P. 1877/69
POLITECHNIKA ŚLĄSKA BIBLIOTEKA GŁÓWNA

maszyny matematyczne

zastosowania

w gospodarce

technice

i nauce

5
1969

Zeszyt jest poświęcony problemom matematycznym w maszynach matematycznych i w przetwarzaniu informacji

SPIS TREŚCI

	Str.
Antoni Mazurkiewicz — „Matematyka w przetwarzaniu informacji"	1
Zdzisław Pawlak — „Uwagi o teorii maszyn cyfrowych"	4
Leon Łukaszewicz — „EOL — język do przetwarzania symboli"	8
Andrzej Włodzimierz Mostowski — „Maszyny matematyczne w logice i algebrze" cz. I	11
Janusz Stanisław Bień — „Algorytmizacja fleksji polskiej — problemy i perspektywy" . .	15
Stanisław Chrobot, Włodzimierz Raczyński — „Programowanie maszyny ZAM-2 GAMMA pracującej na bieżąco"	19
WIADOMOŚCI PKAPI . .	20
Z KRAJU i ze ŚWIATA	
Jerzy Dańda — „Kongres IFIP 68"	21
„KONKURS-69" . . .	IV okł.

СОДЕРЖАНИЕ

А. Мазуркевич — „Математика в обработке информации"

З. Павляк — „Некоторые вопросы теории цифровых вычислительных машин"

Л. Лукашевич — „Язык EOL для обработки символов"

А. В. Мостовски — „Электронные вычислительные машины в логике и алгебре". Часть 1

Я. С. Бень — „Алгоритмизация польской флексии — проблемы и перспективы"

С. Хробот, В. Рачиньски — „Программирование в реальном масштабе времени на электронной вычислительной машине"

ХРОНИКА

CONTENTS

A. Mazurkiewicz — "Mathematics in information processing"

Z. Pawlak — "Some problems of the theory of digital computers"

L. Łukaszewicz — "EOL — symbol manipulation language"

A. W. Mostowski — "Computers in logic and algebra" Part I

J. S. Bień — "Algorithmic approach to Polish inflection. Problems and perspectives"

S. Chrobot, W. Raczyński — "Programming in real time on ZAM-2 GAMMA computer"

CHRONICLE

WYDAWNICTWA CZASOPISM TECHNICZNYCH NOT
Warszawa
Czackiego 3/5

KOLEGIUM REDAKCYJNE

Redaktor naczelny prof. dr Leon ŁUKASZEWICZ

Mgr inż. Jacek KARPIŃSKI, Władysław KLEPACZ, dr Antoni MAZURKIEWICZ, inż. Dorota PRAWDZIC (zast. redaktora naczelnego), mgr inż. Andrzej TARGOWSKI (zast. redaktora naczelnego)

Sekretarz Redakcji mgr Wanda KAĆER **Redaktor techniczny** Alicja BIL

RADA PROGRAMOWA

Prof. dr inż. Jerzy Bromirski (przewodniczący), mgr inż. Jan Bursche, doc. Stefan Czarnecki, mgr Michał Doroszewicz, mgr Adam B. Empacher (sekretarz), mgr inż. Bolesław Gliksman, mgr inż. Józef Knysz, mgr inż. Ludwik Mebel, doc. dr Tadeusz Peche, inż. Zdzisław Puzdrakiewicz, doc. mgr inż. Józef Thierry (wiceprzewodniczący), dr Tadeusz Walczak, mgr Stefan Wojciechowski, dr inż. Henryk Woźniacki, mgr inż. Jan Z. Żydowo

Redakcja: Warszawa, ul. Emilii Plater 20 m. 15, tel. 21-13-91. Zastępca redaktora naczelnego tel. 28-37-29

Zakład Kolportażu WCT NOT, Warszawa, ul. Mazowiecka 12

Zakł. Graf. „Tamka". Z. 2. Zam. 199. Papier powlekany V kl. 80 g. A-1. Obj. 3 ark. druk. Nakład 2300. P-98

Cena egzemplarza zł 8.— **Prenumerata roczna zł 96.—**

P. 1847/69

maszyny matematyczne

zastosowania w gospodarce, technice i nauce

Nr 5

MIESIĘCZNIK

1969

ROK V

Maj

Organ Pełnomocnika Rządu do Spraw Elektronicznej Techniki Obliczeniowej i Naczelnej Organizacji Technicznej

ANTONI MAZURKIEWICZ
Instytut Maszyn Matematycznych
Warszawa

51.004.14:681.3.001+681.3.06+65.011.56

MATEMATYKA W PRZETWARZANIU INFORMACJI[1)]

Autor rozważa pewne wymagania, które stawia przed naukami matematycznymi rozwój zastosowań maszyn matematycznych. Głównymi pojęciami „matematyki informacyjnej" są: pamięć, czynność, stan, informacja, przepływ informacji. Występuje potrzeba nowych teorii i metod sformalizowanego opisywania systemów informacyjnych. Jaskrawym przykładem tego jest problem systematyzacji programowania maszyn cyfrowych.

Nie ma chyba bardziej kontrowersyjnych opinii na temat usytuowania jakiejś nauki w rodzinie nauk ścisłych niż poglądy, dotyczące nauki o przetwarzaniu informacji i maszynach matematycznych. Nauka ta była zaliczana bądź do cybernetyki, bądź matematyki, a niektórzy traktują ją jako swoistą inżynierię informacji, obejmującą również technikę elektroniczną.

Wydaje mi się, że sprawa polega na pomieszaniu tej nauki ze środkami, przy pomocy których się ją uprawia. Technika elektroniczna jest niewątpliwie narzędziem tej nauki: dostarcza ona jedynie fizycznych przyrządów do prac w dziedzinie przetwarzania informacji. Sprzęt elektroniczny odgrywa bowiem analogiczną rolę w informatyce (takiej nazwy będziemy dalej używali), jak np. mikroskop elektronowy we współczesnej biologii czy medycynie. Co więcej, informatyka nie jest też fragmentem matematyki, jak sądzą niektórzy. Jest natomiast nauką wykorzystującą te dziedziny matematyki, które do tej pory uważane były za oddalone w znacznym stopniu od praktycznych zastosowań. Stosunkowo mało docenianym narzędziem informatyki jest naukowa organizacja pracy, systemy klasyfikacji, ekonomia. Każda z dyscyplin służących informatyce, spełnia w niej określoną rolę.

W artykule tym rozważymy rolę matematyki, a ściślej mówiąc niektóre z wymagań, jakie stawia informatyka naukom matematycznym. Nie będziemy więc wyliczać działów matematyki z punktu ich przydatności do zagadnień przetwarzania informacji, lecz pokusimy się o postawienie prognozy, jakiego typu matematyka rozwinie się w przyszłości dla potrzeb informatycznych.

1. Rzut oka w przeszłość

Bogato rozgałęziona na liczne działy i kierunki matematyka wyrosła na gruncie otaczających nas zjawisk. Zbadanie drzewa genealogicznego tej nauki wskazuje na ścisłe jej powiązanie ze zjawiskami fizycznymi. Ukoronowaniem rozwoju matematyki było pojawienie się analizy matematycznej z jej rachunkiem różniczkowym i całkowym o przejrzystej interpretacji fizycznej. Wokół tego jądra, matematyka rozwijała się bądź to „wzwyż" (równania różniczkowe, całkowe, rachunek wariacyjny, teoria dystrybucji, rachunek operatorowy), bądź „wszerz" (analiza funkcjonalna), bądź wreszcie „w głąb" (teoria mnogości, logika, podstawy matematyki).

Przez rozwój „wzwyż" rozumiemy tutaj odkrywanie nowych faktów, udowadnianie nowych twierdzeń, zastosowanie opracowanej metodologii, czyniące teorię przydatną dla coraz to bardziej skomplikowanych zagadnień.

Rozwój „wszerz" polega na uogólnianiu uzyskanych wyników i metod na dziedziny pozornie odległe od siebie, tworzenie wspólnej metodologii dla różnych gałęzi omawianej nauki. Wreszcie istotą rozwoju „w głąb" jest systematyzowanie stosowanych pojęć, ugruntowanie ich na wspólnych podstawach logicznych, zbadanie pierwotnych pojęć wchodzących w zakres badanej nauki.

Podobna sytuacja miała miejsce i z geometrią. Powstała w XVII w. geometria analityczna rozwinęła się potem w geometrię różniczkową, rachunek tensorowy, geometrię Riemanna (widać tu wyraźny wpływ i wzajemne powiązanie z nowoczesnymi teoriami fizycznymi, np. z teorią względności). Podob-

1) Artykuł opracowano na podstawie referatu wygłoszonego na I Ogólnokrajowym Sympozjum „Naukowe problemy maszyn matematycznych", Zakopane, październik 1968 r.

nie, jak w przypadku analizy, rozwój geometrii przebiegał „wszerz" i „w głąb", w kierunku uogólnień i systematyzacji (topologia, klasyfikacja Kleina, geometrie nieeuklidesowe).

Dyscyplina o ogólnomatematycznym znaczeniu to algebra, dostarczająca dogodnego aparatu dla innych dziedzin i teoria podstaw matematyki, starająca się ugruntować pojęcia matematyki na logice i teorii mnogości.

Jak wynika z tego krótkiego przeglądu, matematyka rozwijała się wokół tych zagadnień fizycznych (rozumianych w najszerszym sensie), które znajdowały się w centrum zainteresowania ówczesnej wiedzy praktycznej, jak mechanika, astronomia, elektryczność, budowa materii. Nie wspomniane wyżej: rachunek prawdopodobieństwa i statystyka dostarczyły również silnych środków dla różnych gałęzi wiedzy stosowanej.

2. Sytuacja obecna

W drugiej połowie XX wieku powstało narzędzie, które spowodowało swego rodzaju rewolucję, tym razem nie przemysłową, jak w wieku XVIII, lecz raczej „informacyjną". Narzędziem tym jest m a s z y n a m a t e m a t y c z n a.

Historycznie rzecz biorąc, zamierzeniem twórców tej maszyny było zastosowanie jej do wykonywania obliczeń sformułowanych w terminach matematyki klasycznej, skomplikowanych i długich rachunków dla potrzeb fizyki i techniki. Wkrótce jednak sytuacja uległa gruntownej zmianie. Około 90% maszyn matematycznych, działających obecnie na świecie, pracuje dla potrzeb administracji, bankowości i zarządzania. Dzieje się tak dlatego, że współczesny świat — można rzec — informacją stoi. Nie do pomyślenia jest obecnie funkcjonowanie gospodarki i administracji bez rozwiniętego systemu informacyjnego. Jest to prawdopodobnie skutek wielkiej liczby czynników, które trzeba uwzględniać przy podejmowaniu decyzji gospodarczych i bardzo dużej liczbie powiązań administracyjnych w istniejących społeczeństwach. Weźmy pod uwagę dwa dość typowe przykłady.

System rezerwacji miejsc przez towarzystwa lotnicze, wobec niezwykle szeroko rozbudowanych linii połączeń i powiązań z innymi towarzystwami jest w praktyce nie do zrealizowania bez użycia maszyn matematycznych. Tym niemniej, wobec istnienia dużej liczby samolotów, człowiek współczesny może domagać się szybkiego i sprawnego rozplanowania podróży. Bez użycia maszyn sprawa stałaby się paradoksalna — podróż trwałaby, powiedzmy, kilka godzin, natomiast zarezerwowanie miejsca wymagałoby przypuszczalnie jednej doby lub więcej.

Drugim przykładem jest działalność banków. Po wyeliminowaniu złota jako środka obiegowego, rozwój bankowości pójdzie prawdopodobnie w kierunku wyrugowania banknotów, symulujących dotychczas rzeczywiste wartości przedmiotów. Wzrastający z biegiem czasu udział tak zwanego „obrotu bezgotówkowego" wymaga zamiast przepływu pieniędzy, przepływu informacji. Informacje o stanie kont instytucji lub osób prywatnych muszą być więc zapamiętane, modyfikowane, porównywane i wpływać na właściwą działalność administracyjną. Jest to więc realny przykład, jak dalece w nasze życie ingeruje pozornie abstrakcyjna informacja. Zarządzanie i sprawozdawczość, działalności czysto informacyjne, są potwierdzeniem tego faktu.

W ten sposób — obok świata realnego, namacalnego — rodzi się świat inny, abstrakcyjny, naśladujący zjawiska materialne — świat informacji.

Naukę pozwalającą nam wykorzystywać prawa rządzące tym światem jest właśnie i n f o r m a t y k a. Posługuje się ona nie tylko maszynami działającymi „w pojedynkę" — tworzy się systemy wielomaszynowe, w których informacje są zbierane, opracowywane i wzajemnie przekazywane, niemal bez udziału człowieka. Języki programowania i systemy konwersacyjne oddalają ludzi od maszyn, które prawdopodobnie w niedługiej przyszłości stanowić będą „autonomiczne społeczeństwo". Jest oczywiste, że wysokie wymagania będą wówczas stawiane kadrze fachowców.

3. Dwie matematyki

Nowa rzeczywistość tworzy potrzebę powstania nowej dziedziny matematyki. Zbadajmy stosunek tej oczekiwanej matematyki do matematyki klasycznej takiej, jaką studiowaliśmy i jaką stosujemy. O ile tradycyjną matematykę, wywodzącą swój rodowód z analizy, można by nazwać „matematyką fizykalną", o tyle tę nową matematykę, znajdującą się jeszcze w powijakach, można nazwać „matematyką informacyjną". Wykorzysta ona niewątpliwie zdobycze matematyki współczesnej, lecz różnić się będzie od niej chyba jednak znacznie. Wydaje się, że jako wspólne pozostaną w niej tylko pojęcia z teorii podstaw matematyki i logiki. Pojęciami odpowiedzialnymi za rozwój matematyki fizykalnej są pojęcia przestrzeni, ciągłości i ruchu. Pojęcia te w matematyce informacyjnej tracą na znaczeniu. Przestrzeń w problemach informatyki istnieje tylko w zdegenerowanej postaci. Geometria i metryka są praktycznie pozbawione przypisywanego im zazwyczaj sensu. Nie występuje w tej nauce także ruch traktowany jako zmiana położenia w czasie. Miejsca tych pojęć zajmują w matematyce informacyjnej pojęcia takie, jak pamięć, czynność, stan, informacja, przepływ informacji. Nie istnieje jeszcze dostatecznie rozwinięty aparat matematyczny do wygłaszania twierdzeń, dotyczących tych pojęć i pojęć pochodnych, jakkolwiek aparat taki powstanie już chyba w najbliższej przyszłości. Dotychczasowe środki, jak np. pojęcie maszyny Turinga, nie negując ich filozoficznego znaczenia, okazują się zbyt słabe.

Można mniemać, że jesteśmy teraz na tym etapie rozwoju matematyki informacyjnej, który odpowiada w przybliżeniu prenewtonowskiej analizie. Jesteśmy co prawda bogatsi o wiedzę z zakresu logicznych podstaw matematyki i uzbrojeni w cały arsenał metod matematycznych — jest więc zrozumiałe dążenie do formalizacji matematyki informacyjnej w oparciu o najbardziej podstawowe teorie matematyki „fizykalnej". Przyszłość odpowie jednak dokładniej na pytanie o strukturę tej nowej nauki.

4. Maszyny i matematyka

Współcześni matematycy skłonni są na ogół widzieć w maszynie matematycznej narzędzie, które nie jest w stanie samo przez się dostarczyć nowych, interesujących faktów matematycznych. Takie cechy maszyny matematycznej, jak szybkość liczenia, możliwość pamiętania dużej liczby danych, są dla matematyki współczesnej nieistotne; pojęcia te nie są do wysłowienia w teoriach matematycznych. Pogląd ten jest niewątpliwie słuszny. Jednakże niekiedy wyciąga się z niego fałszywy wniosek, że istnienie szybkich i pojemnych maszyn liczących pozostaje bez wpływu na rozwój matematyki. Tak nie jest. Powstanie nowych możliwości w zakresie przetwarzania informacji powoduje powstanie nowych metod, które bez tych możliwości nie miałyby szans istnienia. Poszukiwanie takich metod, ich ocena lub wzajemne porównywanie zmusza do tworzenia pewnych uogólnień i systematyki, skąd nieduży (lecz być może bardzo trudny) krok do skonstruowania pewnej teorii. Co więcej, sama natura procesów przetwarzania informacji jest tak abstrakcyjna, że stanowi doskonałe tworzywo działalności matematycznej. Powstałe w ten sposób teorie należą już do matematyki; informatyka korzysta z nich na równi z innymi teoriami matematycznymi.

Tworzenie takich teorii — trzeba sobie jasno powiedzieć — nie jest jednak celem informatyki. Nauka

ta wytworzy przypuszczalnie swoje własne teorie (nie będące fragmentem matematyki), jej służące i przez nią oceniane. Im bardziej będą one abstrakcyjne, tym bliżej będą związane z matematyką. Dopóki jednak praktyka przetwarzania informacji wyprzedza teorię, nie może być mowy o ich pełnej matematyzacji.

Jak silne są związki informatyki z tak zwaną „czystą matematyką", można się przekonać z artykułu R. Marczyńskiego omawiającego przedmiot nauki, zwanej w krajach anglosaskich *„Computer Science"* [2] lub program kształcenia w dziedzinie tejże nauki, opisany w tak zwanym *„Curriculum 68"* [3].

5. Początki nowej nauki

Maszyna analogowa [4] jest narzędziem matematyki fizykalnej. Stosowanie tych maszyn opiera się na następujących związkach: system fizyczny — opis matematyczny — ustawienie (program) maszyny analogowej. Opis matematyczny będący środkiem pełnego sformułowania problemu, rozwiązywanego przez maszynę analogową, spełnia tutaj rolę abstrakcyjnego pośrednika pomiędzy badaną rzeczywistością a programem. Sytuacja taka jest możliwa tylko dlatego, że istnieje dostatecznie rozbudowany formalizm matematyczny dla zagadnień opracowywanych za pomocą maszyn omawianego typu. Wygodniej czasem bywa wyeliminować opis matematyczny z wymienionego związku; mamy wówczas do czynienia z modelowaniem analogowym.
W przypadku maszyn cyfrowych sytuacja wygląda podobnie dla zagadnień numerycznych, w pełni opisanych przez formuły matematyczne. Natomiast dla zagadnień innych, jak np. administracyjnych, wykorzystanie maszyny opiera się na związku: system informacyjny — program dla maszyny cyfrowej. Dla tych zagadnień mamy więc do czynienia wyłącznie z modelowaniem, tym razem cyfrowym. Brak pośredniego ogniwa jest skutkiem nieznajomości praw matematyki informacyjnej, braku odpowiedniego formalizmu, w którym można byłoby opisać „informacyjną" rzeczywistość. Wypada nadmienić, że próby formalizacji takiej prowadzone są już od dosyć dawna (1959—1960) i w różnych kierunkach.

Jaskrawym przykładem pilnej potrzeby właściwego formalizmu jest sprawa dokumentacji oprogramowania maszyn. Oprogramowanie stanowi rozbudowany i złożony system informacyjny, którego opis jest albo nieczytelny i związany z pewną konkretną maszyną, albo też sporządzony w języku potocznym i nieprecyzyjnym, nie pozwalający na odtworzenie wiernej (pod względem funkcjonalnym) kopii oprogramowania dla innej maszyny. I tutaj również czynione są próby pewnej systematyzacji pozwalającej stworzyć podstawę późniejszych teorii.

Pogląd na programowanie jako na gałąź matematyki reprezentowany jest raczej nielicznie. Mimo niewątpliwie dedukcyjnego charakteru konstruowania programów, nie istnieje dotychczas teoria stawiająca nietrywialne pytania w tej dziedzinie i udzielająca na nie odpowiedzi. Dotychczasowy formalizm jest na tyle nie przystosowany, że nie można w nim formułować pytań o istotnym znaczeniu dla praktyki.

Sprawdzanie poprawności programów odbywa się zazwyczaj eksperymentalnie; powiązanie metod doświadczalnych z dedukcyjnymi czyni z programowania — które ma do czynienia w gruncie rzeczy tylko z pojęciami abstrakcyjnymi — naukę niemal przyrodniczą. Jest oczywiste, że konkretny program oraz znajomość języka, w którym program ten jest napisany, wystarczają do określenia jego działania; mamy bowiem do czynienia z systemem w pełni sformalizowanym. Niestety, przeprowadzenie dowodu, że dla takich to a takich danych program daje określone wyniki, jest w praktyce niemożliwe, przynajmniej dla programów napisanych w tzw. języku maszyny. Ktoś może powiedzieć: sam program jest dowodem, że dla określonych danych otrzymamy określone wyniki; chodzi tu jednak o wykazanie, że „dowód" ten jest poprawny. Stworzenie metod wykazywania poprawności tych programów, lub jeśli kto woli, „dowodów" znajduje się w tej chwili w centrum zainteresowania naukowców-informatyków.

Prace badawcze tego typu są zaczątkiem kształtowania się matematyki informacyjnej. Powstaje ona także i na innej drodze — poprzez przekształcanie się matematyki klasycznej w informatyczną drogą eliminacji pewnych metod i wprowadzania innych. Pewne metody, uznawane za przydatne w „epoce przedmaszynowej" stają się bezwartościowe przy zastosowaniu maszyn; metody ciągłe zastępuje się metodami dyskretnymi; tablice matematyczne zamiast formy książkowej przybierają postać programów, zapisanych w pamięciach zewnętrznych maszyn cyfrowych. Warto zwrócić uwagę na pewien fakt, który stawia obecnie pod znakiem zapytania kwalifikację pewnych prac jako matematycznych; otóż obok dowodów klasycznej formy twórczości matematyka, powstaje nowa forma — konstrukcje matematyczne, realizowane w praktyce i w ten właśnie sposób uzasadniane. Np. zdefiniowanie przydatnego, wygodnego i konsekwentnego języka zewnętrznego dla maszyny cyfrowej, który wytrzymuje próbę praktyki, lub zdefiniowanie „eleganckiego" systemu operacyjnego nie stanowi klasycznego wyniku matematycznego, wymaga jednak tej samej precyzji myślenia, tego samego wysiłku intelektualnego i stopnia abstrakcji oraz dostarcza tej samej satysfakcji, co niejeden dowód twierdzenia matematycznego.

