3 1971

informatyka

SPIS TREŚCI

WYDAWNICTWA CZASOPISM TECHNICZNYCH NOT
Warszawa
Czackiego 3/5

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. 55 Papier powlekany V kl. 80 g. Obj. 3 ark. druk. Nakład 3000. U-48

Cena egzemplarza zł 8.— **INDEKS 36707** **Prenumerata roczna zł 96.—**

Informatyka

dawniej Maszyny Matematyczne

zastosowania w gospodarce, technice i nauce

Nr 3

MIESIĘCZNIK

1971

ROK VII

Marzec

Organ Krajowego Biura Informatyki i Polskiego Komitetu Automatycznego Przetwarzania Informacji Naczelnej Organizacji Technicznej

Od Redakcji

W dniu 12 lutego 1971 roku Prezydium Rady Ministrów podjęło uchwałę w sprawie rozwiązania Biura Pełnomocnika Rządu d.s. Elektronicznej Techniki Obliczeniowej i powołania z dniem 1 marca 1971 roku

KRAJOWEGO BIURA INFORMATYKI

i jednocześnie **ZJEDNOCZENIA INFORMATYKI** powstałego z przekształcenia Zakładów ETO, które zostały włączone do resortu Komitetu Nauki i Techniki.

Redakcja naszego miesięcznika składa nowym władzom życzenia pomyślnego kierowania rozwojem naszej informatyki.

LEON ŁUKASZEWICZ
Warszawa

681.322.06"311"(438)

Automatyzacja programowania w Polsce do roku 1970[1]

Autor daje przegląd zrealizowanych w Polsce języków programowania zarówno proceduralnych, jak i języków specjalistycznych i systemów operacyjnych.

1. Wstęp

Zainteresowanie automatycznym programowaniem w Polsce rozpoczęło się z chwilą pojawienia się pierwszych maszyn cyfrowych w naszym kraju. W roku 1957 uruchomiono w Instytucie Maszyn Matematycznych w Warszawie maszynę XYZ, a wkrótce potem pojawiły się maszyny produkowane półseryjnie lub seryjnie, jak UMC-1, ZAM-2 oraz ODRA 1003.

Dla każdej z wymienionych maszyn zbudowano język typu assembler oraz odpowiednio prosty język proceduralny. Bardziej złożone języki, w tym język ALGOL 60 i COBOL, zostały zrealizowane dopiero dla większych maszyn krajowych drugiej generacji, jak ZAM-41 lub ODRA 1204. Dla maszyn tych rozpoczęto również budować bardziej złożone systemy operacyjne, umożliwiające ich pracę np. w systemie wieloprogramowym.

Ogólnie biorąc, należy zanotować w Polsce dość dużą aktywność w zakresie automatyzacji programowania. W niniejszym artykule przedstawiono najważniejsze — zdaniem autora — osiągnięcia, a więc przede wszystkim takie, które odegrały u nas rolę pionierską, mają cechy oryginalności lub też zdobyły sobie szersze kręgi użytkowników.

Załączony na końcu artykułu spis zawiera około 30 systemów opracowanych w naszym kraju i niemal wyłącznie dla maszyn krajowych. Spis ten zawiera 4 języki wzorowane na FORTRAN II, 8 realizacji języka ALGOL 60, 1 realizację COBOL-u oraz 10 realizacji języków innego typu.

2. Języki typu FORTRAN II

Żadna wersja FORTRANu nie była w Polsce zrealizowana. Nie mniej opracowane zostały w Polsce dwa opisane poniżej języki, których ogólna struktura była wzorowana na języku FORTRAN II.

2.1. Język i translator SAKO dla maszyny XYZ oraz maszyn ZAM 2, ZAM 21 i ZAM 41

Język SAKO jest opracowaniem w znacznej mierze oryginalnym, a pierwszy jego translator uruchomiono w roku 1962. Było to opracowanie w Polsce pionierskie i przez wiele lat w dziedzinie języków proceduralnych i w tej skali — jedyne.

Następne translatory SAKO uruchomiono dla maszyn ZAM 21 oraz ZAM 41. Łącznie SAKO znalazło w Polsce dość szerokie rozpowszechnienie, aczkolwiek obecnie jest już w znacznej mierze wyparte przez język ALGOL 60.

Język SAKO przystosowany był pierwotnie do maszyn stołoprzecinkowych, przy czym programista miał pełną możność sterowania skalą obliczeń. W realizacjach tego języka dla ZAM 21 oraz dla ZAM 41 obliczenia można również wykonywać w zmiennym przecinku. Rozkazy SAKO mogą też być przeplatane z rozkazami maszyny, zapisanymi w języku maszyny SAS dla ZAM 2 przy zachowaniu tych samych identyfikatorów.

SAKO przewiduje podział programu na rozdziały, z których każdy może być kompilowany oddzielnie. Wprowadzone też zostały operacje bulowskie na słowach binarnych maszyny. Ponadto SAKO posiada prawie wszystkie operacje występujące w FORTRAN II, zakodowane w polskiej wersji językowej, przy czym niektóre cechy tego języka stawiają go na równi z FORTRANEM IV.

2.2. ALGUM — język algorytmiczny dla UMC 1 i UMC 10

Zakończony w roku 1965 w Centralnym Ośrodku Badań i Rozwoju Techniki Kolejnictwa, język ten został pomyślany jako język uniwersalny dla bardzo małych maszyn.

Operacje arytmetyczne określone są jedynie dla liczb stałoprzecinkowych, ze skalowaniem wartości podobnie, jak w SAKO. Analiza programu źródłowego jest jednoprzebiegowa. Program wynikowy umieszczany jest na stałych miejscach pamięci.

3. Języki typu ALGOL 60

Językiem najbardziej rozpowszechnionym w Polsce jest niewątpliwie ALGOL 60. Z tego też powodu translatorom tego języka poświęcono w Polsce sporo uwagi i wysiłków.

3.1. Translator ALGOL 60 na ZAM 21

Zakończony w Instytucie Maszyn Matematycznych w roku 1966, jest to translator prawie pełnej wersji ALGOLu z niewielkimi tylko ograniczeniami. Translator jest 5-przebiegowy i wraz z funkcjami i procedurami standardowymi zajmuje około 8000 rozkazów. W translatorze tym na uwagę zasługuje podział programu wynikowego na segmenty bez wnikania w jego strukturę. Program wynikowy zapisany jest na bębnie; wymiana segmentów między bębnem a pamięcią operacyjną odbywa się według oryginalnej metody, która zdała praktyczny egzamin.

---

[1]) Skrót referatu przeglądowego, wygłoszonego przez autora na międzynarodowej konferencji „Metody automatyzacji programowania i automatyzacji eksploatacji maszyn liczących", odbytej w Warszawie w dniach od 12 do 16 października 1970 r.

3.2. Translator UMC ALGOL dla maszyn UMC 10

Wykonany w roku 1967 w Centralnym Ośrodku Badań i Rozwoju Techniki Kolejnictwa. W stosunku do ALGOLu 60 wprowadzono daleko idące ograniczenia. Tłumaczenie programu źródłowego odbywa się w jednym pasmie translacji bezpośrednio na miejsce programu wynikowego. Przydzielanie pamięci jest statyczno-dynamiczne z tym, że pamięć jest przydzielana dynamicznie tylko tablicom programu.

3.3. Translator ALGOL 60 dla ZAM 41

Translator ten został zakończony w roku 1968 w Instytucie Maszyn Matematycznych. Jest to translator prawie pełnej wersji ALGOL 60, wzorowany na Wheatstone Compiler, opracowany dla maszyny angielskiej KDF9.

Tłumaczenie programu źródłowego na program wynikowy odbywa się w dwóch przebiegach. Program wynikowy zapisany jest w języku makrorozkazów, wykonywanych przez interpreter. Zarówno program wynikowy, jak i stos podzielone są na segmenty po 256 słów i umieszczone w pamięci bębnowej. Wymianą segmentów pomiędzy pamięcią bębnową a operacyjną steruje interpreter.

Język ALGOL 60 uzupełniony został przez dogodne procedury wejścia-wyjścia, zgodne ze standardami ALGAMS.

3.4. MINAL — podzbiór ALGOL 60 i jego translator dla maszyny ODRA 1204

Zakończony w 1968 roku w Centralnym Ośrodku Badań i Rozwoju Techniki Kolejnictwa. Zachowując formę języka ALGOL 60 zastosowano w języku MINAL daleko idące ograniczenia, przyśpieszające wykonanie programu w wyniku wprowadzenia statycznego przydziału pamięci i uproszczeń w komunikacji z procederami. Ponadto w każdym miejscu może występować sekwencja napisana w prostym języku symbolicznym o wygodnej komunikacji z miejscami roboczymi całego programu.

Translator MINAL pozwala ponadto na dodatkowe przyspieszenie wykonania programów wynikowych przez usunięcie z nich po uruchomieniu wszelkich rozkazów dokonujących kontroli.

3.5. Translator ALGOL 1204 dla maszyny ODRA 1204

Wykonany w roku 1970 w Katedrze Metod Numerycznych Uniwersytetu Wrocławskiego. Jest to, zdaniem autora, najbardziej interesująca ze wszystkich implementacji ALGOLu wykonanych dotychczas w Polsce. Dopuszcza się mianowicie dynamiczne tablice z mianem **own** i wszystkie rodzaje rekursji procedur.

Język ALGOL 1204 zawiera dodatkowe funkcje matematyczne oraz możliwość programowania w kodzie wewnętrznym maszyny ODRA 1204; rozkazy maszynowe mają postać instrukcji procedur tłumaczonych na jedno słowo maszynowe.

Tłumaczenie — bezpośrednio na ciąg rozkazów maszynowych — odbywa się w dwóch fazach: w fazie pierwszej analizuje się strukturę blokową programu, a w fazie drugiej wykonuje się pełną weryfikację syntaktyczną programu wynikowego i optymalizację pętli i odwołań do zmiennych indeksowych oraz procedur standardowych. Podział czasu maszyny pomiędzy te procesy jest sterowany kolejnymi stanami rozbioru syntaktycznego programu. Obie fazy translatora są sterowane za pomocą mechanicznie wygenerowanych tablic przejść istotnie zmodyfikowanych według oryginalnych koncepcji autorów.

Translator ALGOLu 1204 zawiera również rozbudowany aparat do drukowania informacji o błędach znalezionych w tłumaczonym programie, a także umożliwiających automatyczną korektę programu według żądań programisty.

Program wynikowy może być przygotowany w częściach — tłumaczonych niezależnie od siebie — na taśmach binarnych kolejno wprowadzanych — ewentualnie wielokrotnie — do pamięci operacyjnej dopiero w czasie działania programu (segmentacja na taśmie); umożliwia to uruchamianie niektórych programów o objętości kilkakrotnie przekraczającej pojemność pamięci operacyjnej. Przewiduje się odpowiednią adaptację tej możliwości dla zestawów maszyny z dużą pamięcią zewnętrzną.

4. Translator języka COBOL na ZAM 41

Translator ten największy ze względu na liczby rozkazów i stopień złożoności ze wszystkich translatorów wykonanych dotychczas w Polsce zakończono w Instytucie Maszyn Matematycznych w roku 1970. Jest on dziesięcioprzebiegowy i zawiera około 18 000 rozkazów.

W skład przyjętej reprezentacji języka wchodzą następujące elementy standardowego opisu języka COBOL 1:

jądro	poziom 2,
dostęp sekwencyjny	„ 2,
sortowanie	„ 1,
segmentacja	„ 1.

Jak widać więc, jest to dość duży podzbiór języka COBOL. Do wzorcowego opisu języka COBOL dołączono szereg zwrotów deklaracyjnych, dzięki którym umożliwiono przetwarzanie informacji zapisanej na nośnikach zewnętrznych w sposób niepozycyjny. Takie rozwiązanie jest szczególnie wygodne w przypadku stosowania taśmy perforowanej jako jednego z nośników. W trakcie wczytywania do pewnego pola pamięci, informacja niepozycyjna jest automatycznie przekształcana do postaci zgodnej z opisem tego pola w programie.

W realizacji translatora warto zwrócić uwagę na trzy sprawy:

a. Organizacja danych na taśmach magnetycznych została przyjęta zgodnie z zaleceniem ISO.

b. Duża część translatora napisana została w języku EOL, dzięki czemu można było precyzyjnie zdefiniować języki wejściowe i wyjściowe poszczególnych przebiegów, szybko napisać i uruchomić programy przebiegów oraz sporządzać czytelną dokumentację translatora.

c. Programista może posługiwać się językiem assemblera SAS, którego instrukcje mogą być włączane w dowolnych miejscach PROCEDURE DIVISION, przy czym można się jednocześnie odwoływać do zmiennych opisanych w DATA DIVISION.

5. Języki specjalistyczne

W ostatnich latach obserwuje się w Polsce tendencje do opracowywania języków specjalistycznych. Następujące z nich zasługują na szczególną uwagę:

5.1. Język modelowania symulacyjnego procesów dyskretnych SYM 69

Język SYM 69 służy do opisu zadań symulacji zachowania się w czasie (modelowania symulacyjnego) złożonych obiektów przedstawionych w postaci modeli dyskretnych.

Język został opracowany w Centralnym Ośrodku Badań i Rozwoju Techniki Kolejnictwa jako narzędzie badania złożonych obiektów transportu (sieci, linii, węzłów, stacji kolejowych).

Zakres zastosowań tego języka jest typowy dla języków tej klasy, co SIMULA, SOL, SIMON.

Język SYM 69 powstaje z języka ALGOL 60 przez dołączenie pewnych konstrukcji programowych, umożliwiających opis wielu zjawisk zachodzących

równolegle oraz dynamiczne operowanie skomplikowanymi strukturami miejsc roboczych.

Translator języka SYM wykonany dla maszyny cyfrowej ODRA 1204 nie zawiera pełnego aparatu języka ALGOL 60, a tylko te jego fragmenty, które okazały się niezbędne w programowaniu zadań symulacji.

Translator eksploatowany jest od połowy 1969 roku. Analiza programu źródłowego odbywa się w 5 przebiegach, za pomocą których jednolity blok informacji (o zmniejszającej się z przebiegu na przebieg objętości) przekształcany jest na program wynikowy, wykonywany interpretacyjnie.

5.2. Język modelowania symulacyjnego procesów ciągłych CEMMA 2

Zarówno język, jak i translator zostały utworzone w Zakładzie Sterowania Instytutu Maszyn Matematycznych dla maszyny ZAM 41. Translator oddany został do eksploatacji w pierwszym kwartale 1970 roku.

Język CEMMA 2 (CYFROWE Modelowanie Maszyny Analogowej) służy do symulowania cyfrowego procesów ciągłych, tradycyjnie rozwiązywanych na maszynach analogowych. Uniwersalność techniki cyfrowej umożliwiła znaczne rozszerzenie zestawu operatorów (jest ich 39). M. in. włączono do języka takie operacje nietypowe dla maszyn analogowych, jak np. funkcje nieliniowe $\sqrt{x}$ oraz log (x), elementy logiczne, funkcje zadane tablicą itp. W jednym zadaniu może występować 500 operatorów, co także przewyższa możliwości stosowanych obecnie maszyn analogowych. Ponadto wprowadzono do języka procedurę optymalizacji, która umożliwia optymalizację statyczną i dynamiczną badanych układów. Programy w języku CEMMA 2 wprowadzane są do maszyny na taśmie perforowanej lub na kartach. Wyniki wypisywane są na drukarkę w postaci tablic albo w postaci graficznej.

5.3. Język do przekształcania symboli — EOL

Jezyk EOL jest językiem do przekształcania symboli i był już opracowany w kilku kolejnych wersjach. EOL-1 oraz EOL-2 zostały opracowane koncepcyjnie w latach 1965—66 w Instytucie Maszyn Matematycznych.

EOL-2 został zrealizowany w roku 1966 na maszynie ZAM 41. EOL-3 opracowany został w roku 1967 na Uniwersytecie ILLINOIS oraz zrealizowany w roku 1968 na maszynach IBM 7094 oraz IBM 360.

Następujące cechy są charakterystyczne dla wszystkich wersji języka EOL:

— są to języki uniwersalne do przetwarzania symboli, przystosowane jednak przede wszystkim do pisania translatorów;

— są koncepcyjnie proste, łatwe do nauczenia i realizacji;

— zawierają operacje niskiego szczebla, a jednocześnie pozwalają użytkownikowi na jego rozszerzenia w dowolnym kierunku za pomocą zarówno podprogramów, zapisanych w języku EOL lub też w języku maszyny.

— Programy napisane w języku EOL prowadzą do sprawnych programów wynikowych.

Realizacja powyższych punktów została ułatwiona m. in. tym, że języki EOL mają zarówno środki dla przetwarzania krótkich, lecz złożonych informacji w pamięci operacyjnej maszyny, jak również środki do przetwarzania plików umieszczonych w pamięci masowej maszyny.

Doświadczenia zdobyte do tej pory potwierdzają użyteczność EOLu przy pisaniu translatorów. W szczególności za pomocą EOL-2 napisano znaczną część translatora języka COBOL.

Języki EOL są stosowane w celu kształcenia w zakresie wiedzy komputerowej zarówno w Polsce, jak i na niektórych uniwersytetach amerykańskich.

5.4. LISP 1.5 dla maszyny GIER

Na Wydziale Matematyki Uniwersytetu Warszawskiego wprowadzono na maszynę GIER (wyposażoną w ALGOL 60) język LISP 1.5 zanutrzony w ALGOLu. Zanurzenie to polega na tym, że zapewnia się stosowanie struktur danych i wykorzystanie pamięci właściwe dla LISP, ale operacje języka mają postać procedur ALGOLu.

Programy pisze się zgodnie ze składnią ALGOLu, wykorzystując procedury odpowiadające operacjom LISP lub zdefiniowane w oparciu o nie.

Wykorzystując ten LISP-ALGOL wprowadzono na GIER język COMIT zanurzony w ALGOLu. Również przez zanurzenie w ALGOLu wprowadzono na GIER język LOGOL (opracowany w kraju, przeznaczony do analizy syntaktycznej i translacji z użyciem automatów skończonych).

Do celów dydaktycznych opracowano na GIER interpreter języka LISP 1.5, napisany w LIPS-ALGOLu w ten sposób, aby w razie potrzeby można było bardzo szczegółowo pokazywać i badać jego działanie; jednak stosowanie go do celów praktycznych nie jest celowe z tego powodu, że interpretowane przezeń programy w LISP 1.5 działają wolniej niż, gdyby były napisane w LISP/ALGOLu.

5.5. LISP 1.5 dla maszyny ODRA 1204

W Katedrze Maszyn Matematycznych Wojskowej Akademii Technicznej uruchomiono w roku 1970 translator języka LISP 1.5. Część stała systemu interpretacyjnego zajmuje około 4000 słów pamięci. W jej skład wchodzą:

— funkcje organizujące pamięć
— funkcje wejścia i wyjścia
— funkcje interpretera.

