartość wszystkich komórek pamięci, poczynając od komórki, w której znajduje się symbol Δ, i zależnie od odczytanego symbolu, zgodnie z wzorem w rozdziale II, § 1, str. 24, dodaje do licznika 0,1 lub −1. Komórka, dla której licznik L przyjmie wartość 0, jest miejscem, gdzie należy wpisać wynik częściowy.

Kolejne stany obliczenia dla porządku W są następujące:

π_0 85− 34+ 9*− **· 51+ 9*− **+ *,

π_1 85− 34+ 9*− *3· 51+ 9*− **+ *,

π_2 85− 34+ 97− *3· 51 +9*− **+ *,

π_3 85− 34+ 97− 23· 51+ 9*− **+ *,

π_4 85− 34+ 97− 23· 51+ 9*− *6+ *,

π_5 85− 34+ 97− 23· 51+ 96− *6+ *,

π_6 85− 34+ 97− 23· 51+ 96− 36+ *,

π_7 85− 34+ 97− 23· 51+ 96− 36+ 9.

Działanie π-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 $\overline{W}$. 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 maszyny.

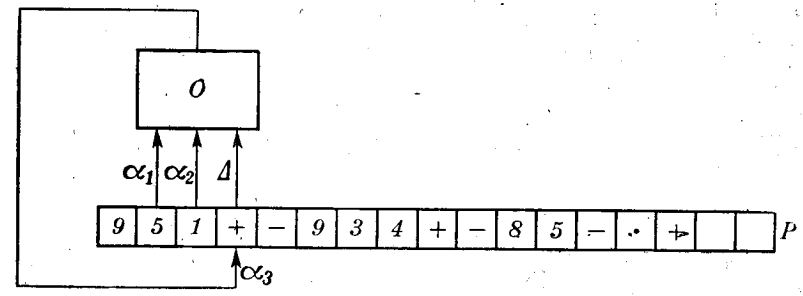
1. W pamięci maszyny znajduje się π-program w języku Łukasiewicza.

2. Niech symbol pierwszej operacji, która ma być wykonana znajduje się w kratce o nr i (1) (patrz rys. 18). Wejście Δ jest nastawione na odczyt

(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.

zawartości kratki i, wejście α_2 — na odczyt kratki $i+1$, oraz wejście α_1 — na odczyt kratki $i+2$.

3. Wyjście α_3 jest nastawione na zapisanie wyniku częściowego w kratce i(1).



Rys. 18

Cykl pracy maszyny.

1. Wejście Δ jest przesuwane do następnej kolejnej kratki pamięci, w której znajduje się następny symbol działania. Wejście α_2 jest przesuwane do kratki sąsiadującej z lewej strony z nowym symbolem działania. Wejście α_1 na podstawie obliczenia wartości funkcji $F_i'(\sigma)$ jest nastawione na odczytanie lewego argumentu(2).

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

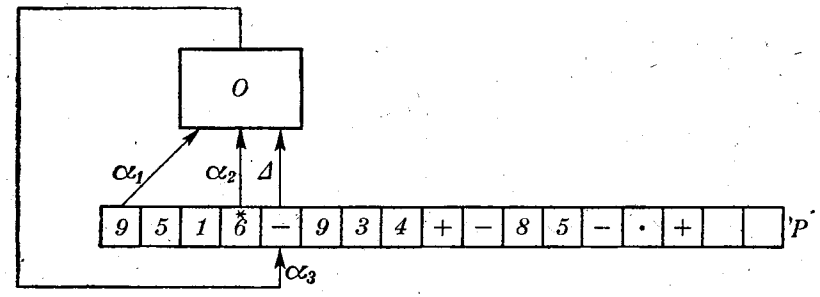
Obliczenie funkcji $F_i'(\sigma)$ jest podobne do obliczenia funkcji $F_i(\sigma)$, określającej miejsce umieszczenia wyniku częściowego w języku podstawowym z porządkiem W. A więc maszyna realizująca język Łukasiewicza posiada licznik L, zwany obecnie *licznikiem lewego argumentu*, którego dzia-

(1) Przyjmujemy, że wyniki częściowe są zapisywane w pamięci na miejsce odpowiadających im symboli działań.

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

łanie jest identyczne jak w maszynie poprzedniej, z tą tylko różnicą, że zawartość początkowa licznika jest 0, a nie jak poprzednio 1.

Po częściowym zrealizowaniu obliczenia w pamięci na miejscu symboli działań znajdują się liczby. Aby było możliwe obliczenie funkcji $F_i(\sigma)$, konieczne jest posiadanie informacji o tym, że w danym miejscu pamięci znajdował się symbol działania. Fakt ten zaznaczymy gwiazdką, jak to pokazano na rysunku 19.



Rys. 19

Kolejne stany procesu są następujące:

π_0 9 5 1 + − 9 3 4 + − 8 5 − · +,

π_1 9 5 1 6 − 9 3 4 + − 8 5 − · +,

π_2 9 5 1 6 3 9 3 4 + − 8 5 − · +,

π_3 9 5 1 6 3 9 3 4 7 − 8 5 − · +,

π_4 9 5 1 6 3 9 3 4 7 2 8 5 − · +,

π_5 9 5 1 6 3 9 3 4 7 2 8 5 3 · +,

π_6 9 5 1 6 3 9 3 4 7 2 8 5 3 6 +,

π_7 9 5 1 6 3 9 3 4 7 2 8 5 3 6 9.

Wyniki częściowe są oznaczone 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 ilością 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 wpisaniu na miejscu symbolu operacji jej wyniku oraz wymazaniu obu argumentów operacji.

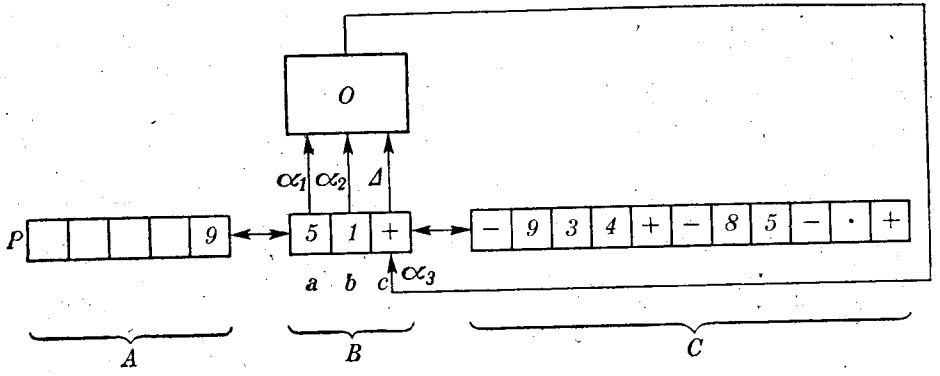
Maszyna działająca na zasadzie redukowania programu, w każdym cyklu wykonuje ostatnie działanie programu(1), a następnie program redukuje. Schemat takiej maszyny pokazany jest na rysunku 20. Aby czytać program od strony lewej do prawej przyjęto porządek $\overline{W}$.

Pamięć maszyny składa się z trzech części A, B, C. 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 B, jak to pokazano na rysunku 20. Działanie maszyny jest następujące:

(1) Dla porządku $\overline{W}$ proces redukcji przebiega identycznie z tym, że redukowane jest zawsze pierwsze działanie z lewej strony.

Stan początkowy maszyny.

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



Rys. 20

Cykl pracy maszyny.

1. Operator odczytuje symbol działania z komórki c pamięci B, oraz argumenty z komórek a i b pamięci B. Wynik działania jest wypisywany 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:

π_0 9 5 1 + − 9 3 4 + − 8 5 − · +,
π_1 9 6 − 9 3 4 + − 8 5 − · +,
π_2 3 9 3 4 + − 8 5 − · +,
π_3 3 9 7 − 8 5 − · +,
π_4 3 2 8 5 − · +,
π_5 3 2 3 · +,
π_6 3 6 +,
π_7 9

§ 6. REALIZACJA JĘZYKA NAWIASOWEGO ZA POMOCĄ REDUKCJI PROGRAMU

Działanie π-maszyny realizującej π-język nawiasowy jest podobne do działania maszyn poprzednich. W pamięci maszyny jest zapisany π-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 wykonanego działania. Dokładne przestudiowanie tego schematu nie przedstawia trudności.

Maszyna realizująca π-język nawiasowy może również działać na zasadzie redukcji programu. Redukcja π-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 następu-

jące:

$$\Phi^0 \quad (((a\,C\,b)\,B\,c)\,A\,(d\,E\,(e\,F\,f))),$$

$$\Phi^1 \quad (((a\,C\,b)\,B\,c)\,A\,(d\,E\,*)),$$

$$\Phi^2 \quad (((a\,C\,b)\,B\,c)\,A\,*),$$

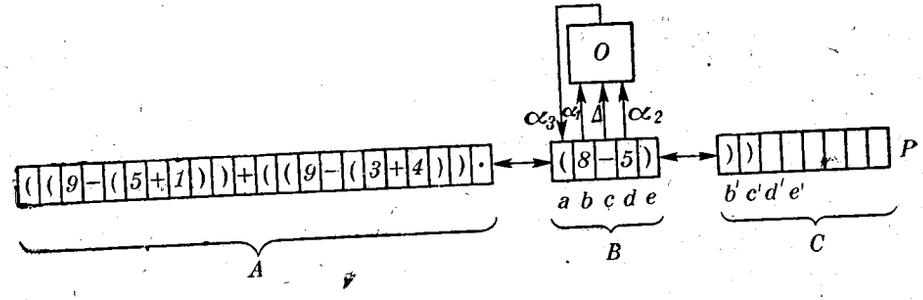
$$\Phi^3 \quad ((*\,B\,c)\,A\,*),$$

$$\Phi^4 \quad (*\,A\,*),$$

$$\Phi^5 \quad *$$

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 $\pi(J_4(W))$, realizującej π-język nawiasowy pokazano na rysunku 21. Pamięć maszyny jest podzielona na 3 części A,



Rys. 21

B, C. Pamięć B posiada pięć komórek a, b, c, d, e. Wejścia i wyjście operatora O są na stałe przyłączone do odpowiednich komórek pamięci B, jak to pokazano na rysunku 21.

Stan początkowy maszyny.

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 maszyny.

1. Zapisanie wyniku operacji w komórce a.
2. Wymazanie 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' znajdzie 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:

$$\pi_0 \quad ((9-(5+1))+((9-(3+4))\cdot(8-5))),$$

$$\pi_1 \quad ((9-(5+1))+((9-(3+4))\cdot \overset{*}{3})),$$

$$\pi_2 \quad ((9-(5+1))+(\overset{*}{(}9-7)\cdot \overset{*}{3})),$$

$$\pi_3 \quad ((9-(5+1))+(\overset{*}{2}\cdot \overset{*}{3})),$$

$$\pi_4 \quad ((9-(5+1))+\overset{*}{6}),$$

$$\pi_5 \quad ((9-6)\overset{*}{)}+\overset{*}{6}),$$

$$\pi_6 \quad (\overset{*}{(}3+6)\overset{*}{)},$$

$$\pi_7 \quad \overset{*}{9}.$$

Dla przejrzystości redukowane wyrażenia zaznaczono klamrami oraz wyniki operacji — gwiazdkami. Oczywiście symbole $*$, $\sqcup$ nie należą do π-języka.

§ 7. UWAGI OGÓLNE O BEZPOŚREDNIEJ REALIZACJI PROCESÓW SEKWENCYJNYCH

Dyskusja realizacji bezpośredniej procesów miała głównie na celu zapoznanie Czytelnika z ogólnymi własnościami języków, które będą przydatne w rozdziałach następnych.

Przedstawione w tym rozdziale zasady realizacji maszyn mają ograniczone perspektywy zastosowania 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 wiele 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 obliczenia 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.

ROZDZIAŁ V

REALIZACJA JĘZYKÓW UPROSZCZONYCH ZA POMOCĄ MASZYNY Z ODDZIELNĄ PAMIĘCIĄ WYNIKÓW CZĘŚCIOWYCH

Obecnie rozpatrzymy drugą grupę maszyn realizujących procesy proste [22]. 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 oznaczymy przez μ_1, drugą przez μ_2, a maszyny należące do odpowiednich klas będziemy nazywali *μ_1-maszynami* lub *μ_2-maszynami*. W niniejszym rozdziale zostanie opisana klasa μ_1, w rozdziale następnym klasa μ_2.

§ 1. POJĘCIE ROZKAZU

Zanim opiszemy schemat μ-maszyn określimy pojęcie *rozkazu* programu. Niech Φ będzie programem uproszczonym i niech σ_i będzie symbolem działania nr i w programie Φ. Niech $l(\sigma_i)$ oraz $p(\sigma_i)$ oznaczają symbole lewego i prawego argumentu programu Φ.

i-tym rozkazem programu Φ, symbolicznie $R_i(\Phi)$ (lub krótko R_i) nazwiemy trójkę symboli $\{l(\sigma_i), p(\sigma_i), \sigma_i\}$(1).

Oczywiście μ-program zawierający n symboli operacji składa się z n rozkazów.

W języku podstawowym rozkazami są kolejne trójki symboli w programie.

Np. w σ-programie

(P) $\qquad abA\ cdB\ {**}C\ *$

(1) Podobnie możemy określić pojęcie *rozkazu* dla języków symbolicznych.

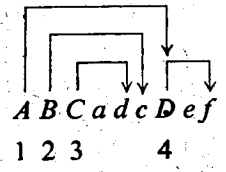
rozkazami są

R_3 abA,
R_2 cdB,
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 beznawiasowych oraz Łukasiewicza symbole rozkazu R_i 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 σ-programie Łukasiewicza



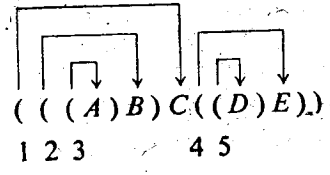
mamy następujące rozkazy:

R_4 $D\,e\,f$
R_3 $C\,a\,d$
R_2 $B\,C\,c$,
R_1 $A\,B\,D$.

W μ-języku 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 składające się na dowolny rozkaz są w programie napisane obok siebie, np. w programie



mamy następujące rozkazy(1):

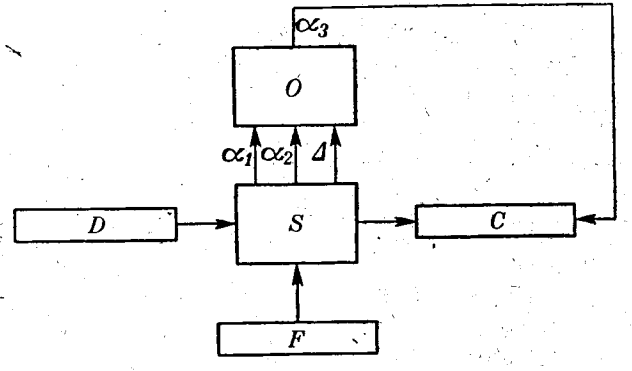
R_5 $(D$,
R_4 $)E$,
R_3 $(A$,
R_2 $)B$,
R_1 $)C($.

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

§ 2. OGÓLNY SCHEMAT μ_1-MASZYNY

Ogólny schemat μ-maszyny jest pokazany na rysunku 22. Maszyna składa się z:

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



Rys. 22

Rola i działanie operatora jest identyczne jak w maszynach opisanych w poprzednim rozdziale. Urządzenie sterujące S analizuje program w pa-

(1) Nawiasy z lewej i prawej strony każdego symbolu działania uważamy za symbole argumentów (patrz rozdział III, § 2).

mięci P w jednym z porządków P, $\overline{P}$, W lub $\overline{W}$. Wynik każdej operacji jest umieszczany według odpowiedniego algorytmu w pamięci wyników częściowych C. Argumenty każdego działania, zależnie od tego czy są danymi, czy wynikami częściowymi, są pobierane z pamięci D lub C.

Jeżeli maszyna pracuje w języku, w którym działania w programie są zapisane w jednym z porządków P, $\overline{P}$, W lub $\overline{W}$, sterowanie wykonuje kolejne operacje programu — w przypadku przeciwnym (np. w języku nawiasowym) 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 wykonywania działań(1).

Działanie μ_1-maszyny jest następujące:

Stan początkowy maszyny.

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

Cykl pracy maszyny.

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(2).

(1) Jest to istotna cecha maszyn klasy μ_1.

(2) Przypominamy, że umówiliśmy się nie podawać jeszcze jednego kroku, a mianowicie — wykonania operacji przez operator O.

Dla porządków P i W pierwszym odczytywanym argumentem jest lewy argument, dla porządków $\overline{P}$ i $\overline{W}$ — 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. PAMIĘĆ WYNIKÓW CZĘŚCIOWYCH

W niniejszym paragrafie określimy dwa rodzaje pamięci wyników częściowych C_1 i C_2, które nazwiemy odpowiednio pamięcią *liniową* i pamięcią *rewersyjną*. Najpierw określimy pamięć liniową C_1. Niech C_1 posiada komórki $\{c_1, c_2, \ldots, c_k\}$. 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). W pamięci w każdej chwili może być przygotowana tylko jedna komórka do odczytu oraz tylko jedna komórka do zapisu. W pamięci można zapisać tylko komórkę przygotowaną do zapisu, natomiast z 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 realizowanych w pamięci C_1:

$(\vec{C_1})$ — odczytaj zawartość komórki c_i i przygotuj do odczytu komórkę c_{i+1},

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

$\vec{[C_1]}$ — 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 tylko 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 $\{s, t\}$. Jeżeli maszyna wykonała i-tą operację, to powiemy, że pamięć (C_1) maszyny jest w stanie $S_i(C_1)$. W szczególności $S_0(C_1)$ oznacza stan pamięci C_1 przed rozpoczęciem liczenia.

Warunkiem koniecznym na to, aby μ_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* $1 \leqslant i \leqslant k$.

Własność ta wynika bezpośrednio z definicji procesu prostego z porządkiem poprzecznym. A więc w pamięci liniowej wyniki każdej operacji są umieszczane w kolejnych miejscach $c_i, c_{i+1}, \ldots, c_{i+n}$ $(i+n \leqslant k)$ pamięci C_1, a argumenty są odczytywane kolejno, poczynając od miejsca c_i (2).

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

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

(2) Numeracja komórek pamięci jest tutaj nieistotna i służy tylko do opisu działania pamięci. Aby uniknąć numeracji komórek, moglibyśmy działanie liniowej pamięci C_1 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 zapisywana ostatnio. Podobnie przy odczycie: do odczytania nowego argumentu z pamięci C konieczne jest zaznaczenie komórki, która była odczytywana ostatnio.

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(1).

Gdybyśmy przyjęli, że po wykorzystaniu każdy wynik częściowy jest z pamięci wymazywany, to można wykazać, że w pamięci liniowej znajduje się jednocześnie co najwyżej $[(n+1)/2]$(2) wyników częściowych(3).

Obecnie rozpatrzymy 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 pojedynczą $\rightarrow$ nad symbolem. Wprowadzimy jeszcze dodatkowe oznaczenie $\left[\vec{C_2}\right]$ oraz $\left[\overleftarrow{C_2}\right]$, które oznaczają odpowiednio:

$\left[\vec{C_2}\right]$ — przygotuj do zapisu komórkę c_{j+1} oraz zapisz w komórce c_{j+1},

$\left[\overleftarrow{C_2}\right]$ — przygotuj do zapisu komórkę c_{j-1} oraz zapisz w komórce c_{j-1}.

Pozostałe oznaczenia jak poprzednio.

(1) Ilość komórek pamięci będziemy nazywali jej *pojemnością*.

(2) $[x]$ — oznacza część całkowitą z x.

(3) Zakładając, że wykorzystane wyniki częściowe są wymazywane oraz że po zapisaniu ostatniej komórki ponownie zapisujemy pierwszą komórkę; do obliczenia procesu zawierającego n operacji potrzebna jest pamięć liniowa o pojemności $[(n+1)/2]$.

Pamięć C_2 nazwiemy *rewersyjną*, jeżeli pamięć C_2 realizuje operacje $(\overleftarrow{C_2})$ i $\left[\overrightarrow{C_2}\right]$ (albo $(\overrightarrow{C_2})$ i $\left[\overleftarrow{C_2}\right]$)(1).

Można wykazać, że:

Warunkiem koniecznym, aby μ_1-maszyna realizowała proces w porządku wzdłużnym (tj. W lub $\overline{W}$) jest, aby pamięć wyników częściowych była pamięcią rewersyjną.

Pamięć rewersyjna działa więc następująco:

1. Pierwszy wynik częściowy jest wpisywany do komórki c_1.
2. Każdy następny wynik częściowy jest wpisywany 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 zależy od języka, w którym maszyna pracuje, a tylko od porządku, w jakim jest proces realizowany.

Czy odczytany argument z pamięci wyników częściowych jest lewym, czy prawym argumentem, mówi o tym formuła realizowanego procesu. Ogólnie, dla procesów z porządkiem normalnym odczytywany jest najpierw lewy argument, dla procesów z porządkiem dualnym odwrotnie, tj. odczytywany jest najpierw prawy argument.

§ 4. REALIZACJA UPROSZCZONEGO JĘZYKA PODSTAWOWEGO Z PORZĄDKIEM POPRZECZNYM (MASZYNA $\mu_1(J_1(P))$).

μ_1-maszynę realizującą uproszczony język podstawowy z porządkiem P oznaczamy przez $\mu_1(J_1(P))$. Rozpatrzymy obecnie szczegółowo działanie maszyny $\mu_1(J_1(P))$. Dla porządku $\overline{P}$ działanie jest analogiczne. Wszystkie pamięci maszyny $\mu_1(J_1(P))$ są pamięciami liniowymi. W języku podstawowym rozkazy są umieszczane w programie, w kolejności ich wykonywania.

(1) Operacje mogą być wykonywane kolejno. Jednoczesne wykonanie operacji jest niedozwolone.

(2) Należy zwrócić uwagę, że chodzi tu o ostatnią w czasie, a nie w przestrzeni zapisaną komórkę.

Odczytywanie programu przez urządzenie sterujące polega na analizowaniu kolejnych trójek symboli w pamięci P. Zależnie od odczytanego rozkazu następuje pobranie argumentów z pamięci D albo C_1. W pamięci D magazynowane są wszystkie dane w takim samym porządku, w jakim występują one w $J_1(P)$ programie.