Charakterystyczne jest wyrobienie matematyczne użytkowników maszyn; wynika ono z narzuconej im przez maszynę konieczności formalnego myślenia i precyzyjnego wysławiania. Posługują się oni na co dzień logiką bardziej rozbudowaną, niż naukowcy z innych dziedzin przyrodniczych. Ludzie związani z maszynami matematycznymi są przygotowani do zrozumienia i zaakceptowania dowolnie abstrakcyjnej i sformalizowanej teorii. Spośród nich właśnie rekrutują się przyszli twórcy matematyki informacyjnej. Stoimy zatem, jak się wydaje, wobec kształtowania się początków matematyki informacyjnej. Zarówno wielorakie problemy, jak i ludzie pracujący przy maszynach, stanowią niezbędny zaczyn umożliwiający jej rozwój. Przyszłość, może nawet najbliższa pokaże, jak dalece przewidywania nasze są słuszne. Nie powinna nas dziwić rezerwa części matematyków wobec prób rozwijania tej nowej dziedziny; jest to fakt znamienny i znany z historii w odniesieniu do różnych działów matematyki, uznawanych dziś za klasyczne.

*

Przedstawione tu poglądy mają niewątpliwie charakter dyskusyjny. Faktem jest jednak, że wpływ maszyn na matematykę jest znaczny. Nie oznacza to oczywiście, że matematyka określana w tym artykule jako klasyczna stanie się przestarzała, czy też nieużyteczna — rozwijać się będzie ona nadal i nie straci swej pozycji wśród nauk ścisłych.

Matematyka, zwana przez nas informacyjną, jeśli powstanie, jak przewidujemy — będzie jednym z działów matematyki, traktowanej jako całość. Pewne jest również, że dla harmonijnego rozwoju informatyki niezbędne jest prowadzenie intensywnych badań nad matematyką w przetwarzaniu informacji i uwzględnienie jej specyfiki w procesie kształcenia.

---

[2] Romuald Marczyński — „Informatyka, czyli maszyny matematyczne i przetwarzanie informacji", „Maszyny Matematyczne" nr 1/1969, str. 1—5.

[3] „Curriculum 68" — Recommendation for Academic Programs in Computer Science. Comm. of the ACM, v. 11 nr 3/1968, str. 151—197.

[4] Mówimy tutaj tylko o tzw. „uniwersalnych" maszynach analogowych takich, jak np. analizatory różniczkowe, a nie o modelach elektronowych pewnych konkretnych zjawisk czy obiektów.

ZDZISŁAW PAWLAK
Warszawa

519.004.14:517.004.14:681.322.0011

UWAGI O TEORII MASZYN CYFROWYCH

Autor przedstawia pewną próbę sprecyzowania pojęcia maszyny cyfrowej. Omawia ogólne pojęcia maszyny matematycznej, opierając się na pojęciach jej pamięci (stanów) oraz sterowania (funkcji przejścia). Następnie definiuje maszynę adresową oraz podaje ideę określenia maszyn programowanych.

Matematyczne podstawy maszyn cyfrowych — tak jak je rozumiem — powinny służyć do rozwiązywania i badania problemów związanych z konstrukcją i zastosowaniami współczesnych maszyn matematycznych. Aby podstawy maszyn matematycznych mogły służyć do realizowania takich właśnie celów, konieczne jest przede wszystkim sprecyzowanie podstawowych pojęć nauki o tych maszynach takich, jak np. samo pojęcie maszyny, program etc. Jeżeli zgodzimy się z takim określeniem teorii maszyn matematycznych, łatwo w konsekwencji przyznać, że jak dotąd teorii takiej właściwie nie ma. W ostatnich latach pojawiło się kilka prac, które można by zaliczyć do tak rozumianej teorii maszyn matematycznych — prace te dotyczą głównie sprecyzowania pojęcia programu — jednakże sporą liczbę tych prac cechuje brak wyraźnej myśli przewodniej: nie wiadomo, czemu mianowicie mają one służyć. Czy mają to być prace o wyraźnie teoretycznym charakterze, interesujące z matematycznego punktu widzenia, czy też prace mające jakieś znaczenie dla ludzi parających się maszynami. Są to jednakże dopiero początki i na pewno w ciągu najbliższych kilku lat sprawa zostanie rozwiązana całkowicie. Dlatego też uważam, że należy na próby te patrzeć przychylnie — nawet gdyby zawierały one pewne luki.

Znacznie gorzej przedstawia się sprawa z pojęciem maszyny cyfrowej. Tutaj można znaleźć może dwie, trzy publikacje zasługujące na uwagę. Mają one jednak tę wadę, że w ramach przyjętych w nich pojęć nie można nawet zdefiniować najprostszej maszyny, np. jednoadresowej. Trudno więc mówić tu o podstawach matematycznych maszyn cyfrowych. Ukazuje się wprawdzie bardzo dużo publikacji na temat maszyn Turinga, automatów skończonych, etc., ale ktoś, kto miał do czynienia z rzeczywistymi maszynami cyfrowymi niełatwo da się przekonać, że są to dobre modele do badania interesujących go problemów, związanych z prawdziwymi maszynami czy programami. Dlatego też, aby można było z sensem mówić o teorii maszyn cyfrowych należy przede wszystkim sprecyzować pojęcia maszyny cyfrowej i programu w ten sposób, żeby definicje te dawały możliwość formułowania i rozwiązania interesujących nas zagadnień.

W artykule tym chciałem krótko przedstawić pewną próbę sprecyzowania pojęcia maszyny cyfrowej. Podana definicja pozwala z niezłym przybliżeniem opisać istniejące maszyny cyfrowe (ściślej — ich organizację) oraz dowodzić różne twierdzenia dotyczące organizacji maszyn. Twierdzeń tych jednak nie będziemy tu podawać. Są one przygotowywane do druku i zainteresowany Czytelnik znajdzie je gdzie indziej.

Doskonale zdaję sobie sprawę z niedostatków proponowanej definicji. Wielu problemów nie da się w niej nie tylko rozwiązać ale nawet sformułować. (Np. zagadnienie przerywania programów w maszynach wieloprogramowych). Tym niemniej uważam, że tego rodzaju próby są niezbędne do stworzenia matematycznych podstaw maszyn cyfrowych.

W części 1 podano ogólne pojęcie maszyny matematycznej. Samo to pojęcie nie jest może zbyt ciekawe, jednakże wprowadzenia takiego ogólnego pojęcia maszyny wydaje się celowe z dwu powodów. Po pierwsze, przemawiają za tym względy dydaktyczne. Warto zdać sobie sprawę z tego, jak rozumiemy w najogólniejszych zarysach pojęcie maszyny. Pojęcie to powinno być takie, aby wszelkie szczególne przypadki maszyn były w nim zawarte. Łatwiej wtedy zadawać sensowne pytania odnośnie do maszyn matematycznych oraz łatwiej wyjaśnić działanie konkretnych maszyn na takim ogólnym modelu. Po drugie, przy badaniu konkretnych maszyn matematycznych zachodzi często potrzeba dowodzenia pomocniczych twierdzeń o tych maszynach, dość ogólnej natury. Dowody tych twierdzeń dla szczególnych przypadków maszyn zajmują po kilka stron. Natomiast jeżeli te same twierdzenia sformułować dla przypadku ogólnego maszyny, to ich dowody skracają się do kilku wierszy maszynopisu.

Część 2 zawiera pojęcie maszyny adresowej. Maszyny te działają według raz na zawsze ustalonego programu (zrealizowanego np. w postaci tablicy połączeń). Historycznie są to więc maszyny sprzed roku 1945, zanim J. Neuman wprowadził koncepcję maszyny z pamiętanym i modyfikowanym w trakcie liczenia programem. Maszyny te są omówione w 3 części zatytułowanej: maszyny programowane. Można pokazać (czego nie czynimy w tym artykule), że maszyny J. Neumana istotnie różnią się od maszyn, które nie posiadają możliwości modyfikowania instrukcji.

Ostatnia wreszcie część 4 zawiera definicję maszyny uniwersalnej dla pewnej klasy maszyn. Ściślejsze zbadanie pojęcia maszyny uniwersalnej pozwala na lepsze zrozumienie istoty pomysłu J. Neumana — maszyny programowanej.

1. Ogólne pojęcie maszyny matematycznej

Niech T będzie pewnym zbiorem (skończonym albo nieskończonym). Natura elementów tego zbioru chwilowo nas nie interesuje. Zbiór ten będziemy nazywać pamięcią, a elementy tego zbioru stanami pamięci i będziemy je oznaczać małymi literami t ewentualnie ze wskaźnikami u dołu. Dalej niech π oznacza funkcję częściową (tzn. nie wszędzie określoną) o argumentach należących do zbioru T i przyjmującą wartości ze zbioru T, co zapiszemy $\pi : T \to T$. Procesem obliczenia (albo krótko procesem lub obliczeniem) będziemy nazywali ciąg

$$t_0, t_1 \ldots \tag{1}$$

jeżeli spełnia on następujące warunki:

1°. Dla każdego i, $t_i \varepsilon T$, tj. elementami ciągu (1) są stany ustalonej pamięci.

2°. Dla każdego i, $t_{i+1} = \pi(t_i)$.

Obliczenie $t_0, t_1, \ldots, t_k$ nazwiemy skończonym wtedy i tylko wtedy, gdy $t_k \varepsilon D\pi$, gdzie $D\pi$ oznacza dziedzinę funkcji, tj. zbiór tych wszystkich elementów zbioru T, dla których funkcja π jest określona. Obliczenie polega więc na tym, że według ustalonej reguły (funkcji π) przechodzimy od jednego stanu pamięci do następnego, tak długo, jak jest to możliwe. Jeżeli natrafimy na taki stan pamięci, że funkcja π nie mówi nam już, co należy dalej robić (tzn. do jakiego następnego stanu pamięci należy przejść), to obliczenie jest zakończone; jeżeli na taki stan nie natrafimy, to obliczenie będzie trwało nieskończenie długo i nigdy w naszym postępowaniu się nie zatrzymamy. Przyjęta definicja obliczenia w ogólnym zarysie oddaje istotę wszelkich obliczeń, a ściślej mówiąc dowolnych manipulacji na symbolach [1]).

[1]) W podobny sposób może być przedstawiony też proces produkcyjny, ale taką interpretacją nie będziemy się zajmowali w tym opracowaniu. Zainteresowanego tym tematem Czytelnika odsyłam do książki Z. Pawlaka „Matematyczne aspekty organizacji produkcji", Państwowe Wydawnictwo Ekonomiczne, 1969.

Mając definicję obliczenia, możemy wprowadzić pojęcie maszyny. Wprowadźmy najpierw relację M między stanami pamięci w następujący sposób: powiemy, że dwa stany t, t' pamięci T są w relacji M wtedy i tylko wtedy, gdy istnieje skończone obliczenie $t_0, ..., t_k$, takie, że $t_0 = t$ oraz $t_k = t'$, tj. obliczenia zaczynające się od stanu t i kończące się w stanie t'. Okazuje się, że przy tak zdefiniowanej relacji M dla każdego t istnieje co najwyżej jedno t' (może też wcale nie istnieć) będące w relacji M z t. A więc stan początkowy obliczenia jednoznacznie wyznacza stan końcowy obliczenia (o ile oczywiście stan taki istnieje). Możemy mówić więc o funkcji M przyporządkowującej stanom pamięci inne stany pamięci, w ten sposób, że jako argument funkcji M bierzemy dowolny stan pamięci T, następnie wykonujemy obliczenie, przechodząc do kolejnych stanów za pomocą funkcji π, tak długo, aż otrzymamy stan końcowy (o ile to jest możliwe). Ten stan końcowy nazywamy wartością funkcji M. Funkcja M jest więc nie zawsze określona. Dla pewnych stanów t wartość funkcji $M(t)$ może być określona, dla innych zaś stanów pamięci wartość funkcji $M(t)$ może być nieokreślona. Funkcję M będziemy nazywać *maszyną*. Nie jest to sprzeczne z intuicyjnym rozumieniem maszyny matematycznej.

Każdą maszynę matematyczną można sobie wyobrazić jako urządzenie, które posiada jakieś stany (pamięć T); jeżeli urządzenie to ustawimy na początku w jakimś stanie, to przechodzi ono w ustalony sposób do następnego stanu itd., i albo zmienia ono stany w sposób nieskończony, albo też może się przy jakimś stanie zatrzymać. T będziemy nazywać pamięcią maszyny M, zaś π — funkcją przejścia albo sterowaniem maszyny M. Dla określenia więc jakiejkolwiek maszyny musimy podać więc jej pamięć T oraz sterowanie π. Ten sposób opisywania maszyn nie odbiega od metod stosowanych w praktyce. Nawet prospekt handlowy dowolnej maszyny, czy też jej opis techniczny zawierają przede wszystkim opis pamięci oraz listę rozkazów, która, jak się przekonamy, jest niczym innym jak właśnie funkcją przejścia π, dla maszyny [2].

Dla pełnego zrozumienia działania maszyny musimy jeszcze określić, co to znaczy, że maszyna oblicza wartości jakiejś funkcji. Załóżmy, że mamy pewną maszynę M i chcemy za jej pomocą obliczać wartości funkcji $f: X \to X$. (Dla uproszczenia przyjmujemy tu funkcję jednoargumentową. Uwzględnienie funkcji wieloargumentowych nie przedstawia trudności). Zgodnie z intuicyjnym rozumieniem obliczania wartości funkcji za pomocą maszyny można przyjąć następującą definicję:

Maszyna M oblicza wartość funkcji f wtedy i tylko wtedy, gdy dla każdego $x \varepsilon \varkappa, f(x) = \delta\{M[\varkappa(x)]\}$, gdzie $\varkappa$ oznaczają odpowiednio funkcję kodującą i dekodującą: funkcja kodująca $\varkappa$ przyporządkowuje argumentom funkcji f stany pamięci maszyny M, zaś funkcja δ (dekodująca) odwrotnie — stanom pamięci maszyny M — wartości funkcji f.

Obliczanie wartości jakiejś funkcji przez maszynę polega więc na odpowiednim „zapisaniu" (czy przedstawieniu) argumentu x — dla którego chcemy obliczyć wartość funkcji f — w pamięci maszyny, tj. przyporządkowaniu zadanemu argumentowi odpowiedniego stanu pamięci maszyny, następnie na uruchomieniu maszyny i jeżeli maszyna się zatrzyma — na odpowiednim zinterpretowaniu (poprzez dekodowanie δ) stanu końcowego pamięci. Oczywiście wymagamy tu, aby przy ustalonej funkcji f sposób kodowania i dekodowania był taki sam dla wszystkich argumentów funkcji f. Wymagamy więc, aby metoda interpretowania działania maszyny była jednolita dla wszystkich możliwych wartości argumentów — inaczej trudno byłoby bowiem mówić sensownie o obliczaniu wartości funkcji f.

Zwróćmy tu przy okazji uwagę na dwie sprawy. Przy podanym sposobie rozumienia liczenia funkcji f, maszyna wykonuje tylko pewną część pracy: znaczną część pracy wykonuje również jej użytkownik przez ustalenie kodowania i dekodowania. Zależnie od tego, jak praca związana z obliczeniem wartości funkcji f jest podzielona między funkcje $\varkappa$, δ i M, więcej pracy może przypadać na obsługującego, bądź też na maszynę. Druga uwaga dotyczy tego, że właściwie każda maszyna liczy w istocie tylko jedną (!) funkcję M. To, że możemy za pomocą tej samej maszyny obliczać wiele różnych funkcji, polega na tym, że możemy różnie interpretować działania maszyny poprzez dobór odpowiednio funkcji kodującej i dekodującej. Mówiąc ściślej, każdą funkcję, której wartość chcemy liczyć, przedstawiamy jako złożenie trzech funkcji $\varkappa$, δ, M, z których tylko jedna jest liczona przez maszynę — a mianowicie funkcja M. Przy ustalonej funkcji M dobierając odpowiednio funkcję kodującą i dekodującą możemy więc otrzymywać różne funkcje. Gdybyśmy wzięli np. za M funkcję tożsamościową, cała praca związana z obliczeniem wartości funkcji f spadałaby na nas, poprzez odpowiedni dobór kodowania i dekodowania. Mielibyśmy więc wtedy do czynienia ze zwykłym rachunkiem za pomocą papieru i ołówka.

Warto może jeszcze zwrócić uwagę na fakt, że f niekoniecznie może oznaczać funkcję liczbową. Jako zbiór argumentów funkcji f możemy przyjąć np. zdania w jednym języku, funkcje f traktować jako algorytm tłumaczenia zdań w jednym języku na zdania w innym języku — wtedy wartością funkcji f będzie zdanie przetłumaczone. Tłumaczenie z języka na język za pomocą maszyn polega więc na tym, że umiemy znaleźć taki sposób interpretowania działania maszyny, że dla pewnych zdań (niekoniecznie wszystkich w danym języku) interpretowanych jako stan początkowy pamięci, maszyna się zatrzyma i otrzymany stan końcowy przy tej właśnie interpretacji jest zdaniem przetłumaczonym. Na tym właśnie polega fakt, że maszyny cyfrowe, budowane pierwotnie do obliczeń numerycznych, mogą być stosowane do spraw, które z rachunkiem nie mają wiele wspólnego.

2. Maszyny adresowe

Jak to stwierdziliśmy w poprzednim rozdziale, określenie każdej maszyny sprowadza się do podania jej pamięci oraz sterowania. Również maszyny, którymi zajmujemy się w tym rozdziale określimy w ten właśnie sposób.

Najpierw zdefiniujemy pamięć maszyn adresowych. Zanim przystąpimy do opisania pamięci, podamy najpierw kilka pojęć pomocniczych.

Niech A będzie zbiorem skończonym albo nieskończonym (przeliczalnym); elementy A będziemy nazywali adresami. W dalszym ciągu przyjmiemy, że A jest zbiorem liczb naturalnych 0, 1, 2, 3, ... W przypadku gdy będziemy czynili odstępstwo od tego założenia wyraźnie to zaznaczymy.

Przez Σ będziemy oznaczali skończony albo nieskończony zbiór (przeliczalny), którego elementy nazwiemy symbolami, zaś sam zbiór Σ — alfabetem. Przyjmiemy, że alfabet również jest zbiorem liczb naturalnych 0, 1, 2, 3, ... W przypadku odstąpienia od tego założenia wyraźnie to zaznaczymy. Będziemy rozpatrywać funkcje $c: A \to \Sigma$. Każdą taką funkcję będziemy nazywać zawartością pamięci. Przyjmiemy, że dowolna zawartość jest zawsze określona dla skończonej liczby wartości argumentów (adresów). Zawartość pamięci możemy więc traktować jako tabelkę, jak to pokazano np. niżej:

adres	0	1	2	3	4	5	6	7	...
symbol			2	0		3			

[2] Nie podajemy tu odpowiednika wejścia i wyjścia maszyny. Można je w razie potrzeby również uwzględnić. Jednakże do celów, którymi będziemy zajmować się w tej notatce, uwzględnienie wejścia i wyjścia maszyny nie jest konieczne.

Należy to rozumieć w ten sposób, że pod pewnymi adresami zapisane są symbole ustalonego alfabetu, przy czym zapisana jest ich zawsze skończona liczba, natomiast dla pozostałych adresów funkcja zawartości jest nieokreślona. Zbiór wszystkich możliwych zawartości maszyny będziemy nazywać *pamięcią zmienną* maszyny i będziemy ją oznaczać przez C [3].

Określimy obecnie pamięć stałą maszyny wprowadzając uprzednio kilka potrzebnych do tego celu pojęć.

Schematem instrukcji będziemy nazywali funkcję postaci:

$$r : A^m \times C \to C, \quad m = 0, 1, 2, \ldots$$

Jeżeli w schemacie instrukcji ustalimy adresy (tzn. adresy traktujemy jako parametry), to tak otrzymana funkcję nazwiemy instrukcją m-adresową (m jest ustaloną liczbą naturalną dla danej instrukcji, $m = 0, 1, 2, 3, \ldots$). Instrukcją jest więc funkcja

$$r(\overline{a}_m) : C \to C, \quad \overline{a}_m = \{a_1, \ldots a_m\},$$

która jedną zawartość pamięci zmiennej przeprowadza w inną zawartość tejże pamięci.

Np. instrukcje dodawania w maszynie trójadresowej możemy rozumieć jako funkcję następującą:

$$[+(a_1, a_2, a_3)](c) = c',$$

gdzie

$$c'(x) = \begin{cases} c(\overline{a}_1) + c(a_2) & \text{gdy } x = a_3, \ x \varepsilon A \\ c(x) & \text{gdy } x \neq a_3. \end{cases}$$

Niech $Q = \{q_0, q_1, \ldots, q_l\}$ będzie pewnym skończonym zbiorem, zaś s funkcją, która każdemu elementowi zbioru Q przyporządkowuje instrukcję. Zbiór Q możemy traktować jako zbiór adresów pamięci stałej, natomiast s jako zawartość tej pamięci.

W pamięci stałej oznaczanej dalej przez S istnieje tylko jedna funkcja zawartości s raz na zawsze ustalona dla danej pamięci. Pod adresami Q są więc tu zapisane instrukcje, które powodują zmianę zawartości pamięci zmiennej. Pamięć maszyny adresowej składać się więc będzie z dwu pamięci — pamięci zmiennej C oraz pamięci stałej S, co zapiszemy $T = C \times S$. Ponieważ w pamięci stałej istnieje tylko jedna funkcja zawartości s i jest ona jednoznacznie wyznaczona przez podanie zbioru Q, zamiast więc pisać $T = C \times S$ wygodniej będzie w dalszym ciągu stosować zapis $T = C \times Q$.

Określiliśmy więc pamięć maszyn adresowych. Teraz przystąpimy do zdefiniowania jej sterowania, tj. funkcji przejścia π. Sterowanie jest funkcją przypisującą jednym stanom pamięci inne stany, tj. $\pi : T \to T$. Ponieważ $T = C \times Q$, więc sterowanie ma postać $\pi : T \times Q \to T \times Q$. Sterowanie π dla maszyn adresowych wygodnie jest traktować jako parę funkcji π_Q, π_C, takich, że

$$\pi_Q : C \times Q \to Q,$$
$$\pi_C : C \times Q \to C.$$

Dla dokładniejszego określenia obu tych funkcji należy wprowadzić kilka pojęć pomocniczych.

Warunkiem w pamięci zmiennej, albo krótko *warunkiem*, będziemy nazywali dowolny, ustalony podzbiór W pamięci zmiennej. Będziemy mówili, że stan pamięci zmiennej c spełnia warunek W wtedy i tylko wtedy, gdy $c \varepsilon W$; w przypadku przeciwnym powiemy, że stan c nie spełnia warunku W [4].