Istnieje możliwość dołączenia dowolnego zestawu funkcji napisanych w języku maszyny, w tym również procedur graficznych.

6. LJAPASS

W Centrum Obliczeniowym PAN do roku 1969 zrealizowano język LJAPASS na maszynie ODRA 1204. Translator języka LJAPASS obejmuje w zasadzie pierwszy poziom tego języka, rozszerzony o operacje nad zmiennymi złożonymi. Uzyskano postać zewnętrzną języka identyczną z jego reprezentacją publikacyjną.

7. Systemy operacyjne

Krajowa produkcja maszyn matematycznych posiadających szereg nowoczesnych cech, jak np. wieloprogramowość stworzyła potrzebę budowy dla nich złożonych systemów operacyjnych.

Ich konstrukcją zajmuja się zarówno sami producenci, zwłaszcza Instytut Maszyn Matematycznych, jak również niektórzy użytkownicy, jak np. Centrum Obliczeniowe PAN.

Następujące systemy operacyjne zaprojektowane w Polsce zasługują na szczególną uwagę:

1. System Operacyjny SO/41 dla ZAM-41

System ten został zakończony w Instytucie Maszyn Matematycznych w roku 1968. Jest to najpełniejszy system operacyjny ze wszystkich opracowanych dotychczas w Polsce. Spełnia on różnorodne zadania. Najważniejszym z nich jest zwiększenie przepustowości maszyny poprzez wykorzystanie jej części centralnej w systemie wieloprogramowym. Dotychczas zrealizowano pracę dwuprogramową przy sztywnym, chociaż dowolnym podziale maszyny na dwa zesta-

wy. System SO/41 zapewnia też prostą obsługę operatorską przez traktowanie maszyny jako urządzenia do przetwarzania kolejnych zleceń, przy czym sposób wykonania każdego zlecenia opisany jest w czołówce za pomocą Języka Operacyjnego Maszyny (JOM).

Należy tu dodać, że zasady obsługi wszystkich urządzeń zewnętrznych są podobne, a wszystkie dane wyjściowe opatrzone łatwo rozróżnialnymi etykietami ustalającymi ich przynależność do przetwarzanych zleceń. System zapewnia również łatwy dostęp do elementów systemu oprogramowania, zawartych w Bibliotece Systemu, umieszczonej na taśmie magnetycznej oraz na bębnie.

Elementami biblioteki są zbiory identyfikowane przez nazwę. System SO/41 zapewnia łatwą adaptację programów do różnych urządzeń zewnętrznych. Jest to zrealizowane przez standaryzację operacji wejścia i wyjścia oraz odwoływanie się do tych urządzeń za pomocą symboli, których znaczenie określa się poza programem przy użyciu języka systemu operacyjnego.

System SO/41 zawiera też w sobie System Generowania Konkretnych SO, dający możliwość automatycznego generowania wersji systemów dostosowanych do różnych konfiguracji maszyny.

SO/41 może być łatwo rozbudowany przez dołączanie nowych programów do Biblioteki Systemu lub też nowych podprogramów wejścia i wyjścia do supervisora. W szczególności do SO/41 można dołączyć stosunkowo łatwo translatory nowych języków.

SO/41 umożliwia łatwą adaptowalność maszyny do zmiennych wymagań eksploatacyjnych danego ośrodka. Osiąga się to przez umożliwianie podjęcia szeregu decyzji o trybie pracy systemu w momencie jego startu. W szczególności można wybrać odpowiednią wersję supervisora, określić rozmieszczenie elementów systemu w pamięciach pomocniczych (taśmie magnetycznej lub bębnie) oraz określić podział maszyny na zestawy.

W czasie pracy SO/41 prowadzi rejestrację pracy oraz rejestrację wykorzystywanych szpul magnetycznych.

Supervisor zajmuje na stałe 3—5 K słów pamięci operacyjnej (24-bitowych) oraz 6-12 K takich słów na bębnie.

System Operacyjny Dwuaktywny SODA dla mc ODRA 1204 został zaprojektowany i zrealizowany w CO PAN w roku 1970. Oparty jest on o oryginalną ideę rozwiązania systemu operacyjnego dla małych maszyn. Głównym celem funkcjonalnym systemu jest usprawnienie pracy ośrodka obliczeniowego, którego rytm pracy rozpada się na dwie składowe — uruchamianie a następnie wykonywanie uruchomionych programów.

SODA organizuje na jednej maszynie dwa pozorne systemy:

SEKS — system eksploatacyjny i SUGAR — system uruchamiania, generowania i rezerwacji umożliwia podział zasobów sprzętowych, w tym także podział czasu pracy jednostki arytmetyczno-logicznej pomiędzy te systemy.

Dodatkowymi założeniami zewnętrznymi na system SODA są: ograniczony i stały zestaw urządzeń peryferyjnych systemu liczącego i nieograniczona różnorodność języków programowania, które — być może — będą w nim pogrążone. Wynika stąd inny niż zazwyczaj rozkład akcentów w pojmowaniu elastyczności systemu: SODA nie przewiduje łatwych zmian konfiguracji sprzętowych, zezwalając jednocześnie na bardzo proste przyłączanie nowych translatorów, na które nie nakłada żadnych ograniczeń organizacyjnych poza wymaganą formą komunikacji między elementami systemu SODA i translatora.

Objętość uzyskanego Systemu — część stała około 5 K słów 24-bitowych (łącznie z tablicami i buforami urządzeń peryferyjnych) wskazuje, że mimo wykorzystania środków automatyzacji projektowania, zachowano dość znaczną zwartość programów.

8. Zakończenie

Powyżej przedstawiono najważniejsze wyniki w dziedzinie automatycznego programowania w Polsce osiągnięte do roku 1970. Wyniki te, jak również szereg innych wyników nie przedstawionych w niniejszym artykule odegrały bardzo ważną rolę w Polsce, gdyż usprawniły wykorzystanie maszyn produkowanych w naszym kraju, rozszerzyły krąg ich użytkowników oraz niepomiernie ułatwiły im pracę. Ponadto wykształciła się w Polsce znaczna kadra projektantów języków i ich realizacji, gotowa podjąć dalsze, poważniejsze postawione przed nimi zadania.

9. Podziękowanie

Autor pragnie w tym miejscu podziękować tym wszystkim osobom, które przez dostarczenie odpowiednich materiałów przyczyniły się do powstania niniejszego artykułu, a więc przede wszystkim **Mgr Janowi Borowcowi, Dr Jerzemu Leszczyńskiemu, Dr Antoniemu Mazurkiewiczowi** oraz **Doc. Władysławowi Turskiemu.**

10. Uwaga

Autor wdzięczny będzie wszystkim Czytelnikom za nadsyłanie pod adresem redakcji wszelkich poprawek lub uzupełnień do niniejszego artykułu celem uwzględnienia ich w następnych jego opracowaniach.

Realizacja języków wyższego szczebla oraz niektórych systemów operacyjnych w Polsce w latach 1958—1970

Lp.	Nazwa	Miejsce opracowania	Maszyna	Rok uruchomienia	Uwagi
1. Języki typu FORTRAN II					
1	SAKO	Instytut Maszyn Matematycznych (wówczas PAN)	XYZ	1962	Opracowanie języka oryginalne, słowa kluczowe polskie. Arytmetyka stałoprzecinkowa. Pamięć dwupoziomowa.
	SAKO	Instytut Maszyn Matematycznych (wówczas PAN)	ZAM 2	1963	
	SAKO	Instytut Maszyn Matematycznych	ZAM 41	1970	
2	ALGUM	Centralny Ośrodek Badań i Rozwoju Techniki Kolejnictwa	UMC 1	1964	Prosty język algorytmiczny wzorowany na FORTRAN II. Arytmetyka stałoprzecinkowa.
	ALGUM	Centralny Ośrodek Badań i Rozwoju Techniki Kolejnictwa	UMC 10	1965	
3	FORTRAN-B	Przemysłowy Instytut Telekomunikacyjny	CONTROL DATA 160-A	1969	Modyfikacja FORTRAN-A, zwiększająca kilkakrotnie jego szybkość.
4	FORPIT	Przemysłowy Instytut Telekomunikacyjny	CONTROL DATA 160-A	1970	Modyfikacja FORTRAN-B polegająca na wbudowaniu do systemu podprogramów generacji liczb losowych.
2. Języki typu ALGOL 60					
1	ZAM 21 ALGOL	Instytut Maszyn Matematycznych	ZAM 21	1966	Prawie pełny ALGOL 60.
2	URAL ALGOL	Centrum Obliczeniowe PAN	URAL 2	1966	Wersja wzorowana na GIER ALGOL III.
3	UMC ALGOL	Centralny Ośrodek Badań i Rozwoju Techniki Kolejnictwa	UMC 10	1965	Wersja nietypowa. Brak warunkowych wyrażeń arytmetycznych i logicznych, brak deklaracji „swith", brak rekursji procedur.
4	ZAM 41 ALGOL	Instytut Maszyn Matematycznych	ZAM 41	1968	Prawie pełny ALGOL 60. Adaptacja Whetstone Compiler dla KDF 9. Interpretacyjne wykonanie programu wynikowego.
5	ODRA ALGOL	Katedra Metod Numerycznych Uniwersytetu Warszawskiego	ODRA 1204	1968	Podzbiór IFIP. Parametrem formalnym procedury nie może być identyfikator procedury.
6	MINAL	Centralny Ośrodek Badań i Rozwoju Techniki Kolejnictwa	ODRA 1204	1967	Patrz uwagi dla UMC ALGOL.
7	ALGOL 1204	Katedra Metod Numerycznych Uniwersytetu Wrocławskiego	ODRA 1204	1970	Prawie pełny ALGOL 60.
8	KAR ALGOL	Instytut Fizyki Doświadczalnej Uniwersytetu Warszawskiego	KAR 65	1970	Brak deklaracji „switch", brak rekursji procedur.
3. Języki typu COBOL					
1	ZAM COBOL	Instytut Maszyn Matematycznych	ZAM 41	1969	Dość znaczny podzbiór języka COBOL
4. Inne języki					
1	MOST 1	Katedra Metod Numerycznych Uniwersytetu Wrocławskiego oraz Wrocławskie Zakłady Elektroniczne ELWRO	ODRA 1003	1965	Język klasy MRK III/ dla maszyny ELLIOTT 803.
2	MOST 2	Wrocławskie Zakłady Elektroniczne ELWRO	ODRA 1204	1968	
3	EOL 2	Instytut Maszyn Matematycznych	ZAM 41	1967	Język do manipulacji symbolami przeznaczony w szczególności do pisania translatorów.
4	LISP 1.5	Wydział Matematyki Uniwersytetu Warszawskiego	GIER	1967	Zanurzenie języka LISP w języku GIER ALGOL.
5	COMIT	Wydział Matematyki Uniwersytetu Warszawskiego	GIER	1969	Zanurzenie języka COMIT w GIER ALGOL. Język specjalistyczny, operujący aparatem automatów skończonych. Zanurzony w GIER ALGOL.
6	LOGOL	Wydział Matematyki Uniwersytetu Warszawskiego	GIER	1969	
7	LISP 1.5	Katedra Maszyn Matematycznych Wojskowej Akademii Technicznej	ODRA 1204	1970	System interpretacyjny zapisany w języku symbolicznym JAS.
8	CEMMA	Instytut Maszyn Matematycznych	ZAM 41	1969	Język symulujący działanie maszyny analogowej.
9	LJAPASS	Centrum Obliczeniowe PAN	ODRA 1204	1969	Pierwszy poziom języka LJAPASS rozszerzony o operacje nad zmiennymi
10	SYM 69	Centralny Ośrodek Badań i Rozwoju Techniki Kolejowej	ODRA 1204	1969	Zmodyfikowany podzbiór SIMULA służący do symulacji procesów nieciągłych.
Systemy operacyjne					
1	SO 41	Instytut Maszyn Matematycznych	ZAM 41	1966	Praca dwuprogramowa.
2	SO 141	Instytut Maszyn Matematycznych	ZAM 41	1968	Praca dwuprogramowa.
3	SOW	Wrocławskie Zakłady Elektroniczne ELWRO	ODRA 1204	1967	Praca jednoprogramowa.
4	SO KAR 65	Instytut Fizyki Doświadczalnej Uniwersytetu Warszawskiego	KAR 65	1970	Praca trójprogramowa.
5	TRAN	Instytut Techniczny Wojsk Lotniczych	ZAM 41	1970	Praca sześcioprogramowa. Transmisja międzyprogramowa i międzyzadaniowa
6	SODA	Centrum Obliczeniowe PAN	ODRA 1204	1970	Praca dwuaktywna.

JAN MADEY
Zakład Obliczeń Numerycznych
Uniwersytetu Warszawskiego

Mgr Jan Madey ukończył w roku 1964 Uniwersytet Warszawski, Wydział Matematyki i Fizyki. Od maja 1964 r. do maja 1970 r. pracował jako zastępca kierownika Zakładu Obliczeń Numerycznych UW, obecnie jest konsultantem Dyrekcji Zakładu, a jednocześnie (od grudnia 1968 r.) pełni funkcję zastępcy przewodniczącego Prezydium Oddziału Warszawskiego PKAPI. Przebywał za granicą na stażach naukowych: w Anglii na Uniwersytecie Londyńskim w Institute of Computer Science w okresie 1967—1968 i kilkakrotnie w Danii w firmie REGNECENTRALEN.
Zajmuje się szczególnie teorią i językami a także systemami programowania.

LEON ŚWIDERSKI
Zakład Obliczeń Numerycznych
Uniwersytetu Warszawskiego

Mgr inż. Leon Świderski ukończył w roku 1952 Politechnikę Warszawą, Wydział Geodezji. Po studiach pracował w Państwowym Przedsiębiorstwie Fotogrametrii. W latach 1963—1966 — w Instytucie Elektroniki w Centralnym Resortowym Ośrodku Przetwarzania Informacji, początkowo jako st. projektant, a następnie jako kierownik Zakładu Techniki Obliczeniowej, pełniąc jednocześnie funkcję zastępcy kierownika CROPI. Od roku 1966 pracuje w Zakładzie Obliczeń Numerycznych Uniwersytetu Warszawskiego jako zastępca kierownika d.s. technicznych, a od 1.VII.1969 r. jako zastępca dyrektora Zakładu.
Zajmuje się zagadnieniami związanymi z organizacją ośrodków obliczeniowych oraz niektórymi problemami związanymi z procesami produkcyjnymi.

Zakład Obliczeń Numerycznych Uniwersytetu Warszawskiego — historia i działalność

512.12:378.4(438.11):06.05

W artykule omówiono historię powstania Zakładu Obliczeń Numerycznych — ośrodka obliczeniowego Uniwersytetu Warszawskiego oraz ciekawsze uwagi i wnioski z jego siedmioletniej działalności.

Historia powstania Zakładu Obliczeń Numerycznych Uniwersytetu Warszawskiego

W listopadzie i grudniu 1963 roku odbyła się w Warszawie wystawa duńskiego komputera GIER. Wystawa ta została zorganizowana przez Uniwersytet Warszawski i duńską firmę REGNECENTRALEN z inicjatywy rektora Uniwersytetu prof. dr Stanisława Turskiego oraz dyrektora REGENCENTRALEN Nielsa Ivara Becha. Maszyna GIER wraz z zestawem urządzeń peryferyjnych została zainstalowana w Sali Kopernikowskiej Pałacu Kultury i Nauki w Warszawie.

Celem wystawy było zaprezentowanie maszyny i jej oprogramowania zarówno szerokiemu ogółowi, jak i specjalistom z dziedziny elektronicznej techniki obliczeniowej. Obok akcji typowo wystawowych, jak realizacja programów pokazowych, prelekcje dla zwiedzających, skoncentrowano uwagę na działalności naukowej.

GIER udostępniony był do normalnej eksploatacji pracownikom naukowym Uniwersytetu Warszawskiego, z czego najbardziej w tym czasie korzystali fizycy, matematycy, astronomowie i ekonomiści.

Ponadto w okresie trwania wystawy zostało zaproszonych do Warszawy kilku wybitnych specjalistów duńskich z zakresu języków i systemów programowania (m. in. Peter Naur), metod numerycznych i innych dziedzin związanych z ETO. Wygłosili oni szereg wykładów przeznaczonych dla pracowników Uniwersytetu i zaproszonych gości z innych ośrodków. W styczniu 1964 roku podjęto decyzję kupna maszyny GIER i utworzenia uniwersyteckiego ośrodka obliczeniowego, który podlegałby bezpośrednio Rektorowi Uniwersytetu Warszawskiego. Zakupiono następujący zestaw:

1. Jednostka centralna charakteryzująca się parametrami:

a. Pamięć operacyjna ferrytowa 1K, słowo 42 bity, czas dostępu 10 μsek.

b. Bęben magnetyczny 12 800 słów

c. Wbudowana arytmetyka stało- i zmiennoprzecinkowa, o czasach realizacji operacji (wraz z modyfikacją adresu) w mikrosek.:

- dodawanie: 49 (st. p.) 93 (zm. p.)
- mnożenie: 182 (st. p.) 167 (zm. p.)
- dzielenie: 271 (st. p.) 217 (zm. p.)

2. Pamięć ferrytowa buforowa 4K, czas dostępu 6—13 μsek.

3. Pamięć taśmowa karuzelowa, 64 szpule × 16 bloków × 512 słów = 524 288 słów.

4. Urządzenia wejścia-wyjścia:
a. Czytnik taśmy papierowej 8-kanałowej, RC 2000, szybkość czytania 2000 zn/sek, możliwość czytania taśmy 5-, 6- i 7-kanałowej.

b. Perforator taśmy 8-kanałowej, FACIT, PE-1500, szybkość perforowania 150 zn/sek.

c. Konsola wejścia-wyjścia, IBM typewriter, szybkość do 15 zn/sek.

5. Flexowritery do przygotowania taśmy i odczytywania wyników.

W styczniu i lutym 1964 roku przygotowywano pomieszczenie dla zestawu komputera GIER na IX piętrze Pałacu Kultury i Nauki. W tych i dwu następnych miesiącach maszyna była nadal eksploatowana przez pracowników naukowych Uniwersytetu, spełniających funkcję operatorów flexowriterów i maszyny, programistów, a nierzadko i techników-konserwatorów.

W chwili oficjalnego powołania Zakładu Obliczeń Numerycznych UW, tzn. dnia 1 maja 1964 roku, zespół pracowników liczył niespełna 10 osób. Kierownictwo Zakładu oddano w ręce profesora Turskiego, a jego ekipa składała się z paru matematyków (głównie studentów V roku), jednego inżyniera elektronika oraz dwóch pracowników do obsługi flexowriterów.