Wykonanie rozkazu można opisać następującą tabelką:

1	p	1	2	3
0	0	$(\overrightarrow{C_1})$	$(\overrightarrow{C_1})$	$\left[\overrightarrow{C_1}\right]$
0	1	$(\overrightarrow{C_1})$	$(\overrightarrow{D})$	$\left[\overrightarrow{C_1}\right]$
1	0	$(\overrightarrow{D})$	$(\overrightarrow{C_1})$	$\left[\overrightarrow{C_1}\right]$
1	1	$(\overrightarrow{D})$	$(\overrightarrow{D})$	$\left[\overrightarrow{C_1}\right]$

W kolumnach 1 i p podano symbole lewego i prawego argumentu w odczytanym rozkazie. Liczby 1, 2, 3 oznaczają kolejne kroki w cyklu pracy maszyny.

Realizacja procesu przedstawionego na rysunku 2 w maszynie $\mu_1(J_1(P))$ przebiega następująco:

Rozkaz	Pamięć D								Pamięć C_1						
1 1+	$\overline{3}$	$\overline{4}$	5′	1	9	8	5	9	7′						
1 1+	3	4	$\overline{5}$	$\overline{1}$	9′	8	5	9	7′	6					
1 0−	3	4	5	1	$\overline{9}$	8′	5	9	$\overline{7}$	6′	2				
1 1−	3	4	5	1	9	$\overline{8}$	$\overline{5}$	9′	7	6′	2	3			
1 0−	3	4	5	1	9	8	5	$\overline{9}$	7	$\overline{6}$	2′	3	3		
0 0	3	4	5	1	9	8	5	9	7	6	$\overline{2}$	$\overline{3}$	3′	6	
0 0+	3	4	5	1	9	8	5	9	7	6	2	3	$\overline{3}$	$\overline{6}$	9

Primy przy liczbach oznaczają, które komórki pamięci są przygotowane do zapisania bądź odczytania w każdym rozkazie. Dla ułatwienia, argumenty wykonywanych działań zaznaczono kreskami u góry.

§ 5. REALIZACJA UPROSZCZONEGO JĘZYKA PODSTAWOWEGO Z PORZĄDKIEM WZDŁUŻNYM (MASZYNA $\mu_1(J_1(W))$)

W maszynie $\mu_1(J_1(W))$ pamięci są następujące:

D — pamięć liniowa,
P — pamięć liniowa,
C_2 — pamięć rewersyjna.

Tablica cyklu pracy ma postać:

1	p	1	2	3
0	0	$\overleftarrow{(C_2)}$	$\overleftarrow{(C_2)}$	$\left[\overrightarrow{C_2}\right]$
0	1	$\overleftarrow{(C_2)}$	$\overrightarrow{(D)}$	$\left[\overrightarrow{C_2}\right]$
1	0	$\overrightarrow{(D)}$	$\overleftarrow{(C_2)}$	$\left[\overrightarrow{C_2}\right]$
1	1	$\overrightarrow{(D)}$	$\overrightarrow{(D)}$	$\left[\overrightarrow{C_2}\right]$

Przebieg procesu obliczenia ma postać następującą:

Rozkaz	Pamięć D								Pamięć C_2 przed wykonaniem działania		Pamięć C_2 po wykonaniu działania	
11−	$\bar{8}$	$\bar{5}$	3′	4	9	5	1	9			3′	
11+	8	5	$\bar{3}$	$\bar{4}$	9′	5	1	9	3		3	7′
10−	8	5	3	4	$\bar{9}$	5′	1	9	3	$\bar{7}$	3	2′
00·	8	5	3	4	9	5′	1	9	$\bar{3}$	$\bar{2}$	6′	
11+	8	5	3	4	9	$\bar{5}$	$\bar{1}$	9′	6		6	6′
10−	8	5	3	4	9	5′	1	$\bar{9}$	6	$\bar{6}$	6	3′
00+	8	5	3	4	9	5	1	9	$\bar{6}$	$\bar{3}$	9′	
0	8	5	3	4	9	5	1	9	9			

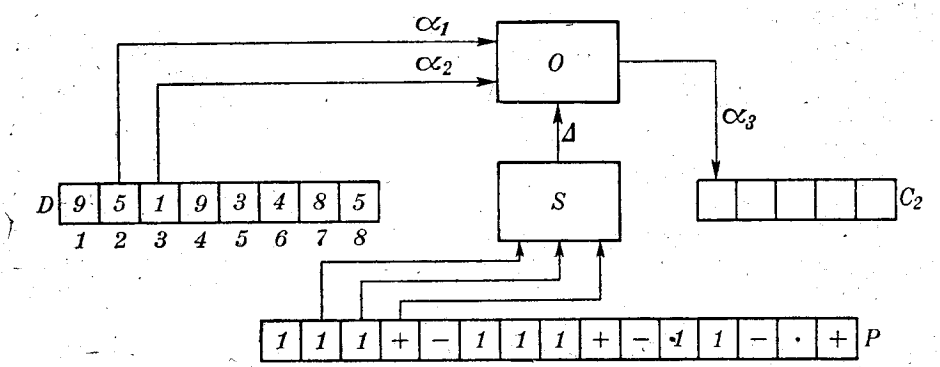
Na podstawie definicji pamięci rewersyjnej oraz tabelki cyklu pracy, prześledzenie przebiegu procesu nie przedstawia trudności. Oznaczenia jak w maszynie poprzedniej. W rozkazach 00· oraz 00+ ostatnia liczba w pamięci C_2 jest lewym argumentem, a przedostatnia — prawym.

§ 6. REALIZACJA 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 uproszczonym 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 te, które rozważaliśmy do tej pory. Dla przykładu rozpatrzymy maszynę $\mu_1(J_3(\overline{W}))$.

Na rysunku 23 jest przedstawiona maszyna $\mu_1(J_3(\overline{W}))$ w stanie początkowym. W pamięci P jest program Łukasiewicza procesu przedstawionego na rysunku 2. Dla prostoty wejścia α_1, α_2 operatora O są narysowane bezpośrednio do pamięci D, a nie za pośrednictwem sterowania S, jak to



Rys. 23

ma miejsce w rzeczywistości (porównaj schemat na rysunku 22). Symbole danych są kolejno ponumerowane liczbami 1, 2, ..., 8, zwanymi w dalszym ciągu *adresami danych*. Pamięć danych D składa się z komórek $d_1, d_2, \ldots, d_8$. W komórce d_i znajduje się argument oznaczony w programie adresem i(1).

Odszukanie kolejnych rozkazów jest oczywiste.

Wykonanie rozkazu opisuje tabelka:

1	p	1	2	3
0	0	$\overleftarrow{(C_2)}$	$\overleftarrow{(C_2)}$	$\left[\overrightarrow{C_2}\right]$
0	1_i	$\overleftarrow{(C_2)}$	(d_i)	$\left[\overrightarrow{C_2}\right]$
1_i	0	(d_i)	$\overleftarrow{(C_2)}$	$\left[\overrightarrow{C_2}\right]$
1_i	1_j	(d_i)	d_j	$\left[\overrightarrow{C_2}\right]$

Zero w tabelce oznacza, że lewy bądź prawy argument jest wynikiem częściowym. Symbole danych napisano z adresami u dołu, (1_i) oznacza odczytanie i-tej komórki pamięci D.

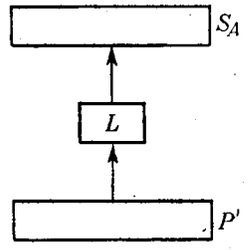
Realizacja procesu będzie przebiegać następująco:

Rozkaz	Pamięć D								Pamięć C_2 przed wykonaniem działania			Pamięć C_2 po wykonaniu działania		
	1	2	3	4	5	6	7	8						
$1_2\,1_3+$	9	$\overline{5}$	$\overline{1}$	9	3	4	8	5				6		
1_1+-	$\overline{9}$	5	1	9	3	4	8	5	$\overline{6}$			3		
$1_5\,1_6+$	9	5	1	9	$\overline{3}$	$\overline{4}$	8	5	3			3	7	
1_4+-	9	5	1	$\overline{9}$	3	4	8	5	3	$\overline{7}$		3	2	
$1_7\,1_8-$	9	5	1	9	3	4	$\overline{8}$	$\overline{5}$	3	2		3	2	3
$--\cdot$	9	5	1	9	3	4	8	5	3	$\overline{2}$	$\overline{3}$	3	6	
$-\cdot+$	9	5	1	9	3	4	8	5	$\overline{3}$	$\overline{6}$		9		

(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.

§ 7. OBLICZANIE ADRESÓW DANYCH W UPROSZCZONYM JĘZYKU ŁUKASIEWICZA ZA POMOCĄ PAMIĘCI REWERSYJNEJ

W maszynie opisanej w poprzednim paragrafie odczytywanie rozkazów programu było dość niewygodne. Ponadto nie podano, w jaki sposób obliczane są numery danych. W niniejszym paragrafie podamy prostą metodę, która pozwala na wyszukanie z pamięci D każdej danej potrzebnej do obliczenia. 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 obliczenie adresów danych.



Rys. 24

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

S_A — 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 kolejne komórki pamięci P'. Jeżeli w odczytanej komórce pamięci P' jest symbolem danej, do licznika danych L jest dodawana jedynka i zawartość licznika jest umieszczana w pamięci S_A. Jeżeli w odczytywanej komórce pamięci P' jest symbol działania, to operator jest nastawiony na odczytane działanie oraz pobrane są argumenty według adresów odczytanych ze skorowidza(2).

Po wykonaniu operacji i zapisaniu jej wyniku w pamięci C_2 — w skorowidzu S_A jest zapisywane zero. Jeżeli adresem argumentu jest zero — argument jest odczytywany z pamięci wyników częściowych C_2.

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

(2) To znaczy, że argumenty są pobierane według dwu ostatnich adresów w skorowidzu.

Działanie skorowidza ilustruje następująca tabelka:

Program	Adres danej	Skorowidz S_A przed wykonaniem działania				po wykonaniu działania		
1	1	1						
1	2	1	2					
1	3	1	2	3				
+		1	$\overline{2}$	$\overline{3}$		1	0	
−		$\overline{1}$	$\overline{0}$			0		
1	4	0	4					
1	5	0	4	5				
1	6	0	4	5	6			
+		0	4	$\overline{5}$	$\overline{6}$	0	4	0
−		0	$\overline{4}$	$\overline{0}$		0	0	
1	7	0	0	7				
1	8	0	0	7	8			
−		0	0	$\overline{7}$	$\overline{8}$	0	0	0
·		0	$\overline{0}$	$\overline{0}$		0	0	
+		$\overline{0}$	$\overline{0}$			0		

W każdym wierszu tabelki podano stan skorowidza. Jeżeli symbolem odczytanym z programu jest symbol działania — podano stany skorowidza przed i po wykonaniu działania(1).

Adresy z-kreskami u góry są adresami argumentów każdego działania. Jeżeli adres jest równy zeru, argument jest odczytywany z pamięci C_2, według algorytmu pracy pamięci rewersyjnej. Dla łatwiejszego odczytania w tablicy podano również adresy wszystkich danych oraz strzałkami zaznaczono lewe argumenty każdego działania — o ile są one danymi. W maszynie ze skorowidzem adresów czytanie danych jest znacznie prostsze niż w maszynie, w której adresy są obliczane na innej zasadzie.

(1) Zwracamy uwagę, że komórka, w której zapisano zero, nie jest równoważna komórce pustej.

§ 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 rysunku 20; zamiast danych są tutaj zapisywane w pamięci symbole danych.

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

§ 9. REALIZACJA UPROSZCZONEGO JĘZYKA NAWIASOWEGO

Zależnie od przyjętego porządku obliczenia pamięć wyników częściowych może być pamięcią liniową bądź rewersyjną. Porządek działań oraz symbole argumentów każdego działania można obliczyć na podstawie wzorów podanych w rozdziale II. Bliżej zajmować się tym 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 realizowania obliczenia w kolejności wzdłużnej łatwo jest otrzymać kolejne rozkazy na zasadzie redukcji formuły.

Jeżeli w uproszczonym programie nawiasowym procesu przedstawionego na rysunku 2, symbole danych kolejno ponumerujemy, to otrzymamy

$$((-(+))+((-(+))\cdot(-)))$$
adresy danych 1 2 3 4 5 6 7 8

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
1 2 3
działania — jest wynik częściowy oraz że lewym argumentem działania + jest dana.

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

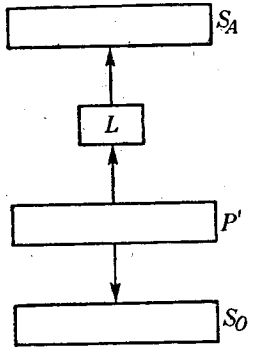
postać

$$\begin{array}{c}(-)\\7\ \ 8\\(+)\\5\ \ 6\\(-(\\4\ \ 5\\)\cdot(\\7\\(+)\\2\ \ 3\\(-(\\1\ \ 2\\)+(\end{array}$$

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, dochodzą tylko dodatkowe urządzenia, służące do obliczenia adresów danych, jak to pokazano na rysunku 25. Rola pamięci P', S_A oraz licznika L jest identyczna jak w maszynie, realizującej język Łukasiewicza, natomiast pamięć S_O jest również pamięcią rewersyjną i służy do magazynowania symboli operacji. Pamięć S_A nazwiemy *skorowidzem adresów*, natomiast pamięć S_O — *skorowidzem operacji*.



Rys. 25

Dla łatwiejszego opisu działania maszyny zmodyfikujemy uproszczony język nawiasowy 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 odczytanego symbolu wykonuje następujące czynności:

	Odczytany symbol	Czynność
1.	(—	1. Odczytanie następnego symbolu.
2.	1 —	1. Dodanie do licznika L jedynki i zapisanie liczby, znajdującej się w L, w pamięci S_A. 2. Odczytanie pamięci P'.
3.	Symbol działania	1. Zapisanie odczytanego symbolu działania w pamięci S_O. 2. Odczytanie pamięci P'.
4.	)	1. Odczytanie symbolu działania z pamięci S_O. 2. Nastawienie operatora na odczytane działanie. 3. Odczytanie dwu ostatnich adresów z pamięci S_A i wpisanie na miejsce przedostatniego adresu 0(1). 4. Pobranie argumentów operacji według odczytanych adresów. 5. Wykonanie operacji. 6. Zapisanie wyniku operacji w pamięci C_2. 7. Odczytanie pamięci P'.

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

Stan skorowidzów (str. 76 — tabelka) jest podany po wykonaniu odpowiednich działań. Pamięć wyników częściowych jest pamięcią rewersyjną C_2. Dane są w pamięci D, w komórkach o odpowiadających im adresach. Kreski nad adresami mają znaczenie identyczne jak poprzednio(2).

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

(2) Zauważmy, że nawiasy lewostronne nie odgrywają tu żadnej roli. Mogą więc w programie nie występować. Języki nawiasowe, zawierające tylko nawiasy lewostronne, bądź prawostronne, zostały wprowadzone przez prof. Kalmára.

Działanie maszyny ilustruje tabelka:

Program	Adres danej	Skorowidz adresów S_A przed wykonaniem operacji				Skorowidz operacji S_O		
(								
(								
1	1	1						
−		1				−		
(		1				−		
1	2	1	2			−		
+		1	2			−	+	
1	3	1	$\overline{2}$	$\overline{3}$		−	+	
)		$\overline{1}$	$\overline{0}$			−		
)		0						
+		0				+		
(		0				+		
(		0				+		
1	4	0	4			+		
−		0	4			+	−	
(		0	4			+	−	
1	5	0	4	5		+	−	
+		0	4	5		+	−	+
1	6	0	4	$\overline{5}$	$\overline{6}$	+	−	+
)		0	$\overline{4}$	$\overline{0}$		+		
)		0	0			+		
·		0	0			+	·	
(		0	0			+	·	
1	7	0	0	7		+	·	
−		0	0	7		+	·	−
1	8	0	0	$\overline{7}$	$\overline{8}$	+	·	−
)		0	$\overline{0}$	$\overline{0}$		+	·	
)		$\overline{0}$	$\overline{0}$			+		
)		0						

Wrócimy 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 np. +).

Teraz już nietrudno podać zasadę działania skorowidza adresów i skorowidza operacji. Zainteresowany Czytelnik rozwiąże to zadanie z łatwością samodzielnie.

§ 11. UWAGI OGÓLNE O μ_1-MASZYNACH

Najciekawszym uproszczonym językiem — z technicznego punktu widzenia — jest język podstawowy (J_1). Analiza programu, pobieranie danych jest tu bardzo proste i polega na badaniu kolejnych 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 aktualnie potrzebne argumenty. Wyszukiwanie rozkazów oraz obliczanie numerów danych jest szczególnie niewygodne w języku nawiasowym. Język Łukasiewicza jest pod tym względem nieco wygodniejszy. Realizacja języka nawiasowego ze skorowidzem adresów jest tutaj dość wygodna technicznie. Najprostszym jednak do realizacji technicznej jest niewątpliwie język podstawowy z porządkiem W lub $\overline{W}$. 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 μ_1-maszyny realizującej język beznawiasowy.

ROZDZIAŁ VI

REALIZACJA JĘZYKÓW UPROSZCZONYCH ZA POMOCĄ MASZYNY Z PAMIĘCIĄ ROBOCZĄ

W rozdziale tym podamy drugi sposób realizowania języków uproszczonych. Jest on szczególnie wygodny dla języka Łukasiewicza oraz języka nawiasowego. Nasze rozważania ograniczymy więc tylko do tych dwu języków oraz do porządków wzdłużnych. Maszyny należące do omawianej klasy oznaczymy przez μ_2 wraz z podaniem języka realizowanego przez maszynę. Np. $\mu_2(J_3(W))$ oznacza maszynę z pamięcią roboczą, realizującą język Łukasiewicza w porządku W.

§ 1. OGÓLNY SCHEMAT MASZYNY

Ogólny schemat jest identyczny ze schematem maszyny z pamięcią wyników częściowych (rys. 22), z tą jedynie różnicą, że działanie maszyny jest obecnie inne.

μ_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 podobną rolę. Pamięć robocza zawiera — w trakcie liczenia — zarówno dane, jak i wyniki częściowe, natomiast pamięć wyników częściowych mogła zawierać tylko liczby otrzymywane w czasie wykonywania operacji.

W rozdziale tym będzie wygodniej przyjąć nieco inne pojęcie rozkazu aniżeli poprzednio. Mianowicie *rozkazem* będziemy w niniejszym rozdziale nazywali *każdy* symbol programu.

μ_2-maszyna analizuje kolejne rozkazy programu i postępuje zależnie od odczytanego rozkazu. Działanie maszyny opiszemy, podając sposób realizowania wszystkich możliwych rozkazów.

§ 2. REALIZACJA UPROSZCZONEGO JĘZYKA ŁUKASIEWICZA

Dla języka Łukasiewicza działanie maszyny jest następujące:

	Odczytany rozkaz	Czynność μ_2-maszyny
1.	Symbol danej	1. Odczytaj pamięć D i przygotuj do odczytania. Odczytaną liczbę zapisz w pamięci R. 2. Odczytaj następny rozkaz z P'.
2.	Symbol operacji	1. Wykonaj odczytaną operację na dwu ostatnich liczbach z pamięci roboczej R. 2. Wynik operacji zapisz na miejscu przedostatniej liczby w pamięci roboczej R. 3. Odczytaj następny rozkaz z P'.

Tablicę tę można by zapisać krócej następująco:

Rozkaz	Czynność μ_2-maszyny
Symbol danej	$\overrightarrow{(D)} \rightarrow \left[\overrightarrow{R}\right], \overrightarrow{(P')}$
Symbol operacji	$\overleftarrow{(R)}\ \Delta\ \overleftarrow{(R)} \rightarrow \left[\overrightarrow{R}\right], \overrightarrow{(P')}$

Symbol $\rightarrow$ oznacza przesyłanie.

Jak widzimy, zasada działania maszyny jest bardzo prosta.

Sterowanie maszyny analizuje kolejne symbole formuły zapisanej w pamięci programu. W przypadku odczytania symbolu danej następuje przesłanie odpowiedniej danej z pamięci danych D do pamięci roboczej R; w przypadku odczytania symbolu operacji maszyna wykonuje odczytaną operację na dwu ostatnich liczbach znajdujących się w pamięci roboczej R i wynik operacji umieszcza w pamięci R na miejscu przedostatniego argumentu. Oba argumenty wykonywanej operacji są oczywiście po odczytaniu wymazywane.

Działanie maszyny dla rozpatrywanego przykładu obliczenia ma następujący przebieg:

	Pamięć D								Pamięć R						
	9	5	1	9	3	4	8	5	przed				po		
1	×								9						
1		×							9	5					
1			×						9	5	1				
+									9	$\bar{5}$	$\bar{1}$		9	6	
−									$\bar{9}$	$\bar{6}$			3		
1				×					3	9					
1					×				3	9	3				
1						×			3	9	3	4			
+									3	9	$\bar{3}$	$\bar{4}$	3	9	7
−									3	$\bar{9}$	$\bar{7}$		3	2	
1							×		3	2	8				
1								×	3	2	8	5			
−									3	2	$\bar{8}$	$\bar{5}$	3	2	3
·									3	$\bar{2}$	$\bar{3}$		3	6	
+									$\bar{3}$	$\bar{6}$			9		

Krzyżyki w tabelce wskazują, która 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 $\mu_2(J_4(W))$ (lub $\mu_2(J_4(\overline{W}))$) zawiera skorowidz operacji S_O, który działa tak, jak to podano w poprzednim rozdziale. Pozostałe elementy maszyny są identyczne jak w maszynie realizującej uproszczony język Łukasiewicza. Ponieważ w języku nawiasowym operacje nie są zapisane w formule w kolejności ich wykonywania, konieczny jest skorowidz operacji S_O, magazynujący symbole operacji w takiej kolejności w jakiej będą one potrzebne w trakcie liczenia.

Znaczenie poszczególnych rozkazów jest następujące:

Rozkaz	Czynność maszyny
(	1. Odczytaj następny rozkaz z P'.
1	1. Odczytaj kolejną 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 symbolicznie otrzymamy:

Rozkaz	Czynności μ_2-maszyny
(	$\vec{(P')}$
1	$\vec{(D)} \to \left[\vec{R}\right]$, $\vec{(P')}$
Symbol operacji	$\Delta \to \left[\vec{S_O}\right]$
)	$\overleftarrow{(S_O)}$, $\overleftarrow{(R)}$, $\overleftarrow{(R)} \to \left[\vec{R}\right]$, $\vec{(P')}$

Maszyna ta działa więc bardzo podobnie do maszyny realizującej język Łukasiewicza. Różnica między obu maszynami polega na tym, że realizacja języka nawiasowego wymaga dodatkowego zmieniania kolejności symboli operacji za pomocą skorowidza S_O. O kolejności wykonywania operacji (dla porządku W) decyduje rozmieszczenie prawostronnych nawiasów w formule.