Np. w maszynie jednoadresowej często spotykanym warunkiem jest badanie, czy zawartość akumulatora jest równa zeru, czy też nie. Tzn. warunek ten w podanej tu terminologii wyrazimy w ten sposób, że W jest zbiorem tych wszystkich stanów pamięci zmiennej, dla których zawartość akumulatora jest równa zeru.

Wprowadzimy jeszcze dwie funkcje pomocnicze φ_1, φ_2 przyporządkowujące elementom zbioru Q elementy tego samego zbioru. Możemy teraz już określić sterowanie w następujący sposób:

$$\pi_C(c, q) = c',$$

gdzie

$$c' = [s(q)](c)$$

zaś

$$\pi_Q(c, q) = \begin{cases} \varphi_1(q), & \text{gdy } c \varepsilon W, \\ \varphi_2(q), & \text{gdy } c \overline{\varepsilon} W. \end{cases}$$

Sterowanie opisuje więc jednoznacznie działanie maszyny. Gdy zadamy jej stan początkowy c, q, sterowanie powoduje wykonanie instrukcji przyporządkowanej q, tzn. zmienia zawartość pamięci zmiennej C, a następnie sprawdza, czy otrzymana zawartość spełnia warunek W, czy też nie — i zależnie od tego przechodzi do wykonania następnej instrukcji z pamięci stałej. Jeżeli funkcje φ_1 i φ_2 przyporządkowują jakiemuś q tę samą wartość q', to znaczy, że wykonanie instrukcji następnej nie zależy faktycznie od spełnienia warunku W. Mówimy wtedy, że przejście do następnego stanu jest *bezwarunkowe*; w przypadku przeciwnym mówimy o przejściu *warunkowym*. Nie powiedzieliśmy jeszcze, w jaki sposób maszyna się zatrzymuje. W tym celu najwygodniej jest wprowadzić instrukcje Stop. Możemy wprowadzić dwa rodzaje takich instrukcji: instrukcje stopu statycznego lub instrukcje stopu dynamicznego. Instrukcje te określimy następująco:

$$\text{Stop}_s(c) = \text{nie określone dla dowolnych } c,$$

natomiast

$$\text{Stop}_d(c) = c \text{ dla dowolnych } c.$$

W przypadku pierwszego stopu instrukcja jest nieokreślona, a więc również funkcja przejścia jest nieokreślona i maszyna zakończy obliczenie przy natrafieniu na taką instrukcję; w drugim natomiast przypadku obliczenie będzie nieskończone, jednakże od pewnego miejsca stany pamięci przestaną się zmieniać. Te nie zmieniające się stany możemy też uważać za stany końcowe (co wszakże nie jest zgodne z definicją obliczenia skończonego podaną w pierwszym rozdziale).

3. Maszyny programowane

Definicja maszyny programowanej jest żmudniejsza, niż maszyny adresowej, dlatego nie będziemy jej przytaczać dokładnie, a podamy tylko ogólną ideę takich maszyn.

Maszyny programowane są szczególnym przypadkiem maszyn adresowych. Pamięć ich składa się również z dwu pamięci, pamięci zmiennej oraz pamięci stałej. O ile w maszynach adresowych nie nakładaliśmy szczególnych warunków na pamięci, to dla maszyn programowanych pamięci te muszą posiadać dodatkowe własności. Odnośnie do pamięci zmiennej wymagamy mianowicie, aby wśród jej adresów znajdował się adres wyróżniony, który oznaczać będziemy literą l, a nazywać go będziemy licznikiem rozkazów. Pamięć stała też jest szczególnej budowy. Nie będziemy jej dokładnie omawiać, a podamy tylko w zarysie najważniejsze jej cechy. W zbiorze Q wyróżnimy element początkowy i oznaczymy go przez q_0. Każde obliczenie, takie że $c_0, q_0, c_1, q_1 \ldots c_k, q_k$, nazwiemy cyklem pracy maszyny programowanej, zaś ciąg instrukcji $s(q_0), s(q_1), s(q_2), \ldots, s(q_{k-1})$ nazwiemy makroinstrukcją maszyny programowanej [5].

Każda makroinstrukcja jest też oczywiście funkcją o argumentach należących do C i wartościach należących również do C. Przyjmiemy ponadto, że każda

[3] Czasem wygodnie jest rozpatrywać pamięć zmienną jako podzbiór wszystkich możliwych zawartości. Jednakże dla rozpatrywanych tu celów, pierwsze określenie pamięci jest wystarczające.

[4] Wprowadziliśmy tu tylko jeden warunek. Czasem wygodnie jest wprowadzić więcej niż jeden warunek w pamięci i badać spełnienie jednego z nich.

[5] Czasem zamiast makroinstrukcja używany jest termin instrukcja, natomiast zamiast instrukcja mówimy mikroinstrukcja.

makroinstrukcja zmienia co najmniej zawartość licznika rozkazów w pamięci zmiennej. Przyjmiemy dalej, że każdej makroinstrukcji odpowiada symbol alfabetu pamieci zmiennej; odwzorowanie to oznaczymy literą Ψ. Funkcję przejścia (sterowanie) dla maszyny programowanej możemy wtedy określić w sposób następujący:

$$c' = \{\Psi\,[c\,(c\,(l))]\}\,(c)$$

Wykonanie każdej makroinstrukcji polega więc na odczytaniu symbolu zapisanego w miejscu pamięci zmiennej wskazywanym przez aktualną zawartość licznika rozkazów, następnie symbol ten jest interpretowany jako ciąg instrukcji zapisanych w pamięci stałej, zaczynajacy się i kończacy w miejscu wyróżnionym q_0. Łatwo zauważyć, że tak określona maszyna programowana jest jednoznacznie wyznaczona przez podanie jej pamięci oraz zbioru wszystkich jej makroinstrukcji (ściślej: schematów makroinstrukcji). Zbiór makroinstrukcji maszyny programowanej nazywamy jej *listą rozkazów*. Zauważmy różnicę między maszynami programowanymi a maszynami adresowymi. Maszyna adresowa nie była wyznaczona przez jej pamięci i listy instrukcji. Działanie maszyny adresowej zależało jeszcze od tego, jaka była zawartość pamięci stałej. Podobnie jest w istocie w maszynach programowanych, jednakże zawartość pamięci stałej jest jednoznacznie wyznaczona poprzez listę makroinstrukcji, a więc lista ta determinuje działanie maszyny we wszsytkich możliwych przypadkach.

Uściślenie podanej definicji nie przedstawia trudności i w ten sposób możemy dokładnie określić pojęcie maszyny programowanej, programu wewnętrznego maszyny, etc.

Pozwala to w konsekwencji stawiać i rozwiązywać różne problemy związane z maszynami programowanymi, jak np. jakie należy dobrać instrukcje, aby można z nich złożyć zadana liste makroinstrukcji, czy też pytanie odwrotne: jaki zbiór makroinstrukcji możemy otrzymać przy zadanym zbiorze instrukcji. Możemy też sformułować różne pojęcia równoważności maszyn i badań, kiedy dwie maszyny czy klasy maszyn są równoważne.

4. Maszyny uniwersalne

W celu lepszego zrozumienia pojęcia współczesnych maszyn matematycznych wygodnie jest wprowadzić pojęcie maszyny uniwersalnej. Pojęcie to pozwoli dokładniej zrozumieć, czym są w istocie maszyny programowane.

Wprowadzimy najpierw pojęcie naśladowania (symulowania, zawierania) maszyny M' przez maszynę M. Powiemy, że maszyna M naśladuje maszynę M', wtedy i tylko wtedy, gdy spełnione są następujące dwa warunki:

1. Maszyny M oraz M' posiadają jednakową pamięć [6].
2. Dla każdego obliczenia $t_0, t_1, \ldots, t_k$ w maszynie M istnieje obliczenie $t'_0, t'_1, \ldots, t'_l$ w maszynie M' takie, że $t_0 = t'_0$, $t_k = t'_l$ oraz dla każdego i $(0 \leqslant i \leqslant k)$ istnieje takie j $(i \leqslant j \leqslant l)$, że $t_i = t'_j$.

Naśladowanie maszyny przez maszynę polega na tym, że jeżeli obie maszyny będą miały jednakowe stany początkowe pamieci, to również skończą obliczenie w jednakowych stanach pamięci oraz każdemu stanowi pamięci w obliczeniu maszyny M będzie odpowiadał stan w obliczeniu maszyny M'.

Definicja ta jest naturalna i odpowiada ona technicznemu naśladowaniu działania jednej maszyny przez inna maszynę. Jeżeli maszyna M naśladuje maszyne M', to zapiszemy $M'CM$. Powiemy, że maszyna M jest maszyną uniwersalną dla klasy maszyn K, jeżeli maszyna M może naśladować działanie każdej maszyny należącej do klasy K. Inaczej, M jest maszyną uniwersalną dla klasy K, jeżeli dla każdej maszyny $M \varepsilon K$, $M'CM$ [7].

Określmy teraz, co to jest klasa maszyn adresowych. Powiemy, że dwie maszyny adresowe należą do tej samej klasy, jeżeli posiadają one jednakową pamięć zmienną oraz ich instrukcje należą do ustalonego zbioru instrukcji. Klasa takich maszyn jest więc wyznaczona przez pamięć zmienną oraz zadany zbiór instrukcji. Oczywiście istnieje nieskończenie wiele maszyn w tak określonej klasie maszyn, gdyż każda taka maszyna jest wyznaczona przez pamięć stałą, a tych właśnie pamięci, nawet przy skończonej liczbie instrukcji, jest nieskończenie wiele. Można pokazać, że dla każdej klasy maszyn adresowych istnieje maszyna programowana, która jest maszyną uniwersalna dla tej klasy maszyny i ta maszyna uniwersalna nie należy do tej klasy maszyn adresowych. Inaczej mówiac, programowana maszyna uniwersalna dla klasy maszyn adresowych jest maszyną istotnie różną od maszyn, które naśladuje. Naśladowanie przez tę maszynę uniwersalna dowolnej maszyny adresowej z ustalonej klasy polega na tym, że musimy umieć odpowiednio „zapisać" działanie dowolnej maszyny adresowej w pamięci maszyny uniwersalnej. Ponieważ działanie maszyny adresowej jest jednoznacznie wyznaczone przez zawartość jej pamięci stałej, wystarczy odpowiednio „przenieść" zawartość pamięci stałej maszyny, której działanie chcemy naśladować, do pamieci zmiennej maszyny uniwersalnej. Maszyna uniwersalna musi oczywiście posiadać makroinstrukcje, które pozwolą symulować instrukcje maszyny adresowej. Zawartość stałej pamięci maszyny adresowej gra więc w maszynie programowanej rolę programu!

Jeżeli natomiast zdefiniujemy klasę maszyn programowanych jako klasę maszyn posiadających jednakowa pamieć oraz makroinstrukcje należace do skończonego, ustalonego zbioru schematów instrukcji, to ponieważ każdy zbiór schematów instrukcji jednoznacznie wyznacza maszyne programowana, więc klasa taka może zawierać tylko skończona liczbe maszyn programowanych. Łatwo znów wykazać, że dla każdej klasy maszyn programowanych istnieje programowana maszyna uniwersalna dla tej klasy i maszyna ta należy do tej klasy maszyn. Co więcej, każda maszyna programowana jest maszyną uniwersalną dla pewnej klasy maszyn.

Pojęcie maszyny uniwersalnej dokładnie więc podaje różnice między maszynami adresowanymi i maszynami programowanymi, wprowadzonymi przez von Neumana. Maszyny programowane istotnie się różnią od maszyn adresowych nie tylko z praktycznego punktu widzenia, ale i własności abstrakcyjne tych maszyn są różne.

Tak sobie wyobrażam teorię matematyczną maszyn cyfrowych. Uważam, że przede wszystkim powinna ona sprecyzować pojęcie maszyny cyfrowej, a następnie pozwalać na badanie różnych własności tych maszyn. Oczywiście zależnie od tego, jakie własności maszyn chcemy badać, definicje maszyny cyfrowej mogą być różne, uwzględniające różne cechy maszyn. Podany tu przykład definicji maszyny miał na celu zilustrowanie myśli przewodniej artykułu i nie stanowi oczywiście rozwiazania całego problemu. Tym niemniej nawet w ramach tej definicji można udowodnić szereg interesujących twierdzeń o maszynach cyfrowych.

Jestem przekonany, że w przyszłości teoria maszyn matematycznych nie tylko będzie nadążała za rozwojem technologii, co obecnie niestety nie ma miejsca, ale nawet będzie wyprzedzać postęp technologiczny — wytyczając nowe kierunki rozwoju tej dyscypliny nauki.

---

[6]) Warunek o jednakowych pamięciach nie jest konieczny. Przyjęliśmy go tutaj jedynie dla uproszczenia.

[7]) Pojęcie naśladowania możemy też zastosować do określenia równoważności maszyn (albo równoważności klas maszyn) w następujący sposób. Powiemy, że maszyny M oraz M' są równoważne wtedy i tylko wtedy, gdy M naśladuje M' oraz M' naśladuje M. Zagadnieniem równoważności nie będziemy się tu jednakże zajmowali.

LEON ŁUKASZEWICZ
Instytut Matematyczny PAN
Warszawa

681.3.06

EOL – Język do przetwarzania symboli

EOL jest językiem uniwersalnym, przeznaczonym do przetwarzania symboli, a w szczególności może być pomocny przy pisaniu translatorów dla innych języków automatycznego programowania. EOL pozwala na dowolną w zasadzie postać ciągu znaków wejściowych i wyjściowych, a ponadto jest wyposażony w środki zarówno do przetwarzania niezbyt długich, lecz za to dowolnie złożonych wyrażeń, jak i do operowania na plikach, zawierających dużą ilość informacji. Instrukcje języka EOL mają prostą strukturę, a bardziej złożone operacje mogą być zapisane w postaci procedur, które mogą być też zakodowane w języku maszyny.

EOL jest stosunkowo prostym językiem, służącym do przetwarzania symboli przedstawionych jako ciągi znaków (sznury). Zakres zastosowań EOL-u jest dość szeroki, aczkolwiek język ten jest przeznaczony w szczególności do pisania translatorów, czyli programów tłumaczących z jednego języka programowania na drugi.

Pierwotnym zastosowaniem EOL-u było przetwarzanie wyrażeń arytmetycznych lub logicznych, stąd też pochodzi nazwa języka (*Expression Oriented Language*). Przykładem podobnego zastosowania jest symboliczne różniczkowanie funkcji, zadanej w postaci wyrażenia algebraicznego. Zadanie to nie jest specjalnie trudne, ponieważ reguły różniczkowania funkcji są stosunkowo proste. Nie wymaga ono wielu obliczeń liczbowych, wymaga natomiast pewnej analizy i manipulacji symbolami, wchodzącymi w skład badanego wyrażenia.

Innym przykładem zastosowań języka EOL do manipulacji symbolami może być program wyznaczania funkcji pierwotnej dla całki ze zmienną górną granicą całkowania. Najprostsze byłoby tutaj przedstawienie wyniku różnicy dwóch funkcji pierwotnych. Jednakże reguły znajdowania funkcji pierwotnych są dość złożone i nie zawsze prowadzą do celu, gdyż często funkcje pierwotne nie dają się przedstawić jako wyrażenie skończone, zawierające jedynie funkcje elementarne. Z tego powodu szukany program powinien jedynie rozpocząć swoje działanie od próby rozwiązania problemu przez znalezienie funkcji pierwotnej, stosując przy tym metody manipulacji symbolami. Jeżeli jednak droga ta nie prowadzi do oczekiwanego rezultatu, należy zastosować jedną z metod całkowania numerycznego, przy czym funkcja poszukiwana przedstawiona już będzie w postaci tabelarycznej. W ten sposób ostateczna metoda rozwiązania tego zadania jest ustalona przez sam program, a wysiłek programisty zostaje zredukowany do minimum. W szczególności zaś omawiany program uwalniałby programistę od poszukiwania funkcji pierwotnej w przypadku, kiedy to rozwiązanie tą drogą jest możliwe.

Uogólniając powyższy przykład można powiedzieć, że najczęstszym zadaniem języka wyspecjalizowanego (*problem oriented language*) jest uwolnienie programisty od formułowania procedury, prowadzącej do rozwiązania danego problemu.

Przykładem takiego języka może być język do rozwiązywania układu równań różniczkowych zwyczajnych. W przypadku tym, programista wypisuje zadany układ równań, warunki początkowe, żądaną precyzję obliczeń oraz podaje, w jakiej postaci pragnie otrzymać wynik.

Opisaną specyfikację problemu należy następnie przełożyć na program dla maszyny cyfrowej, zapisany w jednym z języków proceduralnych (np. w języku ALGOL). Należy ułożyć więc pewien translator, który tłumaczy z języka wyspecjalizowanego na język proceduralny i właśnie ułatwienie pisania tego typu translatorów stanowi jedno z ważniejszych zadań języka EOL.

Implementacja języków proceduralnych, takich jak ALGOL, FORTRAN lub COBOL, polega najczęściej na przekładzie programów napisanych z tych językach na programy napisane w języku symbolicznym (*assembly language*), to jest w języku, który stosuje symbole literowe dla kodów operacji i adresów. A więc i tutaj występuje problem przekładu z jednego ciągu symboli na drugi. Ponieważ zaś języki proceduralne używane są na ogół dosyć często, tłumaczenie takie przebiegać powinno bardzo sprawnie. Zostało to szczególnie uwzględnione przy projektowaniu języka EOL.
Język EOL już był opracowany w kilku kolejnych wersjach.

EOL-1 oraz EOL-2 były opracowane koncepcyjnie w latach 1965/66 w Instytucie Maszyn Matematycznych w Warszawie [1].

EOL-2 został zrealizowany na maszynie ZAM-41 przez zespół: Zdzisława Wrotek, Iwona Messner i Jan Walasek [2, 3].

EOL-3 opracowany został w roku 1967 na Uniwersytecie Illinois [4, 5] i zrealizowany w roku 1968 na maszynach IBM 7094 i IBM 360.

Obecnie w trakcie opracowywania jest EOL-4 jako wspólne przedsięwzięcie Uniwersytetu Illinois i Uniwersytetu Colorado. Projekt ten jest obecnie realizowany na maszynach IBM 360 oraz CDC 6400.

Język EOL zawiera wiele idei zapożyczonych od innych języków do manipulacji symbolami, jak np. COMIT czy IPL-V, jednak jako całość jest od nich wszystkich różny.
W niniejszym artykule przedstawiono język EOL bardzo pobieżnie i w wersji EOL-3.

Założenia języka EOL

Następujące założenia zostały przyjęte przy opracowaniu wszystkich wersji języka EOL:

1. Powinien być to język dobrze przystosowany do przetwarzania symboli, a przede wszystkim do najważniejszych zastosowań tego przetwarzania: pisania translatorów oraz manipulacji formułami algebraicznymi.

2. Powinien być koncepcyjnie prosty, łatwy do nauczenia się i realizacji. Raz zrealizowany — powinien być łatwy do przeniesienia z jednej maszyny na drugą.

3. Powinien zawierać operacje niskiego szczebla, a jednocześnie pozwalać użytkownikowi na jego rozszerzanie w dowolnym kierunku, a w szczególności — tworzenie wyrażeń wyższego szczebla.

4. Programy napisane w języku EOL powinny prowadzić do sprawnych programów wynikowych w po-

---

1) Niniejszy artykuł stanowi rozszerzenie komunikatu, wygłoszonego na Sympozjum „Naukowe problemy maszyn matematycznych", Zakopane, październik 1968 r.

równaniu do równoważnych programów, napisanych w innych dostępnych obecnie językach.

Nie jest zadaniem EOL-u pokrycie pełnego zakresu zagadnień w dziedzinie przetwarzania symboli. Przeznaczeniem jego jednak jest być w tej dziedzinie tym, czym jest sprawny język symboliczny oraz system makroinstrukcji dla normalnej maszyny, a mianowicie narzędziem, które użytkownik łatwo może dostosować do swych specyficznych celów.

Maszyna EOL

Język EOL może być opisany jako język programowania hipotetycznej maszyny EOL, przedstawionej na rysunku. Maszyna ta jest z reguły symulowana na normalnej maszynie, jest więc zbudowana jedynie przy użyciu odpowiednich programów. Maszyny binarne i o stałej długości słowa nadają się szczególnie dobrze do symulacji maszyny EOL.

Struktura danych

Maszyna EOL posiada pewną liczbę **wejść** i **wyjść**, oznaczanych przez zmienne

I1, I2, ..., Iα (wejścia)
Q1, Q2, ..., Qβ (wyjścia)

W praktyce, poszczególne wejścia i wyjścia odpowiadają takim urządzeniom, jak czytniki kart, drukarki liniowe, taśmy magnetyczne itp. Liczby α i β są więc zazwyczaj niewielkimi liczbami całkowitymi. Dane wejściowe i wyjściowe tworzą sznury, zbudowane z dowolnych znaków, jak litery, cyfry, operatory arytmetyczne itp.

W celu zanotowania wartości sznurów, występujących na odpowiednich wejściach lub wyjściach, piszemy przykładowo:

I1: ABCD
Q3:RESULT⊔=1⊔;

Pierwszy z tych wierszy wyraża fakt, że na wejściu I1 znajduje się sznur złożony z czterech znaków ABCD.

Drugi wiersz przedstawia sznur znaków, jaki został wypisany na wyjściu Q3, W notacji tej znak ⊔ oznacza pojedynczy odstęp pomiędzy sąsiednimi znakami (spacja).

Dane wejściowe są czytane przez odpowiednie programy EOL-u, znak po znaku, przy czym każdy znak przeczytany zostaje usunięty z sznura wejściowego. Jednocześnie sznury zawierające dowolną ilość znaków wejściowych mogą być składane w większe jednostki, zwane słowami i umieszczone na początku lub na końcu wskazanego stosu. Podobnie, sznury wyjściowe są tworzone przez program i wyprowadzane na zewnątrz przez ich dopisanie na końcu wskazanego sznura wyjściowego.

Przetwarzanie sznurów w maszynie EOL wykonywane jest przede wszystkim przy użyciu pewnej ilości **stosów**, oznaczonych przez zmienne:

S1, S2, ..., Sγ

gdzie γ zależy od stosowanej implementacji. Typowo zawiera się w granicach od 32 do 256.

Stosy są liniowymi listami jednostek, zwanymi **składnikami**, które z kolei są sznurami złożonymi ze znaków i poprzedzonymi przez jeden ze **znaczników** ¯ lub ° lub ^ . Znaczniki te nie wchodzą w skład znaków podstawowych języka EOL, lecz służą jedynie do zaznaczenia struktury listy i typu jej składników. Mają one natomiast swoje odpowiedniki w wewnętrznej reprezentacji danych w maszynie.