Ponieważ maszyna była eksploatowana w tym czasie już ponad jedną zmianę, więc funkcję operatorów maszyny spełniali w dużej części także i klienci Zakładu.

Lata 1964 i 1965 były niezwykle trudne dla bardzo młodego grona pracowników. Jednak ogromny entuzjazm, poświęcenie w pracy i zgranie całego zespołu pozwalały nadrabiać brak doświadczenia i umożliwiały — pod opieką władz Uniwersytetu — realizację zadań Zakładu. Już w tym czasie dają się zarysować cztery podstawowe kierunki działania Zakładu:

- merytoryczny
- szkoleniowo-dydaktyczny
- obliczeniowy
- wydawniczy.

Działalność merytoryczna, szkoleniowo-dydaktyczna i wydawnicza prowadzona była i jest nadal przy współudziale pracowników Instytutu Maszyn Matematycznych Uniwersytetu Warszawskiego, który kierowany jest także przez profesora Turskiego i powstał z uprzedniej Katedry Metod Numerycznych UW.

W roku 1966 nastąpił wydatny wzrost kadry Zakładu, który umożliwił wyodrębnienie grup pracowników o ściśle sprecyzowanym zakresie pracy. Pracownicy ze średnim wykształceniem, po przeszkoleniu wewnętrznym w Zakładzie, weszli w skład zespołów przygotowania danych i operowania maszyną, natomiast podstawowa kadra matematyków utworzyła zespół programowania. Zwiększona liczba inżynierów elektroników zapewniła stałą konserwację maszyny cyfrowej GIER.

W następnych latach zespół pracowników ulegał nieznacznemu powiększeniu, zapewniającemu rytmiczną, trzyzmianową pracę maszyny cyfrowej z tym, że podział na opisane grupy nie ulegał w zasadzie zmianom.

Podstawowe kierunki działalności Zakładu

I. Działalność merytoryczna

Matematycy Zakładu i Instytutu Maszyn Matematycznych UW opracowali teoretycznie i praktycznie szereg procesów obliczeniowych. Kierunki badań inspirowane były zarówno potrzebami użytkowników Zakładu, jak i własnymi zainteresowaniami. Wyniki opracowań są bezpośrednio stosowane w obliczeniach prowadzonych na maszynie GIER, wzbogaciły istotnie bibliotekę programów i procedur, a także znalazły swoje odbicie w działalności wydawniczej Zakładu. Wyróżnić można następujące grupy problemowe:

1. Metody numeryczne (głównie dla zadań algebry liniowej i nieliniowej oraz równań różniczkowych).

2. Systemy, języki i teoria programowania.

3. Optymalizacja.

4. Probabilistyka i statystyka.

5. Zagadnienia inżynieryjno-konstrukcyjne.

II. Działalność szkoleniowo-dydaktyczna

W akcji szkoleniowo-dydaktycznej można wydzielić następujące kierunki:

1. Ogólnodostępne kursy z zakresu programowania w językach zewnętrznych (głównie ALGOL 60 i jego reprezentacje) oraz metod numerycznych.

W ciągu 7 lat przeprowadzono 19 kursów, na których przeszkolono ponad 1200 osób. Kursy prowadzone były głównie w Warszawie dla pracowników szkół wyższych, instytutów i zakładów przemysłowych, ale również na życzenie instytucji spoza Warszawy organizowane były w innych miejscowościach (Gdańsk, Puławy, Włocławek). Uczestnictwo w kursach organizowanych na terenie Zakładu było bezpłatne. Szczegółowy wykaz kursów podaje zestawienie nr 1 (na końcu artykułu, przyp. red.). Ponadto współpracownicy Zakładu prowadzili wykłady i konsultacje na kilku kursach i wystawach zagranicznych, a mianowicie:

- Budapeszt (1964 r.)
- Praga (1965 r., 1966 r. — podczas wystawy INCOMEX 66, 1970 r., podczas wystaw INCOMEX 1970 r.)
- Sofia (1966 r., 1967 r. — dla Ministerstwa Transportu).

2. Szkolenie studentów Uniwersytetu Warszawskiego.

Poważną ilość czasu pracowników i zestawu maszyny GIER przeznacza Zakład corocznie na szkolenie studentów, głównie matematyki, a także fizyki, astronomii i ekonometrii. Studenci ci odbywają cykliczne ćwiczenia przy maszynie. Również szereg magistrantów wykorzystuje maszynę do prac dyplomowych.

3. Praktyki studenckie i staże.

Corocznie Zakład przyjmuje około 20 studentów na 4-tygodniowe praktyki studenckie. Umowy są zawierane zarówno z wydziałami Uniwersytetu Warszawskiego, jak i z uniwersytetami Wrocławskim, Poznańskim, Toruńskim, a od roku 1970 także z Uniwersytetem im. Humboldta w Berlinie.

Ponadto Zakład przyjmuje sporadycznie na długie staże (od paru miesięcy do roku) pracowników naukowych różnych instytucji i uczelni.

4. Pokazy i prelekcje.

Przeciętnie kilka razy miesięcznie przeprowadzane są pokazy maszyny, połączone z prelekcjami. Przeznaczone są one głównie dla młodzieży szkolnej, ale także i dla pracowników i studentów szkół wyższych oraz przedstawicieli różnych instytucji.

III. Działalność obliczeniowa

Wyposażenie Zakładu stanowi zestaw maszyny cyfrowej GIER wzbogacony w stosunku do omówionego na wstępie — o następujące urządzenia:

1. Drukarka wierszowa ANELEX, 64 różne znaki, szybkość pracy do 667 wierszy/min, 160 znaków w wierszu

2. Urządzenie wyjściowe optyczne (*display*), wyświetlające 2000 znaków alfanumerycznych/min.

3. Maszyny pomocnicze, w tym

a. Flexowritery FRIDEN typ SFD (5 sztuk)

b. Flexowritery FRIDEN typ 2303 (2 sztuki)

c. Automat piszący OPTIMA typ 527 (1 sztuka)

d. Urządzenie do reprodukcji taśmy ZON-1301 (1 sztuka).

4. Zapasowe urządzenia i elementy (czytnik taśmy, perforator, dysk do karuzeli).

Z wymienionych urządzeń, dwa wykonane są całkowicie własnymi środkami w Zakładzie Obliczeń Numerycznych UW. Są to:

- *display* wykonany w roku 1967 jako pierwszy w kraju. Informacje wyświetlane są na ekranie lampy kineskopowej w formie napisów, cyfr lub wykresów. Mogą być one wyświetlane wielokrotnie, co umożliwia bezpośredni odczyt lub jednorazowo i wtedy są rejestrowane na mikrofilmie. Wyświetlane teksty mogą zawierać 32 wiersze po 64 znaki w wierszu. Znaki mogą być wyświetlane w trzech poziomach jednego wiersza, co umożliwia pisanie indeksów i potęg oraz w dwu jasnościach.

- ZON-1301 — urządzenie służące do kopiowania, sprawdzania i poprawiania taśmy perforowanej z jednoczesną kontrolą parzystości. ZON-1301 może pracować na taśmie 8-, 7-, 6- i 5-kanałowej.

Zasadniczo używanymi językami programowania są: język symboliczny SLIP oraz reprezentacje ALGOLu 60 — GIER ALGOL III i GIER ALGOL 4. Reprezentacje te — a zwłaszcza druga z nich — stanowią istotne rozszerzenie języka wzorcowego przy minimalnych ograniczeniach. Translatory ALGOLu na maszynę GIER uznawane są przez fachowców za jedne z najlepszych na świecie, biorąc pod uwagę parametry maszyny. Zarówno efektywność kodu wynikowego, jak i szybkość translacji jest bardzo duża, a diagnostyka w czasie tłumaczenia prawie pełna.

Niezależnie od wymienionych języków, maszyna wyposażona jest w translatory kilku specjalnych języków (LISP, LOGOL, EULER, COMIT, IMP) opracowane głównie w Uniwersytecie Warszawskim oraz w program FOGOL, tłumaczący z FORTRANU II na GIER ALGOL III. Języki te są jednak rzadziej wykorzystywane w użytkowej eksploatacji maszyny.

Eksploatacja maszyny GIER w latach 1964—1970 jest zilustrowana w tablicy.

Jak widać z tablicy, pierwsze dwa lata działalności Zakładu mają dosyć wysoki wskaźnik czasu nieefektywnego, co wynikało z trudności obiektywnych — zbyt szczupły i niedostatecznie doszkolony personel operatorski. W okresie tym maszyna była eksploatowana na niepełne dwie zmiany. Pełną pracę dwuzmianową osiągnął Zakład w roku 1966, a od 1 października 1966 przeszedł na pracę trzyzmianową.

Stały wzrost zapotrzebowania na obliczenia spowodował, że od roku 1968 Zakład eksploatuje maszynę także w niektóre niedziele i święta. W efekcie więc, przy przeciętnie 306 dniach roboczych w roku, maszyna była eksploatowana przez 319,5 doby, w r. 1968 346 dób w r. 1969 i 352,5 doby w roku 1970.

W ogólnej liczbie godzin pracy maszyny, około 40% wykorzystane zostało przez Uniwersytet Warszawski na działalność dydaktyczną i naukową. Szczególnie Instytut Fizyki, a także Wydział Chemii i Instytut Astronomii zgłaszały duże potrzeby obliczeniowe. W czasie 7-letniej działalności, spośród wielu użytkowników spoza Uniwersytetu, wyłoniła się grupa stałych klientów Zakładu (p. zestawienie nr 2).

IV. Działalność wydawnicza

Działalność wydawnicza prowadzona jest w dwóch kierunkach:

- opracowywanie podręczników z zakresu elektronicznej techniki obliczeniowej
- publikowanie ciekawszych prac w postaci zeszytów pod nazwą „Sprawozdania Instytutu Maszyn Matematycznych i Zakładu Obliczeń Numerycznych Uniwersytetu Warszawskiego".

Opracowane dotąd podręczniki dotyczyły języków programowania i przeznaczone były głównie dla studentów i uczestników kursów. Prace publikowane w zeszytach „Sprawozdań" stanowią informacje o ciekawych wynikach, uzyskanych przez zespół Zakładu i Instytutu Maszyn Matematycznych UW oraz bardziej interesujących pracach wykonanych w Zakładzie. Informacje te przeznaczone są dla szerokiego kręgu użytkowników maszyn cyfrowych.

Pełny wykaz wydanych podręczników i „Sprawozdań" znajduje się w zestawieniu nr 3 (na końcu artykułu).

Struktura organizacyjna Zakładu

W pierwszym etapie istnienia Zakładu, współpraca z użytkownikami miała charakter *„open shop"*, tzn. zasadniczo użytkownicy sami programowali i mieli bezpośredni dostęp do maszyny, a pracownicy Zakładu głównie szkolili i udzielali konsultacji. Było to konieczne ze względu na szczupłość grona personelu Zakładu i początkowo — brak doświadczeń. Z drugiej strony sprawna realizacja takiego trybu była możliwa jedynie dlatego, że:

- obsługa operatorska maszyny jest prosta
- urządzenia wejścia-wyjścia są niezawodne
- język programowania (reprezentacja ALGOLu) jest łatwy do nauki i wykorzystania
- większość użytkowników rekrutowała się z pracowników nauki i miała w związku z tym wysokie kwalifikacje.

Taki tryb współpracy miał dwie istotne i oczywiste wady.

Po pierwsze, obsługiwanie urządzenia zewnętrznego przez niedostatecznie przeszkoloną technicznie osobę zwiększało szansę uszkodzenia tegoż urządzenia i awarii systemu.

Po drugie, dało się zauważyć duże marnotrawienie czasu jednostki centralnej ze względu na powolną obsługę oraz stosowany zazwyczaj konwersacyjny tryb realizacji programów.

Tablica

Rok	Czas ogółem	Czas nieefekt.	%	Czas efekt.	%	Średnia miesięczna liczba godzin pracy EMC
1964	2080,00	433,02	21,8	1626,98	78,2	297,14
1965	4318,47	515,03	11,9	3803,44	88,1	359,87
1966	4852,12	330,22	6,8	4521,90	93,2	404,34
1967	6743,53	351,89	5,2	6391,64	94,8	561,96
1968	7667,46	383,91	5,0	7283,55	95,0	638,95
1969	8308,25	371,07	6,9	7737,18	93,1	692,35
1970	8462,51	325,25	3,8	8137,26	96,2	705,21

Dlatego też pierwszym krokiem organizacyjnym było przejmowanie od użytkowników bezpośredniej obsługi maszyny przez doszkalany personel Zakładu.

Drugim krokiem było ograniczanie konwersacyjnego trybu realizacji programów i przechodzenie na imitowanie trybu wsadowego: użytkownicy zostawiali swoje programy wraz z dokładną instrukcją dla operatora maszyny u dyspozytora Zakładu i obliczanie tych programów odbywało się bez obecności ich autorów przy maszynie. Umożliwiało to efektywniejsze wykorzystanie jednostki centralnej, ale było często niechętnie przyjmowane przez osoby programujące, ponieważ narzucało na nie dodatkowe obowiązki (pełne zaprojektowanie reakcji na wszystkie możliwe drogi pracy programu, oszacowanie maksymalnego czasu realizacji programu) oraz wykluczało dialog z maszyną.

Dlatego też do chwili obecnej, oprócz pozostawiania programów do liczenia u dyspozytora, istnieje możliwość także zarezerwowania określonego przedziału czasu maszyny oraz obecności przy realizacji programu.

Wąskim gardłem w efektywnym wykorzystaniu systemu okazała się mało wydajna współpraca z perforatorem jako urządzeniem wyjściowym. Zarówno niewielka szybkość perforowania taśmy w stosunku do szybkości pracy maszyny powodowała przestoje jednostki centralnej, jak i konieczność odczytywania wyników na flexowriterach blokowała na długie okresy czasu te urządzenia.

Kolejnymi krokami usprawniającymi wykorzystanie zestawu maszyny był zakup drukarki, konstrukcja *displaya*, konstrukcja urządzenia ZON-1301, zakup nowych flexowriterów i OPTIMY. Ponadto z chwilą wprowadzenia do eksploatacji translatora języka GIER ALGOL 4 oraz systemu operacyjnego HELP 3, stało się możliwe łatwe wykorzystanie pamięci karuzelowej do przechowywania różnych wersji translatorów, programów użytkowych i bibliotecznych, co z kolei wydatnie zmniejszyło czas tracony na wprowadzenie informacji z taśmy papierowej.

Kadra merytoryczna Zakładu rekrutuje się zasadniczo z absolwentów sekcji metod numerycznych Uniwersytetu Warszawskiego. Podstawowy tryb współpracy z użytkownikami polega na szkoleniu, konsultacjach oraz przygotowywaniu programów i procedur o charakterze standardowym. Jednakże dla dużej grupy zleceniodawców zarówno z Uniwersytetu Warszawskiego, jak i innych instytucji, wykonywane też są pełne prace obliczeniowe poprzez analizę zagadnienia i wybór algorytmu, do programowania i realizacji na maszynie.

W chwili obecnej Zakład zatrudnia 37 osób na pełnych etatach oraz 25 w niepełnym wymiarze godzin. Druga grupa osób obejmuje głównie pracowników naukowo-dydaktycznych Instytutu Maszyn Matematycznych współpracujących merytorycznie z Zakładem.

W pierwszych latach swej działalności — Zakład będący jednostką na własnym rozrachunku — stosował tabele płac przewidziane w układzie zbiorowym dla przemysłu metalowego. Z dniem 1 lipca 1969 roku Zakład uzyskał zgodę Ministra Oświaty i Szkolnictwa Wyższego na stosowanie postanowień Uchwały nr 215/68 w sprawie zasad wynagradzania pracowników w ośrodkach przetwarzania danych i obliczeń numerycznych.

Funkcję dyrektora Zakładu pełni prof. dr Stanisław Turski.

Zestawienie nr 1

Działalność szkoleniowo-dydaktyczna

Jeszcze przed decyzją o powołaniu Zakładu Obliczeń Numerycznych przeprowadzono w październiku 1963 r. kurs języka ALGOL 60
wykładowcą był dr W. Turski
uczestniczyło 30 osób.

Niżej podajemy wykaz wszystkich późniejszych kursów:

Kurs nr 1 — przeprowadzony w marcu 1964 r.
w Zakładzie Obliczeń Numerycznych UW
Wykładowca: dr W. Turski
Temat: ALGOL 60, GIER ALGOL III
Uczestniczyło 85 osób

Kurs nr 2 — przeprowadzony w maju 1964 r.
w Zakładzie Obliczeń Numerycznych UW
Temat: ALGOL 60 i GIER ALGOL III
Wykładowca: mgr J. Madey
Uczestniczyło 60 osób

Kurs nr 3 — przeprowadzony w czerwcu 1964 r.
w Gdańsku dla Sekcji Okrętowców Stowarzyszenia Inżynierów i Techników Mechaników Polskich
Temat: ALGOL 60
Wykładowca: dr A. Kiełbasiński
mgr J. Madey
Uczestniczyło 25 osób.

Kurs nr 4 — przeprowadzony w listopadzie 1964 r.
w Zakładzie Obliczeń Numerycznych UW
Temat: ALGOL 60 i GIER III
Wykładowca: mgr J. Madey
Uczestniczyło 60 osób

Kurs nr 5 — przeprowadzony w okresie marzec-czerwiec 1965 r.
w Zakładzie Obliczeń Numerycznych UW
Temat: ALGOL 60 i GIER ALGOL III
Wykładowca: mgr J. Madey
Uczestniczyło 131 osób.

Kurs nr 6 — przeprowadzony w okresie kwiecień-maj 1966 r.
w Zakładzie Obliczeń Numerycznych UW
Temat: ALGOL 60 i GIER ALGOL III
Wykładowca: mgr J. Madey
Uczestniczyło 112 osób.

Kurs nr 7 — przeprowadzony w okresie kwiecień-maj 1967 r.
w Zakładzie Obliczeń Numerycznych UW
Temat: ALGOL 60 i ELEMENTY METOD NUMERYCZNYCH
Wykładowca: mgr W. Pankiewicz
Uczestniczyły 122 osoby.

Kurs nr 8 — przeprowadzony w kwietniu 1968 r.
w Ośrodku Zastosowań Elektronicznej Techniki Cyfrowej ZBPB Etoprojekt
Temat: ALGOL 60, GIER ALGOL III, GIER ALGOL 4
Wykładowca: mgr M. Tessarowicz
Uczestniczyło 25 osób.

Kurs nr 9 — przeprowadzony w okresie kwiecień-maj 1968 r.
w Zakładzie Obliczeń Numerycznych UW
Temat: ALGOL 60, GIER ALGOL 4
Wykładowca: mgr M. Tessarowicz
Uczestniczyło 80 osób.