Działanie μ_2-maszyny, realizującej uproszczony język nawiasowy — dla dyskutowanego przykładu — przebiega następująco:

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

Widzimy, że tutaj również są zbędne lewostronne 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, jednak podane rozwiązanie wydaje się do zastosowań praktycznych wygodniejsze.

§ 4. UWAGI OGÓLNE O MASZYNACH Z PAMIĘCIĄ ROBOCZĄ

Opisana grupa maszyn ma wiele 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 właściwości znacznie upraszczają konstrukcję maszyny, kosztem oczywiście czasu działania. Ta sama dana jest bowiem najpierw odczytywana z pamięci D, następnie zapisywana w pamięci R i przechowywana w pamięci R tak długo, aż będzie potrzebna do obliczenia. Następnie znów jest ona odczytywana i przesyłana do operatora. 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 zmodyfikować tak, aby liczba odczytów i zapisów została zredukowana do niezbędnego minimum. Nie będziemy tego bliżej dyskutowali, gdyż sprawa ta wiąże się z działaniem arytmometru, którym — jak już wspominaliśmy — nie będziemy się bliżej zajmowali.

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

ROZDZIAŁ VII

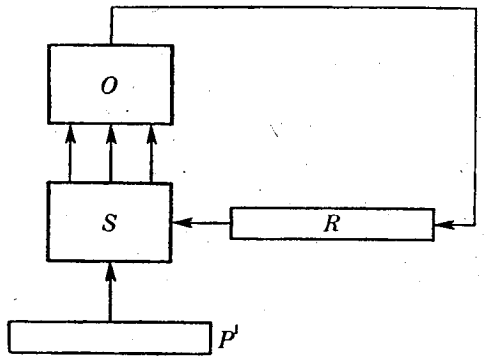
REALIZACJA JĘZYKÓW UPROSZCZONYCH ZA POMOCĄ MASZYNY Z ADRESOWĄ PAMIĘCIĄ ROBOCZĄ

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 ma ona takich zalet jak w przypadku języków: Łukasiewicza i nawiasowego.

§ 1. OGÓLNY SCHEMAT MASZYNY

Uproszczony schemat maszyny jest przedstawiony na rysunku 26. Maszyna składa się z:

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



Rys. 26

Działanie operatora O oraz pamięci P' jest identyczne z rozważaniami poprzednimi. 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. REALIZACJA JĘZYKA ŁUKASIEWICZA

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. Na przykład 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	15

W stanie początkowym dane są umieszczone pod odpowiadającymi im adresami w pamięci R. Obliczanie adresów argumentów oraz wyników częściowych najwygodniej zrealizować za pomocą skorowidza adresów. Działanie skorowidza jest identyczne z działaniem skorowidza przedstawionym na rysunku 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 odczytanych ze skorowidza. 3. Wykonaj odczytaną operację. 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 tak wymazywane, 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 i to 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 pokazany na następującym przykładzie:

Program ((1 − (1 + 1)) + ((1 − (1 + 1)) · (1 − 1)))
Adresy 1 2 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 ma 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 adresów L zapisz w skorowidzu adresów S_A. 2. Dodaj jeden do licznika adresów L. 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 do tablicy dla języka Łukasiewicza, dlatego nie będziemy jej powtarzali.

ROZDZIAŁ VIII

REALIZACJA JĘZYKÓW SYMBOLICZNYCH

W językach symbolicznych różne dane są oznaczone różnymi symbolami. Ponieważ symbole danych można ponumerować, więc zamiast o symbolach możemy mówić o ich numerach. Numery te będziemy nazywali *adresami*. Dla człowieka wygodniejsze jest posługiwanie się symbolami niż adresami, natomiast z punktu widzenia konstrukcji maszyny wygodniejsze jest używanie adresów [21].

W dalszym ciągu — w celu uproszczenia terminologii — wprowadzimy pojęcia 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 $(\mathbf{5}+\mathbf{10})$. 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 — jednak należy pamiętać, że w maszynie faktycznie na miejscu adresów symbolicznych występują w programie adresy liczbowe.

Pojęcie adresu ma tutaj nieco inny sens niż w poprzednim rozdziale. Obecnie adres to symbol danej. Poprzednio adresem był numer symbolu danej — nie występujący w programie, a obliczany przez maszynę, przy czym dane były numerowane kolejno. Obecnie sposób numeracji danych jest całkowicie dowolny. Istotna różnica między oboma 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, tak jednak, żeby różne dane miały różne numery. Taka koncepcja adresu narzuca od razu określone rozwiązanie maszyny.

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

Ogólny schemat maszyny, realizującej dowolny język symboliczny jest taki sam, jak na rysunku 22. Działanie σ-maszyny różni się od działania μ_1-maszyny jedynie sposobem odczytywania danych z pamięci D. Pozostałe elementy działają bez zmiany, tj. pamięć wyników częściowych — zależnie od przyjętego porządku — jest pamięcią liniową lub rewersyjną, a analiza σ-programu przez urządzenie sterujące może być przeprowadzona na podstawie jednej z podanych poprzednio zasad.

W pamięci danych D wszystkie komórki są oznaczone dowolnymi symbolami danych; np. kolejnymi małymi literami alfabetu łacińskiego(1).

Każda komórka jest oznaczona inną literą, przy czym porządek oznaczenia nie gra tutaj roli. Jeżeli argumenty działania mają adresy, np. x i y, to argumenty tego działania znajdują się w pamięci D, w komórkach oznaczonych przez x i y. Wyniki częściowe są zapisywane i odczytywane identycznie jak w μ-maszynie.

§ 1. POBIERANIE DANYCH

Działanie σ-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 umieszczane w skorowidzu bezpośrednio z programu. Interpretowanie zmiennych bezpośrednio jako adresów jest bardzo wygodne przede wszystkim z następującego powodu: jeżeli w formule jakaś zmienna występuje wielokrotnie, wartość jej jest zapisana w postaci D tylko jednokrotnie, w przeciwieństwie do maszyn omawianych w poprzednim rozdziale — w których każda wartość zmiennej musiała być magazynowana w pamięci danych tyle razy ile razy dana zmienna występowała w formule. Ilustruje to następująca tabelka:

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

Program	Skorowidz adresów			
a	a			
b	a	b		
c	a	$\overline{b}$	$\overline{c}$	
$+$	$\overline{a}$	$\overline{0}$		
$-$	0			
d	0	d		
e	0	d	e	
f	0	d	$\overline{e}$	$\overline{f}$
$+$	0	$\overline{d}$	$\overline{0}$	
$-$	0	0		
g	0	0	g	
h	0	0	$\overline{g}$	$\overline{h}$
$-$	0	$\overline{0}$	$\overline{0}$	
$\cdot$	$\overline{0}$	$\overline{0}$		
$+$	0			

Podobnie jak poprzednio po wykonaniu działania oba adresy argumentów są ze skorowidza usuwane, a na miejsce przedostatniego argumentu wypisywane jest 0, które tutaj nie jest adresem, lecz wskazuje, że odpowiedni wynik znajduje się w pamięci wyników częściowych. Dla przejrzystości adresy argumentów każdej operacji są oznaczane kreskami u góry.

Podobnie działa maszyna dla symbolicznego języka nawiasowego. Skorowidz adresów i operacji zapewniają najprostszą realizację techniczną. Ogólny schemat maszyny nie ulega w zasadzie zmianie. Skorowidz adresów służy do wyszukiwania argumentów każdej operacji, a skorowidz operacji porządkuje symbole operacji zgodnie z kolejnością wymaganą w czasie liczenia. Przypominamy, że o kolejności wykonywania operacji decyduje rozmieszczenie prawostronnych nawiasów w formule, natomiast nawiasy lewostronne — przy takiej organizacji maszyny — są zbędne.

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	$\overline{b}$	$\overline{c}$		$-$	$+$	
)	$\overline{a}$	$\overline{0}$			$-$		
)	0						
$+$	0				$+$		
(	0				$+$		
(	0				$+$		
d	0	d			$+$		
$-$	0	d			$+$	$-$	
(	0	d			$+$	$-$	
e	0	d	e		$+$	$-$	
$+$	0	d	e		$+$	$-$	$+$
f	0	d	$\overline{e}$	$\overline{f}$	$+$	$-$	$+$
)	0	$\overline{d}$	$\overline{0}$		$+$	$-$	
)	0	0			$+$		
$\cdot$	0	0			$+$	$\cdot$	
(	0	0			$+$	$\cdot$	
g	0	0	g		$+$	$\cdot$	
$-$	0	0	g		$+$	$\cdot$	$-$
h	0	0	$\overline{g}$	$\overline{h}$	$+$	$\cdot$	$-$
)	0	$\overline{0}$	$\overline{0}$		$+$	$\cdot$	
)	$\overline{0}$	$\overline{0}$			$+$		
)	0						

Oznaczenia w tablicy są identyczne z poprzednimi.

Zrozumienie działania σ-maszyny, realizującej język nawiasowy nie przedstawia trudności, należy tylko pamiętać, że litery w programie oznaczają adresy w pamięci danych D.

§ 2. ADRESY I ZMIENNE

Działanie σ-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 wpisywane na miejsce odpowiednich symboli, a następnie wykonywany jest rachunek według tak otrzymanego π-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 odzwierciedlać fakt, że na miejsce odpowiedniej litery można wpisać dowolną liczbę. Natomiast w języku maszynowym możliwa jest również inna interpretacja liter występujących w formule, a mianowicie — jako *adresów*. Interpretacja ta zależy więc 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, 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 żadnych danych na miejsce adresu.

Gdybyśmy w π-maszynie zamiast π-programu wpisali najpierw program symboliczny, a następnie na miejsce symboli danych napisali odpowiadające im dane, to zmienne grałyby rolę nazw komórek pamięci, w których zostały te zmienne zapisane.

§ 3. REALIZACJA RÓŻNYCH PROCESÓW

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).

Jeżeli $\Phi_1, \Phi_2, \ldots, \Phi_n$ są odpowiednimi programami procesów $\mathfrak{A}_1$, $\mathfrak{A}_2, \ldots, \mathfrak{A}_n$, to ciąg $\Phi_1, \Phi_2, \ldots, \Phi_n$ nazwiemy *ω-programem*.

Ogólny schemat maszyny realizującej ω-procesy w zasadzie nie różni się od schematów maszyn podawanych poprzednio. W stanie początkowym, w pamięci wejściowej D znajduje 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 z pamięci P program, dane, czy też 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.

---

(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.

Maszyna może również posiadać oddzielne pamięci wejściowe dla danych, parametrów oraz programów. Wtedy po zrealizowaniu jednego programu, następny program jest odczytywany z wejściowej pamięci programów.

ROZDZIAŁ IX

PROCESY WIELOKROTNE WIELOKROTNA REALIZACJA PROGRAMÓW

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 [18], [24], [25].

Proces prosty określany w rozdziale I nazwiemy *procesem jednokrotnym* lub krótko *ρ-procesem*, a proces, polegający na wielokrotnym realizowaniu programu procesu prostego dla różnych danych nazwiemy *procesem wielokrotnym* lub krótko *λ-procesem*.

W niniejszym rozdziale podamy ogólne zasady realizowania procesów wielokrotnych. Zanim przejdziemy do opisu realizacji procesów wielokrotnych rozszerzymy pojęcie języka tak, aby za jego pomocą można było również opisywać procesy wielokrotne.

§ 1. JĘZYKI PROCESÓW WIELOKROTNYCH

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

$$\Phi\lambda_n x_n,$$

gdzie $n \geqslant 1$ nazwiemy *programem wielokrotnym* lub *λ-programem*. W szczególnym przypadku $\Phi\lambda_1 x_1$ oznacza jednokrotne wykonanie programu Φ. Program Φ może być programem uproszczonym lub symbolicznym. Dla

π-programów podana definicja nie obowiązuje. Np. wyrażenie

$$((a+b)\cdot c)\,\lambda_2,\ 5,\ 7,\ 8,\ 3,\ 2,\ 1$$

jest λ-programem, oznaczającym, że należy najpierw obliczyć program $((a+b)\cdot c)$ dla danych $a=5, b=7, c=8$, a następnie wykonać ten sam program dla danych $a=3, b=2, c=1$. Oczywiście program ten możemy również zapisać w postaci

$$\begin{aligned}&((a+b)\cdot c)\,\lambda_2,\\ &a=5,\\ &b=7,\\ &c=8,\\ &a=3,\\ &b=2,\\ &c=1.\end{aligned}$$

W szczególności λ-program

$$\begin{aligned}&((a+b)\cdot c)\,\lambda_1,\\ &a=5,\\ &b=7,\\ &c=8\end{aligned}$$

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

Dla uproszczonego języka nawiasowego powyższy λ-program będzie miał postać

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

Podobnie możemy przedstawić λ-programy, jeżeli Φ jest programem podstawowym, beznawiasowym lub Łukasiewicza(1).

Wyrażenie $\Phi\lambda_n$ będziemy nazywali *λ-programem uproszczonym* programu $\Phi\lambda_n x_n$, np. $((a+b)\cdot c)\,\lambda_2$ lub $((+)\cdot)\,\lambda_2$.

§ 2. REALIZACJA PROCESÓW WIELOKROTNYCH

Można podać kilka różnych zasad realizacji procesów wielokrotnych. Tutaj zajmiemy się tylko jedną z nich, a mianowicie przypadkiem, gdy Φ jest ρ-programem uproszczonym(1). Dla języków symbolicznych zasada działania jest podobna.

(1) λ-programy przypominają tzw. *λ-system Churcha* [10].

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. wyrażenie $\Phi\lambda_n$. 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. Po n-krotnym powtórzeniu Φ-programu, λ-maszyna kończy działanie. Liczba powtórzeń programu Φ dana jest w λ-programie. Zamiast liczby 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 adresową 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 *wejściem λ-maszyny*, a pamięć W — *wyjściem λ-maszyny*.

§ 3. PARAMETRY I STAŁE

Przy wielokrotnym wykonaniu λ-programu, niektóre dane nie ulegają zmianie w każdym obliczeniu; dane takie nazywamy *parametrami*. Jeżeli dane posiadają tę samą wartość dla wszystkich obliczeń — mówimy o *stałych*. W dalszym ciągu nie będziemy odróżniać parametrów od stałych; jedne i drugie nazwiemy parametrami.

(1) Należy odróżnić ρ-program uproszczony oraz λ-program uproszczony. Np. $((+)\cdot)$ jest ρ-programem uproszczonym, a $((a+b)\cdot c)\,\lambda_3$ jest λ-programem uproszczonym.

W językach symbolicznych parametry będziemy oznaczali małymi literami z primem, np. a'. W językach tych parametry nie wymagają specjalnego traktowania, gdyż zmiana parametrów obliczenia sprowadza się do wpisania nowych parametrów w pamięci D. Jeżeli parametry nie ulegają zmianie, to 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. Liczba 0 oznacza więc wyniki częściowe, 1 — dane, 2 — parametry. Można wprowadzić oczywiście również inne oznaczenia np. c, d, p — oznaczające odpowiednio wyniki częściowe, dane oraz parametry. μ-maszyna natomiast musi zawierać specjalną pamięć parametrów, z której parametry są w miarę potrzeby pobierane.

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

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

$$\Phi\lambda, p_1, x_1, x_2, \ldots, x_k, p_2, x_{k+1}, x_{k+2}, \ldots, x_t, \ldots, p_n, x_{t+1}, \ldots, x_n,$$

lub krócej:

$$\Phi\lambda_n^m x_t\, p_j,$$

gdzie $p_1, p_2, \ldots, p_m$ są kolejno zbiorami parametrów procesu, a x_1, x_2, $\ldots, x_n$ kolejnymi zbiorami danych. $\Phi\lambda_n^m$ oznacza, że program Φ ma być obliczony dla m — zbiorów parametrów $p_1, p_2, \ldots, p_m$ oraz dla n — zbiorów danych $x_1, x_2, \ldots, x_n$.

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 poprzedzane w pamięci D specjalnymi symbolami sygnalizującymi czy po nich następują dane, czy parametry.

Np.

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

oznacza, że w programie $((a'+b')\cdot c)/f)$ parametrami są a' i b', a danymi c i f, 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 wpisaniu 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 PODPROGRAMÓW

Pisanie i czytanie zbyt długich programów jest niewygodne. W praktyce stosowane są różne metody, służące do czytelniejszego przedstawiania długich wyrażeń matematycznych [24], [25].

Np.

$$a=A+B,$$

gdzie

$$A=ef-g$$

oraz

$$B=xy+z.$$

Podany przykład jest bardzo prosty i jego rozbicie na prostsze części 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 konsekwencjami 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 opracowany jednorazowo i sprawa prostoty układania programu nie ma tutaj specjalnego znaczenia zarówno praktycznego, jak i ekonomicznego(1).

Jeżeli jednak maszyna jest przeznaczona do rozwiązywania bardzo często różnych zadań, a więc często muszą być opracowywane nowe programy, sprawa wygody opracowywania programów ma zasadnicze znaczenie. Kilka takich prostych metod rozważymy w dalszym ciągu, nie wyczerpując oczywiście tematu całkowicie.

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

§ 1. PODPROGRAMY I PROGRAMY ZŁOŻONE

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

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

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

Program podprocesu $\mathfrak{A}'$ będziemy nazywali *podprogramem* procesu $\mathfrak{A}$. Program procesu $\mathfrak{A}$, składający się z podprogramów nazwiemy *programem złożonym* procesu $\mathfrak{A}'$(1).

Dokładniejsze definicje programów złożonych podamy w dalszych paragrafach.

§ 2. OGÓLNY SCHEMAT MASZYNY REALIZUJĄCEJ PROGRAMY ZŁOŻONE

Maszyna realizująca programy złożone posiada dodatkową pamięć wyników pośrednich K, w której są magazynowane wyniki końcowe każdego podprogramu. Program złożony znajduje się w pamięci wejściowej. Kolejne podprogramy są przepisywane z pamięci wejściowej do pamięci programów. Najwygodniej jest, gdy dane i parametry każdego programu są umieszczone w pamięci wejściowej, bezpośrednio po odpowiadającym im programie.

Wygodny jest również schemat z oddzielną 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 za-

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

stosowanej pamięci wyników pośrednich K, mogą być stosowane różne sposoby składania podprogramów.

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

§ 3. REALIZACJA PROGRAMÓW ZŁOŻONYCH ZA POMOCĄ MASZYNY Z LINIOWĄ PAMIĘCIĄ WYNIKÓW POŚREDNICH

Niech $\mathfrak{A}'$ będzie podprocesem procesu $\mathfrak{A}$. Jeżeli istnieje taka operacja w podprocesie $\mathfrak{A}'$, że argument α tej operacji jest wynikiem częściowym operacji Ω, nie należącej do $\mathfrak{A}'$, a należącej do $\mathfrak{A}$, to α nazwiemy *wynikiem pośrednim* oraz oznaczymy w podprogramie symbolem $\#$. Dane i wyniki częściowe będziemy oznaczać jak poprzednio przez 1, 0.

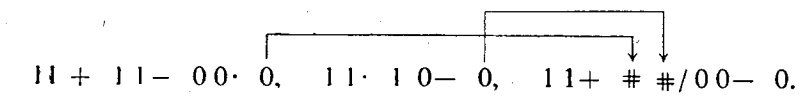
Niech $\mathfrak{A}_1, \mathfrak{A}_2, \ldots, \mathfrak{A}_n$ będą podprocesami procesu $\mathfrak{A}$, niech $\Phi_1, \Phi_2, \ldots, \Phi_n$ będą podprogramami odpowiednich podprocesów.

Jeżeli wynik końcowy podprogramu Φ_i jest wynikiem pośrednim w podprogramie Φ_j, to napiszemy $\Phi_i \prec \Phi_j (j>i)$ oraz $\mathfrak{A}_i \prec \mathfrak{A}_j$, gdzie $\mathfrak{A}_i$ i $\mathfrak{A}_j$ są podprocesami odpowiadającymi podprogarmom Φ_i, Φ_j. Ciąg $\Phi_1, \Phi_2, \ldots, \Phi_n$ nazwiemy *programem złożonym* procesu $\mathfrak{A}$, jeżeli dla każdego i $(1 \leqslant i < n)$ istnieje dokładnie jedno takie j $(n \geqslant j > i)$, że $\Phi_i \prec \Phi_j$(1).

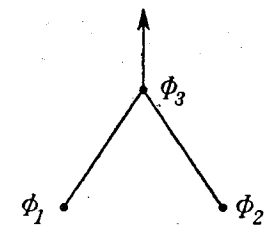
Interpretując podprogramy programu złożonego jako punkty, wyniki pośrednie natomiast jako gałęzie, program złożony możemy również przedstawić w postaci drzewa (danych oraz wyników częściowych na drzewie nie przedstawiamy). Występuje wtedy również problem numeracji punktów drzewa, tzn. kolejności wykonywania podprogramów. Możemy tu oczywiście zastosować identyczne zasady numeracji jak to uczyniliśmy w rozdziale drugim dla obliczeń prostych. W konsekwencji do magazynowania wyników pośrednich możemy również zastosować, zależnie od porządku podprogramów, pamięć liniową bądź rewersyjną. Nie będziemy tej sprawy przedstawiali szczegółowo, gdyż w znacznym stopniu byłoby to powtórzeniem materiału podanego w rozdziale drugim. Podamy natomiast proste przykłady programów złożonych w porządku P i W. Na przykład program złożony w porządku P może mieć postać:

$$\begin{aligned} \Phi_1 &\quad 1\,1+\ 1\ 1-\ 0\,0\cdot\ 0, \\ \Phi_2 &\quad 1\,1\cdot\ 1\ 0-\ 0, \\ \Phi_3 &\quad 1\,1+\ \#\,\#/\ 0\,0-\ 0. \end{aligned}$$

Wynik podprogramu Φ_1 jest lewym argumentem drugiego działania w podprogramie Φ_3, natomiast wynik podprogramu Φ_2 jest prawym argumentem drugiego działania w podprogramie Φ_3. Pisząc podprogramy w jednej linii oraz zaznaczając symbole wyników pośrednich i odpowiadające im podprogramy strzałkami otrzymamy



Zasada umieszczania strzałek oznaczających, gdzie należy umieścić wynik obliczenia każdego podprogramu jest tutaj podobna do zasady podanej dla obliczeń prostych z porządkiem P, tj. wynik obliczenia każdego podprogramu należy wstawić na miejsce najbliższego wolnego symbolu wyniku pośredniego $\#$. Drzewo powyższego programu pokazano na rysunku 27.