Składnik może być typu słowo, liczba lub adres zapisu.

Słowo zbudowane jest z dowolnego sznura znaków, poprzedzonego znacznikiem. **Liczby** składają się z dowolnego sznura cyfr z ewentualnym znakiem algebraicznych, poprzedzonego znacznikiem.

Adresy zapisów reprezentowane są przez pewne dowolne sznury znaków, poprzedzone znacznikiem. Zawartości stosów zaznaczamy, jak w następujących przykładach:

S8: ¯ Y ¯ = ¯ A1 ¯ * ¯ + ¯ B ¯ + ¯ ALFA
S1: ¯ TO ¯ BE ¯ OR ¯ NOT ¯ TO ¯ BE
S4: ¯ X1°3°18 ^ PLACE

Powyższe formuły informują, że:

- zawartością stosu S8 jest ciąg złożony z siedmiu słów, które mogą przedstawić sobą na przykład zapis formuły arytmetycznej.
- Zawartością stosu S1 jest ciąg złożony z sześciu słów, które mogą przedstawić sobą na przykład pewien literacki tekst.
- Zawartością stosu S4 jest ciąg złożony z czterech składników, z których pierwszy jest słowem, dwa następne — liczbami, a ostatni — adresem pewnego zapisu w jednym z używanych plików.

W języku EOL dysponujemy wieloma instrukcjami, pozwalającymi na przetwarzanie zawartości stosów, np.:

a. Przesłanie określonej ilości składników z początkiem jednego stosu na początek lub koniec innego stosu, przy czym pierwotny porządek tych składników może być zachowany lub odwrócony.

b. Złożenie wielu słów w jedno słowo lub rozłożenie jednego słowa na pojedyncze znaki.

c. Testowanie, czy początkowy lub końcowy składnik stosu jest równy innemu słowu.

Przy realizacji języka EOL, stosy są przechowywane w pamięci głównej maszyny w postaci list złożonych z par słów maszyny. Przydział miejsc pamięci na nowe składniki stosów lub odkładanie par słów maszyny dla przyszłego użytku odbywa się dynamicznie przy użyciu techniki „wolnej listy" *(free storage list)*.

Stos Adresów Instrukcji (SAI) służy do przechowywania adresów instrukcji przy wołaniu procedur.

Rejestr logiczny H służy do zapamiętywania wyników testów (prawda lub fałsz).

Pamięć masowa jest reprezentowana w EOL-u za pomocą **plików**, oznaczanych przez zmienne:

P1, P2, ..., Pδ

Zazwyczaj pliki P1 oraz P2 są przechowywane w pamięci wewnętrznej maszyny, natomiast pozostałe odpowiadają takim pamięciom, jak bębny, dyski lub taśmy magnetyczne. Każdy plik przedstawia liniową listę, złożoną z dowolnej ilości zapisów. Zapisy są tworzone jako wartości stosów, dołączone do wybranych plików.

Programy

Instrukcje EOL-u powodują wykonanie pewnych prostych na ogół operacji, jak zmiana zawartości stosu lub zmiana normalnej sekwencji wykonywania programu. Każda instrukcja EOL-u rozpoczyna się od słowa kluczowego takiego, jak CLEAR, TEST lub READ, które określa pewną klasę operacji. Po słowie kluczowym następują nie więcej niż trzy argumenty. Wiele argumentów składa się z litery zaopatrzonej indeksem, np. A1 lub Z3. Litera służy tutaj do bliższego sprecyzowania sposobu wykonania operacji. Indeks określa numer wejścia, stosu, pliku lub wyjścia, na którym należy dokonać tej operacji.

Przy realizacji języka EOL, instrukcje są zapisywane w jednym lub czasami dwóch słowach maszyny (32 bity wystarczają dla większości instrukcji). Słowa te są czytane przez program zwany **EOL-Interpreter**, powodujący wykonanie kolejnych instrukcji języka EOL.

Struktura procedur w języku EOL jest podobna do struktury procedur PL/I, która pozwala na tworzenie bloków wewnętrznych w procedurach oraz niezależną kompilację procedur zewnętrznych tworzących jeden program.

I1: A⊔ +3⊔ +BETA S3:⁻X⁻=

Jako przykład podamy procedurę, której zadaniem jest przeczytanie kolejnego elementu formuły arytmetycznej, zapisane jako sznur znaków na wejściu I1, a następnie dopisanie tego elementu na końcu stosu S3. Za element formuły uważamy operator arytmetyczny, pojedynczy nawias lub dowolny łańcuch, złożony z liter lub cyfr. Wszystkie spacje w sznurze wejściowym są pomijane. Np. niech:

Jeżeli powyżej określona jest wywoływana cztery razy, otrzymujemy:

I1: BETA S3⁻X⁻=⁻A⁻+⁻3⁻+

Procedura ta, zwana ELEMENT, przedstawia się w języku EOL następująco:

```
ELEMENT: PROC
         CLEAR I1, LDR
         TEST I1, LD
         GOPL L1
         READ I1, Z3, 1
         RETURN
     L1: READ I1, Z3, RB
         RETURN
         END
```

W celu wykonania powyższej procedury należy ją wywołać za pomocą instrukcji CALL ELEMENT.
W podanym przykładzie, indeksy występujące po literach są liczbami o ustalonej wartości. Stosując system makrodeklaracji definiowanych przez użytkownika można stosować indeksy symboliczne. Jednocześnie za pomocą tego typu makrodeklaracji można definiować nowe operacje, które zapisane są, podobnie jak instrukcje.

Rozszerzenie języka EOL

EOL jest językiem uniwersalnym, wyposażonym w podstawowy zbiór instrukcji, służących do przetwarzania symboli. Operacje bardziej złożone mogą być określone jako procedury lub za pomocą makrodefinicji. Jednakże czas wykonania tak określonych operacji może być w niektórych przypadkach nadmiernie długi. Np. czas wykonania prostej manipulacji nad pojedynczym znakiem lub bitem jest znacznie wydłużony potrzebą interpretacji każdej instrukcji języka EOL.

Podobnie, korzystając jedynie ze standardowych instrukcji języka EOL, trudno jest nieraz uzyskać sprawne wykorzystanie pamięci wewnętrznej lub zewnętrznej maszyny. Aby więc wprowadzić możliwość usprawnienia programów EOL, wprowadzono możliwość pisania procedur w języku maszyny. Każda z tych procedur może być traktowana jako rozszerzenie języka EOL o nową operację, przystosowaną do rozpatrywanego problemu.

Procedury EOL pisane w języku maszyny są często najpierw pisane i uruchamiane w języku EOL, a następnie przekładane na język maszyny przez programistę, który jest zaznajomiony ze strukturą danych EOL-u oraz z Interpreterem. System taki pozwala na uzyskanie pełnej dokumentacji programu w języku EOL i łatwego przeniesienia tego programu na inną maszynę, wyposażoną w Interpreter EOL-u.

Zastosowanie języka EOL do pisania translatorów

Język EOL stosowany był już w wielu problemach, np. w programach manipulacji formułami lub w programach automatycznego składania tekstów [8], jednakże głównym jego dotychczasowym zastosowaniem było pisanie translatorów. Największy tego typu translator, napisany dla języka COBOL i maszyny ZAM-41 w Instytucie Maszyn Matematycznych w Warszawie, zawiera ponad 10 000 instrukcji języka EOL.

Dotychczasowe doświadczenia w tym zakresie skłaniają do następujących wniosków:

1. Język EOL jest najbardziej przydatny w początkowych, lecz najbardziej złożonych etapach translacji, takich, jak:

- analiza syntaktyczna (rozpoznanie składni) programu źródłowego wraz z analizą błędu,
- konstrukcja tablic identyfikatorów wraz z odpowiadającymi im opisami,
- generacja programu wynikowego w języku przejściowym.

Język przejściowy może mieć instrukcje w postaci makrowołań, które zostają następnie rozwinięte w ciąg instrukcji maszyny na podstawie dostarczonych makrodefinicji.

2. Użycie języka EOL pozostawia programiście pełną swobodę w wyborze algorytmów dla jego translatora. Wynika to stąd, że EOL zaopatrzony jest w środki dla zapisania w sposób prosty większości typowych operacji, używanych w translatorach. W ten sposób osiągnięcie pełnej sprawności programów wynikowych dla danego języka źródłowego nie jest w niczym ograniczone, gdyż sprawność ta zależy przede wszystkim od przyjętych algorytmów translacji, a nie od języka, w którym translator został napisany.

3. Szybkość działania translatora, napisanego wyłącznie w języku EOL, nie jest początkowo zbyt wysoka, dlatego też najbardziej czasochłonne procedury translatora należy zastąpić procedurami napisanymi w języku maszyny. Procedury te są na ogół dość proste, chociaż używane często tak, że ich wymiana nie zmienia poważnie ogólnej logicznej struktury translatora, natomiast przyspiesza znacznie sam proces translacji.

Łącznie, zastosowanie języka EOL ułatwia znacznie pisanie translatorów, chociażby nawet w końcu poważna część translatora napisana była w języku maszyny. Wynika to z faktu, że użycie języka EOL pozwala na systematyczne rozwiązywanie tego problemu i ułatwia stworzenie dobrej dokumentacji translatorów, co usprawnia znacznie współpracę wielu osób nad jednym zagadnieniem.

4. Język EOL może być również pomocny przy projektowaniu nowych języków, gdyż pozwala analizować translator takiego języka w pojęciach bardziej ogólnych, nie wchodząc każdorazowo w szczegóły maszyny.

Reasumując — EOL przedstawia narzędzie, ułatwiające znacznie pisanie translatorów.

Wnioski

EOL-1 jest językiem do manipulacji symboli, stosunkowo prostym, wydajnym, giętkim i łatwym do realizacji. Cechy te uzyskane zostały m.in. przez wykorzystanie następujących własności języka:

a. Niezależne środki do przetwarzania krótkich, lecz złożonych informacji (przy użyciu stosów), oraz do przetwarzania informacji masowych (przy użyciu plików).
b. Prosty format instrukcji, połączony z możliwością określania bardziej złożonych operacji za pomocą procedur, które mogą być napisane w języku EOL lub w języku maszyny.

Doświadczenia zdobyte do tej pory potwierdzają użyteczność EOL-u, w szczególności przy pisaniu translatorów.

BIBLIOGRAFIA

[1] EOL-a symbol manipulation language. Leon Łukaszewicz, The Computer Journal, Vol. 10, nr 1, May 1967, pp. 53—59.

[2] EOL-II, część publikacji „ZAM-41 — oprogramowanie", wydawnictwo Instytutu Maszyn Matematycznych w Warszawie, rok 1968.

[3] Język EOL i jego realizacja na EMC ZAM-41. Z. Wrotek i J. Walasek, I Ogólnokrajowe Sympozjum — „Naukowe problemy maszyn matematycznych", Zakopane 1968.

[4] EOL Report, Leon Łukaszewicz and Jurg Nievergelt, Report nr 241, Department of Computer Science, University of Illinois, Sept. 1,1967.

[5] EOL programming examples: a primer, Leon Łukaszewicz and Jurg Nievergelt, Report nr 242, Department of Computer Science, University of Illinois, Sept. 1, 1967.

[6] Symmetrical Stacks in EOL, Leon Łukaszewicz, The University of Chicago, ICR Quaterly Report, Number 1, 1967.

[7] On simplified implementation of the EOL language. Leon Łukaszewicz, ibid.

[8] „Zastosowanie ZAM-41 w poligrafii". Ryszard Łukaszewicz, „Maszyny Matematyczne" nr 1/1969.

[9] Symbol Manipulation Languages. Paul W. Abrahams, Advances in Computers, Vol. 9, 1968, Krótki opis EOL-u na str. 106—108.

WŁODZIMIERZ A. MOSTOWSKI
Warszawa

681.322.004.14:517.1

Maszyny matematyczne w logice i algebrze

Część I

Autor omawia niektóre próby zastosowania maszyn matematycznych w logice matematycznej. Przedstawia pewną metodę automatycznego dowodzenia twierdzeń rachunku zadań. Podaje też opis działania programów w języku ALGOL, realizujących tę metodę.

I. Maszyny matematyczne a teoretyczne koncepcje podstaw matematyki

Daleko odbiegliśmy od używania elektronicznych maszyn cyfrowych wyłącznie do wykonywania rachunków. Maszyny używane są do sterowania bądź urządzeniami, bądź procesami technologicznymi, czy produkcyjnymi. Wiele czynności czy procesów technicznych lub fizycznych, lub wreszcie gospodarczych modeluje się na maszynach. Rezerwacje miejsc w samolotach optymalizuje się przy użyciu maszyn. Na maszynach matematycznych układa się harmonogramy przedsięwzięć przemysłowych, np. metodą PERT i innymi, jak również gra się w szachy, co jest też rodzajem optymalizacji. Za pomocą maszyn zbiera się i sortuje dane i dokumentację naukową. Prowadzi się także rokujące dużą przyszłość próby nauczania przy użyciu maszyn.
Mówiąc o wykorzystaniu maszyn w pracy naukowej mam na myśli wykonywanie przez człowieka pracy naukowej (a nie obliczeń numerycznych) przy użyciu maszyny matematycznej. Jakie są w tym względzie możliwości i czym są one spowodowane omówimy pokrótce.
Matematycy pierwsi w swoich badaniach nad logiką zwrócili uwagę, że wielkie fragmenty logiki lub matematyki można budować w sposób formalny. Wychodząc od jakichś symboli opisujących rzeczywistość matematyczną i podając reguły przekształcania tych symboli można przeprowadzić dowody wielu twierdzeń jedynie przez formalne, zgodne z pewnymi regułami przekształcanie wyrażeń, nie wnikając w treść tego, co wyrażenia te opisują.

Współczesnym prekursorem takich idei budowania matematyki był Dawid Hilbert. Celem, jaki chciał osiągnąć, było takie ugruntowanie logiki podstaw matematyki, aby wszyscy się mogli na nią zgodzić. Uważał, że przez uzyskanie łatwej (przynajmniej w teorii) powtarzalności wyników matematyki i uproszczenie stosowanych w tym celu środków można ją uwolnić od elementów psychologicznych, pokazać jej niesprzeczność, wyjaśnić jej niezawodność i powszechność.

Formalizacja matematyki jako idea teoretyczna nie została i nie zostanie pewnie nigdy doprowadzona do końca. Okazała się niemożliwa i to nie ze względów praktycznych, a teoretycznych. Wynik Gödla o niepełności bogatszych systemów logicznych poucza, że formalizacja matematyki musiałaby spowodować nadmierne i, jak się wydaje, nie do przyjęcia jej zubożenie. Formalizacja matematyki była pierwszą na wielką skalę i dość skutecznie przeprowadzoną próbą rutynizacji rozważań matematycznych.

Następny interesujący nas krok został dokonany w roku 1937, gdy Alan Turing stworzył pojęcie maszyny, zwanej od jego imienia maszyną Turinga. Maszyna taka jest teoretyczną abstrakcją. Składa się z

urządzenia skończenie stanowego, zwanego głowicą maszyny Turinga, mogącego się poruszać skokowo od kratki do kratki po nieskończenie długiej taśmie, na której wydrukowano skończoną ilość symboli. Głowica, posuwając się po taśmie, czyta symbole i w zależności od swojego stanu wewnętrznego zmienia, bądź zostawia, bądź też wymazuje rozpatrywany symbol i rozpoczyna ruch w prawo, w lewo lub znowu rozpatruje wydrukowany przez siebie symbol, w zależności od czytanego ostatnio symbolu i swojego, nowego stanu wewnętrznego. Urządzenie takie przekształca automatycznie wyrażenie w inne i w zależności od swojej budowy może np. sprawdzać, czy ciąg symboli na taśmie jest poprawnie zapisanym wzorem arytmetycznym lub formułą logiczną. Również może automatycznie wyliczać wynik wzoru, czy też pisać dowód, gdy formuła logiczna jest twierdzeniem, względnie sygnalizować, że formuła napisana na taśmie nie jest twierdzeniem.

Maszyna Turinga to koncepcja miniaturowej maszyny cyfrowej. (Ciekawostka! Można w sposób rozsądny przypuszczać, że Turing rozumiał takie urządzenie jako mechaniczne i składające się z pewnej liczby kółek zębatych).

Aż do pojawienia się rzeczywistych maszyn cyfrowych maszyna Turinga była tylko teoretyczną abstrakcją. Pojęcie to pozwoliło jednak sprecyzować w sposób już absolutnie pozbawiony elementów psychologicznych pojęcie przekształcenia formalnego, dowodu formalnego i efektywności postępowania w sformalizowanych teoriach matematycznych.

Tak samo, jak na maszynie Turinga można dokonywać formalnych przekształceń symboli na maszynach cyfrowych. Przy istnieniu dobrych systemów kodowania i dużej wydajności przekształcania informacji jest możliwe już nie tylko teoretyczne, a również praktyczne użycie elektronicznych maszyn cyfrowych do operacji na symbolach.

Formalny zapis symboli i jego przekształcanie przez maszyny został użyty na szeroką skalę i do potrzeb samych maszyn. Najbardziej typowym przykładem jest translator dowolnego języka maszyny. W rzeczywistości jest to program badający poprawność ciągu symboli formalnych (w tym przypadku program w języku zewnętrznym) i przekształcającym automatycznie ten ciąg symboli w inny (w program w języku wewnętrznym maszyny, tzw. program binarny). Innym przykładem jest język FORMAC, służący bezpośrednim zastosowaniom matematycznym. Jest to program służący m.in. do formalnego różniczkowania wyrażeń, oparty o język FORTRAN.
Zapisujemy np. symbolicznie:

$$\cos^2(3x+2) + \ln \operatorname{tg} x$$

i poprzedzamy odpowiednim hasłem. Maszyna wydrukuje pochodną równą:

$$6 \cos(3x+2) \sin(3x+2) + \operatorname{ctg} x \cdot \frac{1}{\cos^2 x}$$

Podobne programy formalnego różniczkowania istnieją również w innych językach, np. w ALGOL-u. Programy takie są często bardzo użyteczne.

Maszyny Turinga można również symulować na maszynach cyfrowych. Z ciekawszych w tym względzie opracowań należy wymienić język LOGOL. Jest to translator opracowany przez St. Waligórskiego i J. Mysiora [16], [17], dzięki któremu, zapisawszy w odpowiedniej symbolice opis maszyny Turinga lub innego automatu skończonego — uzyskamy program i według niego maszyna cyfrowa wykona to, co powinna wykonać maszyna Turinga (o ile rzeczywiście pojemność pamięci, związana z długością obliczenia nie zostanie przekroczona).

Obecnie nie ma jeszcze mowy o jakimś szerszym zastosowaniu maszyn do pracy naukowej w dziedzinie matematyki. Cała sprawa jest i pewnie długo jeszcze będzie w stadium prób. Próby te mają charakter raz teoretyczny, raz praktyczny. Próby zastosowania maszyn matematycznych do dowodzenia twierdzeń z dziedziny logiki matematycznej i geometrii są niezwykle interesujące z punktu widzenia teoretycznego. Zapoczątkowane zostały przez Hao Wanga i są dość daleko rozwinięte. Objęte one są wspólną nazwą „Automatycznego dowodzenia twierdzeń". Wiele prac z tej dziedziny prowadzi się również w Polsce w IM PAN i na Uniwersytecie Warszawskim.

Choć mniej znane, lecz o donioślejszym znaczeniu praktycznym są próby zastosowania maszyn matematycznych do pracy naukowej w teorii liczb i w algebrze. Badania takie, dotyczące teorii liczb, zostały w interesujący sposób wraz z podkreśleniem ich znaczenia heurystycznego i przyszłościowego omówione w artykule S. Ulama: „Maszyny liczące" w „*Scientific American*", który w tłumaczeniu polskim ukazał się w zbiorze „Matematyka w Świecie Współczesnym", PWN 1966 [8] i dlatego nie będziemy ich w tym artykule omawiać.

Zastosowania w algebrze, choć mające naukowo większą wagę (ze względu na znaczenie tej dyscypliny matematyki) były prowadzone w wielu ośrodkach na świecie.

Z uwagi na osobiste zainteresowania autora, jak i nieznajomość tej problematyki w kręgu osób interesujących się zastosowaniami maszyn matematycznych — próby dotyczące zastosowań maszyn w algebrze zostaną omówione w części drugiej tego artykułu nieco szerzej.

II. Automatyczne dowodzenia twierdzeń

Doświadczenia w tym kierunku zostały zapoczątkowane przez Hao Wanga [5, 6, 7] w latach sześćdziesiątych i uwieńczone od razu dużym sukcesem. Przy użyciu odpowiednich programów dowodzono twierdzeń, z historycznego już, a jednak podstawowego w dziedzinie podstaw matematyki dzieła B. Russela i A. N. Whiteheada „*Principia Matematicae*". Szybka maszyna cyfrowa udowodniła większość twierdzeń, zawartych w tej książce w czasie rzędu godziny, a po odpowiednich modyfikacjach programu ten sam rezultat osiągnięto w 9 minut. Oprócz Wanga prace te prowadzili: Putman [12] i Glimor [4].

W kilka lat później, w Polsce zajmowali się tym zagadnieniem: Ehrenfeucht i Pawlak, którzy prowadzili w IM PAN seminarium na ten temat. Wynikiem seminarium było zainteresowanie się szeregu osób tymi problemami.

Ułożono praktycznie działające programy na maszynę GIER: Salwicki i Dąbrowska-Ludwiczuk [1], korzystając z opracowanej przez Rasiową i Sikorskiego [13] metody dowodzenia twierdzeń rachunku zdań. Pawlak wydał książkę pt. „Automatyczne dowodzenie twierdzeń" [11], poświęconą teoretycznym aspektom tego zagadnienia i stnowiącą niejako poseminaryjny skrypt.

Ehrenfeucht i Orłowska [2] oraz Orłowska [10] zajmowali się metodami automatycznego dowodzenia twierdzeń w logikach wielowartościowych. Podali oni algorytmy do przeprowadzania takich dowodów, opracowane pod kątem możliwości bezpośredniego zaprogramowania ich na maszynę cyfrową.

O ile mi wiadomo, nie próbowano praktycznie układać takiego programu, być może z braku (jeszcze obecnie) potrzeb w tym kierunku. Z innych badań w kierunku logik wielowartościowych należy wymienić podany przez Kubrychta (Toruń) algorytm minimalizacji formuł wielowartościowego rachunku zdań, stanowiący analogię algorytmu Quine'a [14] i McCluskiego [9] dla sieci dwustanowych bezmostkowych. Dla takich sieci, algorytmy na maszynę cyfrową opracował Waligórski [15] oraz w szczególnym przypadku Tomaszewski. O ile mi wiadomo — praca Kubrychta dotychczas nie została opublikowana.