Kurs nr 10 — przeprowadzony w okresie kwiecień-maj 1969 r.
w Zakładzie Obliczeń Numerycznych UW
Temat: PROGRAMOWANIE W JĘZYKACH ALGOL 60 i GIER ALGOL 4
Wykładowca: mgr M. Mirkowska
Uczestniczyło 120 osób.

Kurs nr 11 — przeprowadzony w okresie kwiecień-maj 1969 r.
w Zakładach Azotowych — Puławy
Temat: PROGRAMOWANIE W JĘZYKACH SLANG 1 i ALGOL 5

Wykładowcy: mgr M. Grzymkowski
mgr M. Tessarowicz
Uczestniczyło 38 osób.

Kurs nr 12 — przeprowadzony w okresie kwiecień-maj 1969 r.

w Instytucie Techniki Budowlanej
Temat: ALGEBRA WYŻSZA, METODY NUMERYCZNE, ALGOL 60
Wykładowcy: mgr Anna Jurkiewicz
mgr Anna Paluszkiewicz
mgr Jan Madey
Uczestniczyło 30 osób.

Kurs nr 13 — przeprowadzony w okresie listopad-grudzień 1969 r.

w Zakładzie Obliczeń Numerycznych UW
Temat: PROGRAMOWANIE W JĘZYKACH: ALGOL 60 i GIER ALGOL 4
Wykładowca: mgr A. Kreczmar
Uczestniczyło 145 osób.

Kurs nr 14 — (specjalny zamknięty) przeprowadzony w listopadzie 1969 r.

w Ośrodku Badawczym Sprzętu Chemicznego
Temat: PROGRAMOWANIE W JĘZYKACH: ALGOL 60 i GIER ALGOL III Z ELEMENTAMI METOD NUMERYCZNYCH
Wykładowca: mgr W. Pankiewicz
Uczestniczyło 9 osób.

Kurs nr 15 — przeprowadzony w okresie marzec-czerwiec 1970 r.

w Instytucie Techniki Budowlanej
Temat: ALGEBRA WYŻSZA, ALGOL 60 I METODY NUMERYCZNE
Wykładowcy: mgr A. Jurkiewicz
mgr L. Czaja
mgr H. Woźniakowski
Uczestniczyło: 20 osób.

Kurs nr 16 — przeprowadzony w okresie kwiecień-czerwiec 1970 r.

w Zakładzie Obliczeń Numerycznych UW
Temat: GIER ALGOL 4
Wykładowca: mgr L. Czaja
Uczestniczyło: 60 osób.

Kurs nr 17 — przeprowadzony w maju 1970 r.

w Centralnym Biurze Konstrukcyjnym Urządzeń Budowlanych
Temat: WPROWADZENIE W ELEKTRONICZNĄ TECHNIKĘ OBLICZENIOWĄ.
Wykładowca: mgr A. Jurkiewicz
Uczestniczyło: 11 osób.

Kurs nr 18 — przeprowadzony dla pracowników Zakładów Azotowych — Włocławek złożony z dwóch etapów:

I. w maju 1970 r. we Włocławku
Temat: PROGRAMOWANIE W JĘZYKACH ALGOL 5 I SLANG
Wykładowcy: mgr J. Madey
mgr M. Grzymkowski
Uczestniczyło: 10 osób.
II. w okresie listopad — grudzień 1970 r. w Zakładzie Obliczeń Numerycznych UW
Temat: SYSTEM OPROGRAMOWANIA RC 4000
Wykładowcy: mgr M. Grzymkowski
mgr J. Madey
mgr M. Tessarowicz
Uczestniczyło: 11 osób.

Kurs nr 19 — przeprowadzony w okresie listopad-grudzień 1970 r.

w Zakładzie Obliczeń Numerycznych UW
Temat: PROGRAMOWANIE W JĘZYKU GIER ALGOL 4
Wykładowca: mgr A. Kreczmar
Uczestniczyło: 24 osoby.
Dodatkowo, po zakończeniu kursu, zostały zorganizowane trzy wykłady dotyczące: OBLICZEŃ NUMERYCZNYCH Z UWZGLĘDNIENIEM OPROGRAMOWANIA MASZYNY GIER
Wykładowca: doc. dr A. Kiełbasiński

Zestawienie nr 2

Wykaz stałych użytkowników Zakładu Obliczeń Numerycznych

1. Politechnika Warszawska
2. Uniwersytet im. M. Kopernika w Toruniu
3. Uniwersytet Jagielloński
4. Uniwersytet Łódzki
5. Szkoła Główna Gospodarstwa Wiejskiego
6. Politechnika Łódzka
7. Instytut Chemii Fizycznej PAN
8. Zakład Astronomii PAN
9. Instytut Podstawowych Problemów Techniki PAN
10. Zakład Geofizyki PAN
11. Instytut Fizyki PAN
12. Instytut Chemii Organicznej PAN
13. Centrum Obliczeniowe PAN
14. Instytut Filozofii i Socjologii PAN
15. Instytut Fizyki Jądrowej w Krakowie
16. Instytut Lotnictwa
17. Instytut Chemii Ogólnej
18. Państwowy Instytut Hydrologiczno-Meteorologiczny Oddz. Morski w Gdyni
19. Instytut Techniki Budowlanej
20. Instytut Transportu Samochodowego
21. Przedsiębiorstwo Poszukiwań Geofizycznych
22. Biuro Generalnego Dostawcy Kompletnych Obiektów Przemysłowych ZEMAK
23. Stołeczne Przedsiębiorstwo Energetyki Cieplnej
24. Centrum Elektronicznej Techniki Obliczeniowej Przemysłu Budowlanego ETOB
25. Biuro Studiów i Projektów Energetycznych ENERGOPROJEKT
26. Centralny Ośrodek Konstrukcyjno-Badawczy Przemysłu Motoryzacyjnego
27. Biuro Studiów i Projektów Wzorcowych Zaopatrzenia Rolnictwa w Wodę WODROL-PROJEKT
28. Zakłady Transportowe Budownictwa WARSZAWA

Zestawienie nr 3

Działalność wydawnicza Zakładu Obliczeń Numerycznych UW:

I. Podręczniki z zakresu elektronicznej techniki obliczeniowej:
1. Jan Madey — ALGOL 60, GIER ALGOL III
czerwiec 1965 r.
Nakład 1000 egz.
2. A. Jurkiewicz, J. Madey, A. Paluszkiewicz — ALGOL 60
kwiecień 1969 r.
Nakład 1500 egz. + 600 egz.
3. Praca zbiorowa — PROGRAMOWANIE W JĘZYKU WEWNĘTRZNYM MASZYNY GIER (podręcznik programowany).
październik 1968 r.
Nakład 600 egz.
4. E. Kaczmarska, E. Czyżo, M. Grzymkowski — KOMPEDIUM WIADOMOŚCI O MASZYNIE GIER
maj 1969 r.
Nakład 600 egz.
5. L. Czaja — GIER ALGOL 4 (w druku).

II. Wykaz zeszytów serii:

„SPRAWOZDANIA INSTYTUTU MASZYN MATEMATYCZNYCH I ZAKŁADU OBLICZEŃ NUMERYCZNYCH UNIWERSYTETU WARSZAWSKIEGO"

zeszyt nr 1 — Andrzej Kiełbasiński
PROCEDURA PERMUTACJE
styczeń 1967 r.
Nakład 550 egz.

zeszyt nr 2 — Antoni Kreczmar, Andrzej Salwicki
WSPÓŁPRACA MASZYNY GIER Z BUFOREM I KARUZELĄ
styczeń 1967 r.
Nakład 300 egz.

zeszyt nr 3 — Ewa Kaczmarska, Andrzej Salwicki
GENEROWANIE LICZB PSEUDOLOSOWYCH
maj 1967 r.
Nakład 550 egz.

zeszyt nr 4 — Mirosława Mirkowska, Andrzej Salwicki
SORTOWANIE NA MASZYNIE CYFROWEJ
czerwiec 1967 r.
Nakład 550 egz.

zeszyt nr 5 — Anna Jurkiewicz
ZAGADNIENIA TRANSPORTOWE (KOSZT I CZAS)
wrzesień 1967 r.
Nakład 550 egz.

zeszyt nr 6 — Michał Tessarowicz
MINIMALIZACJA FUNKCJI WIELU ZMIENNYCH
wrzesień 1967 r.
Nakład 550 egz.

zeszyt nr 7 — Mirosław Grzymkowski
PROCEDURY KOMBINATORYCZNE
listopad 1967 r.
Nakład 550 egz.

zeszyt nr 8 — Emanuel Czyżo
SUMOWANIE SZEREGÓW WOLNO ZBIEŻNYCH
listopad 1967 r.
Nakład 300 egz.

zeszyt nr 9 — Bronisław Jankowski
ANALIZA SIECI CZYNNOŚCI METODĄ PERT
luty 1968 r.
Nakład 550 egz.

zeszyt nr 10 — Anna Jurkiewicz
ZAGADNIENIE TRANSPORTOWE (MINIMALIZACJA PUSTYCH PRZEBIEGÓW
marzec 1968 r.
Nakład 550 egz.

zeszyt nr 11 — M. Gardias, A. Kreczmar, M. Tessarowicz
OBLICZANIE SIŁ, MOMENTÓW NAPRĘŻEŃ I PRZEMIESZCZEŃ DLA RUROCIĄGÓW ENERGETYCZNYCH
maj 1968 r.
Nakład 300 egz.

zeszyt nr 12 — Henryk Woźniakowski
NUMERYCZNA REALIZACJA METODY BAIRSTOWA
maj 1968 r.
Nakład 300 egz.

zeszyt nr 13 — Feliks Pietras
NUMERYCZNE ROZWIĄZYWANIE RÓWNANIA POISSONA
maj 1968 r.
Nakład 300 egz.

zeszyt nr 14 — Ewa Karafilowska
ANALIZA KOSZTÓW METODĄ PERT
czerwiec 1968 r.
Nakład 300 egz.

zeszyt nr 15 — Anna Jurkiewicz
ZAGADNIENIE TRANSPORTOWE (PLANOWANIE PRZEWOZÓW)
wrzesień 1968 r.
Nakład 300 egz.

zeszyt nr 16 — Andrzej Kiełbasiński
NUMERYCZNE OBLICZANIE WARTOŚCI WIELOMIANU ALGORYTMEM HORNERA
wrzesień 1968 r.
Nakład 300 egz.

zeszyt nr 17 — Ewa Kaczmarska
AUTOMATYCZNE ŁAMANIE TEKSTÓW WYDAWNICZYCH
wrzesień 1968 r.
Nakład 300 egz.

zeszyt nr 18 — Barbara Kiełbasińska
WYZNACZANIE ZER WIELOMIANU ZESPOLONEGO METODĄ PARABOL
wrzesień 1968 r.
Nakład 300 egz.

zeszyt nr 19 — W. Dubnicki, K. Zorychta
O DWÓCH METODACH PROGRAMOWANIA WYPUKŁEGO
listopad 1968 r.
Nakład 300 egz.

zeszyt nr 20 — Wacław Pankiewicz
KONTROLOWANY ALGORYTM ROZWIĄZYWANIA UKŁADÓW RÓWNAŃ LINIOWYCH
styczeń 1969 r.
Nakład 300 egz.

zeszyt nr 21 — Krystian Zorychta
O PEWNYM ODPOWIEDNIKU LINIOWYM KRYTERIUM ILORAZOWEGO
sierpień 1969 r.
Nakład 300 egz.

zeszyt nr 22 — Wacław Pankiewicz
ROZWIĄZYWANIE UKŁADÓW RÓWNAŃ NIELINIOWYCH
październik 1969 r.
Nakład 300 egz.

zeszyt nr 23 — Antoni Kreczmar
KRYTERIUM ISTNIENIA I ALGORYTM UKŁADANIA ROZKŁADÓW ZAJĘĆ SZKOLNYCH
kwiecień 1970 r.
Nakład 300 egz.

zeszyt nr 24 — Emanuel Czyżo
KOMPUTERY W TEORII GRUP. PROCES WYBIERANIA DLA SŁÓW DODATNICH
wrzesień 1970 r.
Nakład 300 egz.

zeszyt nr 25 — Andrzej Kiełbasiński
OSZACOWANIE BŁĘDU W METODZIE ELIMINACJI
(w druku)

zeszyt nr 26 — Jan Madey
NIEJEDNOZNACZNOŚCI ALGOLU 60 I SPOSOBY ICH USUNIĘCIA W GIER ALGOLU 4.
(w druku).

JAN MAŁUSZYŃSKI, JACEK WITASZEK
Instytut Maszyn Matematycznych
Warszawa

Wiedeńska metoda opisu języków programowania

681.322.06

Artykuł omawia na przykładzie bardzo prostego języka pewną metodę opisu języków programowania uwzględniając szczególnie aspekty związane z opisem semantyki. Semantykę języka definiuje się za pomocą abstrakcyjnej maszyny, która odwzorowuje obiekty zwane abstrakcyjnymi programami w ciągi swoich stanów. Przekształcenie konkretnego programu, który jest napisem określonym przez konkretną składnię w abstrakcyjny program, polega na odrzuceniu wszystkich szczegółów związanych z tekstowym przedstawieniem, zachowując w zasadzie strukturę programu. Metoda pozwala opisywać semantykę języków ściśle i równocześnie dość przejrzyście.

Wstęp

Jednym z istotnych problemów związanych z definiowaniem języków programowania jest zagadnienie opisu semantyki, tj. znaczenia wyrażeń tych języków. Zanim przejdziemy do bardziej ścisłych rozważań, warto uświadomić sobie co rozumiemy intuicyjnie pod pojęcem semantyki. Otóż na ogół uważamy, że ktoś zna język programowania, jeżeli mając program potrafi powiedzieć, jak będzie działała maszyna w trakcie wykonywania tego programu. Używając ściślejszej terminologii to samo można wypowiedzieć następująco: przez semantykę danego języka programowania rozumiemy funkcję, która każdemu wyrażeniu tego języka (wyrażenie potocznie nazywa się programem) przypisuje pewien proces (lub zbiór procesów) zachodzący w wybranej maszynie. Proces taki nazywać będziemy interpretacją odpowiadającego mu wyrażenia. Tak rozumianą semantykę języka programowania definiuje np. każdy translator wraz z maszyną, na której został zrealizowany. Translator przekształca wyrażenia języka na programy w kodzie maszyny, a ta z kolei programy te wykonuje. Taka definicja jest jednak na ogół zupełnie nieczytelna dla człowieka. W metodzie opisu języków programowania, której pewne aspekty zostaną omówione w niniejszym artykule, przyjęto podobną koncepcję, główny akcent położono jednak nie tylko na ścisłość, lecz również na komunikatywność tego opisu. Metodę tę [2] opracowano w ośrodku IBM w Wiedniu, w dalszym ciągu będziemy ją nazywać metodą wiedeńską. Została ona praktycznie zastosowana do opisu języków PL-1 i ALGOL 60 [1] [3].

Punktem wyjścia dla naszych rozważań jest język, którego wyrażenia są napisami określonymi za pomocą formalizmu składni nazwanej składnią konkretną. Zamiast programu w kodzie maszyny występuje tutaj tzw. program abstrakcyjny zapisywany krótko i czytelnie w formalizmie składni abstrakcyjnej; zamiast maszyny rzeczywistej określa się maszynę abstrakcyjną, realizującą programy abstrakcyjne.

Autorzy metody wiedeńskiej opracowali aparat formalny tzw. obiektów abstrakcyjnych, nadający się zarówno do opisu składni konkretnej, jak i składni abstrakcyjnej oraz maszyny. Aparat ten pozwala w sposób jednolity a zarazem komunikatywny opisywać języki programowania. Aparat obiektów abstrakcyjnych jest aparatem opisu strukturalnego. Może on być stosowany również i do innych celów nie związanych z tematyką tego artykułu.

W artykule wskazano różnice i podobieństwa między ogólnie znaną konkretną składnią BNF a abstrakcyjną składnią stosowaną w omawianej metodzie. Przyjmując za podstawę konkretną składnię przykładowego języka J omówiono składnię abstrakcyjną tego języka oraz maszynę, która odwzorowuje programy języka J na procesy.

Składnia konkretna

Składnia konkretna języków programowania opisywana jest często w postaci zbioru reguł Backusa-Naura. Ten formalizm jest dość znany, więc objaśnimy go tylko na przykładzie bardzo prostego języka J, który stanowi źródło wszystkich przykładów w tym artykule. Bardziej wyczerpujące informacje na temat notacji BNF znaleźć można w pracach [4] [5].