Rys. 27

Pisanie podprogramów w porządku P jest bardzo proste. Zaczynamy od ostatniego podprogramu i wypisujemy wszystkie podprogramy, odpowiadające symbolom wyników pośrednich, poczynając od strony prawej. Następnie powtarzamy tę czynność dla przedostatniego programu itd. aż do końca. Ponieważ taki sposób pisania podprogramów jest niewygodny, można by przyjąć zasadę pisania podprogramów od ostatniego do pierwszego, pamiętając, że ich wykonanie następuje w kierunku odwrotnym. Tak więc poprzedni przykład możemy zapisać następująco:

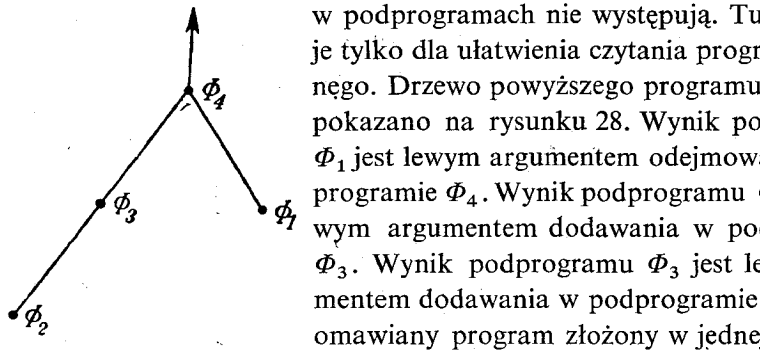
$$\begin{aligned} \Phi_3 &\quad 1\,1+\ \#\,\#/\ 0\,0-\ 0, \\ \Phi_2 &\quad 1\,1\cdot\ 1\ 0-\ 0, \\ \Phi_1 &\quad 1\,1+\ 1\ 1-\ 0\,0\cdot\ 0. \end{aligned}$$

(1) Dla uproszczenia przyjęliśmy, że wynik końcowy każdego podprogramu jest podstawiany tylko do jednego podprogramu. Nietrudno określić program złożony tak, aby wynik każdego podprogramu można było podstawiać do dowolnej ilości podprogramów.

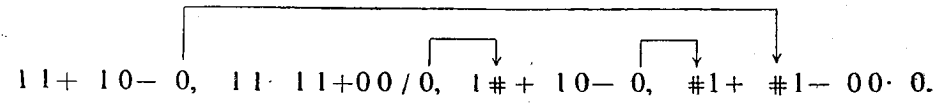
Przykład programu złożonego w porządku W pokazany jest niżej:

$$\begin{array}{llllllll} \Phi_1 & 1 & 1 & + & 1 & 0- & 0, \\ \Phi_2 & 1 & 1 & \cdot & 1 & 1+ & 0\,0/ & 0, \\ \Phi_3 & 1 & \#_2 & + & 1 & 0- & 0, \\ \Phi_4 & \#_3 1 & + & \#_1 1- & 0\,0\cdot & 0. \end{array}$$

Liczbami przy symbolach wyników pośrednich zaznaczono odpowiadające im podprogramy. Oczywiście numery te w podprogramach nie występują. Tutaj podano je tylko dla ułatwienia czytania programu złożonego. Drzewo powyższego programu złożonego pokazano na rysunku 28. Wynik podprogramu Φ_1 jest lewym argumentem odejmowania w podprogramie Φ_4. Wynik podprogramu Φ_2 jest prawym argumentem dodawania w podprogramie Φ_3. Wynik podprogramu Φ_3 jest lewym argumentem dodawania w podprogramie Φ_4. Pisząc omawiany program złożony w jednej linii i zaznaczając podprogramy każdego wyniku pośredniego strzałkami otrzymamy



Rys. 28



Zasada pisania strzałek jest i w tym przypadku podobna do zasady podanej dla programów obliczeń prostych z porządkiem W.

Program złożony w porządku W możemy również pisać poczynając od podprogramu ostatniego i postępując dalej odwrotnie niż dla porządku P, tj. pisząc podprogramy zawsze dla ostatniego symbolu wyniku pośredniego, dla którego podprogram nie został jeszcze napisany. Postępując według tej zasady powyższy program złożony możemy napisać w postaci

$$\begin{array}{llllllll} \Phi_4 & \#_3 1 & + & \#_1 1- & 0\,0\cdot & 0, \\ \Phi_3 & 1 & \#_2 + & 1 & 0- & 0, \\ \Phi_2 & 1 & 1 & \cdot & 1 & 1+ & 0\,0/ & 0, \\ \Phi_1 & 1 & 1 & + & 1 & 0- & 0. \end{array}$$

Dotychczas podane przykłady składania podprogramów dotyczyły tylko języka podstawowego. Jeżeli podprogramy są pisane w języku Łukasiewicza bądź też w języku nawiasowym, zasada pisania podprogramów w porządku poprzecznym lub wzdłużnym może być również zastosowana, jednakże w tym przypadku należy uwzględnić fakt, że zmienne w programie mogą wtedy występować nie w takiej kolejności, w jakiej będą potrzebne do obliczenia. Uwzględnienie tego faktu komplikuje jednak na tyle proponowaną zasadę składania podprogramów, że praktyczne jej zastosowanie dla wymienionych języków jest znacznie ograniczone. Dlatego składania podprogramów w porządku poprzecznym bądź wzdłużnym dla języków Łukasiewicza i nawiasowego nie będziemy omawiali.

§ 4. REALIZACJA PROGRAMÓW ZŁOŻONYCH ZA POMOCĄ MASZYNY Z ADRESOWĄ PAMIĘCIĄ WYNIKÓW POŚREDNICH

Istotą składania podprogramów według zasady podanej w poprzednim paragrafie było dobre uporządkowanie wszystkich podprogramów wchodzących w skład rozpatrywanego programu złożonego. Czasem porządkowanie takie jest kłopotliwe np. wtedy, gdy podprogramy są napisane w języku Łukasiewicza lub nawiasowym. Wygodnie jest wtedy stosować inną zasadę składania podprogramów, którą krótko naszkicujemy. Opisane w tym paragrafie składanie będziemy nazywali *składaniem adresowym*. Zasadę adresowego składania podprogramów przedstawimy najpierw dla języka podstawowego, zakładając, że podprogramy są pisane w języku podstawowym, odpowiednio do tego celu zmodyfikowanym. Przyjmiemy mianowicie, że wynik końcowy obliczenia każdego podprogramu jest oznaczony w podprogramie odpowiednim symbolem, różnym dla różnych podprogramów oraz że, jeżeli w podprogramie Φ_i istnieje działanie, którego argumentem jest wynik końcowy podprogramu Φ_j, to argument ten jest oznaczony symbolem wyniku końcowego podprogramu Φ_j. Np. program złożony adresowo może mieć postać

$$\begin{array}{lll} \Phi_1 & 1\,1+\ 1\,0/ & x, \\ \Phi_2 & 1\,1\cdot\ 1\,1-\ 0\,0\cdot & y, \\ \Phi_3 & 1\,1-\ x\,0\cdot & z, \\ \Phi_4 & z\,1\cdot\ x\,y+\ 0\,0/ & u. \end{array}$$

Wynik podprogramu Φ_1 jest lewym argumentem mnożenia w podprogramie Φ_3 oraz lewym argumentem dodawania w podprogramie Φ_4. Wynik podprogramu Φ_2 jest prawym argumentem dodawania w podprogramie Φ_4, wynik zaś podprogramu Φ_3 jest lewym argumentem mnożenia w podprogramie Φ_4. Przy adresowym składaniu podprogramów każdy podprogram występuje w programie złożonym tylko jednokrotnie, niezależnie od tego, ile razy jest używany w programie złożonym(1).

Dla języka Łukasiewicza oraz języka nawiasowego możemy stosować zapis konwencjonalny, jak np.

$$\begin{array}{lll} \Psi_1 & ((a+b)\cdot c) & =z, \\ \Psi_2 & (b+z) & =u, \\ \Psi_3 & (z+u) & =y. \end{array}$$

Dla realizacji adresowego składania podprogramów maszyna winna posiadać adresową pamięć wyników pośrednich. Wynik obliczenia każdego podprogramu jest wtedy umieszczany w pamięci wyników pośrednich pod podanym w podprogramie adresem i w miarę potrzeby z miejsca tego odczytywany. Np. wynik podprogramu Φ_1 jest zapisany pod adresem x, podprogram Φ_2 pod adresem y, podprogram Φ_3 pod adresem z oraz Φ_4 pod adresem u. W trakcie realizowania podprogramu Φ_3 jako lewy argument mnożenia pobierana jest liczba spod adresu x itd.

Oczywiście adresowe składanie podprogramów jest wygodniejsze od składania bezadresowego, jednak w niektórych przypadkach, np. dla maszyn pracujących według raz na zawsze ustalonego programu, składanie bezadresowe może się okazać korzystniejsze niż składanie adresowe.

(1) Zakładamy tu milcząco, że powtarzające się podprogramy są wykonywane dla tych samych wartości danych oraz parametrów.

ROZDZIAŁ XI

PODPROGRAMY STANDARDOWE

Często wygodnie jest powtarzające się podprogramy nie pisać wielokrotnie w programie złożonym, a zamiast nich pisać ich nazwy (lub numery, jeżeli przyjmiemy, że są one ponumerowane).

Przyjmiemy więc, że każdą maszynę możemy wyposażyć w pewną ilość podprogramów, tzw. *podprogramów standardowych* takich, że w programie złożonym zamiast podprogramów standardowych będziemy używali ich nazw. Przyjmiemy, że duże pisane litery łacińskie oznaczają nazwy podprogramów standardowych. Np. $A(a, b, c, d)$ oznacza podprogram standardowy, natomiast a, b, c, d są symbolami danych. Nazwy podprogramów standardowych grają więc rolę podobną do nazw symboli operacji.

W dalszym ciągu rozpatrzymy dwie metody realizowania podprogramów standardowych przez maszynę. Składanie podprogramów standardowych rozpatrzymy na przykładzie uproszczonego języka podstawowego. Dla innych języków zasada realizowania podprogramów standardowych jest podobna.

§ 1. LINIOWE SKŁADANIE PODPROGRAMÓW

Ciąg

$$\Phi_1, \Phi_2, \ldots, \Phi_n$$

nazwiemy *programem złożonym liniowo*, jeżeli dla każdego i $(1 \leqslant i < n)$, istnieje dokładnie jedno takie j $(i < j \leqslant n)$, że $\Phi_i \prec \Phi_j$, gdzie Φ_i, Φ_j są podprogramami lub nazwami podprogramów(1).

(1) Poprzednio powiedzieliśmy, że podprogramy standardowe, ewentualnie wraz z parametrami są oznaczone dużymi literami łacińskimi. W podanej definicji Φ_i nie są właściwie samymi nazwami podprogramów standardowych, a nazwami wraz z symbolami danych. Aby nie wprowadzać jednak nowych terminów, przez *nazwę* podprogramów standardowych, będziemy rozumieli zarówno pojedynczy symbol, oznaczający ten podprogram, jak i symbol ten wraz z symbolami danych. Ta niejednoznaczność nie powinna prowadzić do nieporozumień, gdyż zawsze wiadomo, którą interpretację mamy na myśli.

Przykład podprogramu złożonego liniowo:

$$\begin{array}{ll} \Phi_1 & 1\ 1+\ 1\ 1\cdot\ 0\ 0/\ 0, \\ \Phi_2 & \#\ 1\ \cdot\ 1\ 0+\ 0, \\ \Phi_3 & F(1,1,1), \\ \Phi_4 & 1\ \#+\ 0\#/\ 0. \end{array}$$

Przyjęliśmy tutaj, że podprogramy są napisane w porządku poprzecznym, co oczywiście nie ma znaczenia, trzy jedynki przy symbolu F oznaczają dane. Podobnie nieistotny jest porządek poszczególnych podprogramów. F jest tutaj nie programem, a nazwą podprogramu standardowego, która może mieć np. postać $1\ 1-\ 1\ 0\cdot\ 0$. Ważne jest tylko to, że w podprogramie F występują trzy dane.

Maszyna realizująca tę zasadę ma dodatkowo adresową pamięć podprogramów standardowych P_s oraz adresowy skorowidz S_s, zawierającą adresy początków podprogramów standardowych w pamięci P_s. Jeżeli sterowanie odczytuje podprogram z pamięci wejściowej, działanie maszyny jest identyczne z opisanym w poprzednim rozdziale. Jeżeli odczytana jest nazwa podprogramu standardowego maszyna działa następująco:

Nazwa podprogramu standardowego jest interpretowana jako adres w skorowidzu S_s, pod którym znajduje się adres początku podprogramu standardowego w pamięci P_s. Następnie z pamięci P_s przesłany jest do pamięci programu znaleziony podprogram standardowy. Jeżeli podprogram standardowy zawiera parametry, to są one przesłane z pamięci P_s do pamięci parametrów. Wybrany podprogram standardowy może być już wykonany identycznie jak zwykłe podprogramy. Maszyna może być również zbudowana w ten sposób, że podprogramy standardowe nie są przepisywane do pamięci programu, lecz odczytywane bezpośrednio z pamięci podprogramów standardowych P_s. Oczywiście danymi w podprogramie standardowym mogą być również wyniki pośrednie. Po wykonaniu podprogramu standardowego maszyna wykonuje kolejny podprogram w programie złożonym.

§ 2. NIELINIOWE SKŁADANIE PODPROGRAMÓW

Jeżeli w dowolnych podprogramach podprogramu złożonego na miejscu symboli danych podstawimy nazwy podprogramów standardowych, to

tak otrzymany program nazwiemy *programem złożonym nieliniowo*. Np.

$$\begin{array}{ll} \Phi_1 & F(1,1,1)G(1)-\ H(1,1)\ 0\cdot\ 0, \\ \Phi_2 & K(1,1)1/\ \#1-\ 0\ 0+\ 0, \\ \Phi_3 & 1\#\cdot\ 0\ 1/\ 0, \end{array}$$

W podprogramie Φ_1 lewym argumentem odejmowania jest wynik podprogramu standardowego $F(1,1,1)$, który ma trzy dane, oznaczone jedynkami w nawiasie(1), natomiast prawym argumentem jest wynik podprogramu $G(1)$.

Lewym argumentem mnożenia w podprogramie Φ_1 jest wynik podprogramu $H(1,1)$. Podobnie w pozostałych podprogramach. Odpowiada to przyjętemu w matematyce zapisowi, np.

$$a+\sin\alpha\cos\alpha,$$

gdzie sin i cos są nazwami pewnych funkcji.

Nieliniowe składanie podprogramów jest wygodniejsze w użyciu, jednak jego realizacja techniczna jest bardziej skomplikowana niż składania liniowego. Nie będziemy jej studiowali szczegółowiej, a podamy tylko ogólny zarys. Oczywiście maszyna realizująca składanie nieliniowe musi posiadać również pamięć podprogramów standardowych oraz skorowidz adresów początków podprogramów standardowych. Jednak w tym przypadku jest to nie wystarczające. Jeżeli bowiem w czasie wykonywania podprogramu nastąpi konieczność wykonania podprogramu standardowego, to maszyna przerywa wykonywanie podprogramu bieżącego i przechodzi do wykonania podprogramu standardowego, co powoduje konieczność zaznaczenia w specjalnym skorowidzu miejsca, w którym zostało przerwane wykonanie programu bieżącego tak, aby po zakończeniu podprogramu standardowego maszyna mogła powrócić do wykonywania programu bieżącego. Oczywiście w czasie wykonywania podprogramu standardowego, ponownie może wystąpić konieczność obliczenia innego podprogramu standardowego itd. Maszyna musi więc być wyposażona w skorowidz, w którym są magazynowane adresy wszystkich miejsc, gdzie przerwane zostało obliczenie. Skorowidz taki jest oczywiście pamięcią rewersyjną.

(1) Nawiasy zostały tu użyte dla przejrzystego zaznaczenia, ile danych ma podprogram. Oczywiście moglibyśmy użyć tu również innych oznaczeń, np. F_3, zamiast $F(1,1,1)$.

ROZDZIAŁ XII

PROCESY ITERACYJNE

Proces prosty $\mathfrak{A}$ nazwiemy *procesem iteracyjnym*, symbolicznie *v-procesem*, jeżeli proces $\mathfrak{A}$ można przedstawić w postaci

$$\underbrace{\mathfrak{A}' \prec \mathfrak{A}' \prec \ldots \prec \mathfrak{A}'}_{n \text{ razy}},$$

gdzie $\mathfrak{A}'$ jest pewnym podprocesem procesu $\mathfrak{A}$. Podproces $\mathfrak{A}'$ będziemy nazywali *iteracją* procesu $\mathfrak{A}$. Proces iteracyjny będziemy też zapisywać krótko

$$\mathfrak{A}' v_n.$$

Program procesu $\mathfrak{A}$ nazwiemy *programem iteracyjnym* albo *v-programem*.

Procesy iteracyjne odgrywają w maszynach matematycznych bardzo dużą rolę. Byłoby rzeczą bardzo pożyteczną pokłasyfikowanie różnego rodzaju procesów iteracyjnych(1).

Tutaj nie będziemy się zajmowali tym zagadnieniem w całej rozciągłości, a podamy tylko najbardziej elementarne przykłady procesów iteracyjnych, ich programy oraz zasadę działania maszyny realizującej procesy iteracyjne.

§ 1. REALIZACJA NIEKTÓRYCH PROCESÓW ITERACYJNYCH W MASZYNACH Z REWERSYJNĄ PAMIĘCIĄ WYNIKÓW CZĘŚCIOWYCH

Maszyna z rewersyjną pamięcią wyników częściowych pozwala na łatwe realizowanie niektórych procesów iteracyjnych.

Rozpatrzmy np. program

$$1\,0+\,0$$

Program ten nie jest poprawny w sensie dotychczasowych definicji, nie przedstawia on bowiem żadnego drzewa. Tym niemniej maszyna ten

(1) W teorii funkcji obliczalnych taka klasyfikacja była robiona, jednak do naszych celów jest ona nieprzydatna.

program wykona. Działanie maszyny będzie polegało w tym przypadku na dodaniu liczby z pamięci wejściowej do liczby z pamięci rewersyjnej i umieszczeniu wyniku dodawania na miejscu liczby odczytanej z pamięci wyników częściowych. Jeżeli program ten powtórzymy n razy, to maszyna wykona obliczenie sumy

$$a_1+a_2+\ldots+a_n=\sum_{i=1}^{n} a_i,$$

gdzie a_i są umieszczone w kolejnych miejscach liniowej pamięci danych (albo pamięci wejściowej).

Program obliczenia sumy n wyrazów możemy więc zapisać

$$\underbrace{1\,0+\,1\,0+\,\ldots+1\,0+}_{n \text{ razy}}\,0,$$

lub krócej:

$$1\,0+\,v_n,$$

gdzie v_n oznacza n-krotne powtórzenie programu $1\,0+\,0$. Dla prostoty zera na końcu programu nie piszemy(1).

Inny przykład procesu iteracyjnego. Program

$$1\,1\cdot\,0\,0+\,0$$

jest również wyrażeniem niepoprawnym, jednak maszyna program ten wykona następująco: zostaną odczytane dwie kolejne dane z pamięci danych, ich iloczyn umieszczony w pamięci rewersyjnej. Zgodnie z zasadą działania tej pamięci, jeżeli ostatnią zapisaną komórką w pamięci rewersyjnej była komórka c_i, to iloczyn będzie zapisany w komórce c_{i+1}. Następnie zostanie dodana zawartość komórki c_i oraz c_{i+1} i wynik zostanie umieszczony w komórce c_i. Jeżeli program zostanie powtórzony n razy, to maszyna obliczy wyrażenie

$$\sum_{i=1}^{n} a_i\, b_i,$$

gdzie a_i, b_i są danymi umieszczonymi na przemian w pamięci danych(2).

(1) Przyjęliśmy milcząco, że w komórce c_i przed rozpoczęciem liczenia zapisane było zero.

(2) Można przyjąć, że mamy dwie pamięci danych: $a_1, a_2, \ldots, a_n$ są umieszczone

Jest to więc iloczyn skalarny dwu wektorów(1).

Program iloczynu skalarnego możemy zapisać więc w postaci

$$\underbrace{11\cdot 00+\,11\cdot 00+\ldots+11\cdot 00+}_{n\text{ razy}}\,0,$$

lub krócej:

$$11\cdot 00+\,\nu_n.$$

Aby maszyna realizowała program w postaci skróconej, konieczny jest licznik iteracji liczący, ile razy ma być powtórzona iteracja. Zamiast licznika iteracji można zastosować specjalny symbol w pamięci danych, który po odpowiedniej ilości iteracji przerwie wykonywanie programu.

Wartość wielomianu

$$a_n+x\,(a_{n-1}+x\,(a_{n-2}+x\,(\ldots x(a_1+xa_0)\ldots)))$$

możemy obliczyć za pomocą wzorów

1. $z_0=a_0$,
2. $z_{i+1}=z_i\cdot x+a_{i+1}$.

W języku podstawowym wielomian ten zapiszemy

$$0x\cdot\,10+\,\nu_n,$$

gdzie x oznacza odczytanie rejestru, w którym znajduje się wartość x, a współczynniki wielomianu $a_1, a_2\ldots, a_n$ znajdują się w liniowej pamięci danych.

W stanie początkowym w pierwszej odczytywanej komórce c_i pamięci rewersyjnej znajduje się współczynnik a_0. Liczba a_0 jest pomnożona przez x i wynik jest umieszczony w komórce c_i, na miejscu współczynnika a_0. Następnie do a_0x dodawany jest a_1 i wynik jest umieszczany w pamięci c_i itd. Po n-krotnej iteracji otrzymamy w komórce c_i wartość wielomianu.

Wielomian interpolacyjny Newtona

$$a_n+a_{n-1}(y-x_n)+a_{n-2}(y-x_n)(y-x_{n-1})+\ldots+a_0(y-x_n)(y-x_{n-1})\ldots(y-x_1)$$

w jednej pamięci, natomiast $b_1, b_2, \ldots, b_n$ w drugiej, i obie pamięci są odczytywane na przemian.

(1) Przyjęliśmy milcząco, że w komórce c_i przed rozpoczęciem liczenia zapisane było 0.

możemy zapisać

$$a_n+(y-x_n)[a_{n-1}+(y-x_{n-1})[a_{n-2}+\ldots+(y-x_2)[a_1+(y-x_1)a_0]\ldots]],$$

lub inaczej:

1. $z_0=a_0$,
2. $z_{i+1}=z_i(y-x_i)+a_i$.