Interesujące będzie przedstawienie tu pokrótce metody, jaką posługujemy się przy realizacji automatycznego dowodzenia twierdzeń rachunku zdań. Pójdziemy za opracowaniem Dąbrowskiej-Ludwiczuk. Formuły rachunku zdań są wyrażeniami zbudowanymi ze zdań najprostszych, zwanych zdaniami ato-

mowymi [1]), przy użyciu spójników zdaniowych takich, jak:

— , negacja (czytamy nie ...)
=> , implikacja (czytamy jeżeli ... to ...)
$\vee$, alternatywa (czytamy ... lub ...)
$\wedge$, koniunkcja (czytamy ... i ...)

Najdogodniejszym sposobem zapisu formuł znanym od dawna logikom, choć mniej czytelnym dla osób przyzwyczajonych do tradycyjnego zapisu nawiasowego jest zapis beznawiasowy Łukasiewicza.

Odczytanie formuły i operowanie na formule jest wtedy dla maszyny naprostsze. Formuły mają postać:

(1)
— p (nieprawda, że p)
=> pq (jeżeli p to q)
$\vee pq$ (p lub q)
$\wedge pq$ (p i q)

Za symbole p i q można wstawiać zarówno zdania atomowe, jak i dowolne inne formuły poprawne.

Np. wyrażenie

(2) => $\vee pq$ => — pq

które można przeczytać

(3) „jeżeli (p lub q) to (jeżeli nie p to q)"

można ją otrzymać z formuł (1) przez wstawienie za zmienne p, q innych formuł poprawnych.
Podobną formułą jest wyrażenie

(4) — => $\wedge pq$ => — pq

które możemy przeczytać „nie (jeżeli p i q to jeżeli nie p to q)"

Wyrażenie

(5) => => $pq \vee prs$

nie jest formułą poprawną, nie ma ono sensu i nie moglibyśmy przeczytać go tak, jak np. wzoru (3).

Formuła

$$f(p, q, ..., r)$$

jest tautologią rachunku zdań (twierdzeniem logiki matematycznej), jeżeli po wstawieniu za $p, q, ..., r$ dowolnych zdań, obojętne prawdziwych czy fałszywych, otrzymujemy zdanie prawdziwe.

Program realizuje następujące operacje:

1. Odczytanie formuły i sprawdzenie, czy jest ona poprawnie zbudowana; gdy nie — sygnalizowany jest błąd.

2. Rozkład formuły lub ciągu formuł na ciąg lub ciągi formuł prostszych według następujących reguł, których jest 7.

R1. Ciąg formuł F, $\vee$, α, β, G zastępujemy ciągiem F, α, β, G

R2. Ciąg formuł F, $\wedge$ α β, G zastępujemy dwoma ciągami formuł F, α, G oraz F, β, G

R3. Ciąg formuł F, => α β, G zastępujemy ciągiem F, —α, β, G

R4. Ciąg formuł F, ——α G zastępujemy ciągiem F, α, G

R5. Ciąg formuł F, —$\vee$ α β, G zastępujemy dwoma ciągami formuł F, —α, G oraz F, —β, G

R6. Ciąg formuł F, —$\wedge$ α β, G zastępujemy ciągiem F, —α, —β, G

R7. Ciąg formuł F, — => α β. G zastępujemy dwoma ciągami F, α, G oraz F, —β, G.

---

1) Zdania atomowe są zdaniami nie zawierającymi spójników zdaniowych.

Na przykład usuwając w formule (2)

=> $\vee pq$ => — pq

pierwszą implikacją => przy użyciu reguły R3 otrzymamy ciąg dwóch formuł

(6) — $\vee pq$, => — pq

Z otrzymanego ciągu, stosując regułę R5 dla usunięcia spójnika alternatywy otrzymamy dwa ciągi formuł

(7) — q, => — pq

(8) — p, => — pq

Ciągi te przekształcamy dalej, usuwając dalsze spójniki, aż do otrzymania ciągów składających się ze zdań atomowych p i q, ewentualnie opatrzonych znakiem — (negacja). Pełny proces kolejnego usuwania spójników podany jest poniżej:

(9) => $\vee pq$ => — pq

1 krok, reguła R3 <:

— $\vee pq$, => — pq

2 krok, reguła R5:

— p, => — pq — q, => — pq

5 krok, reguła R3: 3 krok, reguła R3:

— p, — — p, q — q, — — p, q

6 krok, reguła R4: 4 krok, reguła R4:

— p, p, q — q, p, q

Słowami:

(nie p lub p lub q) i (nie q lub p lub q)

Istnieje dowód, że każdą formułę można rozłożyć w skończonej ilości kroków na ciągi zdań atomowych, przy czym niektóre z nich opatrzone są znakiem — (negacji).

Jeżeli w każdym z uzyskanych ciągów jakieś zdanie atomowe występuje przynajmniej raz zanegowane, a raz nie — to formuła ta jest tautologią rachunku zdań (twierdzeniem logiki matematycznej).

Jeżeli choć w jednym uzyskanym tak ciągu, żadna zmienna nie występuje raz ze znakiem negacji, a raz bez, to formuła nie jest tautologią rachunku zdań.

Np. rozkładając formułę (4) otrzymamy

— => $\wedge$ pq => — pq

1 krok, reguła R7:

$\wedge pq$, — => — pq

2 krok, reguła R7:

—p, —q

(nie p lub q)

3 krok, reguła R2:

p, q

Już po drugim kroku otrzymujemy ciąg złożony tylko z jednej zmiennej —q. Formuła nie jest tautologią (twierdzeniem logiki).

Stwierdzenie prawdziwości czy fałszu formuły przez maszynę powoduje ewentualny druk dowodu. Np. formuła (4) będzie fałszywa, jeżeli za p wstawimy zdanie fałszywe i za q również zdanie fałszywe i wówczas nie ma potrzeby wykonywać dalszych kroków. Poniżej podajemy dowód fałszywości formuły (4), wypisany na monitorze przez maszynę:

1. — => $\wedge pq$ => — pq;

2. [1/R7] $- => - pq$;

3. [2/R7] $- q$; NIEPRAWDA

Podajemy jeszcze dowód formuły (2), która jest tautologią, wydrukowaną przez maszynę.

Po napisaniu na monitorze

$$=> \vee pq => - pq$$

maszyna drukuje w odpowiedzi:

1. $=> \vee pq => \quad - pq$;
2. [1/R3] $- \vee pq, => - pq$;
3. [2/R5] $- q => \quad - pq$;
4. [3/R3] $- q, - - p, q$;
5. [4/R4] $- q, p, q$; PRAWDA

W tym momencie maszyna rozłożyła drugi ciąg powstały z rozkładu formuły (6), tzn. ciąg (7). Numery z lewej strony jedynie numerują wydrukowane wiersze. W maszynie jest zapamiętany numer kroku rozkładu formuły, (podany w schemacie (9)) oraz numer reguły, stosowanej w kolejnym kroku. Następnie maszyna rozkłada ciąg formuł (8).

6. [5/R5] $-p, => - pq$;
7. [6/R3] $-p, - - p, q$;
8. [7/R4] $-p, p, q$; PRAWDA CBDO

Symbol CBDO świadczy, że w przeprowadzonym postępowaniu udowodniliśmy o formule, że jest ona tautologią rachunku zdań.

Jeżeli wyrażenie wprowadzone do maszyny (co dokonuje się przez wypisanie formuły na monitorze) będzie niepoprawne, to maszyna już w pierwszym kroku, nie mogąc zinterpretować formuły, zasygnalizuje błąd.

Np. odpowiedź maszyny na formułę wypisaną na monitorze:

$$=> => pq \vee prs;$$

jest $=> => pq \vee prs$; BLAD (Błąd).

Z punktu widzenia logiki postępowanie maszyny polega w istocie na przekształcaniu formuły na równoważną (tzw. koniunkcyjno-alternatywną) postać normalną, zawierającą koniunkcję alternatyw zmiennych, zanegowanych lub nie.

Formuła (3) jest równoważna formule

(nie q lub nie p lub q) i (nie p lub p lub q)

zaś formuła (4) jest równoważna formule

(p lub q) i (nie p lub nie q)

Wyciągnięte przez nas wnioski o prawdziwości formuły opierają się na tym, że formuła w takiej postaci jest prawdziwa wtedy i tylko wtedy, gdy każdy człon koniunkcji jest prawdziwy, co ma miejsce w przypadku formuły (3).

Opisane postępowanie zaprogramowane zostało w języku ALGOL na maszynę GIER. Tabulogram programu liczy sobie wprawdzie około 2 m, ale szybkość działania jest ogromna, praktycznie ograniczona jedynie szybkością druku. Jedynie dla specjalnie wybranych formuł tak, aby dowód maszynowy był możliwie najdłuższy i składał się z tysięcy przesłanek, maszyna pracowała nad dowodem rzędu minut, oczywiście nie drukując dowodu. Z opisem programu, jak również ze szczegółowym opisem metody, czytelnik może się zapoznać z pracy [1] Dąbrowskiej-Ludwiczuk) w posiadaniu ZON Uniwersytetu Warszawskiego.

Podobną metodą przeprowadza się dowodzenie twierdzeń elementarnego rachunku kwantyfikatorów, tzn. twierdzeń logiki wyrażonych przez formuły, zawierające już nie zdania, a funkcje zdaniowe, tzn. zdania, których prawdziwość lub fałszywość zależna jest od parametrów, zwanych zmiennymi elementarnymi. Oprócz spójników zdaniowych, formuły buduje się przez stosowanie operacji kwantyfikatora $\forall_x p(x)$

co czytamy: „dla każdego x zachodzi $p(x)$" oraz $\exists_x p(x)$ co czytamy: „istnieje x takie, że zachodzi $p(x)$".
W elementarnym rachunku kwantyfikatorów — kwantyfikatory można stosować tylko do zmiennych elementarnych.

Oprócz tworzenia nowych formuł przez opatrywanie ich znakami kwantyfikatorów, można je budować także przez podstawienie w funkcjach zdaniowych za jedne zmienne wyrażeń zależnych od innych zmiennych (nakładając przy tym na operację podstawiania pewne ograniczenia).

Metoda i programy dowodzenia twierdzeń elementarnego rachunku kwantyfikatorów są znacznie bardziej skomplikowane niż w przypadku rachunku zdań. O ile mi wiadomo — na świecie nie poszło się dotąd w udoskonalaniu tej metody wiele dalej niż w pracach Wanga (patrz op. cit.). Zachodzi tu zresztą istotna różnica w porównaniu z automatycznym dowodzeniem twierdzeń w rachunku zdań. W rachunku zdań, stosując opisaną metodę — musieliśmy zawsze, po pewnej skończonej ilości kroków osiągnąć zamierzony cel — albo udowodnić twierdzenie, albo pokazać, że jest ono fałszywe. Rachunek zdań jest teorią rozstrzygalną i opisana powyżej metoda daje algorytm rozstrzygania, czy dane wyrażenie jest twierdzeniem, czy nie.

Inaczej jest w przypadku dowodzenia twierdzeń w elementarnym rachunku kwantyfikatorów. Puszczając program automatycznego dowodzenia twierdzeń otrzymamy — przynajmniej teoretycznie — zawsze dowód formuły w przypadku, gdy formuła ta jest twierdzeniem. Jeżeli jednak formuła nie jest twierdzeniem, to nie musimy, choć możemy, po skończonej ilości kroków otrzymać od maszyny żądanej informacji. Nie jest to wada używanych metod. Żadną metodą, czy to przy użyciu maszyny czy innych urządzeń typu automatu skończonego, nie można rozstrzygnąć dla dowolnej formuły elementarnego rachunku kwantyfikatorów, czy jest ona twierdzeniem czy nie.

Dla każdego algorytmu, zawierającego operacje finitystyczne, będą istnieć szerokie klasy formuł, dla których algorytm nie da wyniku, ponieważ rachunek kwantyfikatorów jest teorią nierozstrzygalną.
Badania nad automatycznym dowodzeniem twierdzeń wyszły poza dziedzinę twierdzeń logicznych. Gelertner [3] badał możliwości dowodzenia prostszych twierdzeń z geometrii rzutowej przy użyciu maszyn. W Polsce badania takie prowadził L. W. Szczerba. Podał on metody automatycznego dowodzenia prostych twierdzeń geometrii płaskiej, dotyczących relacji porządku na prostej, tzn. leżenia punktu między punktami danymi. Publikacja na ten temat jest w przygotowaniu.

O ile mi wiadomo, programował on swoją metodę na maszynę GIER, nie uzyskał jednak dotąd, mimo zalet metody, sprawnie działającego programu.
Prace nad automatycznym dowodzeniem twierdzeń zrealizowały w takim zakresie, jak to było możliwe, odwieczne marzenia o wnioskowaniu logicznym bez użycia mózgu ludzkiego, a jedynie za pomocą postępowania mechanicznego. Główne znaczenie tych prac tkwi jednak w czymś innym.

Prace nad automatycznym dowodzeniem twierdzeń wpłynęły na znaczną intensyfikację badań nad strukturą dowodów formalnych, ich długością i ilością potrzebnych do wykonania operacji i długością wyrażeń występujących w stadiach poprzednich. Badania te są prowadzone obecnie już nie pod kątem potrzeb badań teoretycznych nad zbudowaniem prostych i niesprzecznych logicznie podstaw matematyki, lecz pod kątem praktycznych możliwości przeprowadzania

przez maszynę różnych operacji formalnych, uwzględniając narzucone parametrami maszyny ograniczenia. Jak dotąd, przy istniejących obecnie maszynach o niewysokich parametrach pojemności pamięci i czasu działania, wnioski są pesymistyczne. Na pewno w nauce nie będzie się jeszcze przez jakiś czas prowadzić praktycznie i masowo procesu dowodzenia i wnioskowania przy użyciu maszyn matematycznych. Trudno jednak powiedzieć, jak będzie w przyszłości.

(cdn.)

BIBLIOGRAFIA

[1] H. Dąbrowska-Ludwiczuk, Program dla automatycznego dowodzenia twierdzeń metodą R-S. Praca magisterska, złożona w Katedrze Algebry Uniwersytetu Warszawskiego, 1966.
[2] A. Ehrenfeucht, E. Orłowska, Mechanical proof procedure for propositional calculus, Bull. Acad. Polon. Sci., ser. mat. fiz. 15/67, 25—30.
[3] H. Gelertner, Realisation of a geometry theorem proving machine, Proc. of the International Conference on Information Processing, UNESCO, Paris 1959.
[4] P. C. Glimor, A proof method for quantification theory, IBM Journal of Research and Development, 4 (1960), 28—35.
[5] Hao Wang, Toward mechanical mathematics, IBM Journal of Research and Development, 4 (1960), 2—22.
[6] Hao Wang, Proving theorem by pattern recognition, I, Communications of ACM, 3 (1960), 220—234.
[7] Hao Wang, Proving theorem by pattern recognition, II, Bell system Technical Journal, 40 (1961), 1/41.
[8] Matematyka w świecie współczesnym, PWN 1966.
[9] McCluskey, Minimazation of Boolean Functions, The Bell System Technical Journal, 1956, 1417—1444.
[10] E. Orłowska, Mechanical Proof Procedure for the n-valued Propositional Calculus, Bull. Acad. Polon. Sci., ser. mat. fiz. 15/67, 537—542.
[11] Z. Pawlak, Automatyczne dowodzenie twierdzeń, PZWS 1965.
[12] D. M. H. Putnam, A computing procedure for quantification theory, Journal of the Association for the Computing Maschinery, 7/60, 201—215.
[13] H. Rasiowa, R. Sikorski, The Mathematics of Metammatics, PWN 1963.
[14] W. V. Quine, The Problem of Simplifying Truth Functions, Amer. Math. Monthly, 59 (1952), 521.
[15] S. Waligórski, Calculations of the Quines table for truth functions, Prace ZAM PAN, praca A15.
[16] J. Mysior, S. Waligórski, LOGOL — język algorytmiczny dla przetwarzania napisów, Biuletyn Informacyjny IMM, Nowości Techniczno-Ekonomiczne, 2 (1965), str. 15—18.
[17] J. Mysior, S. Waligórski, LOGOL. A string Manipulation Language, IFIP, Working Conference on Symbol Manipulation Languages, Pisa 1966.

JANUSZ STANISŁAW BIEŃ
Uniwersytet Warszawski

681.3.06.004.14.801.55:808.4

Algorytmizacja fleksji polskiej – problemy i perspektywy

W artykule przedstawiono niektóre problemy związane z mechaniczną odmianą wyrazów w języku polskim. Szczególną uwagę poświęcono koniugacji czasowników na przykładzie programu „Verb 1", który został uruchomiony przez autora w czerwcu 1968 r. w ramach seminarium z teorii maszyn, prowadzonego przez dra inż. Stanisława Waligórskiego. Dzięki wczesnemu zainteresowaniu się polonistów użyciem maszyn cyfrowych do badań nad językiem, istnieją obecnie — stwierdza autor — dobre podstawy teoretyczne do stworzenia pełnego algorytmu fleksji, pierwszego kroku na drodze do polskiego systemu konwersacyjnego, czyli „maszyny mówiącej po polsku".

Maszyny matematyczne osiągnęły obecnie wysoki poziom doskonałości pod względem szybkości wykonywania działań, niezawodności, sprawności urządzeń wejścia i wyjścia oraz innych cech określanych łącznie terminem *„hardware"*. Także w dziedzinie *„software'u"*, tj. oprogramowania, osiągnięto duży postęp tak, że dzisiejsze maszyny cyfrowe można stosować zarówno do obliczeń numerycznych, jak i przetwarzania danych, wykorzystując przy tym różnorodne języki.

Zdaniem wielu specjalistów, dalszy rozwój maszyn matematycznych charakteryzować się będzie dążeniem do względnie najlepszego dostosowania ich do człowieka tak, aby uczynić z nich wygodne i wszechstronne narzędzia ludzkiej pracy. Potwierdzeniem słuszności tej koncepcji jest rozwój prac nad tzw. systemami konwersacyjnymi, czyli programami umożliwiającymi porozumiewanie się z maszyną przy pomocy języka naturalnego. Zastosowanie ich może być bardzo szerokie. Prosty program tego typu stanowi np. część „laboratorium matematycznego" [1], przeznaczonego dla inżynierów do podręcznych obliczeń. Problemy podaje się maszynie do rozwiązania przy pomocy ustalonych wyrazów lub zwrotów, wyniki natomiast opatrywane są komentarzem w języku naturalnym. Jeśli podczas liczenia okaże się niezbędna dodatkowa informacja, maszyna zwraca się po nią — również w języku naturalnym — do użytkownika. Przykładami skomplikowanych systemów konwersacyjnych są programy z rodziny „ELIZA" [2] (ponieważ każdy kolejny program „mówił" coraz lepiej po angielsku, otrzymały one imię bohaterki „Pigmaliona" — znanej komedii G. B. Shawa). Ostatni z tych programów posiada pokaźną ilość wymiennych scenariuszy *(scripts)*, dzięki którym potrafi m.in. uczyć, egzaminować, przeprowadzać wstępny wywiad w szpitalu lub odgrywać rolę psychoanalityka.

Wspólną cechą dotychczas uruchomionych systemów jest korzystanie z języka angielskiego. Wpłynął na to nie tylko wysoki poziom techniki obliczeniowej w krajach anglosaskich, ale także fakt, że uboga fleksja i stały szyk wyrazów w zdaniu predysponują ten język do maszynowego przetwarzania. Przykładowo zmiana osoby z pierwszej na drugą dowolnego czasownika w czasie *Simple Present* odbywa się przez

proste zastąpienie zaimka osobowego „I" lub „we" przez „you"; nie stwarza więc problemu przekształcenia zdania „I carry" w zdanie „You carry", a utworzenie czasu Simple Past — np. „They carried" — jest niewiele trudniejsze. Polskie tłumaczenie tych zwrotów: „niosę" — „niesie" — „nieśli, niosły" daje przedsmak trudności mechanicznej fleksji w języku polskim.