<program> : : = <blok>
<blok> : : = **begin** <lista dekl> ; <lista instr> **end**
<lista dekl> : : = <deklaracja> | <lista dekl> ; <deklaracja>
<deklaracja> : : = **integer** <zmienna> | **logical** <zmienna>
<lista instr> : : = <instr> | <lista instr> ; <instr>
<instr> : : = <instr podst> | <blok>
<instr podst> : : = <zmienna> : = <wyraż>
<wyraż> : : = <stała> | <zmienna> | <operacja>
<zmienna> : : = <identyfikator>
<operacja> : : = <wyraż> <op> <wyraż>
<stała> : : = **true** | **false** | <liczba>
<liczba> : : = <cyfra> | <liczba> <cyfra>
<identyfikator> : : = <litera> | <identyfikator> <litera> | <identyfikator> <cyfra>
<op> : : = + | — | ∧ | ∨
<litera> : : = x | y | z
<cyfra> : : = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

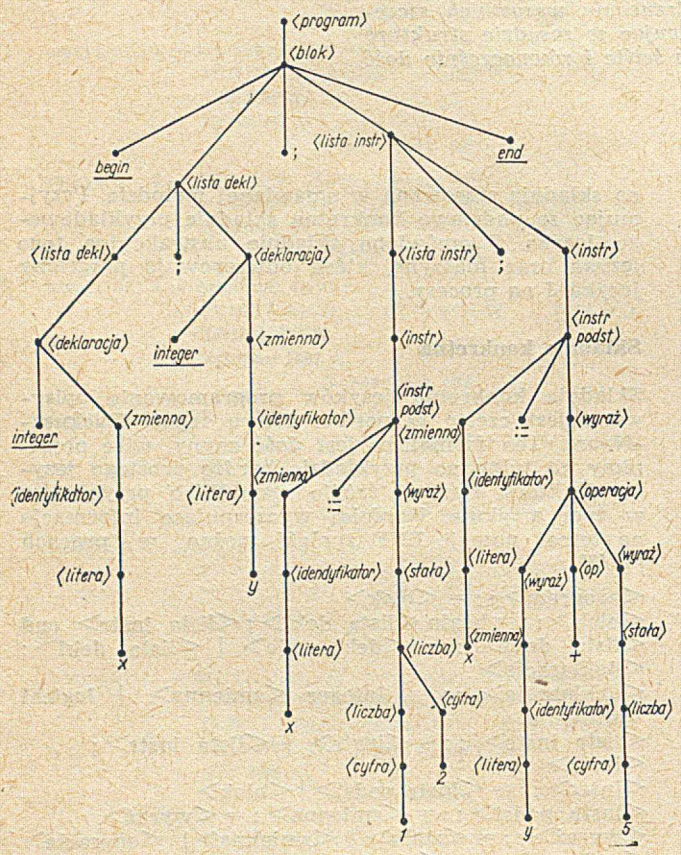
Alfabet terminalny składa się z symboli: **begin, end, true, false, integer, logical,** ;, : =, +, —, ∧, ∨, x, y, z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Alfabet nieterminalny stanowią wszystkie lewe strony reguł. Wyróżnionym symbolem nieterminalnym jest <program>. Wystąpienia symbolu metajęzykowego | dzielą prawe strony reguł na składniki. Zastępując dowolny symbol nieterminalny występujący w pewnym słowie przez dowolny składnik takiej reguły, której lewą stronę stanowi wybrany symbol nieterminalny otrzymujemy nowe słowo. Jeżeli słowo to zawiera jeszcze symbole nieterminalne, to można je znowu przekształcić w podobny sposób. Wszystkie słowa terminalne, które można otrzymać w ten sposób z wyróżnionego symbolu <program> nazywają się wyrażeniami języka opisywanego przez dany zbiór reguł. Wyprowadzimy pewne słowo przykładowego języka:

<program> => <blok> => **begin** <lista dekl>;
<lista instr> **end** => **begin**

<lista dekl> ; <deklaracja> ;
<lista instr>
end => ... =>

=> **begin**
integer x ; **integer** y ;
y := 12 ;
x := y + 5
end

Powyższy proces przekształcania symbolu wyróżnionego w wyrażenie języka można również przedstawić w postaci grafu zwanego drzewem wywodu (rys. 1). Wierzchołki tego drzewa przyporządkowane



Rys. 1

są symbolom występującym w przekształcalnych słowach. Dwa wierzchołki połączone są krawędzią wtedy i tylko wtedy, gdy jeden z nich jest odpowiednikiem zastępowanego symbolu nieterminalnego a drugi — symbolu występującego w słowie zastępującym eliminowany symbol. Tekstowy porządek symboli w tym słowie, określony przez odpowiednią regułę, uwidoczniono na rysunku przez rozmieszczenie odpowiadających im wierzchołków kolejno od lewej strony ku prawej.

Kolejność stosowania reguł (kolejność zastępowania symboli nieterminalnych), określająca postać kolejno przekształcanych słów, nie znajdzie odzwierciedlenia graficznego. Drzewo wywodu programu jest więc jednakowe dla wszystkich wywodów różniących się jedynie kolejnością stosowania produkcji. Język nazywa się jednoznacznym, jeśli każdy program ma tylko jedno drzewo wywodu. Warunek jednoznaczności języka jest na ogół zachowywany. Drzewo wywodu opisuje strukturę syntaktyczną danego programu. Jednostkami syntaktycznymi są te fragmenty programu, które są rozwinięciami symboli nieterminalnych. Jak widać symbole nieterminalne reguł powinny być dobierane z uwzględnieniem przewidywanej interpretacji wyrażeń języka.

Alfabet terminalny określony przez każdy zbiór reguł BNF jest z założenia skończony. Pod pewnymi względami wygodniejszy byłby często nieskończony alfabet terminalny; np. określając interpretację wyrażeń języka J zapytujemy często, czy dwa fragmenty programu są wystąpieniami tego samego identyfikatora, nie interesując się wcale strukturą tych identyfikatorów. Dlatego niektóre symbole nieterminalne i związane z nimi reguły można zwykle uważać za aparat pomocniczy określający nieskończony alfabet terminalny. W naszym przykładzie są to symbole: <liczba>, <identyfikator> <litera>, <cyfra>.

Tak więc zbiór reguł BNF określający język programowania spełnia zazwyczaj następujące zadania:

— określa dopuszczalne wyrażenia języka jako słowa nad pewnym nieskończonym alfabetem terminalnym,

— określa reprezentację symboli tego alfabetu za pomocą skończonego zbioru znaków pisarskich,

— określa strukturę dopuszczalnych wyrażeń języka.

Tekstowy porządek symboli w regułach BNF określa porządek na niektórych podzbiorach wierzchołków drzewa wywodu. Ze względu na semantykę języka nie jest to konieczne w każdym przypadku. I tak np. w języku J porządek instrukcji w bloku jest bardzo istotny, natomiast w przypadku deklaracji nie odgrywa roli.

Łatwo zauważyć, że w języku J niektóre symbole terminalne jak np. „;" pełnią jedynie rolę znaków przestankowych, oddzielających sąsiadujące tekstowo jednostki syntaktyczne i nie są istotne przy rozważaniu semantyki języka.

Jak widać, różnorodne zadania spełniane przez reguły BNF utrudniają wyodrębnienie cech składni użytecznych przy opisie semantyki.

Składnia abstrakcyjna

W metodzie wiedeńskiej jako podstawę opisu semantyki przyjmuje się taki opis składni, który nie niesie żadnych informacji zbytecznych przy opisie semantyki. Podstawowe elementy takiego opisu stanowią abstrakcyjne „obiekty elementarne", które tworzą nieskończony zbiór E podzielony na skończoną liczbę rozłącznych klas. Wśród obiektów elementarnych wyróżnia się obiekt pusty Ω. Obiekty elementarne można uważać za abstrakcyjne odpowiedniki tych symboli nieskończonego alfabetu terminalnego, które nie są znakami przestankowymi. Z obiektów elementarnych tworzy się obiekty o bardziej złożonej strukturze; w tym celu korzysta się z tzw. zbioru selektorów S, który również może dzielić się na rozłączne klasy.

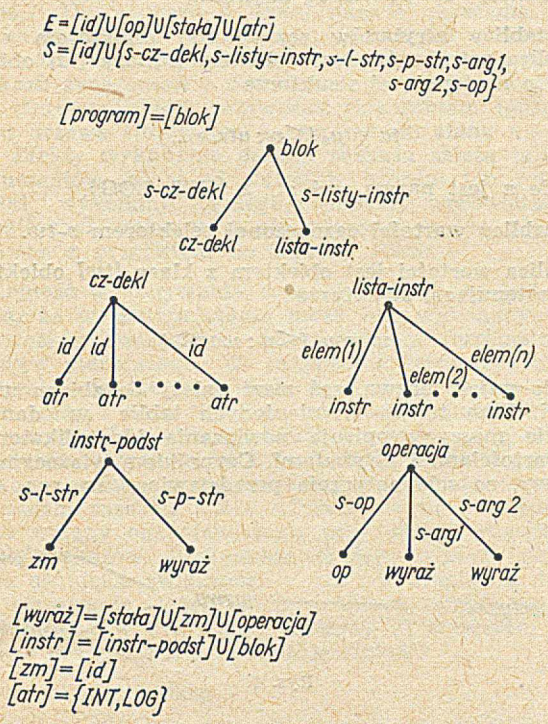
Obiektem abstrakcyjnym nazywa się obiekt elementarny lub skończony zbiór par uporządkowanych; pierwszy element każdej pary jest selektorem, drugi — obiektem abstrakcyjnym, przy czym w zbiorze tym nie mogą występować dwie różne pary zawierające ten sam selektor. Oznaczając obiekty abstrakcyjne będziemy korzystać z następującego zapisu:

$$\{<s_1 : A_1>, <s_2 : A_2>, \ldots <s_n : A_n>\}$$

gdzie s_i $i = 1,2,\ldots n$ oznacza selektor, zaś A_i — obiekt. Zapisy różniące się jedynie kolejnością występowania oznaczeń par oznaczają ten sam obiekt. Tak sformułowana definicja pozwala graficznie przedstawić obiekt abstrakcyjny w postaci drzewa, którego wierzchołkami są obiekty abstrakcyjne. Krawędziom takiego drzewa można przypisać odpowiednie selektory. W ogólnym przypadku obiekty abstrakcyjne nie muszą być zbiorami uporządkowanymi, jeśli jednak tak jest, to zamiast przypisywać poszczególnym elementom selektory można je ponumerować

kolejnymi liczbami naturalnymi zgodnie z określoną relacją porządkującą i podać funkcję *elem* przypisującą selektory liczbom naturalnym. Tak uporządkowany obiekt abstrakcyjny nazywa się *listą*.

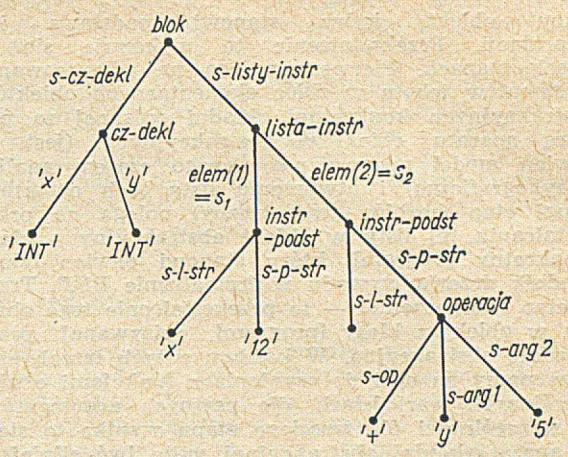
Powyższy aparat został zastosowany do opisu abstrakcyjnej składni języków programowania. Język programowania określa się jako klasę abstrakcyjnych obiektów zwanych *programami abstrakcyjnymi*. Określając klasy obiektów abstrakcyjnych korzysta się z zadanego podziału na klasy zbiorów S i E. Klasę obiektów określa się nie specyfikując szczegółowo wszystkich składowych pewnego obiektu, lecz podając jedynie, do jakich zbiorów należą wskazane elementy par. Zilustrujemy to na przykładzie (rys. 2), określając składnię abstrakcyjną języka J.



Rys. 2

Nie uciekając się do objaśniania szczegółów ścisłej notacji wprowadzonej w pracy [2], klasy obiektów zdefiniujemy bądź graficznie w postaci drzew, bądź jako teoriomnogościową sumę innych klas, bądź też wyliczając w sposób jawny oznaczenia wszystkich elementów klasy (w przypadku obiektów elementarnych lub selektorów). Definicja w postaci drzewa określa klasę obiektów objętych wspólną nazwą wskazaną przy początkowym wierzchołku drzewa.

Nazwy wskazane przy terminalnych wierzchołkach drzewa oraz przy krawędziach wskazują ustalone obiekty (selektory) lub klasy tych obiektów (selektorów), z których może być utworzony obiekt definiowanej klasy. Obiekt z danej klasy powstaje przez wybranie ustalonych elementów z klas wskazanych w definicji. Wybór ten jest w zasadzie dowolny, nie może jednak naruszyć zasady jednoznaczności selektorów określonej w definicji obiektu abstrakcyjnego. Rys. 3 przedstawia program abstrakcyjny stanowiący odpowiednik wyrażenia, którego drzewo wywodu przedstawiono na rys. 1. Dzięki temu, że klasa identyfikatorów należy zarówno do zbioru S, jak i E uzyskano możliwość sformułowania w składni abstrakcyjnej pewnych dodatkowych warunków, których nie da się opisać za pomocą bezkontekstowej składni BNF [4]. Przykładem jest definicja części deklaracyjnej, która mówi, że ten sam identyfikator nie może być w jednym bloku zadeklarowany jednocześnie jako INT i jako LOG.



Rys. 3

Definicja obiektu abstrakcyjnego umożliwia określenie następujących operacji szczególnie ułatwiających opis interpretacji programu:

— operacja *ekstrakcji* [1], której argumentami są selektor i obiekt. Wynikiem tej operacji jest obiekt tworzący wraz z zadanym selektorem składową parę zadanego obiektu (lub obiekt pusty, jeśli taka para nie istnieje). Dla obiektu A z rys. 3 operacja ekstrakcji: s_1 (*s-listy-instr*. (A)) daje w wyniku obiekt z klasy [instrukcja podstawienia]:

{<*s-1-str*: x>, <*s-p-str* ; 12>}. Złożenie operacji ekstrakcji można zapisywać bez użycia nawiasów: $s_1 \cdot$ *s*-listy-instr (A)

— operacja *mutacji*, której argumentami są: obiekt A i skończony zbiór par $\{\langle s_1 : A_1\rangle, \langle s_2 : A_2\rangle, \ldots \langle s_j : A_j\rangle\}$

gdzie $s_i = s_{i_1} \cdot s_{i_2} \cdot \ldots \cdot s_{ik_i}$ $s_{ik_i} \in S$ $i = 1,2 \ldots j$ $l = 1,2 \ldots k_i$

Wynikiem tej operacji jest obiekt $\bar{A}$ taki, że dla dowolnego selektora s zachodzi równość:

$$s(\bar{A}) = \begin{cases} A_i & \text{jeśli } s = s_i \quad i = 1, 2, \ldots j \\ s(A) & \text{w przeciwnym przypadku} \end{cases}$$

Selektory występujące w zbiorze par muszą ponadto spełniać tzw. warunek „niezależności", który jest konsekwencją warunku jednoznaczności spełnianego przez obiekt wynikowy i wszystkie jego składowe. Operację mutacji oznaczamy μ i zapisujemy w sposób, który zilustrujemy przykładem

μ (*A*; <*s-cz-dekl*: Ω>, <s_2 : Ω>, <*s-p-str* $\cdot$ s_1 $\cdot$ *s-listy-instr* : 25>)

W wyniku otrzymujemy obiekt {<*s-l-str* : *x*>, <*s-p-str* : 25>}

— operacja *konstrukcji* [1]; jest to operacja mutacji, której pierwszym argumentem jest obiekt pusty. Operację konstrukcji oznaczamy μ_0 i zapisujemy w następujący sposób:

$\mu_0(\langle s_1 : A_1\rangle, \langle s_2 : A_2\rangle \ldots \langle s_j : A_j\rangle)$, gdzie podane pary mają analogiczną budowę i spełniają te same ograniczenia jak w przypadku operacji mutacji.

Składnię konkretną zapisaną w postaci BNF można wyrazić również za pomocą aparatu obiektów abstrakcyjnych, który znacznie ułatwia opis przekształcania struktury. Obiekty elementarne są w tym przypadku odpowiednikami nieskończonego alfabetu terminalnego nie wyłączając znaków przestankowych, a wszystkie inne obiekty są uporządkowane przez relacje stanowiące odpowiednik relacji tekstowego porządku w regułach BNF.

Jak już wspominaliśmy, omawiana metoda przewiduje przekształcenie tekstu programu w abstrak-

[1]) termin zapożyczony od W. Turskiego.

cyjny obiekt, który stanowi podstawę interpretacji. Przekształcenie to wykonuje się w trzech etapach. Pierwszy polega na odwzorowaniu fragmentów tekstu w zbiór elementarnych obiektów abstrakcyjnych konkretnej składni opisanej za pomocą aparatu obiektów abstrakcyjnych (jest to pewien model procesu nazywanego czasem analiza leksykograficzną) i utworzeniu listy tych obiektów. Drugi etap (rozbiór gramatyczny) polega na przekształcaniu tej listy w obiekt abstrakcyjny z klasy [konkretny program], który stanowi pewien odpowiednik drzewa wywodu w formalizmie BNF. Trzeci etap (tłumaczenie) — to przekształcenie tego obiektu w obiekt z klasy [program] opisywanej przez składnię abstrakcyjną. W każdym z tych trzech etapów mamy na ogół do czynienia z funkcjami o nieokreślonych wartościach, dla pewnych argumentów. W szczególności dla trzeciego etapu wynika to stąd, że aparat składni abstrakcyjnej może być silniejszy niż bezkontekstowy aparat składni konkretnej (jak to miało miejsce w przypadku języka J). Jeśli dla danego argumentu jedna z powyższych funkcji nie jest określona, to mamy do czynienia z błędem syntaktycznym. Te trzy etapy kontroli nie wykluczają jednak na ogół występowania w programie abstrakcyjnym pewnych błędów kontekstowych wykrywanych dopiero w czasie interpretacji.

Maszyna i proces

Dotychczas omówiono pojęcia składni konkretnej i abstrakcyjnej na przykładzie języka J. Pozostaje nam zatem zdefiniować maszynę, której procesy będą stanowić interpretację wyrażeń tego języka.

Maszynę można scharakteryzować podając zbiór jej stanów Σ oraz funkcję przejścia Λ. Funkcja Λ jest funkcją częściową na zbiorze Σ a jej wartościami są podzbiory zbioru Σ. Jest to więc maszyna niedeterministyczna.

Procesem nazywać będziemy ciąg stanów maszyny

$$\zeta_0, \zeta_1, \zeta_2 \ldots$$

taki, że $\zeta_{k+1} \in \Lambda(\zeta_k)$ dla $k = 0, 1, \ldots$

W przypadku opisywanej metody zarówno zbiór stanów, jak i funkcję przejścia definiuje się stosując aparat obiektów abstrakcyjny. Odstąpimy jednak od tej zasady przy wprowadzaniu definicji drzewa sterującego oraz graficznej formy zapisu drzewa sterującego. Konsekwentne stosowanie tego aparatu w tych przypadkach rozszerzyłoby bowiem znacznie rozmiary artykułu. W dalszym ciągu maszynę skonstruowaną dla potrzeb opisu semantyki naszego języka J oznaczać będziemy przez M. W celu zdefiniowania maszyny M rozszerzymy zdefiniowany uprzednio zbiór obiektów elementarnych włączając do niego:

1) nieskończony ciąg nazw $n_1, n_2, \ldots$ taki, że $n_i \neq n_j$ dla $i \neq j$. Nazwy odpowiadają w pewnym sensie adresom pamięci maszyny rzeczywistej;

2) zbiór wartości $I \cup L$ złożony ze zbioru I wartości liczb całkowitych (tj. wartości typu INT) oraz zbioru L wartości logicznych (tj. zbioru wartości typu LOG).