W języku podstawowym program ten zapisujemy

$$y\,1-\,00\cdot\,10+\,\nu_n,$$

gdzie y oznacza odczytanie rejestru, w którym znajduje się wartość y, oraz a_i, x_i są umieszczone w pamięci danych według schematu

$$x_1, a_1, x_2, a_2, \ldots, x_n, a_n,$$

a w pierwszej odczytywanej komórce c_i pamięci rewersyjnej znajduje się przed rozpoczęciem liczenia współczynnik a_0.

Przebieg obliczenia przedstawia się następująco: najpierw wykonywane jest odejmowanie $y-x_i$, którego wynik umieszczony jest w pamięci rewersyjnej c_{i+1}, następnie a_0 jest mnożone przez $y-x_i$, i wynik jest umieszczony w komórce c_i, po czym do zawartości komórki c_i dodawane jest a_1 z pamięci danych i proces ten jest powtarzany n razy.

W podobny sposób można zapisać wiele innych procesów iteracyjnych. Język podstawowy jest bardzo wygodny do zapisywania procesów iteracyjnych.

Wszystkie podane procesy można również zapisać w innych językach, jednak są one mniej wygodne do realizacji technicznej od języka podstawowego.

§ 2. REALIZACJA PROCESÓW ITERACYJNYCH ZA POMOCĄ MASZYNY Z REJESTREM WYNIKÓW ITERACJI

Obecnie podamy ogólny schemat maszyny, realizującej procesy iteracyjne, niezależnie od tego, który rodzaj pamięci zastosowano w maszynie, jako pamięć wyników częściowych, czy też pamięć roboczą.

Przyjmiemy, że maszyna posiada rejestr I, w którym zapisywany jest wynik końcowy każdej iteracji. Oczywiście maszyna ma również licznik

iteracji, którego działania nie będziemy tutaj omawiali. Jeżeli argumentem jakiegoś działania w następnej iteracji jest wynik końcowy poprzedniej iteracji, to argument ten oznaczymy symbolem i. Jeżeli w czasie wykonywania programu urządzenie sterujące odczyta symbol i, powoduje to pobranie z rejestru I liczby jako argumentu działania. Poprzednio podany przykład programu sumy n wyrazów obecnie zapiszemy

$$1\, i+\ v_n.$$

Potęgowanie x^y, gdzie y jest liczbą naturalną, możemy zapisać następująco:

1. $x^0=1$,
2. $x^{y+1}=x^y\cdot x$.

Program potęgowania będzie miał postać:

$$i x\cdot\ v_y.$$

Podobnie można przedstawić wiele innych procesów iteracyjnych.

Podane metody nie wyczerpują oczywiście wszystkich możliwych, spotykanych w praktyce schematów iteracyjnych. Chodziło nam tu bowiem tylko o wskazanie, w jaki sposób procesy iteracyjne mogą być realizowane w maszynach bezadresowych.

§ 3. REALIZACJA ZŁOŻONYCH PROGRAMÓW ITERACYJNYCH

Obecnie będziemy rozpatrywali takie procesy, w których program iteracji jest programem złożonym. W praktyce takie procesy iteracyjne mają duże znaczenie. W dalszym ciągu podamy indukcyjną definicję złożonych programów iteracyjnych.

Niech Γ będzie programem złożonym o postaci

$$\Phi_1, \Phi_2, \ldots, \Phi_k,$$

gdzie Φ_t są podprogramami programu złożonego Γ.

Wyrażenie Γv_n^i, $1\leqslant i\leqslant k$, nazwiemy *złożonym programem iteracyjnym*, gdzie n oznacza ilość iteracji, a i numer podprogramu, od którego należy rozpoczynać iterację. Podprogram Φ_i będziemy nazywali *początkiem iteracji*. Początek iteracji będziemy czasem zaznaczali pisząc Φ_i' lub strzałkę,

jak to pokazano niżej:

$$\Phi_1, \Phi_2, \ldots, \Phi_i', \ldots, \Phi_k \quad v_n$$

lub

$$\Phi_1, \Phi_2, \ldots, \Phi_i, \ldots, \Phi_k\ v_n.$$

Realizacja programu iteracyjnego złożonego polega na realizacji kolejno podprogramów $\Phi_1, \Phi_2, \ldots, \Phi_k$ po podprogramie Φ_k podprogramu Φ_i aż do Φ_k itd. n razy. Jeżeli $1\leqslant j<i$, to Φ_j nazwiemy *podprogramem wolnym* dla symbolu iteracji v_n^i. Jeżeli $i\leqslant j\leqslant k$, to Φ_j nazwiemy *programem związanym* dla symbolu iteracji v_n^i. A więc podprogramy znajdujące się pod strzałką są związane, a pozostałe wolne dla symbolu iteracji v_n^i.

Niech Γ_1 i Γ_2 będą programami iteracyjnymi złożonymi z podprogramami odpowiednimi $\Phi_1, \Phi_2, \ldots, \Phi_s$ oraz $\Phi_{s+1}, \Phi_{s+2}, \ldots, \Phi_t$. Jeżeli istnieje co najmniej jeden taki podprogram Φ_i $(s<i\leqslant t)$ w Γ_2, że wynik końcowy programu Γ_1 jest argumentem w Φ_i, to zapiszemy $\Gamma_1\prec\Gamma_2$. Jeżeli $\Gamma_1\prec\Gamma_2$, to $\Gamma_1\Gamma_2$ jest również programem iteracyjnym złożonym. Jeżeli Γ jest iteracyjnym programem złożonym, to Γv_n^i jest również programem złożonym, wtedy gdy Φ_i jest podprogramem wolnym w Γ.

Przykłady złożonych programów iteracyjnych

$$\Phi_1, \Phi_2, \Phi_3, \Phi_4\ v_{n_1}\ v_{n_2},$$

$$\Phi_1, \Phi_2\ v_{n_1}\ \Phi_3, \Phi_4, \Phi_5\ v_{n_2}\ v_{n_3}.$$

Sens tych programów jest oczywisty.

Maszyna realizująca złożone programy iteracyjne ma dodatkowy skorowidz rewersyjny adresów początków iteracji. Maszyna wykonując kolejne podprogramy umieszcza w skorowidzu adresy początków iteracji. Po wykonaniu ostatniego podprogramu w danej iteracji odczytywany jest ze skorowidza adres początku iteracji itd. aż do zakończenia programu. Adres początku iteracji jest wymazywany ze skorowidza po zakończeniu iteracji.

ROZDZIAŁ XIII

MASZYNY ADRESOWE

W rozdziale tym nie będziemy zajmowali się konwencjonalnymi maszynami adresowymi, a pokażemy nowe koncepcje maszyn adresowych, które wynikają z dotychczasowych rozważań. Nie będziemy rozpatrywali wszystkich możliwych nowych koncepcji maszyn adresowych, a przedstawimy tylko dwie: jedną realizującą nieco zmodyfikowany język podstawowy, drugą realizującą pewną modyfikację języka Łukasiewicza. W pierwszym przypadku będziemy mieli maszynę jednoadresową, w drugim — maszynę dwuadresową. Jednak dwie te koncepcje nie odpowiadają żadnym, istniejącym zasadom organizacji maszyn ani jednoadresowych ani dwuadresowych. Nie chodzi nam tutaj o podkreślenie praktycznej przydatności naszkicowanych koncepcji maszyn adresowych, ale o wskazanie pewnych nowych możliwości w tym zakresie.

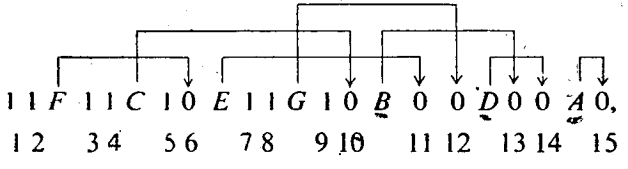
Pierwszą z wymienionych maszyn nazwiemy maszyną 0+0+1 *adresową*, co ma symbolizować, że oba argumenty są odczytywane bezadresowo, natomiast wynik operacji jest umieszczony w pamięci na zasadzie adresu, czyli rozkazy są jednoadresowe. Maszynę drugą nazwiemy maszyną 1+0+1 *adresową*, co ma oznaczać, że w rozkazie występują dwa adresy: adres lewego argumentu i adres wyniku operacji; adres prawego argumentu nie jest podany w rozkazie, a jest obliczony w prosty sposób w maszynie.

§ 1. MASZYNA 0+0+1 ADRESOWA

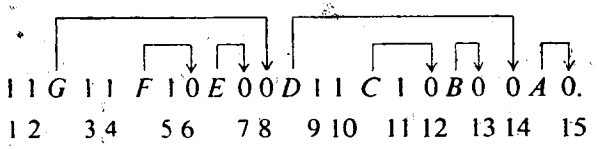
Określimy najpierw pojęcie rozkazu oraz programu w maszynie 0+ +0+1 adresowej. Definicje te można podać w sposób formalny, jednak dla przejrzystości wyjaśnimy je na przykładzie.

Rozpatrzmy uproszczony program w języku podstawowym procesu z porządkiem P lub W

(P)



(W)



Jeżeli w programie ponumerujemy kolejno symbole danych i wyników częściowych, to możemy symbol wyniku częściowego zaznaczyć przez podanie, przy każdym symbolu działania, numeru symbolu wyniku częściowego, który mu odpowiada. Poprzednie programy zapiszemy wtedy

$$(P) \qquad 1\,1\,F_6\;1\,1\,C_{10}\;1\,0\,E_{11}\;1\,1\,G_{12}\;1\,0\,B_{13}\;0\,0\,D_{14}\;0\,0\,A_{15}\;0,$$

$$(W) \qquad 1\,1\,G_8\;1\,1\,F_6\;1\,0\,E_7\;0\,0\,D_{14}\;1\,1\,C_{12}\;1\,0\,B_{13}\;0\,0\,A_{15}\;0.$$

Jak to pokażemy w dalszym ciągu, przy odpowiedniej organizacji maszyny, programy powyższe mogą mieć następującą postać:

$$(P) \qquad F_6\,C_{10}\,E_{11}\,G_{12}\,B_{13}\,D_{14}\,A_{15},$$

$$(W) \qquad G_8\,F_6\,E_7\,D_{14}\,C_{12}\,B_{13}\,A_{15}.$$

Symbole danych oraz wyników częściowych nie występują w tych programach.

Programy te składają się wyłącznie z symboli działań oraz adresów wyników operacji. Rozkazami tych programów są wyrażenia G_8, E_7 itp. Symbol operacji jest nazywany *częścią operacyjną* rozkazu, natomiast stojący przy nim wskaźnik *częścią adresową* rozkazu.

Maszyna składa się z:

O — operatora,
R — adresowej pamięci roboczej,
P — liniowej pamięci programu,
S — sterowania.

Maszyna działa następująco: operator O pobiera jako argumenty każdego działania kolejne pary liczb, mieszczące się w pamięci roboczej R, poczynając od komórek 1, 2. Sterowanie analizuje kolejne rozkazy w pamięci P, nastawia operator na odpowiednią operację oraz powoduje umieszczenie wyniku operacji pod adres podany w części adresowej rozkazu. Przed rozpoczęciem liczenia dane są umieszczane w pamięci R, pod

odpowiednimi adresami. Jeżeli rozkazy i dane są umieszczane od pierwszej komórki pamięci P i R, to oczywiście argumenty rozkazu znajdującego się w komórce p_i znajdują się w pamięci R w komórkach r_{2i-1} oraz r_{2i}. Pozwala to na bardzo proste rozwiązanie sterowania maszyny.

Pracą obu pamięci R i P steruje jeden licznik L, który jest jednocześnie licznikiem rozkazów oraz licznikiem argumentów. Jeżeli w liczniku znajduje się liczba i, to sterowanie powoduje pobranie z pamięci P zawartości komórki i oraz z pamięci R odczytywanie zawartości komórek $2i-1$, $2i$, jako argumentów operacji. Po wykonaniu rozkazu zawartość licznika zwiększona jest o jeden.

Oczywiście maszyna może pracować zarówno w porządku P, jak i w porządku W, gdyż sprawa porządku nie gra tu istotnej roli i sprowadza się do doboru odpowiednich adresów w programie.

Przykład realizowania programu w przypadku P pokazany jest w poniższej tablicy.

Porządek P

Numer rozkazu	Rozkaz	Pamięć R														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	+ 6	$\bar{3}$	$\bar{4}$	5	1	9	$*7$	8	5	9						
2	+10	3	4	$\bar{5}$	$\bar{1}$	9	7	8	5	9	$*6$					
3	−11	3	4	5	1	$\bar{9}$	$\bar{7}$	8	5	9	6	$*2$				
4	−12	3	4	5	1	9	7	$\bar{8}$	$\bar{5}$	9	6	2	$*3$			
5	−13	3	4	5	1	9	7	8	5	$\bar{9}$	$\bar{6}$	2	3	$*3$		
6	·14	3	4	5	1	9	7	8	5	9	6	$\bar{2}$	$\bar{3}$	3	$*6$	
7	+15	3	4	5	1	9	7	8	5	9	6	2	3	$\bar{3}$	$\bar{6}$	$*9$

Z tablicy wyraźnie widać sposób pobierania argumentów oraz umieszczenia wyników operacji. W przykładzie nie pokazano natomiast w jaki sposób wprowadza się dane przed obliczeniem do pamięci R.

Dla porządku W obliczenie przebiega podobnie.

Porządek W

Numer rozkazu	Rozkaz	Pamięć R														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	− 8	$\bar{8}$	$\bar{5}$	3	4	9			$*3$	5	1	9				
2	+ 6	8	5	$\bar{3}$	$\bar{4}$	9	$*7$		3	5	1	9				
3	− 7	8	5	3	4	$\bar{9}$	$\bar{7}$	$*2$	3	5	1	9				
4	·14	8	5	3	4	9	7	$\bar{2}$	$\bar{3}$	5	1	9			$*6$	
5	+12	8	5	3	4	9	7	2	3	$\bar{5}$	$\bar{1}$	9	$*6$		6	
6	−13	8	5	3	4	9	7	2	3	5	1	$\bar{9}$	$\bar{6}$	$*3$	6	
7	+15	8	5	3	4	9	7	2	3	5	1	9	6	$\bar{3}$	$\bar{6}$	$*9$

Kreski nad liczbami oznaczają argumenty operacji, a gwiazdka jej wynik.

Oczywiście możliwe jest również zastosowanie zamiast dwu pamięci — jednej. Zasada pracy pozostaje i w tym przypadku bez zasadniczej zmiany, musi być tylko dokonana poprawka adresów uwzględniająca, że program, dane oraz wyniki częściowe znajdują się we wspólnej pamięci. Czyli argumenty rozkazu o numerze i znajdują się wtedy w komórkach o adresach $2i+n-1$ oraz $2i+n$, gdzie n oznacza ilość rozkazów w programie(1); do adresu wyniku częściowego również należy dodać n.

A więc sterowaniem pobierania rozkazów i argumentów może rządzić jeden licznik, podobnie jak w przypadku maszyny z dwoma pamięciami.

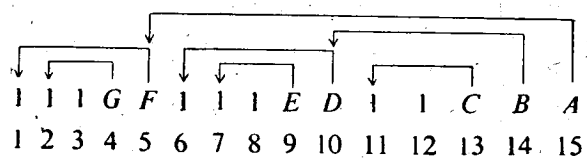
§ 2. MASZYNA 1+0+1 ADRESOWA

Analizując język Łukasiewicza, podobnie jak to uczyniliśmy z językiem podstawowym w poprzednim paragrafie, możemy również otrzymać nową koncepcję maszyny adresowej.

Ponumerujemy w programie Łukasiewicza wszystkie symbole kolej-

(1) Zakładamy tu, że program jest umieszczony w pamięci, poczynając od adresu 1.

nymi liczbami jak niżej:



Z przykładu widać, że jeżeli symbol działania ma numer i, to symbol prawego argumentu ma numer $i-1$. Numer symbolu lewego argumentu można określić na podstawie programu.

Program Łukasiewicza można więc zapisać następująco:

$$G_{4,2}\ F_{5,1}\ E_{9,7}\ D_{10,6}\ C_{13,11}\ B_{14,10}\ A_{15,5}\,.$$

Podwójne wskaźniki przy każdym symbolu operacji są adresami. Pierwszy wskaźnik nazwiemy *adresem wyniku*, drugi *adresem lewego argumentu*. Ponieważ adres prawego argumentu jest o jeden mniejszy od adresu wyniku, mamy zatem adresy obu argumentów oraz wyniku. Maszyna realizująca ten język może być zbudowana również z dwiema bądź z jedną pamięcią. Jeżeli maszyna ma oddzielną adresową pamięć roboczą oraz oddzielną liniową pamięć programu, to schemat jej jest zbliżony do schematu maszyny, dyskutowanej w poprzednim paragrafie.

Dla ilustracji rozpatrzmy przebieg obliczenia w postaci tablicy.

Rozkaz	Pamięć R														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
+ 4,2	9	$\bar{5}$	$\bar{1}$	$*\bar{6}$		9	3	4			8	5			
− 5,1	$\bar{9}$	5	1	$\bar{6}$	$*\bar{3}$	9	3	4			8	5			
+ 9,7	9	5	1	6	3	9	$\bar{3}$	$\bar{4}$	$*\bar{7}$		8	5			
−10,6	9	5	1	6	3	$\bar{9}$	3	4	$\bar{7}$	$*\bar{2}$	8	5			
−13,11	9	5	1	6	3	9	3	4	7	2	$\bar{8}$	$\bar{5}$	$*\bar{3}$		
·14,10	9	5	1	6	3	9	3	4	7	$\bar{2}$	8	5	$\bar{3}$	$*\bar{6}$	
+15,5	9	5	1	6	$\bar{3}$	9	3	4	7	2	8	5	3	$\bar{6}$	$*\bar{9}$

Warto zwrócić uwagę, że numery rozkazów nie grają tu żadnej roli. W przypadku obu maszyn adresy mogą być też obliczane automatycznie przez maszynę i wtedy otrzymamy jeszcze jeden schemat maszyny bezadresowej.

ROZDZIAŁ XIV

AUTOMATYCZNE PROGRAMOWANIE

Przedstawione tu koncepcje mogą być również zastosowane do automatycznego programowania maszyn konwencjonalnych. Dla przykładu rozpatrzymy tu w sposób uproszczony zastosowanie języka Łukasiewicza do automatycznego programowania. Przyjmiemy porządek W, jest on bowiem wygodniejszy do posługiwania się, gdyż pozwala na czytanie programów od strony lewej do prawej. Dla uproszczenia przyjmiemy, że maszyna ma realizować λ-programy uproszczone, a więc realizować zadany program wielokrotnie dla różnych danych. Następnie przyjmijmy, że dane są zapisane w kolejności ich występowania na wejściu. Parametry programu znajdują się natomiast w pamięci maszyny i w trakcie obliczenia są nie zmieniane.

Aby maszyna mogła wykonać obliczenie według programu Łukasiewicza, musimy do maszyny wprowadzić specjalny program, który będzie analizował realizowaną formułę Łukasiewicza i zależnie od odczytanego symbolu wykonywał odpowiednie operacje. Maszyna zaopatrzona w taki program symuluje więc działanie odpowiedniej maszyny bezadresowej. Celem dokonania symulacji musimy w pamięci maszyny wydzielić fragmenty spełniające rolę odpowiednich pamięci modelowanej maszyny bezadresowej, tj. pamięci danych, pamięci formuły i pamięci wyników częściowych (lub pamięci roboczej); musimy również przewidzieć specjalne miejsce w pamięci, w którym umieścimy program modelujący.

Dokładny podział pamięci pokazany jest w paragrafie 2.

§ 1. LISTA ROZKAZÓW

Rozpatrzymy ogólną strukturę programu maszyny jednoadresowej, który analizuje kolejne symbole programu uproszczonego Łukasiewicza i realizuje według niego obliczenie. Tak więc zaprogramowanie jakiegoś obliczenia polega w tym przypadku na wprowadzeniu do maszyny uproszczonego programu Łukasiewicza, problemu, który ma być przez maszynę obliczony.

Struktura programu modelującego w pewnym stopniu zależy od listy rozkazów maszyny. Zagadnieniem tym jednakże nie będziemy się w zasadzie tutaj zajmowali i przyjmiemy następującą konwencjonalną listę rozkazów:

Kod	Skrót	Znaczenie rozkazu
0	$R(a)$	Zawartość komórki o adresie a prześlij do rejestru rozkazów
1	$C(a)$	Prześlij zawartość komórki o adresie a do akumulatora
2	$P(a)$	Prześlij zawartość akumulatora do komórki o adresie a
3	$D(a)$	Dodaj zawartość komórki o adresie a do akumulatora
4	$E(a)$	Odejmij zawartość komórki o adresie a od akumulatora
5	$M(a)$	Pomnóż zawartość komórki o adresie a, przez zawartość rejestru mnożnika Q
6	$U(a)$	Prześlij zawartość komórki o adresie a do rejestru Q
7	$Q(a)$	Podziel zawartość rejestru Q przez liczbę znajdującą się w komórce a
8	$V(a)$	Odczytaj kolejne wejście i odczytane słowo umieść pod adresem a
9	$S(a)$	Skocz do komórki o adresie a.
10	$S(a)?$	Skocz warunkowo do komórki o adresie a, jeżeli zawartość akumulatora $\leqslant 0$
11	$N(a)$	Do części adresowej słowa w komórce o adresie a dodaj 1
12	$I(a)$	Od części adresowej słowa w komórce o adresie a odejmij 1
13	$O(a)$	Wyzerowanie komórki o adresie a

§ 2. PODZIAŁ PAMIĘCI

Pamięć maszyny podzielimy na bloki według schematu.

Numer bloku	Adresy	Przeznaczenie
B 0	0 1 . . . a_1-1	Pomocnicze
B 1	a_1 a_1+1 . . . a_2-1	Podprogramy czytania formuły Łukasiewicza (Φ_1)
B 2	a_2 a_2+1 . . . a_3-1	Podprogram wprowadzenia danych do pamięci roboczej (Φ_2)
B 3	a_3 a_3+1 . . . a_4-1	Podprogram wprowadzania parametrów do pamięci roboczej (Φ_3)
B 4	a_4 a_4+1 . . . a_5-1	Podprogram dodawania (Φ_4)

Numer bloku	Adresy	Przeznaczenie
B 5	a_5 a_5+1 . . . a_6-1	Podprogram odejmowania (Φ_5)
B 6	a_6 a_6+1 . . . a_7-1	Podprogram mnożenia (Φ_6)
B 7	a_7 a_7+1 . . . a_8-1	Podprogram dzielenia (Φ_7)
B 8	a_8 a_8+1 . . . a_9-1	Formuła Łukasiewicza
B 9	a_9 a_9+1 . . . $a_{10}-1$	Parametry
B 10	a_{10} $a_{10}+1$ . . . $a_{11}-1$	Pamięć robocza

Dla odróżnienia programów jednoadresowych od programu Łukasiewicza, ten ostatni będzie nazywany *formułą Łukasiewicza*, lub krótko: *formułą*.