Na szczęście problemem tym dość wcześnie zainteresowali się językoznawcy. W roku 1958 został wydany pierwszy tom „Słownika języka polskiego" pod redakcją prof. W. Doroszewskiego, w którym „dobór wzorów w tabelach, odsyłaczy do nich i form podawanych przy haśle jest tak ułożony, by korzystający ze Słownika mógł odtworzyć pełny zasób form fleksyjnych danego wyrazu w zakresie uwzględnionym w Słowniku [3] (w 1968 roku ukazał się ostatni, dziesiąty tom tego dzieła opartego na nowoczesnych założeniach metodologicznych). W roku 1961 doc. dr Jan Tokarski na łamach „Poradnika językowego" [4] (notabene miesięcznik wydawany przez redakcję „Słownika języka polskiego") wyraził nadzieje, jakie wiązać można ze statystycznymi badaniami dokumentacji zgromadzonej podczas prac nad Słownikiem oraz przedstawił korzyści dla wiedzy o języku, jakie wynikłyby z realizacji „mechanizmu" tłumaczącego. Sprawom tym poświęcił następnie w latach 1961—1964 obszerny cykl artykułów pod ogólnym tytułem „Fleksja polska, jej opis w świetle możliwości mechanizacji w urządzeniu przekładowym" [5], interesując się zarówno problemem syntezy, tj. konstruowania nowych form w oparciu o formę podstawową, jak i analizy, czyli określenia części mowy i formy fleksyjnej dla danego wyrazu. Obecnie wiadomo, że maszynowe tłumaczenie jest sprawą o wiele trudniejszą niż sądzono początkowo. W przypadku polisemii (posiadania kilku znaczeń przez jeden wyraz) w celu znalezienia odpowiednika w innym języku należy dokładnie zbadać kontekst. Stwarza to dla algorytmu tłumaczącego olbrzymie wymagania, które nie dotyczą systemów konwersacyjnych. Te ostatnie, mając bezpośredni kontakt z użytkownikiem, mogą zażądać innego sformułowania wypowiedzi lub wyjaśnienia znaczenia użytego zwrotu. Oczywiście, większość rozważań polonistów na temat urządzeń tłumaczących pozostaje aktualna; wydaje się więc, że skonstruowanie polskiego odpowiednika „ELIZY" będzie zadaniem realnym w dość bliskiej przyszłości. Z myślą o częściowym sprawdzeniu w praktyce metod wskazanych przez lingwistów opracowano na Uniwersytecie Warszawskim program „Verb 1" — jak

```
VERB

BEGIN

algol<
begin comment
 Program tworzy 17 form prostych czasownikow na podstawie informacji o odmianie - w zasadzie
rownowarnej odpowiedniej czesci hasla w ,,Slowniku jezyka polskiego,, pod red. W.Doroszew-
skiego.Sa to formy:
1.czas terazniejszy (dla niedokonanych) lub przyszly prosty (dla dokonanych),
2.tryb rozkazujacy - tylko 2. os.l.p. i 1.,2. os. l. mn.,
3.imieslow przeszly - wszystkie liczby i rodzaje,
4.imieslowy wspolczesne przymiotnikowy i przyslowkowy (tylko dla niedokonanych).
Informacja potrzebna do poprawnej odmiany:
1.aspekt - ,,1,, dla dokonanych lub ,,2,, dla niedokonanych,
2.grupa - okresla numer wzoru odmiany w tablicy stanowiacej czesc niniejszej tasmy,
3.okreslenie tematow rornych od tematu bezokolicznika przez podanie
  dla wszystkich form cz. ter. - formy 3. os.l.p.
  dla 1. os.l.p. i 3. os.l.mn. cz. ter. - formy 1. os.l.p.
  dla trybu rozk. - formy 2. os.l.p.
  tylko dla grup 7,19 i 20:
  dla wszystkich form imieslowu przeszl. - formy l.p. rodz.mesk.
  j.w. oprocz l.p. r. mesk. - formy l.p. rodz. rensk.,
4.pojedyncze formy o temacie odmiennym od wymienionych wyrej.
Dla grup 8-11,14,19,20 tryb rozkazujacy tworzy sie od tematu 3. os.l.p. cz.ter.,uwzglednia
sie przy tym wymiany typu ,,lapie,,-,,lap,,.
Numer grupy dla danego symbolu odmiany ze Slownika morna okreslic znajdujac jego numer w
ciagu:I,II,III,IV,Va,Vb,Vc,VIa,VIb,VIIa,VIIb,VIIIa,VIIIb,IX,Xa,Xb,Xc (typ -mij),Xc (typ
-nij),XI,bez nr.gr..Ze wzgledu na to,re niektore informacje przyhaslowe w Slowniku sa nie-
wystarczejace,sprawdzenie ich jest zawsze wskazane.Szczegolna uwage nalery zwrocic na
formy trybu rozkazujacego oraz czasowniki grupy 8 i 10.
<dane>::=<inf. o czas.>|<inf. o czas.><dane>
<inf. o czas.>::=<aspekt><grupa><ilosc form><bezokolicznik><pozostale formy><komentarz>
<komentarz>::=<pusty>|<ciag symboli nie zawierajacy cyfry>
Dla danych i wynikow przewidziano transkrypcje stosowana w tym komentarzu.Wyrazy nalery
oddzielac spacjami,CR lub znakami przestankowymi.Dane wprowadzane sa z tasmy.Wyniki wypro-
wadzane sa na drukarke.Jednemu czasownikowi odpowiada 10 wierszy (1/6 strony) po110 znakow.
Realizacja programu:
Po wywolaniu Helpu 3 wybrac ,,r,, jako wejscie i wczytac tasme VERB.Po zgloszeniu sie
programu podac z monitora ilosc odmienianych czasownikow,nastepnie wprowadzic tasme danych.;
boolean p,r;
integer grupa,typ,i,j,k,l,m,n,ilosc,licz;
integer array SLOWO[1:20],ZAK[1:20,1:17,0:7],VERB[1:17,0:20],FORMA[1:25,-1:5];
integer procedure Temat(forma); integer forma;
  begin integer i,j,k,l;
    k:=ZAK[grupa,forma,0];
    j:=l:=VERB[forma,0]mod 20;
```

Rys. 1. Fragment pierwszej części taśmy VERB. Zawiera ona program „Verb 1" oraz instrukcje dla systemu operacyjnego Help 3

```
1.
ac,am,asz,a,amy,acie,aja
aj,ajmy,ajcie,ajacy,ajac
al,ala,alo,ali,aly
2.
ec,em,esz,e,emy,ecie,eja
ej,ejmy,ejcie,ejacy,ejac
al,ala,alo,eli,aly
3.
ec,eje,ejesz,eje,ejemy,ejecie,eja
ej,ejmy,ejcie,ejacy,ejac
al,ala,alo,eli,aly
4.
owac,uje,ujesz,uje,ujemy,ujecie,uja
uj,ujmy,ujcie,ujacy,ujac
owal,owala,owalo,owali,owaly
```

Rys. 2. Fragment drugiej części taśmy VERB. Zawiera tablice zakończeń poszczególnych grup tematowych, które zostają wczytane przed właściwą taśmą danych

```
2,1,1,czytac
1,1,2,dac,dadza
2,3,3,stroic,stroje,strój
1,11,1,uslyszec
2,19,1,rac
2,20,5,jechac,jade,jedzie,jedz,jechal,
1,1,1,poznac
2,2,1,rozumiec
2,3,1,szalec
2,4,1,rysowac
1,5,1,dzwignac
2,8,1,tropic
2,8,3,nosic,nosze,nos
2,8,3,snic,snie,snij
2,9,2,tworzyc,tworz
2,8,2,wodzic,wodz
```

Rys. 3. Taśma danych

mówi sama nazwa, jego zadaniem jest odmiana czasowników. Wybrano właśnie to zagadnienie, gdyż koniugacja jest najbardziej skomplikowanym i chyba najtrudniejszym działem polskiej fleksji; za podstawę pracy przyjęto przepis ze „Słownika języka polskiego" na stworzenie form nie podanych przy haśle. Trzeba tutaj podkreślić, że właściwa definicja czasownika — i w ogóle podział na części mowy — nie jest sprawą prostą [6]. Szkolna zasada, że są to wyrazy „określające czynności i stany przedmiotów" nie jest zadowalająca, bo przecież czynności i stany określają także rzeczowniki, np. „bieg", „lot", „choroba", a poza tym formy złożone („będę pisać", „tyś to zrobił") nie podpadają pod tę definicję. Wprowadzono zatem trafniejsze określenie czasownika jako „klasy wyrazów odmieniających się przez osoby oraz ich pochodnych o żywych i prawidłowych związkach słowotwórczych", a jako kryterium zastosowano warunek seryjności. Wynikiem tego było uznanie za czasowniki tzw. odsłowników („czytanie"), gdyż można je utworzyć od każdego czasownika przy pomocy zakończenia „-anie", „-enie" lub „-cie", i odrzucenie np. wyrazu „stojąco", ponieważ dla większości czasowników nie istnieją formy analogiczne („czytająco", „pisząco"). W przypadkach wątpliwych sprawę musi rozstrzygnąć konwencja, np. formy typu „pożółkło", „wyblakło" Słownik nazywa czasownikami wbrew dotychczasowej tradycji.

Inna sprawa, o której należy pamiętać, to fakt, że każdy algorytm fleksji stanowi całość dopiero razem ze słownikiem, zawierającym odpowiednie informacje o odmianie. W pewnych granicach istnieje zawsze możliwość regulowania ilości tych informacji przyhasłowych dla poszczególnych wyrazów, oczywiście przy równoczesnej zmianie właściwego algorytmu. Można zatem budować różne warianty mechanicznej fleksji o różnej objętości słownika. Najgroźniejszym błędem, z jakim można się spotkać, to niedopasowanie haseł słownika do danego wariantu algorytmu — wymaga bowiem krytycznego podchodzenia do każdej informacji przyhasłowej. Niestety, ilość niekonsekwencji już wykrytych w „Słowniku języka polskiego" skłania do obaw, że tego błędu redakcja Słownika nie uniknęła.

Koniugacja czasowników stanowi lwią część fleksji polskiej; wpływa na to możliwość równoczesnej odmiany przez strony, tryby, czasy, osoby, liczby i rodzaje. Zbiór wszystkich form jednego czasownika, liczący ponad sto elementów, można podzielić ze względu na strukturę na formy proste, aglutynacyjne i złożone. Najciekawsza z nich jest pierwsza grupa — stanowi ona podstawę całej koniugacji, chociaż liczy tylko 22 formy. Dysponując kompletem form prostych można łatwo zbudować dowolną formę złożoną („będę czytał", „niech czyta") lub aglutynacyjną (tj. z ruchomą końcówką: „jam to czytał", „ja to czytałem"). Należy podkreślić, że pomimo prostoty tworzenia użycie nieprostych form w zdaniu jest trudne, wymaga bowiem uwzględnienia skomplikowanych zasad składni; za przykład może służyć „wyłączanie przed nawias" niektórych końcówek („sameś nie przyszedł i brata nie przyprowadził" zamiast „sam nie przyszedłeś i brata nie przyprowadziłeś").

Biorąc pod uwagę te fakty, zarówno redakcja Słownika, jak i autor omawianego programu uznali za słuszne ograniczyć działanie algorytmu do tworzenia tylko form prostych, w których skupiają się wszystkie kłopoty mechanicznej koniugacji. Zasada działania tego algorytmu została dokładnie omówiona przez J. Tokarskiego w jego pracach [7]. Polega ona na podziale czasowników na tzw. grupy tematowe i zbudowaniu dla każdej grupy tablicy zakończeń form prostych. Dzięki temu, że zakończenia obejmują także część tematu, tablice te opisują systematyczne zmiany tematowe, np. „**czytam**" — „**czyta**jący". Dla dużej części czasowników podanie numeru grupy w zupełności określa sposób ich odmiany. Istotnym problemem polskiej koniugacji są jednak te czasowniki, u których wymiany tematu mają charakter cech indywidualnych (wytłumaczenie ich często można znaleźć tylko w historii języka — zademonstrowane

Rys. 4. Wyniki programu dla ciekawszych czasowników

zmiany samogłosek w czasowniku „nieść" wywodzą się z okresu, gdy rozróżniano w wymowie samogłoski długie i krótkie). Dało się jednak i tutaj wykryć pewne prawidłowości, mianowicie zmiany tematu występują przeważnie równocześnie w kilku formach. Pełną informację o tematach form prostych można więc przekazać przy użyciu tylko kilku form nietypowych, które zostają odpowiednio wykorzystane przez odsyłacze użyte w tabelach zakończeń.

Z przykrością trzeba stwierdzić, że opis koniugacji w Słowniku nie został dopracowany do końca. Pomijając nawet oczywisty błąd w odsyłaczu szóstym i niejasne sformułowanie innych odsyłaczy, to szokujący jest fakt, że ścisłe stosowanie się do przepisu nie daje poprawnej odmiany dla 6 z 25 czasowników przytoczonych jako przykład. Część błędów wynikła z niewłaściwego sposobu tworzenia trybu rozkazującego („robić", „wziąć", „żąć", „jechać"), jeden z niejednoznacznego podania formy obocznej („zmoknąć") i jeden z niedostatecznej ilości form nietypowych („robić"). Oczywiście, uwzględniono to podczas przygotowywania algorytmu dla programu.

Program „Verb 1" został napisany w GIER ALGOLU [4] (reprezentacja języka ALGOL 60 dla duńskiej maszyny GIER). Ze względu na kłopoty z pomieszczeniem w pamięci pełnej tablicy zakończeń zrezygnowano z tworzenia siedmiu mniej ciekawych form. Ponieważ informacje przyhasłowe mają dość skomplikowaną strukturę, np.:

dać dk I, dadzą, dany — **dawać** ndk IX, daje, dawaj, -any... wprowadzono kodowanie postaci (dokonany czy niedokonany) i grupy tematowej przy pomocy liczb. Brak symboli polskich liter zmusił do stosowania niezbyt przejrzystej transkrypcji: „ą" — „a", „ę" — „e", „ż" — „r", „ź" — „z" itp., przy której np. wyraz „żąć" zapisuje się jako „rac". Zbyt mała ilość czasowników (kilkadziesiąt), na których był testowany program, nie pozwala jeszcze na podsumowanie. Wydaje się jednak, że przez rozbicie grupy VIa na podgrupy: jedną typu „robić", drugą typu „gryzmolę" oraz przez odpowiednią zmianę zakończeń dla grupy VIIb udałoby się zmniejszyć ilość form potrzebnych dla poprawnej koniugacji czasowników z tych grup. Dużo kłopotu sprawiają niewłaściwe informacje przyhasłowe — dotyczy to także wydanego niedawno „Małego słownika języka polskiego".

W takiej sytuacji za zadanie na najbliższą przyszłość należy przyjąć ulepszenie programu przez bardziej ekonomiczny zapis zakończeń w pamięci oraz rozbudowanie algorytmu do pełnej fleksji. Po odpowiednio dokładnym testowaniu otrzymalibyśmy wiarygodny słownik poprawnej odmiany, który mógłby także służyć szerszym kręgom czytelników (w „Słowniku poprawnej polszczyzny" St. Szobera użyto opisu fleksji identycznego z wyżej omówionym, co — jak było widać — nie gwarantuje np. cudzoziemcowi uzyskania rzeczywiście poprawnych form fleksyjnych). Istnienie niezawodnych algorytmów fleksji i rozpoznawania form oraz wspomnianego słownika uczyniłoby język polski równie wygodnym do maszynowego przetwarzania jak angielski. Można by wtedy — korzystając z bogatych doświadczeń zachodnich — rozpocząć prace nad szeregiem interesujących zagadnień: od automatycznego sporządzania bibliografii do maszyny rozmawiającej po polsku.

BIBLIOGRAFIA

[1] Zob. Manowe, S. Bloon, C. Engelmi: Rational Function in MATHLAB. Conference on Symbol Manipulations Languages, Pisa 1966.
[2] Zob. J. Weizenbaum: Contextual understanding by computer. Communications of the ACM 10, No 8 (Aug. 1967).
[3] J. Tokarski: Formy fleksyjne. „Słownik języka polskiego" t. I. Warszawa 1958; s. XLIX.
[4] J. Tokarski: Metody ilościowe w językoznawstwie wobec nowych perspektyw. „Poradnik językowy" r. 1961; z. 6 (191), s. 241.
[5] J. Tokarski: Fleksja polska, jej opis w świetle możliwości mechanizacji w urządzeniu przekładowym. „Poradnik językowy": Założenia ogólne. Deklinacja przymiotnikowa. r. 1961, s. 3; Deklinacja rzeczownikowa i inne nieprzymiotnikowe, z. 4, 8; Koniugacja, r. 1962, z. 4;
Rozpoznawanie form fleksyjnych. r. 1963, z. 1, 2, 3—4, 5—6;
Granice paradygmatów fleksyjnych i formantów słowotwórczych w obrębie deklinacji rzeczownikowej, z. 9;
Seryjne powiązania fleksyjno-słowotwórcze rzeczowników z przymiotnikami i przysłówkami, r. 1964, z. 4;
Typy krzyżowania się fleksyjnego i słowotwórczego paradygmatów koniugacyjnych, z. 5;
Seryjne powiązania fleksyjno-słowotwórcze czasowników z innymi częściami mowy, z. 6.
[6] Por. przypis 5): „Koniugacja".
[7] J. Tokarski: „Czasowniki polskie". Warszawa 1951. Por. przypis 6).

W CZERWCU B. R.
ŚWIATOWY „SEJM" AUTOMATYKÓW W WARSZAWIE

Po Moskwie, Bazylei i Londynie — Warszawa jest miejscem spotkania ponad 2 tys. automatyków zrzeszonych w Międzynarodowej Federacji Automatyki — IFAC. Organizację IV Kongresu IFAC powierzono Polsce, a konkretnie NOT-owi. Patronat nad Kongresem objął premier — Józef Cyrankiewicz. Od dawna działa Komitet Organizacyjny, w którego prezydium znajdują się wybitni specjaliści w dziedzinie automatyki. Na czele Komitetu stoi minister przemysłu maszynowego — mgr inż. Janusz Hrynkiewicz.

We wrześniu 1956 r. w Heidelbergu odbyło się Międzynarodowe Sympozjum Automatyki, na którym 6 naukowców, a w tym Polak — prof. dr P. J. Nowacki, powzięło decyzję utworzenia Międzynarodowej Federacji Automatyków. W rok później na zebraniu konstytucyjnym w Paryżu powołano do życia Międzynarodową Federację Automatyki. Polska znalazła się wśród 19 krajów założycieli IFAC, zrzeszającej obecnie 33 państwa. W myśl statutu Federacji tylko jedna organizacja w danym kraju ma prawo reprezentowania go w IFAC. Polskę reprezentuje Polski Komitet Pomiarów i Automatyki NOT.

Co 3 lata odbywają się na świecie kongresy automatyków. Celem IFAC jest pobudzenie rozwoju automatyzacji oraz rozpowszechnienie osiągnięć wśród rzesz specjalistów różnych dziedzin techniki wykorzystujących automatyzację. Merytoryczna działalność IFAC prowadzona jest w komitetach technicznych: teorii, zastosowań, elementów, szkolenia i bibliografii.

Przygotowania do Kongresu, który rozpocznie swoje obrady 16 czerwca w salach Pałacu Kultury i Nauki w Warszawie, są już daleko zaawansowane. Komitet Selekcyjny wybrał 303 z ponad 400 nadesłanych referatów. Codziennie do biura sekretariatu IFAC w warszawskiej siedzibie NOT napływają zgłoszenia. Ustalono już szczegółowy program obrad i tzw. imprez towarzyszących.

Obrady prowadzone będą w 50 sesjach technicznych. Kongres zapoczątkuje przemówienie powitalne Prezydenta IFAC — prof. P. J. Nowackiego na inauguracyjnej sesji plenarnej. W dyskusjach „okrągłego stołu" i na sesjach omawiane będą między innymi problemy: sterowania cyfrowego, pneumatyki, hydrauliki, sterowania systemami wielkimi, diagnostyka jakościowa i niezawodność. Ciekawym zagadnieniem jest sterowanie pojazdami i ruchem, nawigacja i sterowanie w przestrzeni kosmicznej, automatyzacja w różnych gałęziach przemysłu.

Językami obrad są: angielski, rosyjski, francuski, niemiecki. Przewiduje się tłumaczenie poszczególnych wystąpień systemem symultanicznym na język polski. Koszt uczestnictwa w IV Kongresie IFAC wynosi dla uczestników 1200 zł, dla autorów przyjętych referatów — 800 zł, dla osób towarzyszących — 300 zł. Przygotowywane są również komplety wszystkich referatów w 50 broszurach w wersji angielsko-rosyjskiej.

STANISŁAW CHROBOT, WŁODZIMIERZ RACZYŃSKI
Instytut Techniczny Wojsk Lotniczych
Warszawa

Programowanie maszyny ZAM-2 GAMMA pracującej na bieżąco

Autorzy przedstawiają problemy programowania maszyny cyfrowej ZAM-2 GAMMA współpracującej z urządzeniem transmisji danych za pośrednictwem kanału przetwarzania na bieżąco oraz ze wskaźnikiem optycznym. Skupiono uwagę na trzech zasadniczych problemach:

- *synchronizacji szybkości przetwarzania informacji z szybkością jej napływu;*
- *stopniowym ograniczaniu funkcji systemu w momentach przeciążenia;*
- *specyfice uruchamiania programów.*

W zakończeniu pokazano zrealizowane rozwiązania tych problemów na tle zagadnień występujących w dużych i wyżej zorganizowanych systemach.

Celem artykułu jest uwypuklenie niektórych aspektów programowania maszyny cyfrowej pracującej w systemie przetwarzania na bieżąco. Konkretnym przykładem będzie tu stosunkowo nisko zorganizowana maszyna ZAM-2 GAMMA współpracująca z kanałem przetwarzania na bieżąco i wskaźnikiem optycznym.

Po ogólnym scharakteryzowaniu systemu z podkreśleniem wszelkich zmian konstrukcyjnych umożliwiających omawianą współpracę, rozważony będzie szerzej wpływ tej współpracy na strukturę programu, kwestie czasowe tu występujące oraz tendencje rozwojowe w systemach wyżej zorganizowanych.

1. Ogólna charakterystyka systemu

Jednostką centralną w systemie jest maszyna ZAM-2 GAMMA. Średni czas wykonania rozkazu wynosi tu około 1 ms, pojemność pamięci operacyjnej — 1024 słów 18-bitowych. Pamięcią pomocniczą są trzy bębny magnetyczne o łącznej pojemności 49152 słów 36-bitowych.

Poza tym maszyna współpracuje z dwoma czytnikami taśmy papierowej (300 zn/sek.) perforatorem taśmy (150 zn/sek.), czytnikiem kart (400 kart/min), kanałem przetwarzania na bieżąco oraz wskaźnikiem optycznym.

Kanał przetwarzania na bieżąco (KCR) [1] jest modułem pośredniczącym między częścią centralną i urządzeniem średniej transmisji (UST) będącym w tym przypadku odbiornikiem informacji z łącza telefonicznego.

UST umożliwia przekazywanie słów informacyjnych w odstępach czasowych 30 lub 60 m/sek.

Wskaźnik optyczny dołączony wymiennie z drugim perforatorem służy do wyświetlania na lampie oscyloskopowej informacji w formie punktów świetlnych z maksymalną szybkością 1000 punktów/sek.

Maszyna wyposażona jest w translatory języków SAS, SAKO i SAKO-M.

2. Specyfika programowania na bieżąco

Podstawową cechą, pozwalającą stwierdzić, że współpraca z KCR odbywa się na bieżąco, jest fakt, że informacja napływa do maszyny w sposób synchroniczny, z częstotliwością dyktowaną przez UST. Z drugiej strony przetwarzania informacja wyświetlana jest na wskaźniku optycznym. Jest to więc system z bezpośrednimi sprzężeniami *(on-line)*. Drugą cechą są tu bardzo silne w tym przypadku ograniczenia czasowe związane z przyjmowaniem, przetwarzaniem i wysyłaniem informacji.

Podchodząc do zagadnienia z drugiej strony stwierdzamy, że maszyna utraciła tu całkowicie kontrolę nad szybkością napływu informacji na rzecz UST, a szybkość wysyłania informacji uwarunkowana jest czynnikiem zewnętrznym, mianowicie niedopuszczeniem do jej dezaktualizacji. Ta utrata kontroli nad szybkością napływu i wysyłania informacji jest specyficzną cechą systemów pracujących w czasie rzeczywistym.

Dodatkowym utrudnieniem programowania w opisywanym tu systemie jest fakt, że maszyna nie posiada przerwań układowych. Ten fakt w sposób zasadniczy wpływa na strukturę programu. Jeśli bowiem informacja nie może być tracona, to program musi zapewnić wczytanie jej do momentu przyjścia następnego słowa. Oznacza to, że w trakcie realizowania algorytmu przetwarzania informacji w określonych odstępach czasu wywoływany musi być podprogram współpracy z KCR.