Zbiorem Σ stanów maszyny M nazywamy zbiór wszystkich obiektów ζ takich, że

$\zeta = \mu_0$ (<*s-liczn: całkowita*>, <*s-tz: tz*>, <*s-ta: ta*>, <*s-tw:tw*>, <*s-stos:stos*>, <*s-ster:d-ster*>)

Stan jest zatem obiektem złożonym z następujących składników bezpośrednich:

1) licznika zaznaczonego selektorem s-liczn i który jest liczbą całkowitą. Licznik wskazuje indeks pierwszej spośród nazw w ciągu, którym w żadnym z poprzednich stanów nie został przyporządkowany identyfikator. Z rolą licznika czytelnik zaznajomi się studiując definicję rozkazu **int-cz-dekl,**

2) tablicy zmiennych zaznaczonej sektorem *s-tz*. Tablica zmiennych jest obiektem z klasy [tz] obiektów o następującej budowie

$$tz = \mu_0(\langle id : n \rangle, \ldots)$$

gdzie *id* należy do klasy identyfikatorów zaś n jest elementem ciągu nazw. Użycie kropek oznacza, że para $\langle id : n \rangle$ może się powtórzyć dowolną ilość razy bądź może w ogóle nie wystąpić. Tablica zmiennych może więc być obiektem pustym Ω, obiektem

$\mu_0(\langle id_\alpha : n_\alpha \rangle)$, $\mu_0(\langle id_\alpha : n_\alpha \rangle, \langle id_\beta : n_\beta \rangle)$ itd.,

gdzie id_α, id_β itd. należą do klasy identyfikatorów natomiast n_α, n_β, itd., są nazwami,

3) tablicy atrybutów zaznaczonej selektorem s-ta. Tablica artrybutów jest obiektem z klasy [*ta*] obiektów o następującej strukturze

$$ta = \mu_0(\langle n : atr \rangle \ldots)$$

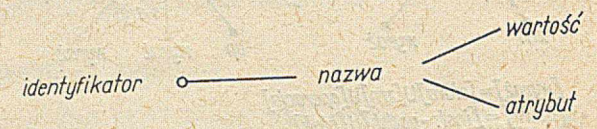
gdzie *n* jest nazwą, zaś *atr* ε {INT, LOG},

4) tablicy wartości zaznaczonej selektorem *s-tw*.

Tablica wartości jest obiektem z klasy *[tw]* obiektów o następującej strukturze

$$tw = \mu_0(\langle n : wart \rangle)$$

gdzie *n* jest nazwą, zaś *wart* należy do zbioru wartości. Stan trzech wymienionych tablic w danym stanie maszyny opisuje powiązania identyfikatorów z wartościami i atrybutami. Czynnikiem wiążącym są nazwy, co schematycznie przedstawiono na rys. 4.



Rys. 4

Jak tworzone są te tablice oraz jaką pełnią rolę dowie się czytelnik studiując definicje rozkazów **aktualizuj, int-cz-dekl., int-instr-podst,**

5) stosu zaznaczonego selektorem s-stos. Stos jest obiektem z klasy [stos] obiektów o następującej budowie

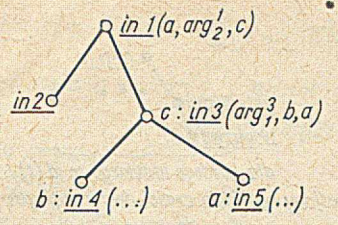
$$stos = \Omega \vee \mu_0(\langle s\text{-}tz : tz \rangle, \langle s\text{-}stos : stos \rangle)$$

Stos służy do przechowywania aktualnego stanu tablicy wartości przy przejściu z bloku obejmującego do wewnętrznego a następnie do jego odtworzenia w chwili powrotu z bloku wewnętrznego do obejmującego — w bloku wewnętrznym bowiem tablica zmiennych ulega na ogół zmianie (por. definicje rozkazów **int-blok, wyjdź-z-bloku),**

6) sterowania oznaczonego przez s-ster i które jest obiektem z klasy [d-st]. Klasy tej formalnie nie zdefiniujemy. Możemy przyjąć, że należą do niej obiekty, które posiadają takie same własności jak opisane niżej drzewa sterujące.

Do zapisu drzew sterujących używać będziemy graficznej notacji, której również nie będziemy formalnie definiować a zadowolimy się jedynie jej opisem wprowadzanym równolegle z opisem drzewa sterującego. Przykład zapisu drzewa sterującego przy użyciu tej notacji przedstawiono na rys. 5.

Przez drzewo sterujące rozumieć będziemy drzewo, którego każdy wierzchołek związany jest z pewną instrukcją. W stosowanej notacji graficznej nazwy



Rys. 5

tych instrukcji zapisane małymi literami będą podkreślone i umieszczone obok lub pod odpowiednim wierzchołkiem [1]).

Szczególnym przypadkiem drzewa sterującego jest drzewo puste — nie zawierające żadnego wierzchołka. Z każdym rozkazem związana jest lista argumentów. Argumenty rozkazów są obiektami abstrakcyjnymi. Dla rozkazu o danej nazwie określona jest liczba argumentów oraz klasy, do których należą poszczególne argumenty. Obiekt będący argumentem rozkazu związanego z dowolnym wierzchołkiem drzewa sterowania jest oczywiście składnikiem (pośrednim) drzewa sterowania a tym samym stanu maszyny. Efekty wykonania danego rozkazu zależą tak od argumentów jak i od stanu, w którym ten rozkaz został wykonany. Polegają one na wprowadzeniu określonych zmian w stanie maszyny, co jest równoznaczne z przejściem do nowego stanu.

Kolejność wykonywania rozkazów określona jest zasadą, że w danym stanie może być wykonany tylko rozkaz związany z wierzchołkiem terminalnym drzewa sterującego. W przypadku większej liczby wierzchołków terminalnych w drzewie należy wybrać jeden z nich. Np. drzewo sterujące przedstawione na rys. 5 ma trzy wierzchołki terminalne odpowiadające rozkazom **in 2, in 4, in 5.** Omówimy teraz tylko te zmiany w stanie maszyny spowodowane wykonaniem rozkazu, które odnoszą się do sterowania. Inne zmiany omawiane będą przy okazji omawiania spisu rozkazów. Wykonanie każdego rozkazu powoduje usunięcie związanego z nią wierzchołka z drzewa sterowania. Jeżeli chodzi o dalsze zmiany w sterowaniu, to w przypadku maszyny M wystarczy dopuścić następujące dwa rodzaje zmian:

1) przesłanie obiektu

2) zastąpienie w drzewie sterującym wierzchołka związanego z wykonywaną instrukcją innym drzewem sterującym.

Pierwszy rodzaj zmian polega (najogólniej rzecz biorąc) na tym, że obiekt wytworzony w wyniku wykonania rozkazu zostaje przesłany do innych wierzchołków drzewa i wstawiony w określone miejsca list argumentów rozkazów związanych z tymi wierzchołkami. Zakładamy przy tym, że przesyłanie takie może odbywać się jedynie w kierunku wierzchołka początkowego. W przypadku drzewa sterującego przedstawionego na rys. 5 obiekt, który powstał w wyniku wykonania rozkazu **in 5**, nie mógłby zostać wstawiony na listę argumentów instrukcji **in 2** lub **in 4.**

W przypadku używanej przez nas notacji, dla zaznaczenia skąd i dokąd obiekty mają być przekazywane, używać będziemy tzw. *etykiet* oznaczanych małymi początkowymi literami alfabetu.

Rozkazy, dające w wyniku wykonania obiekty do przesłania będą poprzedzane etykietami oddzielanymi od nazwy instrukcji dwukropkiem. Np. na rys. 5 etykietami zostały poprzedzone rozkazy **in 3, in 4, in 5.**

Te same etykiety, lecz umieszczone na liście argumentów, oznaczać będą miejsca, w które odpowiednie obiekty mają zostać przesłane. I tak w przypadku drzewa sterującego przedstawionego na rys. 5 obiekt, który powstanie w wyniku wykonania rozkazu **in 5**, zostanie wstawiony jako trzeci argument rozkazu **in 3** i jako pierwszy, argument rozkazu **in 1.**

Podobnie obiekt, który powstanie w wyniku wykonania **in 4** zostanie wstawiony jako drugi argument **in 3.**

Drugi rodzaj zmian nie wymaga dodatkowych objaśnień. Należy tutaj dodać, że w przypadku gdy instrukcja związana z zastępowanym wierzchołkiem była poprzedzona etykietą, wówczas etykieta ta nie jest usuwana, lecz doczepiana do instrukcji związanej z wierzchołkiem początkowym drzewa zastępującego usuwany wierzchołek.

Spis rozkazów

Funkcję przejścia Λ dla naszej maszyny M zdefiniujemy podając spis rozkazów maszyny M wraz z definicjami. Poszczególne rozkazy będziemy definiować posługując się następującym schematem. W pierwszym wierszu, po lewej stronie znaku równości zapisywać będziemy nazwę definiowanego rozkazu wraz z listą argumentów. Po prawej stronie znaku równości oraz w kolejnych wierszach opisane będą zmiany, jakie w poszczególnych składnikach stanu powoduje wykonanie danego rozkazu. Opis zmian danego składnika stanu będzie poprzedzony symbolem selektora tego składnika oddzielony od samego opisu dwukropkiem. Opis będzie podany bądź w postaci operacji na obiektach abstrakcyjnych, bądź w postaci drzewa sterowania w notacji graficznej. Ten drugi przypadek dotyczy rzecz jasna jedynie opisu zmian w sterowaniu. W pewnych przypadkach definicję rozkazu trzeba będzie rozbić na kilka członów, z których każdy odpowiada pewnej podklasie argumentów definiowanego rozkazu. Dla zapisania takiej definicji używać będziemy notacji McCarthy'ego [2]. Dla przypadku jednego argumentu ma ona postać

$$
\begin{aligned}
f(x) = \\
& p_1(x) \rightarrow f_1(x), \\
& p_2(x) \rightarrow f_2(x), \\
& \vdots \\
& p_n(x) \rightarrow f_n(x).
\end{aligned}
$$

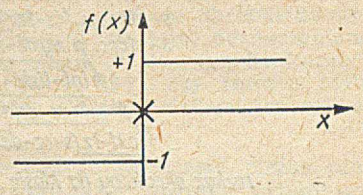
gdzie $p_1, p_2 \ldots p_n$ są warunkami wyznaczającymi podklasy argumentu x, a $f_1 \ldots f_n$ definicjami.

Zapis ten oznacza, że jeżeli dla danego x_0, p_i jest pierwszym spośród warunków $p_1 \ldots p_n$, który jest spełniony dla x_0, to $f(x_0) = f_i(x_0)$

Dla przykładu zapis

$$
\begin{aligned}
f(x) = \\
& x < 0 \rightarrow -1, \\
& x = 0 \rightarrow 0, \\
& T \rightarrow +1
\end{aligned}
$$

gdzie T oznacza warunek zawsze spełniony, definiuje funkcję o wykresie jak na rys. 6.



Rys. 6

Zauważmy tutaj, że np. dla x równego zero spełniony jest zarówno warunek $x = 0$, jak i warunek T (spełniony zawsze). Ponieważ jednak warunek $x = 0$ jest pierwszym, spełnionym spośród wypisanych trzech warunków, zatem przyjmujemy $f(0) = 0$.

W pewnych przypadkach (tak jak np. w przypadku rozkazu **int-cz-dekl**) drzewo sterowania występujące z definicji może mieć takie wierzchołki, że liczba wychodzących zeń krawędzi zależy od argumentów definiowanego rozkazu. Fakt ten zapisywać będziemy w ten sposób, że z owego wierzchołka wy-

prowadzimy tylko jedną krawędź, po której postawimy trzy kropki, pod spodem zaś opiszemy, jak określić liczbę krawędzi. Zapis ten obejmuje również przypadek, gdy liczba wspomnianych krawędzi jest równa zero.

Jeżeli zmiany w sterowaniu będą jedynymi zmianami stanu maszyny wywołanymi wykonaniem definiowanego rozkazu, opisu tych zmian nie będziemy poprzedzali symbolem selektora.

Dla skrócenia zapisu definicji używać będziemy następujących oznaczeń

ST = *s-ster* (ζ)
TA = *s-ta* (ζ)
TZ = *s-tz* (ζ)
TW = *s-tw* (ζ)
L = *s-licz* (ζ)
ST = *s-stos* (ζ)

Zanim przejdziemy do zdefiniowania rozkazów wprowadzimy cztery niezbędne od tego celu funkcje

wartość (t) — funkcja ta podaje dla danej stałej t jej wartość

konwersja (t, u) podaje wartość t przekształconą (jeżeli to potrzebne) do typu określonego przez u (u może być w naszym przypadku INT lub LOG, t jest wartością)

głowa (t) — funkcja określona dla niepustych list, daje w wyniku pierwszy element listy t

ogon (t) — funkcja określona dla niepustych list, daje w wyniku listę, która powstała z listy t przez usunięcie z niej pierwszego elementu.

W zamieszczonym niżej spisie trzy pierwsze rozkazy zdefiniowane zostaną w sposób opisowy.

Opisy rozkazów

1) **int-op** (op, t, v) rozkaz ten daje w wyniku obiekt z klasy wartości będący wynikiem zastosowania operacji op do t i v (t oraz v są wartościami). Otwartą pozostawia się sprawę, co należy robić, jeżeli operacja op „nie pasuje" do typu t lub v,

2) **prześlij** (t) rozkaz określony dla wszystkich obiektów, powoduje przesłanie obiektu t zgodnie z użytymi etykietami,

3) **n-n-r** rozkaz nic nie rób. Jego wykonanie polega na usunięciu go z drzewa sterowania,

4) **int-prog** (t) =

int-blok (t)

rozkaz określony na klasie programów abstrakcyjnych

5) *int-blok*(t)

s-stos: μ_0(⟨*s-tz*: *TZ*⟩, ⟨*s-stos*: *STOS*⟩)
s-ster: • *wyjdź-z-bloku*
• *int-listę-instr*(*s-lista-instr*(t))
• *int-cz-dekl*(*s-cz-dekl*(t))
akt-tz(*s-cz-dekl*(t))

rozkaz określony na klasie bloków

rozkaz określony na klasie instrukcji podstawiania; w przypadku, gdy zmienna stanowiąca lewą stronę nie została zadeklarowana będzie zasygnalizowany błąd,

14) **podstaw** (*n*, v) =

s-tw: μ (**TW**); ⟨*n*: konwersja (v, *n* (**TA**)⟩)

określone dla n należącego do zbioru nazw oraz v należącego do zbioru wartości,

6) *akt-tz*(t) =

• *n-n-r*
• • •
akt-id(id,n)
n: *nazwa*

dla każdego id takiego że id (t) ≠ Ω

rozkaz określony na klasie części deklaracyjnych

7) *akt-id*(id, n) =

s-tz: μ(*TZ*: ⟨*id*: *n*⟩)

rozkaz określony dla id należącego do klasy idendyfikatorów oraz n należącego do zbioru nazw.

8) *nazwa* =

s-ster: •
prześlij(n_L)

s-liczn: *L*+1

9) *int-cz-dekl*(t) =

• *n-n-r*
• • •
int-dekl(*id*(*TZ*), *id*(t))

dla każdego id takiego że id (t) ≠ Ω

rozkaz określony na klasie części deklaracyjnych.

10) *int-dekl*(n, atr) =

s-ta: μ(*TA*: ⟨*n*: *atr*⟩)

rozkaz określony dla n należącego do zbioru nazw oraz atr ∈ {*INT*, *LOG*}

11) *int-listę-instr*(t) =

t jest puste ⟶ • *n-n-r*

T ⟶ • *int-listę-instr*(*ogon*(t))
• *int-instr*(*głowa*(t))

12) *int-instr*(t) =

t jest blokiem ⟶ • *int-blok*(t).

t jest instrukcją podstawiania ⟶ • *int-instr-podst*(t)

rozkaz określony na klasie instrukcji.

13) *int-instr-podst*(t) =

z(*TZ*) ≠ Ω ⟶ • *podstaw*(n_t, v)
v: *int-wyraż*(*s-p-str*(t))

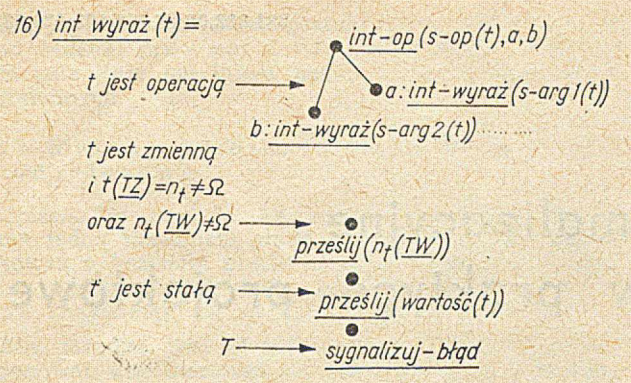
T ⟶ • *sygnalizuj-błąd*

gdzie z=*s-l-str*(t)

15) **wyjdź z bloku**

s-tz:*s-tz* (**STOS**)
s-stos:*s-stos* (**STOS**)

gdzie n_t = t (**TZ**) jest nazwą przypisaną zmiennej t rozkaz jest określony na klasie wyrażeń; w przypadku, gdy zmienna t nie została zadeklarowana lub zostanie zasygnalizowany błąd.



Ponieważ ustalono, że wykonanie każdego rozkazu powoduje usunięcie związanego z nim wierzchołka, efektu tego już nie opisywano w podanych wyżej definicjach.

Nie zdefiniowano również rozkazu **sygnalizuj-błąd** przyjmując, że jego znaczenie jest intuicyjnie oczywiste a definicja mogłaby wprowadzić szereg niepotrzebnych komplikacji.

Interpretacja wyrażeń języka J

W celu określenia interpretacji danego wyrażenia, języka J należy najpierw znaleźć odpowiadający mu program abstrakcyjny p.

Proces, o który nam chodzi, otrzymamy w ten sposób, że jako stan początkowy ζ_0 przyjmiemy stan, w którym drzewo sterowania zawiera jeden wierzchołek związany z rozkazem **int-prog** (p), licznik zawiera liczbę 1, natomiast tablica zmiennych, tablica atrybutów, tablica wartości oraz stos są obiektami pustyni. Poczynając od tak określonego stanu ζ_0 przechodzimy do stanów następnych ζ_1, ζ_2 ... wykonując rozkazy w kolejności określonej przez zasadę węzła terminalnego oraz zgodnie z podanymi w spisie rozkazów definicjami.

Zauważmy, że wykonanie pewnych rozkazów powoduje „rozrastanie się" drzewa sterowania a innych jego „kurczenie". W momencie, gdy dojdziemy do stanu ζ_k, w którym drzewo sterowania jest puste, uznajemy, że jest to stan końcowy (niezależnie od stanu pozostałych elementów). Otrzymany w ten sposób proces ζ_0, ζ_1 ... ζ_k stanowi interpretacje naszego wyrażenia. Czytelnik dla przykładu może opisać proces stanowiący interpretacje wyrażenia podanego jako przykład przy omawianiu składni konkretnej; program abstrakcyjny odpowiadający temu wyrażeniu został przedstawiony na rys. 3.