Działanie podprogramu jest następujące:

Najpierw odczytywany jest blok B 1. Podprogram Φ_1, powoduje czytanie kolejnego symbolu w formule Łukasiewicza, znajdującej się w bloku B 8. Zależnie od odczytanego symbolu, następuje przejście do jednego z bloków B 2, B 3, B 4, B 5, B 6 lub B 7. Jeżeli odczytany symbol formuły Łukasiewicza jest symbolem danej, to podprogram Φ_2 powoduje wpisanie danej z wejścia do pamięci roboczej w bloku B 10. Dane są wpisywane na kolejne wolne miejsca w pamięci roboczej. Jeżeli odczytanym symbolem jest symbol parametru, to podprogram Φ_3 znajdujący się w bloku B 3, wprowadza bieżący parametr z bloku B 9, na kolejne wolne miejsce do pamięci roboczej w bloku B 10. Jeżeli odczytany symbol formuły Łukasiewicza jest symbolem operacji, to zależnie od rodzaju operacji maszyna przechodzi do wykonywania jednego z podprogramów Φ_4, Φ_5, Φ_6 lub Φ_7. Każdy z tych podprogramów powoduje odczytanie dwu ostatnio zapisanych liczb w pamięci roboczej bloku B 10 i przesłanie ich, w zależności od rodzaju operacji, do odpowiednich rejestrów arytmometru. Po odczytaniu, oba ostatnie miejsca w pamięci roboczej są wyzerowane. Po zakończeniu jednego z podprogramów Φ_2, Φ_3, Φ_4, Φ_5, Φ_6, Φ_7 maszyna wraca do wykonywania podprogramu Φ_1, znajdującego się w bloku B 1, który powoduje analizowanie następnego symbolu formuły Łukasiewicza itd., aż do całkowitego przeanalizowania formuły Łukasiewicza.

§ 3. PODPROGRAMY

Przyjmiemy, że wszystkie symbole pierwotne w języku Łukasiewicza są ponumerowane liczbami naturalnymi np.

d	— symbol danej	2
p	— symbol parametru	3
$+$	— symbol dodawania	4
$-$	— symbol odejmowania	5
$\cdot$	— symbol mnożenia	6
$/$	— symbol dzielenia	7

To znaczy, że maszyna na wejściu nie czyta np. symbolu $+$, lecz liczbę 4. W bloku B 8 formuła Łukasiewicza jest umieszczana w ten sposób, że każdy numer symboli formuł Łukasiewicza znajduje się na najmniej znaczącym miejscu słowa. Ponieważ numery te są nieduże, część operacyjna każdego słowa jest równa zeru.

W dalszym ciągu opiszemy zawartości poszczególnych bloków.

Blok B 0. W bloku B 0 znajdują się następujące rozkazy:

Adres	Rozkaz	Objaśnienie
0	—	
1	—	
2	$S(a_2)$	Skok do podprogramu Φ_2
3	$S(a_3)$	Skok do podprogramu Φ_3
4	$S(a_4)$	Skok do podprogramu Φ_4
5	$S(a_5)$	Skok do podprogramu Φ_5
6	$S(a_6)$	Skok do podprogramu Φ_6
7	$S(a_7)$	Skok do podprogramu Φ_7
8	—	
9	—	
10	$a_{10}-1$	

Znaczenie tych rozkazów poznamy w dalszych paragrafach.

W pamięci 10 znajduje się adres $a_{10}-1$, tj. adres pierwszej komórki pamięci roboczej zmniejszony o 1. Część operacyjna słowa w komórce 10 jest równa zeru.

Blok B 1. Podprogram Φ_1 czytania formuły Łukasiewicza jest następujący:

Adres	Rozkaz
a_1	$N(a_1+1)$
a_1+1	$R(a_8-1)$

Rozkaz $N(a_1+1)$ powoduje zwiększenie części adresowej rozkazu $R(a_8-1)$ o jeden tak, że po dodaniu jedynki przyjmuje on postać $R(a_8)$. Rozkaz $R(a_8)$ powoduje przesłanie zawartości komórki a_8 do rejestru rozkazów. Ponieważ w komórce a_8 znajduje się numer n_1 $(2 \leqslant n_1 \leqslant 7)$ pierwszego symbolu formuły Łukasiewicza, więc numer n_1 jest przesłany do rejestru rozkazów. W rejestrze rozkazów numer ten jest interpretowany jako rozkaz przesyłania $R(n_1)$, gdyż w części adresowej zgodnie z przyjętą

umową znajduje się zero. Rozkaz $R(n_1)$ powoduje ponowne pobranie do rejestru rozkazów zawartości komórki n_1. W komórce n_1 natomiast znajduje się rozkaz skokowy $S(a_i)$, $2 \leqslant i \leqslant 7$, a więc zależnie od odczytanego symbolu następuje przejście maszyny do jednego z podprogramów Φ_2, Φ_3, Φ_4, Φ_5, Φ_6, Φ_7.

Blok B 2. Podprogram Φ_2 wprowadzania danych z wejścia do pamięci roboczej

Adres	Rozkaz
a_2	$N(10)$
a_2+1	$C(10)$
a_2+2	$D(a_2+6)$
a_2+3	$P(20)$
a_2+4	$R(20)$
a_2+5	$S(a_1)$
a_2+6	$V(0)$

Pierwszy rozkaz podprogramu Φ_2 zwiększa adres w komórce 10 o jeden. Zawartość komórki 10 gra rolę licznika, wskazującego ostatnie zapisane miejsca w pamięci roboczej. Ponieważ przed rozpoczęciem liczenia pamięć robocza jest nie zapisana zupełnie, więc w komórce 10 znajduje się adres $a_{10}-1$. Po dodaniu jedynki otrzymamy więc w komórce 10 liczbę a_{10}.

Następny rozkaz $C(10)$, powoduje przesłanie liczby a_{10} do akumulatora. Rozkaz $D(a_2+6)$ powoduje dodanie adresu a_{10} do rozkazu $V(0)$. Ponieważ w akumulatorze część adresowa nie jest równa zeru, a w rozkazie $V(0)$ część adresowa równa się zeru, więc w wyniku dodawania otrzymamy rozkaz $V(a_{10})$, który oznacza odczytanie danej z wejścia do pamięci a_{10}. Następny rozkaz $P(20)$ powoduje przesłanie rozkazu $V(a_{10})$ do komórki 20, która służy do przechowywania różnych częściowych obliczeń lub rozkazów, mających pomocniczy charakter. Przyjmujemy, że komórka 20 mieści się jeszcze w bloku B 0. Przyjmiemy także, że $a_1=100$. Komórkę tę nazwiemy *komórką pomocniczą*. Z komórki pomocniczej rozkaz $V(a_{10})$ jest przesyłany, za pomocą rozkazu $R(20)$, do rejestru rozkazów, po czym następuje już wykonanie rozkazu $V(a_{10})$ i przejście do następnego rozkazu. Następny rozkaz $S(a_1)$ jest rozkazem skokowym, powodującym przejście do podprogramu Φ_1, który czyta następny symbol formuły Łukasiewicza w bloku B 3.

Blok B 3. Blok B 3 zawiera podprogram Φ_3 wprowadzenia parametrów do pamięci roboczej.

Adres	Rozkaz
a_3	$N(10)$
a_3+1	$C(10)$
a_3+2	$D(a_3+8)$
a_3+3	$P(20)$
a_3+4	$N(a_3+5)$
a_3+5	$C(a_9-1)$
a_3+6	$R(20)$
a_3+7	$S(a_1)$
a_3+8	$P(0)$

Niech l będzie liczbą znajdującą się w komórce 10. Rozkaz $N(10)$ powoduje, że w komórce 10 znajduje się liczba $l+1$. Rozkaz $C(10)$ umieszcza w akumulatorze liczbę $l+1$. Rozkaz $D(a_3+8)$, dodaje do akumulatora rozkaz $P(0)$ i w rezultacie otrzymamy rozkaz $P(l+1)$. Rozkaz $P(20)$ umieszczono za pomocą rozkazu $P(l+1)$ w komórce 20. Rozkaz $N(a_3+5)$ zwiększa adres w rozkazie $C(a_9-1)$ o jeden tak, że w komórce a_3+5 znajduje się rozkaz $C(a_9)$. Rozkaz $C(a_9)$ przesyła pierwszy parametr do akumulatora. Rozkaz $R(20)$ pobiera z komórki 20 rozkaz przesyłania $P(l+1)$, który powoduje przesłanie parametru z akumulatora do kolejnego miejsca w pamięci roboczej, tj. do miejsca o adresie $l+1$. Rozkaz $S(a_1)$ powoduje przejście maszyny do wykonywania podprogramu Φ_1.

Blok B 4. Podprogram dodawania Φ_4.

	Adres	Rozkaz
	a_4	$C(10)$
1	a_4+1	$D(a_4+14)$
	a_4+2	$P(20)$
	a_4+3	$I(10)$
	a_4+4	$C(10)$
2	a_4+5	$D(a_4+15)$
	a_4+6	$P(21)$

	Adres	Rozkaz
	a_4+7	$C(10)$
3	a_4+8	$D(a_4+16)$
	a_4+9	$P(22)$
	a_4+10	$R(20)$
4	a_4+11	$R(21)$
	a_4+12	$R(22)$
5	a_4+13	$S(a_1)$
	a_4+14	$C(0)$
6	a_4+15	$D(0)$
	a_4+16	$P(0)$

Nie będziemy szczegółowo omawiali programu dodawania, a objaśnimy go tylko szkicowo. Omówienie zaczniemy od grupy ostatniej. W grupie tej znajdują się trzy rozkazy $C(0)$, $D(0)$, $P(0)$ opisujące pobranie obu argumentów z pamięci roboczej, oraz umieszczenie wyniku w pamięci roboczej. Rozkazy te posiadają adresy zero i przed wykonaniem muszą otrzymać właściwe adresy. Dzieje się to za pośrednictwem grup rozkazów 1, 2 i 3. Pierwsza grupa rozkazów powoduje umieszczenie w pamięci 20 rozkazu $C(l)$, gdzie l jest ostatnim zajętym miejscem w pamięci roboczej. Druga grupa powoduje przesłanie do komórki 21 rozkazu dodawania $D(l-1)$. Trzecią grupę umieszcza w pamięci 22 rozkaz zapisu wyniku dodawania $P(l-1)$. Czwarta grupa powoduje wykonanie rozkazów $C(l)$, $D(l-1)$ oraz $P(l-1)$. Rozkaz następny $S(a_1)$ kieruje maszynę do podprogramu Φ_1.

Blok B 5. Podprogram odejmowania Φ_5 jest podobny do podprogramu dodawania, nie będziemy go więc dyskutowali.

Blok B 6. Podprogram mnożenia Φ_6 nie różni się w istocie od podprogramu dodawania. Pierwsze pięć grup rozkazów są identyczne z podprogramem dodawania Φ_4. Szósta grupa natomiast ma postać

Adres	Rozkaz
a_6+14	D (0)
a_6+15	$M(0)$
a_6+16	P (0)

Rozkazy $D(0)$, $M(0)$, $P(0)$ po odpowiednim zmodyfikowaniu adresów powodują pobranie do arytmometru mnożnej i mnożnika oraz wykonanie mnożenia i umieszczenie iloczynu w odpowiednim miejscu pamięci roboczej.

Blok B 7. Podprogram dzielenia Φ_7 ma podobnie do podprogramu mnożenia, pierwszych pięć grup rozkazów identyczne z podprogramem dodawania, natomiast grupa szósta ma rozkazy:

Adres	Rozkaz
a_7+14	$U(0)$
a_7+15	$Q(0)$
a_7+16	$P(0)$

Znaczenie tych rozkazów jest podobne do znaczenia podprogramu mnożenia.

§ 4. UWAGI OGÓLNE O AUTOMATYCZNYM PROGRAMOWANIU JĘZYKA ŁUKASIEWICZA

Podany w poprzednim paragrafie program był raczej szkicem, a nie szczegółowym programem. Tym niemniej widać z niego, że automatyczne programowanie wymaga wielu rozkazów, a tym samym i czasu do realizacji języka Łukasiewicza. Dla języka nawiasowego automatyczne programowanie jest jeszcze bardziej skomplikowane. Widać wyraźnie, że przy tych samych środkach technicznych w maszynie bezadresowej mamy znacznie większą efektywną szybkość działania, gdyż na każdy symbol w języku Łukasiewicza wypada jeden rozkaz.

Gdybyśmy zastosowali inną listę rozkazów, poszczególne podprogramy można by znacznie uprościć, tym niemniej stosunek szybkości nadal pozostanie korzystniejszy dla maszyn bezadresowych. Np. niech w maszynie będą możliwe oprócz rozkazów podanych w liście rozkazy typu $X(a)'$, gdzie X jest dowolną literą z listy rozkazów, natomiast $(a)'$ oznacza adres adresu, tj. wykonanie rozkazu nie na wielkości znajdującej się w komórce a, lecz w komórce, której adres jest podany w komórce a. Wtedy program dodawania i innych działań arytmetycznych znacznie się uprości.

	Adres	Rozkaz
	a_4	$C(10)$
1	a_4+1	$P(20)$
	a_4+2	$I(10)$
	a_4+3	$C(10)'$
2	a_4+4	$D(20)'$
	a_4+5	$P(10)'$
3	a_4+6	$S(a_1)$

Podprogram zamiast 16 rozkazów zawiera tylko 7, co oczywiście przyspiesza szybkość działania maszyny.

Działanie programu jest oczywiste.

Podprogram wprowadzania parametrów Φ_3 ma również prostszą postać

Adres	Rozkaz
a_3	$N(10)$
a_3+1	$N(a_3+2)$
a_3+2	$C(a_9-1)$
a_3+3	$P(10)'$
a_3+4	$S(a_1)$

Zamiast 8 mamy teraz 5 rozkazów. Zrozumienie programu nie przedstawia trudności.

Podprogram wprowadzania danych Φ_2 także się upraszcza.

Adres	Rozkaz
a_2	$N(10)$
a_2+1	$V(10)'$
a_2+2	$S(a_1)$

W przypadku zastosowania automatycznego programowania w maszynach konwencjonalnych nie jest rzeczą obojętną, którą listą rozkazów dysponujemy. Tak więc maszyny te można lepiej bądź gorzej przystosować do zadań automatycznego programowania.

ROZDZIAŁ XV

UWAGI O KONSTRUKCJI MASZYN BEZADRESOWYCH

Dotychczasowe rozważania dotyczyły organizacji maszyn, czyli zasad ogólnej budowy maszyny. Obecnie podamy kilka uwag dotyczących konstrukcji maszyn bezadresowych, uwzględniając już konkretne środki techniczne, za pomocą których możemy maszynę realizować.

§ 1. MAŁA MASZYNA BEZADRESOWA

W paragrafie tym rozważymy najprostszą maszynę bezadresową realizującą język podstawowy z porządkiem W. Maszyna ta jest przeznaczona do wykonywania najprostszych obliczeń technicznych w sposób całkowicie automatyczny.

Założenia:

1. Maszyna realizuje cztery podstawowe działania arytmetyczne: dodawanie, odejmowanie, mnożenie i dzielenie, w zmiennym przecinku(1).
2. Obliczanie jednokrotne bądź wielokrotne programów, zawierających kilkanaście operacji.
3. Przy obliczaniu jednokrotnym programu dane są wprowadzane ręcznie z klawiatury dalekopisu; przy obliczaniu wielokrotnym dane są wprowadzane automatycznie z taśmy perforowanej.
4. Wyniki są drukowane w ustalonym formacie przez dalekopis i ewentualnie dziurkowane na taśmie.
5. Wygodne obliczanie wartości wielomianów; możliwość wprowadzania zmiennej niezależnej, ręcznie z klawiatury.
6. Realizacja programów iteracyjnych.
7. Realizacja programu iloczynu skalarnego dwu wektorów.
8. Szybkość wykonywania — kilkadziesiąt działań arytmetycznych na sekundę.

(1) Liczby na wejściu maszyny są zapisywane z przecinkiem na odpowiedniej pozycji, np. 9,54.

W założeniach nie podano dokładności maszyny, która tu nie gra istotnej roli. Dla wygodnej obsługi i kontroli poprawności działania maszyny przyjmiemy, że możliwe jest wprowadzanie i wyprowadzanie danych z wejścia do dowolnej komórki pamięci oraz rejestrów arytmometru.

Maszyna składa się z:

A — arytmometru,
P — pamięci,
R — rejestru programu,
S — sterowania,
W — wejścia i wyjścia dalekopisowego,
C — czytnika taśmy perforowanej,
M — pulpitu manipulacyjnego.

Uwagi o arytmometrze

Ze względów konstrukcyjnych wygodnie jest, aby arytmometr był zmiennoprzecinkowym arytmometrem dziesiętnym. Tłumaczenie z systemu dwójkowego na dziesiętny za pomocą programów jest w maszynach bezadresowych niewygodne. Można również zastosować arytmometr zmiennoprzecinkowy dwójkowy, z tym, że na wejściu znajdą się odpowiednie układy tłumaczące. Poza tym arytmometr jest konwencjonalny.

Uwagi o pamięci

Do realizacji postawionych w założeniach zadań wystarczy niewielka pamięć, w której będą mieściły się wyniki częściowe oraz parametry. Do tego celu wystarczy pojemność rzędu 30 słów. Ponieważ maszyna jest wolna wystarczy tutaj pamięć typu opóźnieniowego, np. jedna linia magnetostrykcyjna. Wygodnie wtedy byłoby również zastosować linie magnetostrykcyjne, jako rejestry arytmometru.

Rejestr programu może być również rejestrem magnetostrykcyjnym o pojemności kilku słów.

Pulpit manipulacyjny

Sygnalizacja świetlna

1. Nadmiar.
2. Plus akumulatora.

3. Minus akumulatora.
4. Zero akumulatora.
5. Wprowadzić dane.
6. Wykonywana operacja.
7. Stan pamięci.
8. Stop.

Przełączniki

1. Wprowadzanie programu.
2. Obliczenie jednokrotne albo wielokrotne.
3. Praca krokowa albo normalna.
4. Przełącznik wejść.
5. Zerowanie rejestrów arytmometru.

Obsługa maszyny jest bardzo prosta. Najpierw wprowadzany jest program do rejestru programu oraz parametry do pamięci *P*. Jeżeli program jest obliczany jednokrotnie, to dane mogą być wprowadzane z klawiatury dalekopisu. Po wprowadzeniu danej z dalekopisu, maszyna realizuje program tak długo, aż będzie potrzebna nowa dana. Wtedy następuje zatrzymanie maszyny i pojawia się na pulpicie sygnalizacja 5, oznaczająca wprowadzenie danych. Po wprowadzeniu następnej danej maszyna znów realizuje program, aż do pojawienia się w programie symbolu następnej danej itd.

Przy wielokrotnej realizacji programu dane są kolejno pobierane z taśmy, aż do pojawienia się symbolu oznaczającego koniec danych. Wtedy następuje zatrzymanie maszyny. Gdybyśmy dla tych celów chcieli zbudować maszynę adresową, okazałoby się, że niezbędna jest co najmniej dziesięciokrotnie większa pamięć. Maszyna bezadresowa jest więc tutaj znacznie prostsza.

Podobnie można zbudować maszynę pracującą w języku Łukasiewicza lub w języku nawiasowym. W przypadku języka Łukasiewicza i nawiasowego, konieczna byłaby może większa pamięć, gdyż dane przed wprowadzeniem do arytmometru muszą być najpierw wprowadzone do pamięci. Oczywiście pamięć grałaby wtedy rolę pamięci roboczej. Przy realizacji języka nawiasowego konieczny jest jeszcze skorowidz operacji, który może być prostym rejestrem przesuwanym.

§ 2. DUŻA MASZYNA BEZADRESOWA

Dyskusja konstrukcji dużej maszyny bezadresowej jest trudna do przeprowadzenia, gdyż należałoby dokładniej określić jej zastosowania; w ramach krótkiego opisu jest to niemożliwe. Tym niemniej kilka uwag można tu podać:

1. Pamięć wyników częściowych powinna tu być możliwie najszybszą pamięcią ferrytową lub tranzystorową o pojemności 8-16 słów,

2. Pamięć danych, parametrów i wyników pośrednich mogą tu być pamięciami ferrytowymi niezależnymi, bądź oddzielnymi, szczególnie pamięć wyników pośrednich może być niedużą pamięcią ferrytową.

3. Pamięć programów powinna być raczej również niezależną pamięcią ferrytową o pojemności zależnej od wielkości rozwiązywanych problemów.

4. Wszystkie skorowidze powinny być również niazależnymi pamięciami ferrytowymi.

5. Pamięć podprogramów standardowych może być połączona razem z pamięcią danych i parametrów.

W przypadku zastosowania jednej pamięci, spełniającej wszystkie funkcje, sterowanie byłoby zbyt skomplikowane i czas działania maszyny zbyt długi.

ZAKOŃCZENIE

Określiliśmy pojęcie procesu języka opisującego proste procesy oraz podaliśmy różne sposoby ich realizowania. Nie są to gotowe rozwiązania konstrukcyjne, tym niemniej na ich podstawie opracowanie konkretnych konstrukcji nie przedstawia trudności. Oczywiście mogą pojawić się wtedy problemy nie poruszane w tej książce, np. inne rozwiązanie procesów iteracyjnych lub składania podprogramów, zasadnicza myśl jednak nie ulegnie zmianie. Nie podano tutaj również zasad realizowania procesów jednoczesnych, tj. zasad współpracy wielu arytmometrów przy realizacji jednego programu.

Interesujące jest badanie organizacji maszyn bezadresowych z punktu widzenia szybkości liczenia. Prowadzi to do jeszcze innych schematów organizacyjnych, niż te, które były tutaj dyskutowane. Ciekawe jest również badanie organizacji bezadresowych maszyn wieloprogramowych, które w książce nie zostało poruszone. Nie podano również bardzo ważnego zagadnienia rozwiązywania równań liniowych, za pomocą maszyn bezadresowych. Nie dyskutowano tutaj zastosowania do dowodzenia twierdzeń matematycznych, wielokrotnej dokładności i innych. Tematy te pominięto celowo. Po pierwsze — nie wszystkie z nich są jeszcze dostatecznie opracowane; po drugie — chodziło tu raczej o naszkicowanie sposobu postępowania niż o szczegółowe rozwiązania.