Średnia szybkość przetwarzania kolejnych słów musi być, oczywiście, większa od szybkości ich napływu. Biorąc pod uwagę z jednej strony, że czasy przetwarzania kolejnych słów nie są jednakowe, a z drugiej strony, że informacja jałowa jest eliminowana układowo bądź programowo, stwierdzamy, że chwilowe prędkości napływu i przetwarzania informacji są zmienne. Wynika z tego konieczność buforowania informacji wejściowej.

W celu zapisania i odczytania informacji z buforu zorganizowano dwa podprogramy.

Pierwszy z nich wczytuje słowa z KCR i po rozpoznaniu, że nie jest to informacja jałowa, umieszcza je w kolejnym miejscu buforu.

Bufor jest cykliczny, co oznacza, że po jego wypełnieniu, następne słowa wpisywane są od początku buforu na miejsca słów pobranych uprzednio przez program.

Realizacja rozkazów bezpośredniej współpracy z kanałem uniemożliwia wielokrotne wczytanie tego samego słowa, bowiem powtórne usiłowanie czytania tego samego słowa sygnalizowane jest podprogramowi, co powoduje natychmiastowy powrót do programu.

Drugi podprogram pobiera kolejne słowa z buforu i daje je do przetwarzania programowi.

Po pobraniu słowa z końca buforu, następne słowo pobierane jest z początku. Przy poprawnej pracy systemu między indeksami zapisu i odczytu muszą zachodzić takie relacje, aby nie wykorzystana jeszcze informacja nie była zacierana przez nową, ani powtórnie czytana.

Organizacja buforu wejściowego umożliwia stopniowe ograniczanie zdolności przetwarzania informacji w momentach wzrastającego przeciążenia systemu, spowodowanego nadmiernym wzrostem szybkości napływu informacji. Tak więc w momentach, gdy bufor jest całkowicie wypełniony, kolejne słowa zapisywane są „na siebie" aż do momentu zwolnienia następnej pozycji buforu przez program. Tracimy wtedy część informacji, ale nad pozostałą jesteśmy w stanie zachować kontrolę.

Zauważmy, że przy jednomilisekundowym cyklu rozkazowym między kolejnymi wczytaniami informacji z kanału, możemy wykonać około 30 rozkazów. Z te-

[1] Techniczna strona kanału zawarta jest w opracowaniu: „Kanał przetwarzania na bieżąco maszyny ZAM-2 GAMMA", „Maszyny Matematyczne" nr 2/69.

go około 15 rozkazów pochłania buforowanie. Pozostałe 15 rozkazów dla przetworzenia informacji to w większości przypadków stanowczo za mało. W takich sytuacjach poprawna praca programu jest możliwa przy napływie dużej liczby słów jałowych.

Te bardzo silne ograniczenia czasowe prowadzą niejednokrotnie do optymalizacji czasowej programu kosztem zajętości miejsc pamięci operacyjnej lub naruszenia kolejności przetwarzania słów w stosunku do kolejności ich przychodzenia. W tym ostatnim przypadku organizacja buforu jest inna i do przetwarzania jako pierwsze pobierane jest to słowo, które ostatnio przyszło.

Na rytmiczność, a nawet i na kolejność wysyłania informacji na wskaźnik optyczny brak jest silnych ograniczeń ze względu na długą poświatę.

3. Sposób uruchamiania programów

Zauważmy, że w opisywanym przez nas systemie zarówno informacja wejściowa, jak i wyjściowa jest „nietrwała". Nie wykorzystana bowiem w odpowiednim czasie zanika zarówno na wyjściu kanału, jak i na wskaźniku. Zmienia to w zasadniczy sposób metodę uruchamiania programów.

Wstępny etap sprawdzania poprawności oddzielnych algorytmów nie ulega zmianie w stosunku do stosowanych metod programowania konwencjonalnego.

Następny etap uruchamiania polega na sprawdzeniu, czy spełnione są założone relacje czasowe. W tym celu za pomocą prostego programu nagrywamy dane wyjściowe z kanału na bęben dołączając do każdego słowa znacznik czasowy. Ze względu na szybkość obrotów bębna (1 obrót = 40 m/s) można tu nagrywać jedynie transmisję o odstępach czasowych 60 m/sek. Tak utrwalona informacja odczytywana jest z bębna za pomocą uruchamianego programu, w którym podprogram współpracy z kanałem zastępuje się podprogramem czytania z bębna. Wykorzystując znaczniki czasowe w słowach i impulsy z kanału jako zegar, rozpoczynamy przetwarzanie kolejnych słów we właściwych im momentach. Uzyskujemy w ten sposób możliwość wielokrotnego przetwarzania tych samych informacji we właściwej skali czasu, co pozwala na wprowadzanie kolejnych korekt w programie.

4. Podsumowanie

Uzyskane efekty zastosowania maszyny ZAM-2 GAMMA do pracy na bieżąco są raczej skromne na tle aktualnych osiągnięć światowych w tej dziedzinie. Zwróćmy jednak uwagę, że parametry i organizacja maszyny na pierwszy rzut oka świadczą raczej o jej nieprzydatności do tego rodzaju zastosowań. Podstawowymi mankamentami są bowiem

- brak przerwań układowych;
- długi cykl rozkazowy;
- bardzo mała pojemność pamięci operacyjnej.

W tych warunkach osiągnięcie jakichkolwiek efektów jest cenne. Abstrahując bowiem nawet od efektów użytkowych pracy systemu, które nie są wcale bagatelne, stanowi to jedną z wielu możliwości eksperymentowania w czasie rzeczywistym.

Znajdujemy tu bowiem, niekiedy jeszcze w formie zalążkowej, większość problemów spotykanych w większych i wyżej zorganizowanych systemach pracujących na bieżąco, a nade wszystko poznajemy swoistą specyfikę programowania tych systemów.

Jednym z ważniejszych problemów jest tu synchronizacja szybkości napływu informacji z szybkością jej przetwarzania. Buforowanie jest rozwiązaniem zastosowanym również w większych systemach, z tym że jest ono z reguły funkcją Systemu Operacyjnego lub Programu Wykonawczego. Zastosowane tu dwa podprogramy współpracy z buforem wyjściowym można więc traktować jako zalążek obsługi linii transmsji w Systemie Operacyjnym.

Drugim problemem, który warto poruszyć, jest stopniowe ograniczanie funkcji systemu w przypadkach jego przeciążenia. Zastosowane rozwiązanie, aczkolwiek częściowe, uzyskane zostało tu bardzo prostą metodą. W dużych systemach jest to zagadnienie dużej wagi, wymagające rozpracowań teoretycznych.

Również nagrywanie informacji ze środowiska na bęben dla celów uruchamiania programów jest prostym, aczkolwiek efektywnym rozwiązaniem problemów modelowania czy symulacji środowiska stosowanych do tego samego celu w dużych systemach.

WIADOMOŚCI PKAPI

Konferencja „Elektroniczna technika obliczeniowa w pracach naukowo-badawczych, projektowych i konstrukcyjnych"

Konferencja ta została zorganizowana przez Oddział Wojewódzki Polskiego Komitetu Automatycznego Przetwarzania Informacji (PKAPI) przy OW NOT w Katowicach w dniu 10.X.1968 r.

Program obejmował następujące referaty:

1. Perspektywy rozwoju elektronicznej techniki obliczeniowej do celów zarządzania — mgr inż. Wincenty Balasiński
2. Zastosowanie maszyn cyfrowych w pracach konstrukcyjnych — mgr inż. Eugeniusz Bilski
3. O niektórych aspektach stosowania metod numerycznych w obliczeniach naukowych i technicznych — mgr Aleksander Śliżyński
4. Zasady optymalizacji konstrukcji za pomocą maszyn matematycznych — dr inż. Jan Goliński
5. Kosztorysowanie za pomocą maszyn cyfrowych — mgr inż Roman Gocki.

Po wysłuchaniu referatów oraz żywej dyskusji uczestnicy Konferencji wysunęli wnioski, które postulują do realizacji:

1. Należy zapewnić wprowadzenie do eksploatacji 400—500 elektronicznych maszyn cyfrowych w ciągu przyszłego planu 5-letniego w celu realizacji podstawowych wymagań w zakresie zastosowań eto.
2. Należy doprowadzić do ujednolicenia w perspektywie nomenklatury i formy dokumentacji systemowej i programowej w celu stworzenia warunków do ich szerokiego wykorzystania.
3. Należy doprowadzić do inwentaryzacji problematyki z zakresu prac naukowo-badawczych, projektowych i konstrukcyjnych w poszczególnych branżach i między branżami celem koncentracji wysiłków i w rozwiązywaniu problemów przy pomocy elektronicznej techniki obliczeniowej.
4. Należy położyć większy nacisk na opracowanie i stosowanie programów optymalizacyjnych.
5. Należy przyspieszyć publikacje informacyjno-dydaktyczne o produkowanych w kraju maszynach.
6. Należy organizować konferencje i narady poświęcone specjalnie problemom branżowym w przedmiocie elektronicznej techniki obliczeniowej.
7. Należy uwzględnić zapotrzebowanie społeczne na elektroniczną technikę obliczeniową w pracach naukowych i dydaktycznych katedr wyższych uczelni. Sprawie tej należy poświęcić specjalną naradę.
8. Należy intensyfikować wykorzystanie elektronicznej techniki obliczeniowej w pracach biur projektowych i konstrukcyjnych oraz dążyć do śledzenia i przedstawiania ekonomicznych korzyści płynących z jej zastosowania.

Nadesłał: **mgr inż. Jan Rejdych**
Katowice

JERZY DAŃDA
Instytut Maszyn Matematycznych
Warszawa

KONGRES IFIP 68

W dniach od 5 do 10 sierpnia 1968 r. odbył się międzynarodowy kongres, organizowany co trzy lata przez **International Federation for Information Processing (IFIP).**

Ta międzynarodowa organizacja powstała faktycznie w r. 1959 na pierwszej Międzynarodowej Konferencji, poświęconej zagadnieniom przetwarzania informacji, zorganizowanej przez UNESCO w Paryżu. Konferencja owa zgromadziła 1800 uczestników z 37 krajów. Formalnie organizacja IFIP powstała z dniem 1 stycznia 1960 roku zrzeszając przedstawicielstwa towarzystw i organizacji naukowych, zajmujących się przetwarzaniem informacji z 13 państw. W ciągu 8 lat członkami IFIP stało się 28 krajów. Spośród krajów socjalistycznych w IFIP reprezentowane są: Bułgaria, Czechosłowacja, Jugosławia, Kuba, Polska, Węgry i Związek Radziecki.

Przedstawicielem Polski w IFIP jest prof. dr Leon Łukaszewicz — w latach 1964—1968 — wiceprezydent tej organizacji.

Statutowymi zadaniami IFIP jest m.in. organizowanie międzynarodowych konferencji i sympozjów, poświęconych zagadnieniom przetwarzania informacji łącznie z zagadnieniami matematycznymi, inżynieryjnymi oraz handlowymi tej nowej dziedziny wiedzy.

Inną dziedziną działalności jest organizowanie międzynarodowych grup roboczych, mających na celu szczegółowe opracowanie zagadnień, leżących w sferze zainteresowań członków IFIP. Wreszcie IFIP stawia sobie za cel propagowanie szerokiej międzynarodowej wymiany informacji w dziedzinie automatycznego przetwarzania informacji wśród fachowców i rozpowszechnianie wiedzy o maszynach cyfrowych w szerokich kręgach społeczeństwa. Celowi temu służą liczne wydawnictwa — w chwili obecnej głównie specjalistyczne. W okresie istnienia IFIP odbyły się trzy międzynarodowe kongresy: w Monachium w 1962 r. (2800 uczestników z 41 krajów); w Nowym Jorku w 1965 r. (ponad 5000 uczestników z 50 krajów) i ostatni w Edynburgu w 1968 r. — który zgromadził około 3700 uczestników z 47 krajów.

Ponadto IFIP zorganizował szereg konferencji specjalistycznych: w Wiedniu w 1964 r., poświęconą językom do opisu języków formalnych; w Pizie w 1966 r. — poświęconą językom do przetwarzania napisów i w 1967 r. w Oslo — językom symulacyjnym. Odbyły się też konferencje IFIP organizowane wespół z innymi stowarzyszeniami, przede wszystkim z IFAC, zajmującym się zagadnieniami automatycznego sterowania.

Ważną pracą podjętą przez Komitet Techniczny TC-1 IFIP-u jest opracowanie wielojęzycznego słownika terminologii automatycznego przetwarzania informacji. Jest to słownik pojęciowy, z jednakowym układem tych pojęć i jednakowym ich oznakowaniem odpowiednimi symbolami w każdym języku. Taki układ i zasada wydawania słownika w oddzielnych tomach dla każdego języka umożliwia precyzyjne tłumaczenie oraz wydawanie poszczególnych tomów bez czekania na zakończenie całości prac. Słownik IFIP stanie się podstawą do ujednolicania terminologii w dziedzinie przetwarzania informacji, bowiem oprze się na nim w tym zakresie Międzynarodowa Organizacja Standardów (ISO).

Komitet Techniczny TC-2 zajmuje się językami programowania. Dziełem Grupy Roboczej WG 2.1, podległej temu Komitetowi było udoskonalenie ALGOL-u, zaproponowanie IFIP *Subset* ALGOL 60 dla organizacji ISO. Obecnie trwają prace nad propozycją ALGOL-u 68. Działalność TC-3 dotyczy zagadnień nauczania (m.in. kwestia propagowania wiedzy o maszynach matematycznych w szkolnictwie średnim), natomiast TC-4 zastosowania maszyn matematycznych w medycynie.

Kongres w Edynburgu

Podstawowym forum, stwarzającym szerokie możliwości wymiany poglądów pomiędzy specjalistami wielu krajów, są — według założeń IFIP — kongresy, gromadzące co trzy lata większość specjalistów tej dziedziny. Na kongresach wygłaszane są referaty „zaproszone" *(invited papers)* 1-godzinne lub półgodzinne; referaty „zgłoszone" *(submited papers)* 20-minutowe połączone z 10-minutową dyskusją.

Tablica

Kraj	Referaty zaproszone	Referaty nadesłane — przyjęte				Razem
		Matematyka	Oprogramowanie	Sprzęt	Zastosowanie	
Bułgaria					1	1
Czechosłowacja	1	4	1		3	9
NRD				1		1
Polska	1			1		2
Rumunia				1	1	2
ZSRR	4	2	2	2	5	13
Australia	1		1	1	1	4
Dania						
Finlandia			1			1
Francja	2	7	1	1	4	15
Holandia	1	1				2
Izrael				1	1	2
Japonia	1			1	1	3
Kanada	1	3	1	3	2	10
NRF	2	2			1	5
Szwajcaria		1			1	2
Szwecja		1				1
USA	13	14	23	22	38	110
Wenezuela		1				1
W. Brytania	4	4	12	11	17	48
Włochy		1	1			2
Razem:	31	41	41	45	76	234

Referaty te są wybierane przez komitet programowy spośród nadesłanych materiałów. Stosowana jest dość ostra selekcja, a mimo to materiał prezentowany na kongresie jest ogromny.

Tablica zawiera zestawienie referatów „zaproszonych" i „zgłoszonych", zakwalifikowanych do wygłoszenia z ostatniego kongresu w Edynburgu.

Równolegle do sesji odczytowych, odbywała się wystawa sprzętu elektronicznego przetwarzania danych. Niezależnie od ekspozycji na terenie wystawy — poważniejsze firmy zorganizowały oddzielne sesje wykładowo-odczytowe, toczące się niezależnie od obrad kongresu. W jednej z sal projekcyjnych odbywał się przegląd *non-stop* filmów naukowych i popularnonaukowych, związanych z zagadnieniami przetwarzania informacji. Były organizowane wycieczki do fabryk maszyn matematycznych, znajdujących się w pobliżu Edynburga i do ośrodków obliczeniowych, a dla osób towarzyszących delegatom i dla delegatów zorganizowany był tzw. program socjalno-turystyczno-rozrywkowy, w którym większym wydarzeniem była *garden-party*, wydana na terenie *Lauriston Castle* nad morską zatoką *Firth of Forth*.

W takich warunkach — olbrzymiej ilości źródeł informacji — poważnym wielce problemem staje się ich właściwe wykorzystanie. Jest ono tym trudniejsze, im mniej liczna jest delegacja, która w naszym przypadku liczyła tylko 5 osób. Dla przykładu: Czechosłowacja przysłała 44 osoby, Rumunia — 11, Węgry 18.
Bodaj jeszcze trudniejsze jest zrelacjonowanie choćby najważniejszych zagadnień poruszanych na Kongresie, dlatego niniejsze sprawozdanie należy traktować jako pierwsze, rozpoczynające serię artykułów bardziej ukierunkowanych, opartych o materiały kongresu w Edynburgu.

W tym sprawozdaniu postaramy się zreferować te tendencje w dziedzinie automatycznego przetwarzania informacji, które stały się już rzeczywistością — do sprawdzenia naocznie i namacalnie na wystawie, związanej z Kongresem.

Dominujące kierunki w API

Wystawa sprzętu do automatycznego przetwarzania informacji stwarzała dobrą okazję do zaobserwowania aktualnie dominujących kierunków w automatycznym przetwarzaniu informacji. Należą do nich niewątpliwie:

- stosowanie zdalnej transmisji danych, pozwalającej nie tylko na przesyłanie danych sposobem *„off-line"*, lecz na tworzenie systemów wielomaszynowych, jak również sprzęganie licznych urządzeń peryferyjnych z maszynami wielodostępnymi, pracującymi z podziałem czasu między wielu użytkowników,

- coraz szersze stosowanie dużych maszyn z podziałem czasu i wielodostępem,

- gwałtownie zwiększająca się popularność małych uniwersalnych maszyn cyfrowych, przeznaczonych głównie do sterowania przebiegiem złożonych eksperymentów z dziedziny fizyki atomowej, biologicznych i innych, jak również do zbierania i przetwarzania danych, pochodzących z pomiarów — wreszcie do rozwiązywania typowych zagadnień API — wszędzie tam, gdzie mała maszyna jest wystarczająca i gdzie oddanie maszyny do jednoczesnego użytkowania przez tylko jedną osobę posiada istotne uzasadnienie ekonomiczne.
Szerokiemu rozpowszechnianiu się tych maszyn sprzyja ich niska cena — z reguły poniżej 10 000 dolarów, jak również dobrze przemyślana modułowość, pozwalająca użytkownikowi rozbudowywać zainstalowany i pracujący system API w dowolny sposób. Maszyny te mogą również z powodzeniem pracować jako maszyny peryferyjne w systemach wielomaszynowych oraz zastępować w licznych przypadkach urządzenia centralnej rejestracji danych (tzw. *data loggers*), mając nad tymi ostatnimi przewagę, spowodowaną znacznie większą mocą obliczeniową.

- Wielka popularność nowoczesnych urządzeń wyjściowych — pisaków *x—y* (tzw. *x—y plotter*), umożliwiających wyprowadzenie z EMC wyników wprost w postaci wykresów, rysunków, schematów, jak również urządzeń wejściowo-wyjściowych wideograficznych (ang. *display*), podłączonych do maszyny *„on-line"*, przez co uzyskuje się możliwość konwersacyjnej współpracy użytkownika z maszyną cyfrową.

Rys. 1. Mała uniwersalna maszyna cyfrowa firmy VARIAN DATA MACHINES, typ 520/i. Pojemność pamięci od 4 do 32 kilobajtów/8-bitowych, cykl 1,5 μsek. 12 rejestrów na obwodach scalonych, priorytetowy system przerwań (11 linii), arytmetyka wieloprecyzyjna (8-, 16-, 24-, 32-bitowa). Przy przerwaniach przełączenie sterowania na drugi zespół rejestrów, tzw. dual environment, umożliwiające bardzo szybkie załatwianie przerwań, szczególnie cenne przy zastosowaniach maszyny do sterowania i przetwarzania danych pomiarowych. Cena około 10 000 dolarów, zależy od dodatkowego wyposażenia

Wielką powszechność omówionych nowoczesnych środków technicznych można było ocenić na wystawie znacznie lepiej, niż na posiedzeniach Kongresu; większość poważnych firm z dziedziny API demonstrowała urządzenia tego typu, (produkowane już seryjnie) w czasie pokazów unaoczniających realne korzyści, płynące z ich stosowania.

Obrady Kongresu pozwalały natomiast, co zresztą jest oczywiste, na orientację w kierunkach znacznie bardziej wybiegających w przyszłość. Zaliczyć do nich należałoby:

- próby tworzenia języków programowania i maszyn cyfrowych, opartych o nowe koncepcje, charakteryzujące się współbieżnym projektowaniem języka i maszyny w celu osiągnięcia należytej równowagi pomiędzy funkcjami *hardware'u* i *software'u* i wielkiej efektywności systemu, równocześnie przy większej jego uniwersalności. Za przykłady mogą służyć: *„Basic Language Project"* — G. G. Scarotta i J. K. Iliffe'a, prowadzony od 1965 r. w ICT, maszyna cyfrowa (projekt) oparta o pamięć asocjacyjną, opracowywana przez grupę T. Kilburna w *University of Manchester*, maszyna cyfrowa, oparta o pamięć stosowaną — projekt A. N. Myamlin i V. K. Smirnova, a wykonywana w Instytucie Matematyki Stosowanej w Moskwie;

- próby tworzenia nowych struktur maszyn cyfrowych, odpornych na uszkodzenia; przykładem może być projekt A. Avižienisa maszyny samonaprawiającej się, opartej o układy zwielokrotnione, połączone z układami pozwalającymi na kontrolę stanu technicznego, w jakim znajdują się elementy funkcjonalne tej maszyny — przełączane bloki;

- próby systematyzowania zagadnień oprogramowania dla maszyn z podziałem czasu i wielodostępem, ze szczególnym uwzględnieniem problemów występujących przy wzajemnym wykorzystywaniu programów użytkowników, korzystających z jednego systemu obliczeniowego. Uważa się bowiem, że takie właśnie wspólne wykorzystywanie programów, przy równoczesnym zapewnieniu należytej ochrony przed nie uprawnionym korzystaniem z danych cyfrowych poszczególnego użytkownika, przechowywanych w pamięci systemu, jest kluczem do dalszego rozwoju systemów z podziałem czasu. Pozytywne rozwiązanie tych nieprostych zagadnień nie tylko zapewni ekonomiczniejsze wykorzystanie zainstalowanej mocy obliczeniowej oraz pamięci informacyjnej, ale stworzy równocześnie bodziec do szerszego korzystania z systemów wielodostępnych, które będą zapewniały łatwe korzystanie z programów, opracowywanych przez współużytkowników. Systemy takie staną się zapewne współczesną formą klubów użytkowników maszyn, których zadaniem było wspólne tworzenie bibliotek programów. Obecna forma, którą umożliwiają maszyny wielodostępne, likwiduje praktycznie wszelkie zbędne prace dokumentacyjne.