Zakończenie

Wiedeńska metoda opisu języków programowania wymaga, jak podano wyżej, zdefiniowania m. in. składni abstrakcyjnej, przekształcenia zbioru wyrażeń języka w zbiór abstrakcyjnych programów oraz maszyny abstrakcyjnej. Precyzja wszystkich tych definicji wynika z zastosowania formalnego aparatu obiektów abstrakcyjnych, zaś komunikatywność opisu zależy przede wszystkim od definicji składni abstrakcyjnej. Osiągnięto ją dzięki temu, że w odróżnieniu od programu w kodzie maszyny, program abstrakcyjny zachowuje w zasadzie strukturę programu konkretnego z pominięciem elementów zbędnych z punktu widzenia semantyki. Konstrukcja maszyny abstrakcyjnej uwarunkowana jest składnią abstrakcyjną: każda klasa obiektów zdefiniowana przez tę składnię (np. [program] [lista-instr], [instr]) znajduje swój odpowiednik wśród rozkazów maszyny. Istnieje jednak pewna swoboda definiowania stanów maszyny. Nie podano metody definiowania maszyny, więc szereg problemów dotyczących tego zagadnienia pozostaje nadal sprawą intuicji „inżynierskiej".

BIBLIOGRAFIA

[1] LAUER P. Formal Definition of ALGOL 60. IBM Lab. Vienna, TR 25.088, 1968.

[2] LUCAS P., LAUER, P., STIGLEITNER H. Method and Notation for the Formal Definition of Programming Languages. IBM Lab. Vienna, TR 25.087, 1968.

[3] WALK K et al. Abstract Syntax and Interpretation of PL-1. IBM Lab. Vienna, TR 25.082, 1968.

[4] MAŁUSZYŃSKI J. Opis składni języków programowania. „Maszyny Matematyczne" nr 9/1970.

[5] MAZURKIEWICZ A. Problemy języków formalnych w automatycznym przetwarzaniu informacji. Problemy przetwarzania informacji WNT, Warszawa 1970.

[1]) Ze względów typograficznych zamiast napisów podkreślonych w tekście używa się pisma półgrubego.

Wspomnienie pośmiertne

Z żalem pożegnaliśmy mgr inż. STANISŁAWA GRUDZIECKIEGO, którego w wieku 46 lat nagła śmierć wyrwała z naszego grona w październiku 1970 roku.

Był wybitnym specjalistą w zakresie przemysłowych zastosowań maszyn licząco-analitycznych i komputerów do potrzeb zarządzania. Początkowo organizował mechanizację systemów technologiczno-planistycznych w FSO, następnie pracował w Instytucie Organizacji Przemysłu Maszynowego, aby potem na dobre poświęcić się pionierskiemu wdrażaniu Pakietu Obliczeń Produkcyjnych w Zakładach Mechanicznych im. M. Nowotki w Warszawie jako kierownik Działu Przetwarzania Informacji. Poza tym zajmował się konstruowaniem urządzeń do obróbki plastycznej, za co był wyróżniany wysokimi odznaczeniami państwowymi oraz nagrodami w licznych konkursach (np. Mistrza Techniki w r. 1960, czy resortu MPC w r. 1962 — II nagroda). Opatentował w tym zakresie szereg wynalazków.

Pamiętamy Go jako wielkiego entuzjastę, ciężko pracującego w niewdzięcznej fazie wdrożeń przemysłowych. Był zawsze uśmiechnięty, gotów do pomocy, przez wszystkich lubiany. Często sam pomagał w dorabianiu unikalnych części do komputera IBM 1440, dzięki czemu szereg razy uniknięto dłuższych przestojów.

Szkoda to olbrzymia dla kraju, że ubył z polskiej informatyki. Cześć Jego pamięci!

Koleżanki i koledzy
z ZOWARu i Zakładów im. M. Nowotki

JAN GOLIŃSKI
SOETO
Warszawa

681.322.06:65.012.122:621.833

Metody optymalizacyjne stosowane w praktyce projektowej

Autor przedstawia kilka podstawowych metod optymalizacyjnych stosowanych w projektowaniu technicznym. Zalety metody programowania dynamicznego ilustruje zadanie z dziedziny projektowania przekładni zębatej.

Optymizacja jest rzemiosłem, jest pracą wymagającą zręczności i wyobraźni, stosowania różnych narzędzi w celu uzyskania w końcu prostych wyników.

Optymizacja pozwala osiągnąć „najlepsze" w danej sytuacji, co jest naturalną ambicją większości projektantów. Doświadczenie i intuicja mają w tym procesie olbrzymie znaczenie, wydaje się jednak, że bardzo często przymioty te nie są należycie wsparte matematyką. Ażeby zastosować metody matematyczne dla konkretnego zadania technicznego lub ekonomicznego, należy najpierw zadanie to wyrazić w matematycznych terminach. Napotykamy tutaj na pierwsze kłopoty, bowiem nawet proste sytuacje w ekonomice, inżynierii, przemyśle okazują się bardzo skomplikowane po poddaniu skrupulatnej analizie matematycznej. Analiza taka polega na tym, że rozkładamy proces na składowe i poszukujemy opisu matematycznego tych składowych i ich wzajemnych powiązań. Mówimy przy tym o budowaniu modelu matematycznego. Sprawą niesłychanie istotną przy modelowaniu jest dokonywanie poprawnych założeń. Oczywiście, może być konieczne dokonanie różnych uproszczeń, aby zadanie dało się rozwiązać. Zawsze jednak wprowadzone uproszczenie musi być działaniem świadomym, a nie wynikającym z przeoczenia.

Załóżmy zatem, że istniejąca sytuacja i funkcja celu, do ekstremum której dąży projektant, dadzą się opisać w terminach matematycznych. To jednak nie wystarcza. Musimy znać ponadto sposoby rozwiązania tak sformułowanego zadania. Dobrze byłoby również, gdybyśmy byli w stanie odpowiedzieć na pytanie, które ze znanych metod najbardziej nadają się do rozwiązania właściwego zadania.

1. Kilka podstawowych metod stosowanych w optymizacji

1.1. „Obliczenie wprost"

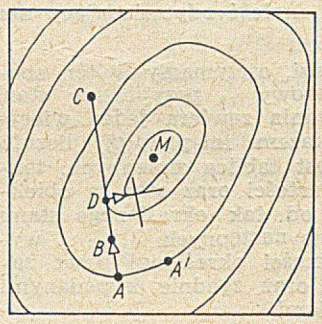
Najprostsze postępowanie polega na przeliczeniu podanych wzorów. Jeżeli model matematyczny składa się ze zbioru równań algebraicznych lub różniczkowych, a funkcja celu jest funkcją tych samych zmiennych, które występują w równaniach, to można czasami zalecić następujące postępowania:

Dokonujemy dowolnego (opartego na intuicji) wyboru niektórych zmiennych. Dla tego wyboru rozwiązujemy równania i obliczamy funkcję celu. Następnie dokonujemy innego wyboru, ponawiając postępowanie. Jeżeli w drugim przypadku otrzymamy lepszą funkcję celu, wówczas ten zbiór zmiennych uważamy za lepszy itd.

Zauważmy, że jeżeli funkcja celu ma tylko jedno ekstremum, to dla przypadku jednej zmiennej istnieje pełne rozwiązanie.

1.2. Metody największego spadku

Załóżmy, że mamy przypadek opisany dwoma zmiennymi x i y i że funkcja celu da się przedstawić warstwicami (rys. 1). Zamknięte linie dają krzywe równej wartości funkcji celu. Jeżeli dla punktu A określimy współrzędne x, y, to np. przesunięcie do A' nie da zmiany wartości funkcji celu. Najbardziej korzystnym kierunkiem przesuwania się jest kierunek AB. Nawet mała zmiana x, y w tym kierunku powoduje wzrost wartości funkcji celu przy poszukiwaniu maksimum.



Rys. 1

Zauważmy jednak, że zbyt duże przesunięcie (np. do punktu C) spowoduje zmniejszenie wartości funkcji celu.

Jeżeli opiszemy powierzchnię funkcją różniczkowalną f(x, y) to kąt, jaki tworzy odcinek AB z osią x jest równy $\frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}}$, a zatem można analitycznie określić kierunek najbardziej korzystnego przesunięcia. Na ogół jednak nie możemy lub trudno nam zróżniczkować funkcję celu. W tym przypadku postępujemy następująco: punkt A ma współrzędne (a, b). Jeżeli δa, δb są małymi przesunięciami, to możemy pochodną w A przedstawić następującym przybliżeniem:

$$\left(\frac{\partial f}{\partial x}\right)_{a,\,b} = \frac{f(a+\delta a, b) - f(a, b)}{\delta a}$$

$$\left(\frac{\partial f}{\partial y}\right)_{a,\,b} = \frac{f(a, b+\delta b) - f(a, b)}{\delta b}$$

Mając obliczoną wartość funkcji celu w 3 punktach [(a,b), (a+δa,b), (a,b+δb)], można znaleźć kierunek największego spadku. Przesuwając się po tak wyznaczonym kierunku tak długo, jak długo wartość funkcji celu wzrasta osiągniemy punkt D (p. rys. 1).

W punkcie tym ponawiamy postępowanie. Seria takich działań prowadzi w okolicę punktu ekstremalnego M. W okolicy M można posłużyć się bardziej

precyzyjnym sposobem poszukiwania, uzależniając to od żądanej przez nas dokładności wyniku. Opisana metoda daje się łatwo uogólnić na większą liczbę zmiennych.

1.3. Metody gradientu swobodnego i metody losowe

Opisana w punkcie 1.2. metoda szybkiego spadku ma jedną zasadniczą wadę. Nie pozwala ona wnioskować o kształcie powierzchni, a zatem nie wiemy, czy wykonać krok mały, czy duży i czy niekiedy nie znajdziemy się za daleko (punkt C na rys. 1).

Aby zaradzić tym kłopotom, D. I. Wilde [1] zaproponował, aby kierować się nie informacją o kierunku największego spadku, lecz informacją o stycznej do warstwicy w danym punkcie. Postępowanie to można zobrazować następująco:
Zaczynając od dowolnie wybranego punktu A (rys. 2) kreślimy prostą KAL, odrzucając badanie powierzchni poniżej tej prostej, której punkty odpowiadają gorszym wartościom funkcji celu. Następnie wybieramy kolejny dowolny punkt w pozostałej części obszaru, np. B na rys. 2. Kreślimy styczną PBN (do warstwicy funkcji celu) i odrzucamy według podanej reguły część powierzchni zakreskowaną na rysunku itd., oczywiście postępowanie to można łączyć z innymi metodami.

Sposób ten zawodzi w przypadku, kiedy istnieje więcej niż jedno ekstremum (punkty 3 i 4 na rys. 3).

1.4. Zastosowanie rachunku różniczkowego

Maszyny cyfrowe preferują metody polegające na „obliczaniu wprost". Nie znaczy to, że należy pomijać rachunek różniczkowy. Z jego zastosowaniem wiążą się spore trudności jako że rachunek ten prowadzi do ekstremum lokalnego, a nie globalnego, który na ogół interesuje inżyniera czy ekonomistę.

Jeżeli mamy ciągłą funkcję $y = f(x)$ zdefiniowaną w przedziale $\langle a, b \rangle$ należy również wziąć pod uwagę punkty na brzegach przedziałów. Zatem wartości $f(x)$ w A, E i G należy porównać z wartością maksimum w B. W ten sposób stwierdzimy, że maksimum absolutne znajduje się w G. To ostatnie zadanie jest znacznie trudniejsze niż znalezienie ekstremum przez proste różniczkowanie i przyrównanie pochodnej do zera.

Oczywiście zdarzają się przypadki, gdzie fizyczna strona zjawiska zapewnia ciągłość i różniczkowalność funkcji oraz istnienie w rozważanym przedziale jednego tylko ekstremum. W przypadku istnienia większej liczby zmiennych, rozwiązanie zadania jest dalece nietrywialne.

1.5. Programowanie liniowe

W przypadku liniowej funkcji celu i liniowych ograniczeń wiadomo, że ekstremum leży w narożach obszaru dopuszczalnego.

Rozważmy banalny przykład maksymizacji funkcji

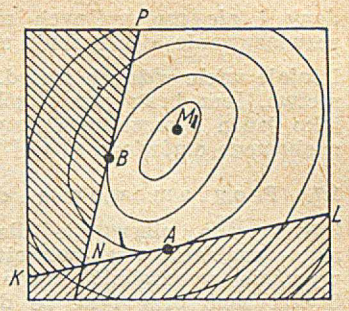
$f(x, y) = 2x + 3y$

przy ograniczeniach

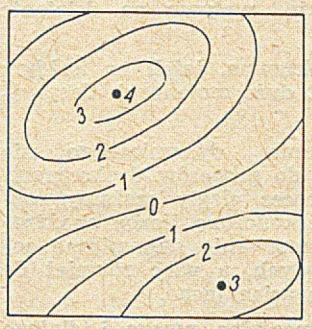
$$x \geqslant 0 \quad y \geqslant 0 \quad x + y \leqslant 1$$

Ograniczenia określają obszar pokazany na rys. 5.

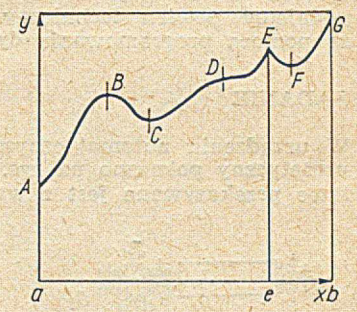
Poszukiwany punkt musi leżeć tutaj w obszarze zakreskowanym. Prostymi, odpowiadającymi rzutom równej wartości funkcji celu są proste nachylone pod kątem —2/3. Strzałką pokazano kierunek zwiększenia się funkcji celu. Maksimum wystąpi zatem w punkcie $x = 0$, $y = 1$. Punkt ten będzie odpowiadał maksimum funkcji $f = ax + by$, jeżeli $b > a$; $b > 0$. Przy $a > b$, $a > 0$ maksimum wystąpi w narożu $x = 1$, $y = 0$. Jeżeli współczynniki a, b są ujemne, to maksimum będzie w 0 ($x = 0$, $y = 0$). Jest to oczywiście przykład najprostszy z możliwych. Typowy przykład z programowania liniowego opisuje się następująco: Znaleźć ekstremum funkcji:



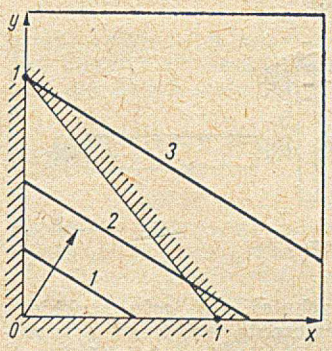
Rys. 2



Rys. 3



Rys. 4



Rys. 5

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

przy spełnieniu nierówności

$$b_{11} x_1 + b_{12} x_2 + \ldots\ldots + b_{1n} x_n \leqslant c_1$$

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

$$b_{m1} x_1 + b_{m2} x_2 + \ldots\ldots b_{mi} x_n \leqslant c_m$$

Przedstawione powyżej postępowanie nie jest dla tego przypadku oczywiste.

Technika poszukiwania [2, 3] znana jako metoda simplex prowadzi w bezpośredni sposób do rozwiązania i jest dzisiaj szeroko stosowana wśród ekonomistów i inżynierów.

1.6. Programowanie nieliniowe

Rozszerzenie programowania liniowego stanowią przypadki, gdy funkcja celu lub ograniczenia są nieliniowe. Ten dział, niestety, nie jest dotychczas zba-

dany i nie ma ogólnych metod pozwalających na rozwiązywanie tego rodzaju zadań.

Istnieje wiele różnych metod, bardzo bogate piśmiennictwo, lecz brak jest uogólnienia. Metody losowe jako jeden z możliwych przypadków programowania ruchomego omówione są w [7].

1.7. Programowanie dynamiczne [4, 6]

Zastosowano pierwotnie dla układu wielodecyzyjnego mającego tę właściwość, że można było wyraźnie wyodrębnić różne stopnie w układzie. Stwierdzono przy tym, że decyzje powzięte na kolejnych stopniach nie mają wpływu na działanie poprzedzających.

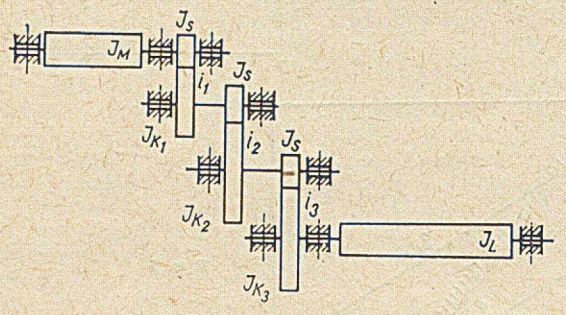
Metoda ta jest bardzo efektywna, jeżeli na każdym stopniu nie występuje zbyt dużo zmiennych decyzyjnych.

Zastosowanie programowania dynamicznego nie ogranicza się jednak tylko do procesów dyskretnych. Uwzględniając fakt, że proces ciągły możemy rozważać w granicy jako nieskończoną liczbę skończonych stopni, można programowanie dynamiczne zastosować do procesów ciągłych, a więc również do zadań z zakresu rachunku wariacyjnego.

Metoda ta zasługuje na specjalną uwagę i dlatego podajemy pewien techniczny przykład rozwiązany za pomocą programowania dynamicznego.

ZADANIE

W urządzeniu automatycznym, którego schemat kinematyczny pokazano na rys. 6, przemieszczenie kątowe przekazywane jest z wału 1 na wał 2 za po-



Rys. 6

średnictwem przekładni zębatej. W celu szybkiego przekazania tego przemieszczenia kątowego przeprowadza się minimizację momentu bezwładności przy założeniu, że bezwładność obciążenia jest odpowiednio mała. Bezwładność kół zębatych przedstawia podstawowy opór dla przyspieszenia.

Sformułowanie zadania ujmiemy następująco:

Jeżeli całkowite przełożenie jest określone przez inne wymagania, jaką należy przyjąć redukcję? (Ile stopni reduktora (1, 2, 3)? i jakie położenia na poszczególnych stopniach?).

Oznaczamy przez:

I_M — moment bezwładności wirnika

I_S — moment bezwładności każdego małego koła zębatego

I_{k1}, I_{k2}, I_{k3} — odpowiednio momenty większych kół zębatych

I_L — moment bezwładności obciążenia

i_1, i_2, i_3 — przełożenia kolejnych stopni

$R = i_1 \cdot i_2 \cdot i_3$ — przełożenia całkowite.

Przyjmujemy następującą regułę redukcji:

- moment bezwładności każdego dużego koła zębatego redukowany jest w odniesieniu do wału silnika przez podzielenie tego momentu przez odpowiedni kwadrat przełożenia;
- bezwładność każdego dużego koła zębatego obliczana jest jako moment bezwładności małego koła pomnożony przez odpowiedni kwadrat przełożenia.