Wydaje się rzeczą zastanawiającą, że inne języki są wygodne do realizacji technicznej, a inne — dla człowieka. Niewątpliwie najwygodniejszym językiem do posługiwania się w rachunkach ręcznych jest język nawiasowy. Pozwala bowiem na łatwe zorientowanie się w strukturze procesu obliczania po jednym spojrzeniu, bez potrzeby szczegółowego analizowania formuły.

Dla celów maszynowych wygodniejszy jest język podstawowy, w którym opis wszystkich czynności, które mają być wykonane w czasie rachunku, występuje w takiej samej kolejności, w jakiej będą one wykonywane w czasie liczenia.

W językach używanych w matematyce istnieje zawsze problem odróżnienia funkcji od wartości funkcji. Ten sam napis może oznaczać jedno i drugie. Np. nie wiadomo czy x^2+y oznacza funkcję, czy jej wartość. Istnieje wiele symbolik, które rozróżniają oba te pojęcia.

W języku podstawowym problem ten nie istnieje. Pojęciu funkcji odpowiada program (nie jest to zbyt ścisłe) i na oznaczenie obiektu otrzymanego, w wyniku zrealizowania programu, przewidziany jest w programie inny symbol.

Z podaną problematyką wiążą się również sprawy natury ogólnej. Chodzi tu o związki wprowadzanych pojęć z innymi pojęciami matematycznymi. Rozważania te jednak nie leżą w granicach kompetencji autora i być może zostaną kiedyś przeprowadzone przez osoby bardziej do tego celu predystynowane.

DODATEK

§ 1. MASZYNA Z JEDNĄ PAMIĘCIĄ

Rozpatrywane do tej pory maszyny posiadały oddzielne pamięci dla danych programu i wyników częściowych. Rozwiązanie takie zapewnia większą szybkość liczenia, kosztem ilości wyposażenia. Każda z pamięci musi bowiem posiadać niezależne układy odczytu i zapisu. Większa szybkość wynika stąd, że jednocześnie można odczytywać kilka pamięci.

Czasem wygodnie jest zastosować wspólną pamięć dla danych, wyników częściowych oraz programu. Dla ilustracji rozpatrzymy organizację maszyny, realizującej język podstawowy, z porządkiem W, oraz ze wspólną pamięcią danych, parametrów oraz wyników częściowach. Niech d, c i p oznaczają symbole danych, wyników częściowych i parametrów.

Maszyna składa się z:

X — pamięci,
A — arytmometru,
a — rejestru adresów,
L_d — licznika danych,
L_c — licznika wyników częściowych,
L_p — licznika parametrów,
L_r — licznika rozkazów,
S — sterowania.

Pamięć X podzielona jest na następujące części:

D — pamięć danych,
R — pamięć parametrów,
C — pamięć wyników częściowych,
P — pamięć programu.

Schemat maszyny pokazany jest na rysunku 29.

Przyjmiemy, że komórki pamięci są ponumerowane liczbami od 0 do k. Jeżeli rejestr a zawiera liczbę i, to powiemy, że wyznaczona jest komórka i. Niech L_d oznacza także liczbę w L_d i podobnie dla L_r, L_c, L_p. Każda

komórka pamięci X może być wyznaczona przez jedną z liczb L_d, L_c, L_p, L_r. Wprowadzimy następujące operacje pamięciowe:

(D) — odczytaj komórkę o adresie L_d,

$[D]$ — zapisz w komórce o adresie L_d,

$\overleftarrow{D}$ — wyznacz komórkę o adresie L_d-1,

$\overrightarrow{D}$ — wyznacz komórkę o adresie L_d+1.

oraz podobne operacje dla pamięci P, W, C.

Tablica operacji dla omawianej maszyny będzie miała wtedy postać

l	p	Operacje pamięci
c	c	$\overleftarrow{(C)}$, $\overleftarrow{(C)}$, $\left[\overrightarrow{C}\right]$
c	d	$\overleftarrow{(C)}$, $\overrightarrow{(D)}$, $\left[\overrightarrow{C}\right]$
c	p	$\overleftarrow{(C)}$, $\overrightarrow{(R)}$, $\left[\overrightarrow{C}\right]$
d	d	$\overrightarrow{(D)}$, $\overrightarrow{(D)}$, $\left[\overrightarrow{C}\right]$
d	c	$\overrightarrow{(D)}$, $\overleftarrow{(C)}$, $\left[\overrightarrow{C}\right]$
d	p	$\overrightarrow{(D)}$, $\overrightarrow{(R)}$, $\left[\overrightarrow{C}\right]$
p	c	$\overrightarrow{(R)}$, $\overleftarrow{(C)}$, $\left[\overrightarrow{C}\right]$
p	d	$\overrightarrow{(R)}$, $\overrightarrow{(D)}$, $\left[\overrightarrow{C}\right]$
p	p	$\overrightarrow{(R)}$, $\overrightarrow{(R)}$, $\left[\overrightarrow{C}\right]$

Przed rozpoczęciem liczenia, zawartości liczników muszą spełniać zależności

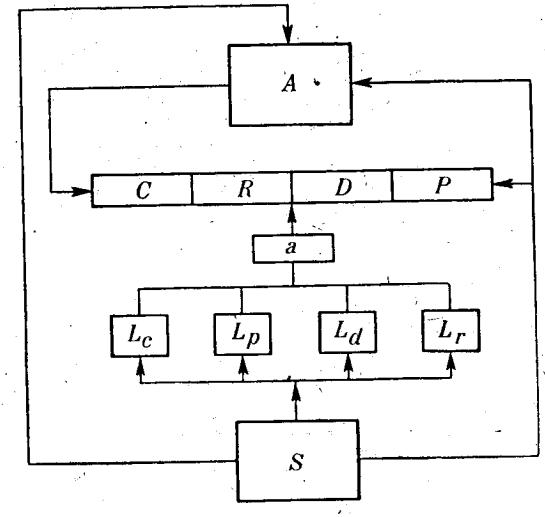
$$L_r=0,$$
$$L_d\geqslant n,$$
$$L_p\geqslant d+L_d,$$
$$L_c\geqslant p+L_p,$$

gdzie d ilością danych oraz p ilością parametrów. Ponadto

$$k=L_c+n,$$

gdzie k jest liczbą wszystkich komórek pamięci(1), n jest ilością operacji w programie.

Liczniki można odpowiednio ustawić w czasie wprowadzania programu z wejścia do pamięci. Licznik L_d liczy ilość symboli operacji, licznik L_p sumę symboli operacji i symboli danych wejściowych, licznik L_c sumę symboli operacji, symboli danych wejściowych i symboli parametrów. Licznik L_r steruje wprowadzenie programu do pamięci P oraz danych i parametrów do odpowiednich pamięci. Po zakończeniu wprowadzania licznik L_r jest nastawiony na zero.



Rys. 29

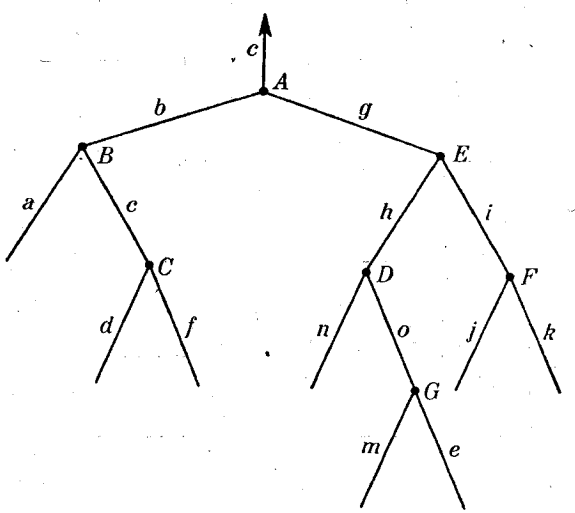
Oczywiście działanie maszyny z wspólną pamięcią jest wolniejsze od maszyny posiadającej oddzielne pamięci, wymaga bowiem kolejnego od-

(1) Dla prostoty przyjęliśmy, że na wyniki częściowe zarezerwowano n komórek pamięci, chociaż wiadomo, że nie wszystkie z nich będą wykorzystane. Można by podać również wzór, który pozwala na automatyczne obliczenie ilości miejsc na wyniki częściowe, ale jest to oczywiście nieopłacalne, bowiem na wyniki częściowe zarezerwowano ostatnią część pamięci i sprawa zaoszczędzenia kilku miejsc nie ma tutaj znaczenia.

czytywania pamięci i dodatkowych czynności, polegających na przesyłaniu liczb z liczników do rejestru adresów. Tym niemniej, w niektórych przypadkach rozwiązanie takie może być celowe.

§ 2. MASZYNA TRÓJADRESOWA

Wychodząc z pojęcia procesu prostego można również określić pojęcie programu trójadresowego. Przyjmiemy, że w procesie operacje są oznaczone dużymi literami łacińskimi, a wszystkie obiekty (dane i wyniki częściowe) literami małymi, jak to pokazano na rysunku 30.



Rys. 30

Programem trójadresowym nazwiemy ciąg

$$a_1\ a_2\ a_3\ D_1\ a_4\ a_5\ a_6\ D_2\ \ldots\ a_{3n-2}\ a_{3n-1}\ a_{3n}\ D_n$$

taki, że dla każdego D_i $a_{3i-2}, a_{3i-1}, a_{3i}$ są odpowiednio symbolami lewego argumentu, prawego argumentu oraz wyniku, oraz dla każdego D_i, $1 \leqslant i < n$, istnieje takie D_j, $j > i$, że zachodzi jedna z dwu możliwości $a_{3j} = a_{3i-2}$ lub $a_{3j} = a_{3i-1}$. To znaczy, że wynik każdego działania, z wyjątkiem ostatniego, jest lewym bądź prawym argumentem w jakiejś dalszej operacji. A więc działania są tu częściowo uporządkowane i nie ma w związku z tym możliwości dokonania uproszczenia, które daje dobre uporządkowanie operacji.

Maszyna realizująca język trójadresowy jest bardzo prosta. Istnieje w maszynie tylko jedna pamięć adresowa. Dane są umieszczone w dowolnych miejscach pamięci, wyniki częściowe są również umieszczone dowolnie, $a_{3i}, a_{3i-1}, a_{3i-2}$ są interpretowane jako adresy. Program może znajdować się w oddzielnej pamięci liniowej i jest realizowany podobnie jak inne programy dyskutowane w tej książce.

§ 3. MASZYNY CZĘŚCIOWO BEZADRESOWE

Można również wprowadzić koncepcję *maszyn częściowo bezadresowych*, tj. takich maszyn, w których jednocześnie stosowano zasadę adresowej i bezadresowej organizacji.

Niech A oznacza zasadę adresową, natomiast B zasadę bezadresową. Wprowadzimy następującą zasadę klasyfikacji maszyn częściowo bezadresowych. Każdą maszynę podzielimy ze względu na trzy cechy: odczytywanie danych początkowych, odczytywanie wyników częściowych oraz zapisywanie wyników częściowych. Wszystkie możliwe typy maszyn względem przyjętej klasyfikacji są przedstawione poniżej:

- *AAA* — maszyna trójadresowa,
- *AAB* — maszyna z bezadresowym zapisem (lub z adresowym odczytem),
- *ABA* — maszyna z bezadresowym odczytem wyników częściowych,
- *ABB* — maszyna z adresowym odczytem danych,
- *BAA* — maszyna z bezadresowym odczytem danych,
- *BAB* — maszyna z adresowym odczytem wyników częściowych,
- *BBA* — maszyna z bezadresowym odczytem (lub z adresowym zapisem danych),
- *BBB* — maszyna bezadresowa.

Podana klasyfikacja może być rozszerzona na dalsze cechy maszyny jak odczytywanie parametrów, rozkazów itp.

W dalszych paragrafach podam dwa przykłady maszyn częściowo bezadresowych, z jedną pamięcią dla programu, danych oraz wyników częś-

ciowych. Wprowadzenie parametrów nie przedstawia trudności i może być rozwiązane w sposób podobny do podanego w poprzednich przykładach.

Dla przykładu rozpatrzymy organizację maszyny *ABB*. Przyjmiemy, że maszyna posiada wspólną pamięć oraz liczy w porządku *W*. Schemat maszyny pokazany jest na rysunku 31.

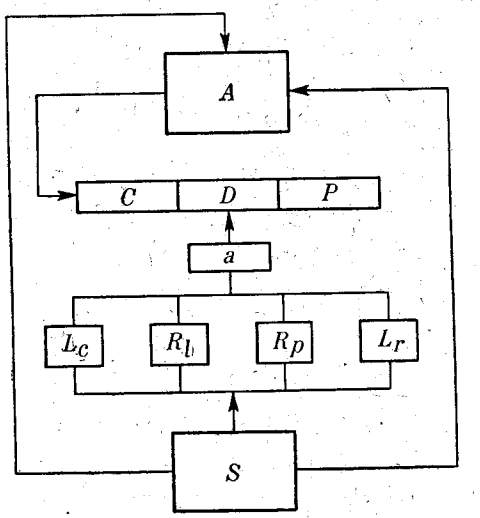
Schemat maszyny nie różni się wiele od schematów maszyn poprzednich. Dodatkowymi elementami są: rejestr lewego argumentu R_l oraz rejestr prawego argumentu R_p. Pozostałe elementy jak w maszynach poprzednich. Pamięć może być przygotowana do odczytu lub zapisu przez przesłanie do rejestru adresów jednej z liczb R_l, R_p, L_c, L_r. Symbolami argumentów są liczby od 0 do k. Liczba 0 jest interpretowana jako symbol dowolnego wyniku częściowego (bezadresowy), natomiast pozostałe liczby jako adresy.

Tablica operacji pamięci ma wtedy postać:

l	p	Operacje pamięci
0	0	$\overleftarrow{(C)}$, $\overleftarrow{(C)}$, $\left[\overrightarrow{C}\right]$
0	i	$\overleftarrow{(C)}$, (d_i), $\left[\overrightarrow{C}\right]$
i	0	(d_j), $\overleftarrow{(C)}$, $\left[\overrightarrow{C}\right]$
i	j	(d_i), (d_j), $\left[\overrightarrow{C}\right]$

(d_i) oznacza odczytanie komórki d_i — pamięci danych.

Oczywiście programy mogą być w maszynach częściowo bezadresowych przedstawiane w dowolnym z dyskutowanych języków. Tak więc maszynę trójadresową, jak i bezadresową możemy traktować jako szczególny przypadek maszyn częściowo bezadresowych. Do niektórych celów maszyny częściowo bezadresowe mogą być korzystniejsze od maszyn bezadresowych (*BBB*).



Rys. 31

§ 4. SPRAWDZANIE POPRAWNOŚCI PROGRAMÓW

Ze względu na niedoskonałości elementów technicznych, z których są budowane maszyny matematyczne, w maszynach są stosowane różnego rodzaju kontrole, np. tzw. kontrola parzystości, pozwalająca stwierdzić, czy w maszynie nie został popełniony błąd przy przesyłaniu. Oczywiście metoda ta pozwala na wykrywanie tylko niektórych błędów. W maszynach bezadresowych możliwa jest kontrola innego typu, polegająca na badaniu poprawności wykonywanych programów. W maszynach adresowych tego rodzaju kontrola jest niemożliwa, nie istnieje bowiem ściśle sprecyzowane pojęcie programu.

W celu badania poprawności programów wprowadzimy dla każdego dyskutowanego języka funkcję sprawdzającą, której argumentami są symbole sprawdzanego programu, a wartościami liczby naturalne i zero. Dla dyskutowanych tutaj języków funkcje te mają następujące postacie:

Język podstawowy. Niech σ_i będzie i-tym symbolem programu podstawowego Φ. Niech $S_1(\sigma_i)$ będzie funkcją sprawdzającą, określaną dla języka podstawowego następująco:

I. $S_1(\sigma_1) = 0$.

II. $S_1(\sigma_{i+1})=\begin{cases} S_1(\sigma_i), \text{ jeżeli } \sigma_{i+1} \text{ jest symbolem działania;} \\ S_1(\sigma_i)+1, \text{ jeżeli } \sigma_{i+1} \text{ jest symbolem danej;} \\ S_1(\sigma_i)-1, \text{ jeżeli } \sigma_{i+1} \text{ jest symbolem wyniku częściowego.} \end{cases}$

Jeżeli dla ostatniego symbolu programu $S_1(\sigma_k)\neq 0$, to program jest niepoprawny. Jeżeli natomiast $S_1(\sigma_i)=0$ dla ostatniego symbolu programu, to o poprawności tego programu nie możemy nic powiedzieć, może się bowiem zdarzyć, że jest on niepoprawny, mimo że funkcja sprawdzająca przyjmuje dla jego ostatniego symbolu wartość zero.

Przykład zastosowania funkcji sprawdzającej do języka podstawowego.

Program	e	f	F	b	c	C	d	$*$	E	g	h	G	a	$*$	B	$*$	$*$	D	$*$	$*$	A	$*$
$S_1(\sigma_i)$	0	1	1	2	3	3	4	3	3	4	5	5	6	5	5	4	3	3	2	1	1	0

Wzór kontrolny jest ważny dla porządków P i W. Dla porządków $\overline{P}$ i $\overline{W}$ należy tylko uwzględnić odwrotny kierunek pisania formuły.

Widzimy, że struktura programu nie zależy w istocie od porządku procesu. Z formalnego punktu widzenia, struktura programów podstawowych nie zależy od porządku procesu i dla wszystkich porządków jest identyczna.

Język Łukasiewicza. Dla języków beznawiasowych oraz Łukasiewicza możemy programy sprawdzić za pomocą funkcji $S_2(\sigma_i)$, określonej następująco:

I. $S_2(\sigma_1)=0$.

II. $S_2(\sigma_{i+1})=\begin{cases} S_2(\sigma_i)+1, \text{ jeżeli } \sigma_{i+1} \text{ jest symbolem działania;} \\ S_2(\sigma_i)-1, \text{ jeżeli } \sigma_{i+1} \text{ jest symbolem operacji.} \end{cases}$

Program	a	b	c	C	B	d	e	f	F	E	g	h	G	D	A
$S_2(\sigma_i)$	0	1	2	1	0	1	2	3	2	1	2	3	2	1	0

Jeżeli dla ostatniego symbolu sprawdzanego programu $S_2(\sigma_k)\neq 0$, to program jest niepoprawny.

Kontrola poprawności nie zależy tutaj, podobnie jak poprzednio, od tego czy jest to program Łukasiewicza, czy beznawiasowy.

Język nawiasowy. Dla języka nawiasowego funkcję kontrolną określimy podobnie.

I. $S_3(\sigma_1)=1$.

II. $S_3(\sigma_{i+1})=\begin{cases} S_3(\sigma_i), \text{ jeżeli } \sigma_{i+1} \text{ jest symbolem danej lub symbolem działania;} \\ S_3(\sigma_i)+1, \text{ jeżeli } \sigma_{i+1} \text{ jest nawiasem lewostronnym (;} \\ S_3(\sigma_i)-1, \text{ jeżeli } \sigma_{i+1} \text{ jest nawiasem prawostronnym).} \end{cases}$

Jeżeli dla ostatniego symbolu programu nawiasowego $S_3(\sigma_k)\neq 0$, to program ten jest niepoprawny.

Program	(	(	a	B	(	b	C	c	)	)	A	(	(	d	E	(	e	F	f	)	)	D	(	g	G	h	)	)	)
$S_3(\sigma_i)$	1	2	2	2	3	3	3	3	2	1	1	2	3	3	3	4	4	4	4	3	2	2	3	3	3	3	2	1	0

Kontrola poprawności programów jest więc bardzo prosta do realizacji technicznej. Funkcje $S_1(\sigma_i)$, $S_2(\sigma_i)$, $S_3(\sigma_i)$ mogą być zrealizowane za pomocą prostego licznika rewersyjnego, który na podstawie analizy programu dodaje bądź odejmuje jedynkę. Sprawdzanie może być przeprowadzane jednocześnie w kilku punktach maszyny, np. przy wprowadzaniu programu do maszyny, lub też przy przepisywaniu z jednej pamięci do drugiej. Byłoby interesujące szczegółowe zbadanie tego problemu.

§ 5. TŁUMACZENIE

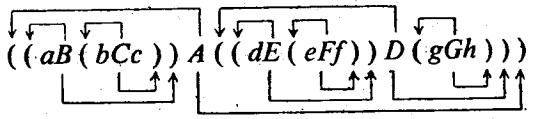
Czasem może być uzasadnione budowanie maszyny pracującej w języku wygodnym do realizacji technicznej, niewygodnym natomiast dla posługującego się maszyną personelu. W takim przypadku można maszynę wyposażyć w urządzenie tłumaczące programy z języka używanego przez personel na język stosowany w maszynie.

Wydaje się, że budowanie maszyny pracującej bezpośrednio w zadanym języku jest racjonalniejsze. Tym niemniej dla ilustracji podamy przykład tłumaczenia programów. W podobny sposób można tłumaczyć i inne podane języki. Dla przykładu rozpatrzymy tłumaczenie z języka nawiasowego na język Łukasiewicza.

W rozdziale II podaliśmy wzory przypisujące każdemu nawiasowi symbol działania. Łatwo zauważyć, że jeżeli symbole działania wpiszemy

na miejsce odpowiadających im nawiasów, to otrzymamy program w języku Łukasiewicza. Jeżeli wpiszemy symbole operacji na miejsce nawiasów prawostronnych i nawiasy lewostronne opuścimy, otrzymamy język Łukasiewicza z porządkiem $\overline{W}$. Jeżeli natomiast wpiszemy symbole operacji na miejsce nawiasów lewostronnych i nawiasy prawostronne opuścimy, otrzymamy język Łukasiewicza z porządkiem W.

Przykład



Przenosząc symbole działań na miejsce nawiasów, wskazane przez górne strzałki, otrzymamy program w języku Łukasiewicza z porządkiem W.

$$A\,B\,a\,C\,b\,c\,D\,E\,d\,F\,e\,f\,G\,g\,h$$

Przenosząc symbole operacji na miejsce wskazane przez strzałki dolne otrzymamy język Łukasiewicza z porządkiem $\overline{W}$.

$$a\,b\,c\,C\,B\,d\,e\,f\,F\,E\,g\,h\,G\,D\,A$$

Realizacja tych algorytmów jest bardzo prosta. Maszyna tłumacząca składa się z trzech pamięci:

N — pamięć liniowa programu nawiasowego,
$Ł$ — pamięć liniowa programu Łukasiewicza,
R — pomocnicza pamięć rewersyjna,

oraz sterowania.