Omówione wyżej zagadnienia rozpatrzone były przede wszystkim w referacie „zaproszonym" F. J. Corbató oraz J. H. Saltzera, jak również w szeregu innych, zawierających opisy systemów operacyjnych maszyn wielodostępnych.

Produkcja software'u

Znaczna liczba artykułów, poświęconych na Kongresie zagadnieniom techniczno-organizacyjnym opracowywania i produkcji *software'u* dobrze odzwierciedla krytyczną sytuację, jaka panuje w tej dziedzinie. Jej uzmysłowieniu mogą pomóc następujące liczby:

- analiza liczby rozkazów zawartych w podstawowym oprogramowaniu dostarczanych w maszynach firmy IBM wykazała 10-krotny wzrost tej liczby w ciągu okresu 5 1/2 roku. Oceniano tempo wzrostu w latach od 1952 do 1968 r. [1])
- koszty oprogramowania rodziny maszyn IBM 360 wyniosły 495 mln dolarów, oczywiście chodzi o koszty poniesione dotychczas (do roku 1968) [2]),
- koszty oprogramowania w stosunku do łącznych kosztów systemu (z włączeniem konserwacji) stanowiły we wczesnych latach pięćdziesiątych 5%, w roku 1965 wzrosły do 50%, a przewiduje się, że w roku 1970 wzrosną do 80%! Należy przy tym zwrócić uwagę na fakt, że przy tej prognozie uwzględniono prawie czterokrotny spadek kosztu opracowania programów w przeliczeniu na jeden rozkaz, jaki był obserwowany na przestrzeni dekady 1955—1965 [3]).

Rys. 2. Wideograf (ekranopis, display) firmy SANDERS — typ alfanumeryczny, przystosowany do lokalnej lub zdalnej współpracy z maszynami IBM 360, wyposażony w klawiaturę alfanumeryczną. Szybkość transmisji danych 47,5 tys. znaków/sek. przy pracy lokalnej lub 2400 bodów przy wykorzystaniu łączy telefonicznych. Wykonany w pełni na obwodach scalonych (z wyjątkiem kineskopu i jego zasilaczy)

Rys. 3. Uniwersalny wideograf firmy IBM-2250, na wystawie współpracujący z maszyną IBM-1130. Ekranopis wykorzystuje pamięć operacyjną (32 tys. słów) o cyklu 2,2 lub 3,6 μsek. maszyny 1130, wyposażony jest w klawiaturę alfanumeryczną, funkcyjną, pióro świetlne. Możliwość wyposażenia maszyny w pamięci dyskowe (do pięciu stacji) czyni zestaw 1130—2250 przydatnym do rozwiązywania szerokiej klasy zagadnień naukowych, wymagających konwersacyjnej graficznej współpracy z użytkownikiem. Maszyna może współpracować z 4 wideografami. Na wystawie demonstrowano szereg zastosowań systemu w wielu dziedzinach techniki

Powyższe liczby wskazują, że (przynajmniej w krajach dysponujących zaawansowaną techniką obliczeniową) zagadnienia obniżki kosztów opracowania *software'u* są warunkiem istotnym obniżenia kosztów przetwarzania informacji. Uważa się, że sprawa musi być rozwiązana radykalnymi metodami, również i z tych względów, że w dziedzinie programowania obserwuje się poważny deficyt sił ludzkich. W tej chwili jednak brak jest jednolitego stanowiska w zakresie środków, jakie mają być użyte do uzyskania celu — obniżenia kosztów *software'u*. Obserwuje się nawet stwierdzenia wręcz przeciwstawne, padające z ust specjalistów. I tak np. prof. S. Michaelson z *Department of Computer Science* Uniwersytetu w Edynburgu widzi rozwiązanie trudności przede wszystkim w odpowiednich krokach organizacyjnych. Postuluje powierzać opracowanie programowania małym grupom pracowników naukowych (np. na uniwersytetach), którym stworzone zostają szczególnie korzystne warunki do twórczej pracy, przede wszystkim przez przekazanie jasnych wytycznych na program jako wynik ich pracy — zażądanie programów o nadzwyczaj prostej strukturze, postawienie wymagań tylko na jakość produktu — terminy wprowadzają tylko niepotrzebne elementy nacisku — dobrze zorganizowana grupa i tak będzie pracowała szybko, możliwie dobre odizolowanie grupy od wszelkich niepotrzebnych doradców, oddanie do dyspozycji grupy maszyny wyposażonej w dobry język programowania i dobre programy diagnostyczne i odsunięcie od tej grupy wszelkich menażerów (sic!).

Mogło by się wydawać, że stwierdzenia takie są bezpodstawne — jednakże artykuł opiera się na licznych faktach, polegających na tym, że większość dobrych systemów oprogramowania została stworzona przez małe grupy, pracujące na uczelniach, podczas gdy analogiczne przedsięwzięcia, prowadzone przez duże firmy skończyły się, łagodnie mówiąc, rozczarowaniem. Z zawartych w artykule wielu przykładów przytoczymy jeden, szczególnie jaskrawy: system TSS (*Time Sharing System*), opracowany przez IBM dla maszyny 360/67 dopuszcza znacznie mniejszą liczbę stanowisk użytkowników i zapewnia mniejszą moc przerobową niż system CTSS, opracowany w *Massachusetts Institute of Technology* już w roku 1964 dla maszyny IBM 7094, która jest maszyną około trzy razy wolniejszą niż maszyna 360/67 [4]).

Wychodząc z założenia, że przedstawione wyżej tezy, dotyczące dużej efektywności małych grup, są słuszne, angielskie Ministerstwo Technologii wespół z firmą EEC zawarło z Uniwersytetem w Edynburgu kontrakt na oprogramowanie maszyny 4/75 z wielodostępem, będącej przybliżonym odpowiednikiem maszyny IBM 360/67. Nad zagadnieniem tym pracuje łącznie 22 osoby, zorganizowane w trzy grupy według prostych zasad, omówionych wyżej.

Rzecznicy poglądów przeciwstawnych upatrują główną przyczynę małej wydajności grup, oprogramowujących maszyny w niedostatkach organizacyjnych innego typu. Twierdzą, że do opracowywania programów należy wprowadzić metody, które zdały egzamin przy klasycznej produkcji, a więc precyzyjne określanie zadań, wprowadzenie etapowości prac: założenia, projekt koncepcyjny, projekt wstępny, modele i prototypy programów, kontrola jakości, odbiory etapów, dokumentacja na wzór dokumentacji technicznej, prowadzonej przy produktach technicznych, stosowanie PERT-u do planowania przebiegu prac, krótko mówiąc, zastosowanie wszystkich metod rozpowszechnionych w fabrykach czy biurach projektowych.

Podkreślają oni, że matematycy, którzy z oczywistych względów stanowią trzon grup programowania, nie otrzymują na uczelniach (z również dotychczas oczywistych względów) nawet niezbędnego minimum wiedzy o organizacji produkcji i praca ich charakteryzuje się bardzo wieloma elementami improwizacji, które w warunkach przemysłowej produkcji przynoszą jak najbardziej opłakane skutki. Dlatego też —

1) DATAMATION, vol. 15, nr 1, January 1969, str. 35.

2) DATAMATION, ibid. str. 82.

3) S. C. Joseph: IFIP Congress 68, Invited papers, str. 147.

4) S. Michaelson: IFIP Congress 68, Invited papers, str. 185.

Rys. 4. Pisak *x-y* firmy CALCOMP, typ 471 (płaski) z przystawką do wprowadzania rysunków na taśmę magnetyczną

twierdzą rzecznicy metod „produkcyjnych" w programowaniu — sytuację trzeba zmienić radykalnie przez pełne wykorzystanie wszystkich doświadczeń, jakie nagromadził przemysł w dziedzinie organizacji produkcji. Oczywiście zdają sobie oni sprawę, że przy wprowadzaniu metod „produkcyjnych" napotkają na opory wśród osób, którym taki, noszący znamiona „twórczego stylu bycia", sposób pracy bardziej odpowiadał.

Metody te wymagają bowiem znacznie większej dyscypliny w pozornie nieważnych drobiazgach (prace dokumentacyjne i to powtarzane dla poszczególnych etapów), jednakże ich zdaniem — są to trudności do pokonania.

Trudno przesądzać w tej chwili, która grupa ma rację, warto jednak zwrócić uwagę, że o ile pierwsza grupa poglądów sprowadza się do podjęcia takich środków organizacyjnych, aby osoby przyzwyczajone do określonego stylu pracy otrzymały warunki najbardziej sprzyjające pracy w stylu, jaki im najbardziej odpowiada — o tyle druga grupa — notabene liczniejsza — postuluje przełamanie dotychczasowych nawyków programistów i skłonienie ich do pracy metodami „przemysłowymi". Prawdopodobnie obie grupy reprezentują nieco skrajne poglądy. Dlatego też autor osobiście jest skłonny podzielać poglądy trzeciej grupy, która przedstawia najbardziej konstruktywne propozycje, omówione w następnym rozdziale.

Oprzyrządowanie pomocnicze dla produkcji software'u

Porównując środki, jakimi dysponuje inżynier-projektant *hardware'u* maszyny — ze środkami, jakimi dysponuje matematyk-projektant *software'u* systemu przetwarzania informacji, musimy stwierdzić, że wyposażenie pierwszego jest znacznie bogatsze [10]. Oscyloskopy, testery, lampki sygnalizacyjne i inne środki informowania o stanach, w jakich znajduje się *hardware* w trakcie wykonywania swych czynności, pozwalają mu stwierdzić zarówno prawidłowość działania, jak i efektywność działania realnych układów i porównać z wartościami założonymi w projekcie.

W odróżnieniu od tej sytuacji w znacznie gorszym położeniu jest programista. W systemach przetwarzania partiowego, które aktualnie posiadają dominujące znaczenie, jedynymi sposobami informowania o działaniu programu są programy, tworzące obraz pamięci *post mortem (dumping routines)* oraz programy tworzące ślady *(tracing routines)*. W innych systemach, umożliwiających pracę konwersacyjną, czy to poprzez oddanie całej maszyny do bezpośredniego użytku programisty czy w systemach wielodostępnych, możliwości uzyskania większej ilości informacji wzrastają.

Równocześnie jednak wzrost komplikacji systemów wieloprogramowych i wielodostępnych pociąga za sobą znaczne niewykorzystywanie potencjalnej mocy przerobowej systemu. Dla oceny, jakie są tego przyczyny, konieczne jest znaczne wzmożenie „wglądu" programisty w sposób wykorzystywania poszczególnych składników systemu, jak np. efektywność wykorzystania części centralnej, pamięci głównej i pamięci pomocniczych, urządzeń peryferyjnych. Przy tym istotna jest znajomość zmienności obciążeń poszczególnych składników w trakcie rozwiązania różnych problemów obliczeniowych.

Sporządzanie histogramów obciążeń rzadko może być uzyskane na drodze programowej, bowiem praca programu zbierającego informacje może poważnie zniekształcić tryb pracy systemu. W związku z tym proponuje się wyposażenie dużych systemów wielodostępnych w maszyny-monitory, pozwalające na ocenę zachowania się zasadniczego systemu w czasie rozwiązywania zadań. Propozycję taką zgłoszono np. dla maszyny IBM 360/67, patrz [10]. Innym nieco rozwiązaniem jest wykorzystanie w tym celu procesora peryferyjnego maszyny CDC 6600 [11].

Omówione rozwiązanie jest kosztowne i potrzebne przede wszystkim przy dużych systemach, czy jednak nie można by było wymyśleć mniej złożonych środków bądź programowych, bądź układowych, pozwalających programiście na ocenę efektywności działania programu? Wydaje się, że tak. A to, że nimi jeszcze nie dysponujemy, w głównej mierze wynika stąd, że zadanie takie nie było wyraźnie precyzowane, a co za tym idzie, nie było wystarczających wysiłków prowadzących do jego rozwiązania. Potwierdzeniem, że zagadnieniom tym poświęcono na Kongresie IFIP 68 dużą wagę, jest fakt zorganizowania trzech dyskusji panelowych, dotyczących m.in. spraw ekonomicznej produkcji *software'u*. Nie-

d.c. na III str. okł.

Rys. 5. Rysunek konstrukcyjny wykonany przy użyciu pisaka CALCOMP, typ 570. Na pisaku wykonywany jest kompletny rysunek łącznie z opisami słownymi. Precyzja rysunku: skoki (odcinki linii o długości 0,1 mm w 8 kierunkach co 45°). Szybkość kreślenia 300 skoków/sek.

Dokończenie ze str. 24

stety, więcej poświęcono uwagi zdefiniowaniu zagadnień niż samym metodom ich rozwiązania.

Interesująca, aczkolwiek odległa, prawdopodobnie w realizacji, wydaje się koncepcja R. W. Bemera z *Information Systems Group* [12] stworzenia fabryki *software'u*. Istotą tej propozycji jest stworzenie w fabryce *software'u* takich warunków, jakie występują w fabrykach znanych nam z doświadczenia, w których zagadnienie rozdziału mocy, stanowisk roboczych, dostaw i wysyłki, technicznej kontroli jakości, kontroli finansowej podlegają określonym regułom. Dalej Bremer proponuje, aby w fabryce *software'u* analogiczne do wyżej omówionych, zagadnienia znajdowały się pod kontrolą supersystemu przetwarzania informacji, który nie tylko byłby „dostawcą" niezbędnej mocy obliczeniowej, potrzebnej do wytwarzania programów-produktów pracy tej fabryki, ale spełniał wszystkie funkcje kontrolne, w szczególności zapewniające przestrzeganie przez programistów odpowiednich rygorów, np. co do postaci dokumentacji przestrzegania terminów, zamykania się w określonym limicie wydatków, czy nawet zabezpieczenie przed pisaniem programów do rozwiązywania problemów, już przez kogoś wcześniej w „fabryce" napisanych. R. W. Bremer podkreśla, że system taki musiałby być zaprojektowany zgodnie z zasadami *„human engineering"* tak, aby praca w takiej fabryce była nie tylko wydajna, lecz również atrakcyjna dla każdego programisty.

Wydaje się, że chociaż niewątpliwie jest to zamierzenie gigantyczne, to jednak jest ono realne i możliwe do osiągnięcia na drodze ewolucyjnej. Nie zakłada ono bowiem pełnej automatyzacji prac nad oprogramowaniem, jakie jest założeniem innych projektów, lecz idzie w kierunku tworzenia oprogramowania w sposób „wspomagany przez maszynę". Nasuwa się tu analogia do „projektowania wspomaganego przez maszynę" *(computer-aided design)*, którym — jak wiadomo — można objąć znacznie szerszą klasę zagadnień projektowych, np. w elektronice, czy konstrukcjach mechanicznych niż za pomocą całkowicie zautomatyzowanych procesów projektowych.

Obecne metody opracowywania oprogramowania nie mają prawie żadnych cech projektowania wspomaganego, mimo że — oczywiście — maszyny są tu wykorzystywane. Wykorzystanie bowiem maszyny cyfrowej przy uruchamianiu sprowadza się (można chyba zaryzykować takie stwierdzenie) do eksperymentalnego wypróbowania zastosowanych algorytmów, a w żadnej mierze — do wspomagania programisty w jego procesie obmyślania budowy programu. Uzyskuje on tylko możliwość względnie szybkiego (zależnego od stopnia konwersacyjności systemu, z którego korzysta) sprawdzania różnych wariantów, często nawet nie przemyślanych do końca. Nie przemyślanych nie dlatego, że się tego nie chce robić, lecz dlatego, że się tego robić nie opłaca, gdyż maszyna „sprawdzi" i wykryje wszystkie konsekwencje błędnych decyzji.

Rys. 6. Schemat czynnościowy wykreślony na pisaku CALCOMP z użyciem programu konwertującego program w FORTRANIE na schemat czynnościowy, wykreślany automatycznie

I na tym chyba kończy się rola maszyny, używanej w obecnie obowiązujący sposób przy tworzeniu programów. Oczywiście, programista zawsze korzysta z takich udogodnień, jak języki adresów symbolicznych, interpretatory i kompilatory, jednakże są to narzędzia stosunkowo niskie w hierarchii narzędzi, jakie należałoby stworzyć, chcąc zbudować prawdziwą fabrykę oprogramowania.

Przechodząc od tych daleko wybiegających w przyszłość fantazji do realiów życia codziennego — warto odnotować pojawienie się programów, odciążających programistów od znacznej ilości prac dokumentacyjnych. Są to programy sporządzające automatycznie schematy czynnościowe *(flow-charts, flow-diagrams)* na podstawie programu, zapisanego w odpowiednim języku.

I tak np. firma CALCOMP specjalizująca się w produkcji pisaków x—y i ich oprogramowania, oferuje program CALCOMP FLOWGEN/F za cenę 4500 dolarów. Program ten jest dostosowany do popularniejszych maszyn i języków na rynku amerykańskim. Przykładem wykresu czynnościowego, sporządzanego za pomocą tego programu, jest wykres na rys. 6. Podobne programy oferuje firma APPLIED DATA RESEARCH, Inc., która zaczęła reklamować swój program „68 AUTOFLOW" pod koniec roku 1968. Na razie ze zrozumiałych względów brak jest bliższych danych o budowie tych programów, jak również nie znane są opinie użytkowników o ich przydatności. Niewątpliwie jest to jednak pożyteczna nowość.

(cdn.)

BIBLIOGRAFIA

[1] G. G. Scarott, J. K. Iliffe: The Basic Language Project, IFIP Congress 68, Edinburgh — August 1968, Software 2, str. C8—C13, North Holland Publishing Company.
[2] T. Kilburn et al.: A System Design Proposal, ibid., Hardware 1, s. D76—D80.
[3] D. Aspinall et al.: Associative Memories in Large Computer Systems, ibid., Hardware 1, str. D81—D85.
[4] A. N. Myamlin, V. K. Smirnov: Computer with Stack Memory, ibid., Hardware 1, s. D91—D96.
[5] A. Avižienis: An Experimental Self-Repairing Computer, ibid., Hardware 2, s. E29—E33.
[6] F. J. Carbató, J. H. Saltzer: Some Considerations of Supervisor Program Design for Multiplexed Computer Systems, ibid., Invited Paper, s. 66—71.
[7] E. C. Joseph: Computers: Trends Toward the Future, ibid., Invited Papers, s. 145—157.
[8] S. Michaelson: How to succeed in software, ibid., Invited Papers, s. 184—185.
[9] W. L. Van der Poel: The Software Crisis, Some Thoughts and Outlooks, ibid., Invited Papers, s. 201—205.
[10] R. W. Bemer, A. L. Ellison: Software Instrumentation Systems for Optimum Performance, ibid., Software 2, s. C39—C42.
[11] D. F. Stevens: System Evaluation on the Control Data 6600, ibid., Software 2, s. C35—C38.
[12] R. W. Bemer: Position paper for panel discussion: The Economics of Program Production, ibid., Supplement, s. I13—I14.

CENTRALNY OŚRODEK BADANIA ODMIAN ROŚLIN UPRAWNYCH
w Słupi Wielkiej, pow. Środa Wlkp.,
tel. Środa Wlkp. 308

zatrudni

MATEMATYKÓW - STATYSTYKÓW jako **problemistów i programistów** w zakresie **biometrii i metodyki eksperymentu.**

Własny ośrodek ETO. Mieszkanie służbowe. Siatka płac instytutów.

WCT/864/69

Redakcja czasopisma „Maszyny Matematyczne"
przy współudziale
Polskiego Komitetu Automatycznego Przetwarzania Informacji
ogłasza otwarty

„KONKURS—69"

na najlepszą wypowiedź na temat:

„Najwłaściwsza tematyka przetwarzania informacji i organizacja sieci ośrodków obliczeniowych w kraju"

Wypowiedź powinna dotyczyć następujących zagadnień:
1. Jaka tematyka przetwarzania informacji jest najbardziej celowa w warunkach krajowych?

2. W jakich działach gospodarki i na jakim szczeblu zarządzania należy przede wszystkim stosować elektroniczną technikę obliczeniową?

3. Jaka powinna być organizacja przetwarzania informacji: czy na zasadach usługowych, np. sieć zakładów elektronicznej techniki obliczeniowej (ZETO), czy w oparciu o własne ośrodki obliczeniowe (np. przyzakładowe, branżowe, resortowe)?

W pracach należy uwzględnić w pełni warunki krajowe, w tym ograniczoną liczbę oraz charakterystykę eksploatacyjną posiadanych maszyn matematycznych. Praca konkursowa może dotyczyć pojedynczego lub syntetycznego ujęcia wszystkich podanych zagadnień.

Celem „KONKURSU-69" jest zebranie opinii co do kierunków rozwoju elektronicznej techniki obliczeniowej w kraju.

Warunki „KONKURSU-69"

- Prace należy nadsyłać w 5 egzemplarzach maszynopisu, nie przekraczającego 15 stron wraz z rysunkami (około 600 wierszy po 50 znaków minus miejsce na rysunki).

- Praca powinna zawierać uzasadnienie wysuniętych tez oraz podsumowanie w postaci zwięzłego sformułowania poglądów, ewentualnie wniosków.

- Prace należy nadsyłać w kopercie opatrzonej godłem, zawierającą obok pracy drugą zaklejoną kopertę wyjaśniającą godło. Koperta ta zostanie otwarta dopiero po zakończeniu prac jury nad pracami zaopatrzonymi godłem.

- Termin nadsyłania prac: **10 września 1969 r.**, pod adresem redakcji „Maszyn Matematycznych" z dopiskiem „KONKURS-69".

Organizatorzy zastrzegają sobie możliwość nieprzyznania I nagrody lub innego podziału nagród w zależności od wartości nadesłanych prac.
Nagrody ufundowało: **Polskie Towarzystwo Ekonomiczne** oraz **Wydawnictwa Czasopism Technicznych NOT.**

I nagroda	zł 3500
II nagroda	zł 2500
III nagroda	zł 1500
5 wyróżnień po	zł 500

Prace opublikowane następnie w czasopiśmie „Maszyny Matematyczne" będą honorowane oddzielnie.