Sterowanie analizuje kolejne symbole programu w pamięci N. Zależnie od odczytanego symbolu działanie maszyny jest następujące:

Symbol odczytany	Czynność maszyny
(	1. Odczytaj następny symbol
Symbol danej	1. Zapisz odczytany symbol w pamięci $Ł$ 2. Odczytaj następny symbol
Symbol operacji	1. Zapisz odczytany symbol w pamięci rewersyjnej R 2. Odczytaj następny symbol
)	1. Odczytaj ostatni symbol z pamięci R i zapisz go w pamięci $Ł$ 2. Odczytaj następny symbol

Przyjęliśmy tu tłumaczenie na język Łukasiewicza z porządkiem $\overline{W}$.

Przebieg tłumaczenia podany jest niżej. W tablicy tej dla przejrzystości, odczytane symbole z pamięci N opuszczono. Oczywiście to nie jest konieczne. Można również podać algorytmy tłumaczenia z języka Łukasiewicza na nawiasowy i inne. Są one jednak bardziej skomplikowane.

Pamięć N	Pamięć $Ł$	Pamięć R
(		
(		
a	a	
B	a	B
(	a	B
b	$a\,b$	B
C	$a\,b$	$B\,C$
c	$a\,b\,c$	$B\,C$
)	$a\,b\,c\,C$	B
)	$a\,b\,c\,C\,B$	
A	$a\,b\,c\,C\,B$	A
(	$a\,b\,c\,C\,B$	A
(	$a\,b\,c\,C\,B$	A
d	$a\,b\,c\,C\,B\,d$	A
E	$a\,b\,c\,C\,B\,d$	$A\,E$
(	$a\,b\,c\,C\,B\,d$	$A\,E$
e	$a\,b\,c\,C\,B\,d\,e$	$A\,E$
F	$a\,b\,c\,C\,B\,d\,e$	$A\,E\,F$
f	$a\,b\,c\,C\,B\,d\,e\,f$	$A\,E\,F$
)	$a\,b\,c\,C\,B\,d\,e\,f\,F$	$A\,E$
)	$a\,b\,c\,C\,B\,d\,e\,f\,F\,E$	A
D	$a\,b\,c\,C\,B\,d\,e\,f\,F\,E$	$A\,D$
(	$a\,b\,c\,C\,B\,d\,e\,f\,F\,E$	$A\,D$
g	$a\,b\,c\,C\,B\,d\,e\,f\,F\,E\,g$	$A\,D$
G	$a\,b\,c\,C\,B\,d\,e\,f\,F\,E\,g$	$A\,D\,G$
h	$a\,b\,c\,C\,B\,d\,e\,f\,F\,E\,g\,h$	$A\,D\,G$
)	$a\,b\,c\,C\,B\,d\,e\,f\,F\,E\,g\,h\,G$	$A\,D$
)	$a\,b\,c\,C\,B\,d\,e\,f\,F\,E\,g\,h\,G\,D$	A
)	$a\,b\,c\,C\,B\,d\,e\,f\,F\,E\,g\,h\,G\,D\,A$	

§ 6. REALIZACJA PAMIĘCI WYNIKÓW CZĘŚCIOWYCH

Kilka uwag o realizacji pamięci wyników częściowych dla porządku P i W (oraz $\overline{P}$ i $\overline{W}$) pozwoli nam na lepsze zrozumienie zasady działania tej pamięci.

Omówimy najpierw działanie pamięci dla porządku poprzecznego. Najprostszą pamięcią liniową jest dziurkowana taśma papierowa. Urządzenie perforujące taśmę, po każdej perforacji przesuwa taśmę o określoną długość, przygotowując ją do następnego dziurkowania. Podobnie, urządzenie czytające po każdym odczycie przesuwa taśmę o jedną pozycję w tym samym kierunku, w którym przesuwa ją urządzenie dziurkujące.

Jeżeli liniowa pamięć wyników częściowych jest pamięcią elektronową (np. ferrytowa, typu opóźnieniowego), to do sterowania pracą pamięci potrzebne są dwa liczniki: licznik zapisu L_z oraz licznik odczytu L_o.

Zapisywanie jest dokonywane na miejsce pamięci, wskazane przez licznik L_z. Po każdym zapisie licznik L_z zwiększa swoją wartość o 1, przygotowując następnie miejsce do zapisu. Odczytywanie pamięci odbywa się według licznika L_o. Po każdym odczycie zawartość licznika L_o zwiększa swoją wartość o 1, przygotowując następne miejsce do odczytu.

Pamięci rewersyjnej natomiast nie można zrealizować na dziurkowanej taśmie papierowej, bowiem po wydziurkowaniu taśmy, nie można jej przywrócić do stanu pierwotnego, co jest koniecznym warunkiem dla działania pamięci rewersyjnej. W przypadku realizacji elektronowej pamięci rewersyjnej do sterowania pracą pamięci wystarczy tylko jeden licznik L, który przed zapisem zwiększa swoją wartość o 1. Natomiast odczyt następuje według aktualnego stanu licznika i po odczycie następuje odjęcie od zawartości licznika jedynki. Możliwa jest jeszcze druga realizacja pamięci rewersyjnej, która w niektórych przypadkach może być interesująca. Pamięć ta może działać na zasadzie rejestru przesuwanego w lewo i prawo z tym, że w poszczególnych ogniwach są magazynowane nie cyfry a liczby.

Działanie takiego rejestru jest więc następujące. Przy zapisie zawartość rejestru jest najpierw przesuwana o jedno miejsce w prawo i na powstałym wolnym miejscu jest zapisywana liczba (1). Przy odczycie zawartość rejestru jest przesuwana w lewo. Odczyt i zapis następują z tego samego punktu rejestru.

(1) Przesuwanie i zapis mogą się odbywać równocześnie.

BIBLIOGRAFIA

[1] Ajdukiewicz, K., *Die Syntaktische Konnexität*, Studia Philos. Vol. 1 (1935), str. 1-27.

[2] Barton, R. S., *A new approach to the functional design of a digital computer*, Proceedings of the Westem Joint Computer Conference, May, 9-11 (1961), str. 393-396.

[3] Curry, H. B., *Some Logical Aspects of Grammatical Structure*, Proceedings of Symposia in Applied Mathematics, Vol. XII, *Structure of Languages and its Mathematical Aspects*, American Mathematical Society (1961), str. 56-58.

[4] — *Foundation of Mathematical Logic*, New York, 1963.

[5] Kalmár, L., *A practical infinistic computer*, Infinistic Methods, Proceedings of the Symposium on Foundations of Mathematics, Warsaw, 2-9 September, 1959, str. 347-362.

[6] — *Über einen Rechenautomaten, der eine mathematische Sprache versteht*, Zeitschrift für Angewandte Mathematik und Mechanik, Band 40 (1960), str. 53.

[7] — *On a digital computer which can be programmed in a mathematical Formula Language* (maszynopis).

[8] Kämmerer, W., *Ziffern rechen automaten mit Programmierung nach mathematischen Formelbild*, Zeitschrift für Angewandte Mathematik und Mechanik, Akademie-Verlag, Band 40 (1960), str. 67-76.

[9] Kuratowski, K., Mostowski, A., *Teoria mnogości*, Warszawa 1952.

[10] Kleene, S. C., *Introduction to metamathematics*, Amsterdam 1963.

[11] Łukasiewicz, J., *Elementy logiki matematycznej*, Warszawa 1958.

[12] Ore, O., *Theory of graphs*, Providence 1962.

[13] Pawlak, Z., *The organization of Digital Computers and Computable Functions*, Biuletyn Polskiej Akademii Nauk, Ser. Techn. Vol. VIII, 1 (1960), str. 41.

[14] — *Organization of the Address-free Digital Computer for Calculating Simple Arithmetical Expresions*, ibid. Vol. VIII, 4 (1960), str. 193.

[15] — *Automatic Programming of Arithmetical Formulae in Parenthesis Notation by the Addressing Function*, ibid. Vol. VIII, 6 (1960), str. 315.

[16] — *On Realization of Recursive Schemes in the Address-free Computer*, ibid. Vol. VIII, 11-12 (1960), str. 685.

[17] — *Organization of Address-free Computer with Seperate Memeory of Partial Results*, ibid. Vol. IX, 2 (1961), str. 123.

[18] — *On the Application of the Rule of Substitution in the Organization of an Address-free Computer*, ibid. Vol. VIII, 11-12 (1960), str. 681.

[19] — *Organization of Address-free Computer B-100*, ibid. Vol. IX, 4 (1960), str. 229.

[20] — *New Conception of Two-address Computer*, ibid. Vol. IX, 5 (1961), str. 313.

[21] — *Some Remarks on Automatic Programming of Arithmetical Formulae*, ibid. Vol. IX, 5 (1961), str. 317.

[22] — *Realization of Memeory of Partial Results in Certain Parenthesis-free Formalisms*, ibid. Vol. IX, 8 (1961), str. 487.

[23] — *On the Utility of Arithmetical Formalisms in Digital Computers*, ibid. Vol. IX, 9 (1961), str. 527.

[24] — *Realization of the Rule of Substitution in Addressless Digital Computer*, ibid. Vol. IX, 9 (1961), str. 531.

[25] — *Realization of the Rule of Substitution in the Addressless Computer without Working Memeory*, ibid. Vol. IX, 10 (1961), str. 579.

[26] — *Realization of Certain Class of Recursive Formulae in the Addressless Computer*, ibid. Vol. IX, 11 (1961), str. 651.

[27] — *Über adreslose digitale Rechen-Maschinen* (w druku).

[28] — *A New Class of Mathematical Languages and Organization of addressless Computers* (w druku).

[29] — *Maszyna i język*, Warszawa 1964.

[30] — *Matematyczna teoria organizacji*, Materiały prakseologiczne, Warszawa 1962.

[31] — *Organization of Addressless computer working in parenthesis notation*, Zeitschrift für Mathematische Logik und Grundlagen der Mathematik, Band 9, Heft 3 (1963), str. 243.

[32] Taub, A. H., *Machine organization in Relation to Problem Types with Particular Emphasis on the Scientific computer*, University of Illinois, File, 320 (1960).

[33] Wang, H., *Toward Mechanized Mathematics*, IBM Journal of Research and Development, Jan. Vol. 20, 53 (1956), str. 28.

[34] Wegner, P., *Zero-address Computers*, Comp. J. 5, 1 (1962), str. 15.

SKOROWIDZ

Adres lewego argumentu 120
— liczbowy 88
— symboliczny 88
— wyniku 120
adresowa część rozkazu 117
adresy 92
— danych 70
alfabet języka J 22
algorytm procesu (program) 22
argument lewy operacji 8
— prawy operacji 8
argumenty działań 39
arytmometr 134

Cykl pracy maszyny 62
— — — w języku Łukasiewicza 51, 54
— — — — — nawiasowym 57
— — — — — podstawowym 46, 48
— — ω-maszyny 93
część adresowa rozkazu 117
— operacyjna rozkazu 117
częściowo bezadresowe maszyny 143

Dana początkowa procesu 9
dowód 9
— formalny 9, 20
drzewo 11, 12
— uporządkowane 12
dualny program beznawiasowy w języku J_2 30
— — w języku Łukasiewicza 33
duża maszyna bezadresowa 136

Formalny dowód 20
formuła Łukasiewicza 126
— procesu (program) 22

Gałęzie wolne drzewa 35
— związane drzewa 35

Implikacja 17
iteracyjny proces (ν-proces) 110
— program (ν-program) 110
— — założony 114
i-ty rozkaz programu 59

Jednokrotny proces (ρ-proces) 95
Język (J) 22
— beznawiasowy (J_2) 29
— Łukasiewicza (J_3) 31, 146
— nawiasowy (J_4) 35, 147
— podstawowy (J_1) 23, 145
— uproszczony (μ-język) 42
— — nawiasowy 42
języki przedmiotowe (π-języki) 40
— symboliczne (σ-języki) 40, 88
— uproszczone 42

Lewy argument operacji 8
licznik lewego argumentu 51
lista rozkazów 122

Łukasiewicza formuła 126
— język (J_3) 31, 146

Maszyna bezadresowa duża 136
— — mała 131
— 1+0+1 adresowa 116
— 0+0+1 adresowa 116
— trójadresowa 142
— z adresową pamięcią wyników pośrednich 105

maszyna z bezadresową pamięcią wyników pośrednich 102
— — jedną pamięcią 139
— — rejestrem wyników iteracji 113
— — rewersyjną pamięcią wyników częściowych 110
— — roboczą pamięcią (μ_2-maszyna) 79
— μ_1 ($J_1(P)$) 66
— μ_1 ($J_1(W)$) 68
maszyny bezadresowe 5
— — częściowo 143

Nawiasowy język 147
nazwa podprogramów 107
nieliniowe składanie podprogramów 108
normalny program beznawiasowy w języku J_2 30
— — procesu 23
— — w języku Łukasiewicza 32
numeracja wzdłużna 31

Obiekty 7
obliczenia 7, 20
obliczenie przez redukcję programu 53
ogólny schemat π-maszyny 45
— — π_1-maszyny 61
operacja końcowa 8
operacji argument lewy 8
— — prawy 8
— wynik 8
operacyjna część rozkazu 117
operator 46

Pamięć 46
— liniowa 63, 64
— rewersyjna 63, 66
— wyników częściowych 63
parametry 97
początek iteracji 114
podprocesy procesu $\mathfrak{A}$ 101
podprogram procesu $\mathfrak{A}$ 101
— wolny 115
podprogramowy 126
podstawowy język 145
podział pamięci 124
pojemność pamięci 65
poprawność programów 145
porządek procesów prostych 13
— poprzeczny 37
— wzdłużny 36
porządki dualne 14
— normalne 14
— poprzeczne 14
— wzdłużne 14
prawy argument operacji 8
proces 12
— $\mathfrak{A}$ 8
— — jednoczesny 8
— — prosty 8
— — sekwencyjny 8
— — w stanie i 10
— — — — końcowym 10
— — — — początkowym 10
— iteracyjny (ν-proces) 110
— jednokrotny (ρ-proces) 95
— prosty 12
— wielokrotny (λ-proces) 95
— wytwarzania 7
produkcja 9
program (algorytm procesu, formuła procesu, schemat procesu) 22
— beznawiasowy dualny 30
— — normalny 30
— dualny w języku Łukasiewicza 33
— iteracyjny (ν-program) 110
— — złożony 114
— normalny w języku Łukasiewicza 32
— podstawowy normalny 23
— trójadresowy 142
— wielokrotny (λ-program) 95
— złożony liniowo 107
— — nieliniowo 109
— — procesu 101, 102
— zredukowany 53
— — jednokrotnie 55
— związany 115

przedmiotowe języki (π-języki) 40
— programy 40

Rachunek liczb naturalnych 9
— logiczny 9
— macierzowy 9
rozkaz 78
— programu 59
— — dla języków symbolicznych 55
— — i-ty 59
rząd symbolu 37

Schemat ogólny π-maszyny 45
— — π_1-maszyny 61
— procesu (program) 22
składanie adresowe 105
skorowidz adresów 71, 74
— danych 74
stałe 97

Tłumaczenie 147
trójadresowy program 142

Uproszczony język (μ-język) 42
— — nawiasowy 42

Wejście λ-maszyny 97
wnioskowanie 9
wolny podprogram 115
— symbol wyniku częściowego 25
wyjście λ-maszyny 97
wynik częściowy procesu 8
wynik końcowy procesu 8
— operacji 8
— pośredni 102
wzdłużna numeracja 31

Złożony program iteracyjny 114
zmienne 92
związany program 115

λ-maszyna 97
λ-proces (proces wielokrotny) 95
λ-program (program wielokrotny) 95
λ-program uproszczony 96
λ-system Chrucha 96

μ-język (język uproszczony) 42
μ-maszyna 45
μ_1-maszyna 59
μ_2-maszyna 59, 79

π-języki (języki przedmiotowe) 40

ρ-język 95
ρ-proces (proces jednokrotny) 95
ρ-programy 95

σ-języki (języki symboliczne) 40

ν-proces (proces iteracyjny) 110
ν-program (program iteracyjny) 110

ω-program 93

SPIS RZECZY

Przedmowa 5

Rozdział I. OBLICZENIA PROSTE 7
§ 1. Procesy 8
§ 2. Procesy i drzewa 11
§ 3. Porządek procesów prostych 13
§ 4. Przykłady procesów prostych 17
§ 5. Przykłady procesów nieprostych 21

Rozdział II. PROGRAMY PROCESÓW PROSTYCH 22
§ 1. Język podstawowy (J_1) 23
§ 2. Język beznawiasowy (J_2) 29
§ 3. Język Łukasiewicza (J_3) 31
§ 4. Język nawiasowy (J_4) 35

Rozdział III. KLASYFIKACJA JĘZYKÓW 40
§ 1. Języki przedmiotowe 40
§ 2. Języki uproszczone 42
§ 3. Definicje formalne języków 42

Rozdział IV. REALIZACJA JĘZYKÓW PRZEDMIOTOWYCH 45
§ 1. Ogólny schemat π-maszyny 45
§ 2. Realizacja języka podstawowego z porządkiem poprzecznym 47
§ 3. Realizacja języka podstawowego z porządkiem wzdłużnym 49
§ 4. Realizacja języka Łukasiewicza 50
§ 5. Realizacja języka Łukasiewicza za pomocą redukcji programu 53
§ 6. Realizacja języka nawiasowego za pomocą redukcji programu . . . 55
§ 7. Uwagi ogólne o bezpośredniej realizacji procesów sekwencyjnych . . . 57

Rozdział V. REALIZACJA JĘZYKÓW UPROSZCZONYCH ZA POMOCĄ MASZYNY Z ODDZIELNĄ PAMIĘCIĄ WYNIKÓW CZĘŚCIOWYCH . 59
§ 1. Pojęcie rozkazu 59
§ 2. Ogólny schemat μ_1-maszyny 61
§ 3. Pamięć wyników częściowych 63
§ 4. Realizacja uproszczonego języka podstawowego z porządkiem poprzecznym (maszyna $\mu_1(\iota_1(J_1P))$) 66

§ 5. Realizacja uproszczonego języka podstawowego z porządkiem wzdłużnym (maszyna $\mu_1(J_1(W))$) 68
§ 6. Realizacja uproszczonego języka Łukasiewicza 69
§ 7. Obliczanie adresów danych w uproszczonym języku Łukasiewicza za pomocą pamięci rewersyjnej 71
§ 8. Realizacja uproszczonego języka Łukasiewicza za pomocą redukcji programu 73
§ 9. Realizacja uproszczonego języka nawiasowego 73
§10. Obliczanie adresów danych w języku nawiasowym za pomocą pamięci rewersyjnej 74
§11. Uwagi ogólne o μ_1-maszynach 77

Rozdział VI. REALIZACJA JĘZYKÓW UPROSZCZONYCH ZA POMOCĄ MASZYNY Z PAMIĘCIĄ ROBOCZĄ 78
§ 1. Ogólny schemat maszyny 78
§ 2. Realizacja uproszczonego języka Łukasiewicza 79
§ 3. Realizacja uproszczonego języka nawiasowego 80
§ 4. Uwagi ogólne o maszynach z pamięcią roboczą 83

Rozdział VII. REALIZACJA JĘZYKÓW UPROSZCZONYCH ZA POMOCĄ MASZYNY Z ADRESOWĄ PAMIĘCIĄ ROBOCZĄ 84
§ 1. Ogólny schemat maszyny 84
§ 2. Realizacja języka Łukasiewicza 85
§ 3. Realizacja języka nawiasowego 86

Rozdział VIII. REALIZACJA JĘZYKÓW SYMBOLICZNYCH. 88
§ 1. Pobieranie danych 89
§ 2. Adresy i zmienne 92
§ 3. Realizacja różnych procesów 93

Rozdział IX. PROCESY WIELOKROTNE. WIELOKROTNA REALIZACJA PROGRAMÓW 95
§ 1. Języki procesów wielokrotnych 95
§ 2. Realizacja procesów wielokrotnych 96
§ 3. Parametry i stałe 97

Rozdział X. SKŁADANIE PODPROGRAMÓW 100
§ 1. Podprogramy i programy złożone 101
§ 2. Ogólny schemat maszyny realizującej programy złożone 101
§ 3. Realizacja programów złożonych za pomocą maszyny z bezadresową pamięcią wyników pośrednich 102
§ 4. Realizacja programów złożonych za pomocą maszyny z adresową pamięcią wyników pośrednich 105

Rozdział XI. PROGRAMY STANDARDOWE 107
§ 1. Liniowe składanie podprogramów 107
§ 2. Nieliniowe składanie podprogramów 108
Rozdział XII. PROCESY ITERACYJNE
§ 1. Realizacja niektórych procesów iteracyjnych w maszynach z rewersyjną pamięcią wyników częściowych 110
§ 2. Realizacja procesów iteracyjnych za pomocą maszyny z rejestrem wyników iteracji 113
§ 3. Realizacja złożonych programów iteracyjnych 114
Rozdział XIII. MASZYNY ADRESOWE 116
§ 1. Maszyna 0+0+1 adresowa 116
§ 2. Maszyna 1+0+1 adresowa 119
Rozdział XIV. AUTOMATYCZNE PROGRAMOWANIE 122
§ 1. Lista rozkazów 122
§ 2. Podział pamięci 124
§ 3. Podprogramy 126
§ 4. Uwagi ogólne o automatycznym programowaniu języka Łukasiewicza . 131
Rozdział XV. UWAGI O KONSTRUKCJI MASZYN BEZADRESOWYCH . 133
§ 1. Mała maszyna bezadresowa 133
§ 2. Duża maszyna bezadresowa 136
ZAKOŃCZENIE 137
DODATEK 139
§ 1. Maszyna z jedną pamięcią 139
§ 2. Maszyna trójadresowa 142
§ 3. Maszyny częściowo bezadresowe 143
§ 4. Sprawdzanie poprawności programów 145
§ 5. Tłumaczenie 147
§ 6. Realizacja pamięci wyników częściowych 150
Bibliografia 151
Skorowidz 153

PAŃSTWOWE
WYDAWNICTWO NAUKOWE

Nakład 1500+200 egz. Ark. wyd. 8. Ark. druk. 10. Pap. druk. sat. kl. V 70 g 61×86 cm. Oddano do składania 9 IV 1964 r. Podpisano do druku 25 II 1965 r. Druk ukończono w marcu 1965 r. M-2 Zamówienie nr 183/41 Cena zł 16,–

Drukarnia Uniwersytetu im. A. Mickiewicza w Poznaniu