Przy założeniu zatem, że momenty bezwładności wszystkich małych kół zębatych są równe sobie —

$$I_{zr} = I_M + I_S + i_1^2 I_S + \frac{I_S}{i_1^2} + \frac{i_3^2 I_S}{i_1^2 i_2^2} + \frac{I_1}{R^2}$$

albo po uporządkowaniu

$$I_{zr} = I_M + (1 + i_1^2) I_S + I_S \frac{1}{i_1^2} + \frac{i_2^2}{i_1^2} I_S + \left(\frac{1}{i_1^2 i_2^2} + \frac{R^2}{i_1^4 i_2^2}\right) I_S + \frac{I_L}{R^2}$$

1. Rozwiązanie zadania metodą analityczną

Obliczając pochodne i przyrównując je do zera, możemy teoretycznie obliczyć optymalne wartości i_1 oraz i_2. Nie należy to w tym przypadku do zadań łatwych.

$$\frac{\partial I_{zr}}{\partial i_1} = \left(2 i_1 = \frac{2}{i_1^3} - \frac{2 i_2^2}{i_1^2} - \frac{2}{i_1^3 i_2^2} - \frac{4R^2}{i_1^5 i_2^4}\right) I_S = 0$$

$$\frac{\partial I_{zr}}{\partial i_2} = \left(\frac{2 i_2}{i_1^2} - \frac{2}{i_1^2 i_2^3} - \frac{4R^2}{i_1^4 i_2^5}\right) I_S = 0$$

sprowadza się to do rozwiązania układu równań:

$$i_1^6 i_2^2 - i_1^2 i_2^4 - i_1^2 i_2^6 - i_1^2 i_2^2 - 2R^2 = 0$$

$$i_1^2 i_2^6 - i_1^2 i_2^3 - 2R^2 = 0$$

Zadanie to rozwiązano [5], korzystając z gradientowego algorytmu IOWA CADET, zakładając ograniczenia dla przełożeń:

$1 \leqslant i_1 \leqslant 3$
$1 \leqslant i_2 \leqslant 3$

Wyniki otrzymano następujące:
reduktor 3-stopniowy

$i_1 = 1,74862$
$i_2 = 2,02090$
$i_3 = 2,82983$

funkcja celu f (x) — 375,5 gcm²

reduktor 2-stopniowy

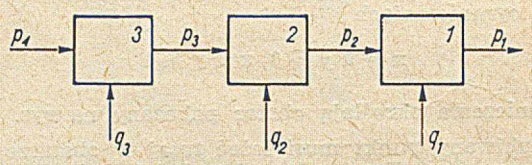
$i_1 = 2,42924$
$i_2 = 4,11651$

funkcja celu f (x) — 523,5 gcm²

Uwaga: Porównujemy tylko moment zredukowany. Do tej wartości dodaje się zawsze stała wartość momentu bezwładności małego koła i momentu bezwładności wirnika silnika.

2.2. Rozwiązanie zadania metodą programowania dynamicznego

Zbudowano schemat (rys. 7) i zdefiniowano następujące pojęcia:



Rys. 7

1) stany — p_i — przełożenia na wejściu i wyjściu ze stopnia

2) decyzja — $q_i = I_i$ — momenty bezwładności na każdym stopniu. Dla uproszczenia liczenia jako, że przykład ten ma stanowić jedynie ilustrację metody

przyjęto, że moment bezwładności koła jest funkcją **tylko** liczby zębów.

W rzeczywistości $I = f(z, m, b, \gamma)$. Przyjęto zatem stałość modułu, szerokości i ciężaru właściwego. Podkreślamy jednak, że dowolna liczba zmiennych decyzyjnych nie zwiększa metodologicznych trudności. Zwiększeniu ulega jedynie czas poszukiwania ekstremum na poszczególnym stopniu,

3) transformacja — p_n — $T_n\,(p_{n+1}, q_n)\,p_n = p_{n+1}\,\frac{z_n}{k}$

k — stała liczba zębów koła mniejszego,

4) więzy — dla uproszczenia przykładu przyjęto tylko $q_i \geqslant k$. W ogólnym przypadku tych ograniczeń może być dowolnie dużo,

5) funkcja celu — $I_{zr} = \sum_{i=1}^{n} \frac{q_i + a}{z_i}$; z_i przełożenia zależne od położenia momentu bezwładności względem wału, do którego przeprowadza się redukcję;

a — stała wartość momentu bezwładności mniejszych kół.

W obliczeniach przyjęto $a = 250$ gcm² — dla z — 14, $m = 0{,}3$ cm, $b = 1$ cm; $\gamma = 8$ gcm⁻³

Przełożenie całkowite układu określone przez inne wymagania $R = 10$.

2.2.1. Zakładamy, że jeden stopień przenosi nadany impuls kątowy.

Funkcja celu wynosi wówczas

$$f_1(p_2) = \min\,[p_1(p_2, q_1)]$$

$$f_1(p_2) = \frac{q_1 + a}{\left(\frac{p_1}{p_2}\right)^2}$$

z warunków zadania $p_1 = 10$;

gdzie

$q_1 = I_1 = (ma)_1 \cdot r^2$

$(ma)_i$ — masa i-tego koła

$r = \frac{2}{3}\,r_p$ promień bezwładności

$r_p = mz/2$ promień koła podziałowego

m — moduł

z — liczba zębów

γ — masa właściwa

$$ma = \frac{\pi\,mz^2}{4\,.\,2} \cdot b = \frac{\pi}{2}\,m^2 z^2$$

$$I_1 = (ma)_1\,r^2 = \frac{\pi}{18}\,m^4 z^4$$

$$f_1(p_2) = \frac{\frac{\pi}{18}\,m_1^4 z_1^4 + a}{\left(\frac{p_1}{p_2}\right)^2}$$

$$f_1(p_2) = \frac{(0{,}0014\,z_1^4 + 250)\,p_2^2}{p_1^2}$$

Z równania transformacji znamy:

$$z_1 = \frac{10 \cdot k}{p_2}$$

Ostatecznie zatem

$$f_1(p_2) = \frac{\frac{54 \times 10^4}{p_2^2} + 250\,p_2^2}{100}$$

Funkcja celu jest w tym przypadku funkcją przełożenia na wejściu.

Tablica I

p_2	$f_1(p_2)$
1	5402,5
2	1360,0
3	622,5
4	377,5
5	278,5
6	240,0
6,5	233,0
7	232,6
7,5	236,5
8	244,5
9	269,0
10	304,0
20	1013,5

Tablica II

p=1,75	
z	$f_2(p_3)$
20	665,0
27	391,0
28	385,0
29	392,5
30	396,5
33	411,2

Dla wymaganego przez nas przełożenia wynosi 5402,5. Zależność dla całego zakresu ujęto tablicą I.

2.2.2. Przy założeniu, że dwa stopnie reduktora wykonają narzucone zadanie konstrukcyjne, funkcję celu w ogólnym zapisie przedstawia się następująco:

$$f_2(p_3) = \min_{q_2}\,[p_2(p_3, q_2) + f_1(p_2)]$$

gdzie p_2 dana jest przez transformację stanu

$$p_2 = p_3 \cdot \frac{z_2}{k}$$

$$f_2(p_3) = \frac{q_2 + a}{\left(\frac{p_2}{p_3}\right)^2} + \frac{f_1(p_2)}{\left(\frac{p_2}{p_3}\right)^2}$$

przedstawiając zdefiniowane wielkości i porządkując, otrzymamy następujące wyrażenie na funkcję celu dla dwóch stopni:

$$f_2(p_3) = \frac{(0{,}0014\,z_2^4 + 250)\,14^2}{z_2^2} + \frac{[54 \times 10^4 + 0{,}0065\,(p_3 z_2)^4]\,p_3^2}{0{,}0026\,(p_3 z_2)^4}$$

Zgodnie z algorytmem programowania dynamicznego należy teraz poszukiwać ekstremum tego wyrażenia dla różnych założonych wartości p_3. W przypadku jednej zmiennej decyzyjnej zadanie jest uproszczone. Wystarczy dla założonego p_3 stablicować $f_2(p_3)$ w funkcji z_2. Podano tablice dla $p_3 = 1$ i dla $p_3 = 1{,}75$ (tablice II i III).

Spróbujmy zanalizować otrzymane wyniki.

Zwracamy uwagę, że dla układu jednostopniowego, reduktor zapewniający zadane przełożenie 10:1 opisany jest wartością funkcji celu równą 5402,5 gcm².

Tablica III

p=1	
z	$f_2(p_3)$
20	1300,0
28	610,0
30	562,0
33	537,0
35	528,0
36	521,0
37	528,0
38	531,0
70	1382,0

Tablica IV

p=1	
z	$f_3(p_4)$
14	585,5
16	477,0
18	410,5
20	371,5
22	348,5
23	342,0
24	338,5
25	336,0
26	342,0
28	348,0

Reduktor dwustopniowy zapewniający przełożenie 10:1 da nam wartość funkcji celu 521,0 gcm³. Odpowiednie przełożenia na stopniach wynoszą:

$i_1 = 2{,}57145$
$i_2 = 3{,}92857$

a liczby zębów na poszczególnych stopniach:

na stopniu 1 — : 14 — 36
na stopniu 2 — : 14 — 55

2.2.3. Zakładamy, że 3 stopnie przenoszą zadany impuls kątowy.

Korzystając z definicji polityki optymalnej podanej przez R. Bellmana [6] dla dalszych rozważań przyjmiemy dla stopnia drugiego wartość z = 28 otrzymaną dla p = 1,75.

$$f_3(p_4) = \min_{q_3} [p_3(p_4, q_3) + f_2(p_3)]$$

gdzie z transformacji stanu $p_3 = p_4 \frac{z_3}{k}$

$$f_3(p_4) = \frac{q_3 + a}{\left(\frac{p_3}{p_4}\right)^2} + \frac{f_2(p_3)}{\left(\frac{p_3}{p_4}\right)^2}$$

po podstawieniach i uporządkowaniu

$$f_3(p_4) = \frac{0{,}274\, z_3^4 + 49 \times 10^3}{z_3^2} + \frac{55 \times 10^3}{z_3^2} + \frac{54 \times 10^4 + 0{,}143\, z_3^4}{9{,}65\, z_3^4}$$

Z założenia nie będziemy tutaj przeprowadzali rozważań dla większej liczby stopni. Dlatego też obliczenia wystarczy wykonać jeden raz dla ustalonego $p_4 = 1$.

Wyniki uzyskane dla różnej liczby zębów ujęto w tablicy IV.

Uzyskane ekstremum dla z = 25 zębów wynosi 336 gcm².

Odpowiednie przełożenia na stopniach wynoszą zatem:

$i_1 = 1{,}64285$
$i_2 = 2{,}00000$
$i_3 = 3{,}07142$

a liczby zębów na poszczególnych stopniach

1 : — 14 — 25
2 : — 14 — 28
3 : — 14 — 43

Porównanie wyników przedstawia się następująco:

reduktor 2-stopniowy —	programowanie dynamiczne	metoda gradientowa
przełożenie i_1 =	2,57145	2,42924
i_2 =	3,92857	4,11651
funkcja celu f x =	521,0	523,5

reduktor 3-stopniowy —	programowanie dynamiczne	metoda gradientowa
przełożenie i_1 =	1,64285	1,74862
i_2 =	2,00000	2,02090
i_3 =	3,07142	2,82987
funkcja celu f (x) =	336,0	375,5

Zaletą programowania dynamicznego jest w tym przypadku również to, że przełożenia te można dokładnie zrealizować, bowiem danymi są tu liczby zębów. Natomiast liczby zębów dla przełożeń w metodzie gradientowej dobiera się później.

Zatem, albo nie uzyskuje się dokładnie wymaganego przełożenia albo też dobiera się większe liczby zębów, a to zwiększa sumaryczny moment bezwładności, bowiem wzrastają wymiary małych kół zębatych, które powinny być jak najmniejsze.

BIBLIOGRAFIA

[1] Wilde D. I. Optimum — seeking Methods, New York: Prentice Hall Inc 1964
[2] Dantzig G. B. Linear Programming and Extensions Princeton University Press, 1962
[3] Vajda S. Mathematical Programming Reading: Addison — Wesley Publishing Co 1956
[4] Bellman R. Dynamic Programming Princeton: Princeton University Press 1957
[5] Charles R. Mischke. An Introduction to Computer — Aided Design Prentice Hall, Inc Englewood Cliffs, N. Y. 1968
[6] Richard E. Bellman, Stuart E. Dreyfus. Programowanie dynamiczne, Państwowe Wydawnictwa Ekonomiczne, Warszawa 1967
[7] Zieliński R. Metody Monte Carlo. WNT, Warszawa 1970.

JANUSZ LIPIŃSKI
Politechnika Łódzka
Katedra Mechaniki Technicznej

681.322.004.14:518.12:512.3

O pewnej metodzie rozwiązywania dużych układów algebraicznych równań liniowych na maszynach cyfrowych o małych pamięciach wewnętrznych

Autor omawia doświadczenia uzyskane przy zastosowaniu jednej z metod iteracyjnych — metody kolejnych nadrelaksacji w połączeniu z procesem Aitkena. Program obliczeń, który został ułożony dla maszyny typu ZAM-2 przewiduje możliwość rozwiązania układu równań liniowych o maksimum 330 niewiadomych.

Szereg problemów technicznych i fizycznych sprowadza się do rozwiązywania dużych układów algebraicznych równań liniowych

(1) $$A \cdot X + B = 0$$

gdzie A — macierz współczynników
X — wektor — kolumna szukanych rozwiązań
B — wektor — kolumna danych wyrazów wolnych.

Mogą to być układy pełne lub też, co bardzo często spotyka się w zagadnieniach technicznych, układy zawierające w każdym równaniu tylko kilka lub kilkanaście współczynników niezerowych, skupionych wokół przekątnej.

Metod numerycznego rozwiązywania układów równań (1) jest bardzo dużo, przy czym należy dokonać ogólnego podziału na metody „dokładne" prowadzące do znalezienia rozwiązania po wykonaniu z góry określonej liczby działań oraz metody iteracyjne, w których liczba działań zależy zarówno od pewnego początkowego przyjęcia rozwiązania przybliżonego, jak i od założonej dokładności rozwiązania.

Metody iteracyjne są szczególnie dogodne do rozwiązywania układów równań zawierających znaczną liczbę elementów zerowych, a zwłaszcza wówczas, gdy posługujemy się maszyną cyfrową posiadającą stosunkowo niedużą pamięć operacyjną, jak np. maszyna ZAM-2 [1]. Z metod iteracyjnych najbardziej znane są: metoda iteracji prostych, metoda Gaussa-Seidla oraz metoda kolejnych nadrelaksacji.

W niniejszej pracy zostaną omówione doświadczenia uzyskane przy stosowaniu metody kolejnych nadrelaksacji.

Metoda kolejnych nadrelaksacji

Wzór macierzowy dla metody nadrelaksacji można przedstawić w postaci wzoru

$$X^{(n+1)} = X^{(n)} + \alpha R^n \tag{2}$$

w którym α — współczynnik nadrelaksacji

R^n — residua w n-tym kroku iteracyjnym

$X^{(n)}, X^{(n+1)}$ — składowe wektora rozwiązań w n-tym i $n+1$ kroku iteracyjnym.

Wzór macierzowy dla residuów można przedstawić w postaci

$$R^{(n)} = (P + G)\, X^{(n)} + D\, X^{(n-1)} + B \tag{3}$$

gdzie P oznacza diagonalne wyrazy macierzy A

G oznacza górną macierz trójkątną bez wyrazów diagonalnych

D oznacza dolną macierz trójkątną bez wyrazów diagonalnych

B jest wektorem-kolumną wyrazów wolnych.

Szybkość zbieżności tej metody zależy od doboru współczynnika nadrelaksacji α. Jednak przyjęcie najkorzystniejszej wartości współczynnika nadrelaksacji jest skomplikowane, gdyż jest równoznaczne ze znajomością wartości własnych macierzy iteracji. Oczywiście obliczenie wartości własnych jest zadaniem jeszcze trudniejszym i wymagającym wykonania większej ilości obliczeń. Wiadomo jednak, że wartość współczynnika nadrelaksacji zawiera się w granicach $1 \leqslant \alpha \leqslant 2$ ($\alpha = 1$ odpowiada metodzie iteracyjnej Seidla).

Proces Aitkena

W celu przyśpieszenia procesu iteracyjnego można zastosować proces Aitkena. Proces ten polega na przeprowadzeniu działań na wektorach X uzyskanych w trzech kolejnych iteracjach zgodnie z wzorem:

$$X^{(n+2)} = X^{(n)} + \frac{X^{(n+1)} - X^{(n)}}{1 - p} \tag{4}$$

p jest współczynnikiem postępu geometrycznego wyznaczanym w zależności:

$$p = \left| \frac{\varepsilon^{(n+1)}}{n} \right| \tag{5}$$

$$\varepsilon^{(n+1)} = \max\,(\varepsilon^n \mid R^n) \tag{6}$$

ε^n — dokładność rozwiązania w n-tym kroku iteracyjnym.

Tablica I

Współczynnik nadrelaksacji	Liczba iteracji		Uwagi
	bez procesu Aitkena	z procesem Aitkena	
1	830	450	interacja Seidla
1,125	647	62	
1,5	318	50	
1,75	142	54	

Proces ten daje korzyści jednak tylko wówczas, gdy dokładność rozwiązań ε^n w kilku krokach iteracyjnych osiąga wartości mało różniące się od siebie, tzn. p przyjmuje prawie stałą wartość. W przeciwnym przypadku, proces Aitkena może zaburzyć zbieżność procesu iteracyjnego.

Program obliczeń

Program obliczeń [1], który został ułożony dla maszyn typu ZAM-2 przewiduje możliwość rozwiązania układu równań liniowych o maksimum 330 niewiadomych (ograniczenie jest spowodowane pojemnością pamięci maszyny). Wartość współczynnika nadrelaksacji może być dowolna (wczytywana z kluczy) i zmieniana w czasie trwania procesu iteracyjnego na podstawie otrzymanych informacji pośrednich. Proces Aitkena jest wykonywany przez program samoczynnie.

Wyniki prób i wnioski

Przeprowadzone próby obliczeń dotyczące rozwiązywania układów równań zilustrowano rozwiązaniem układu równań o 120 niewiadomych przy żądanej dokładności rozwiązania $\varepsilon = 0{,}0001$ (liczba iteracji zależy od żądanej dokładności).

Wyniki prób przedstawiono w tablicy I.

Zilustrowane w tablicy I próby dla układu równań liniowych o 120 niewiadomych, jak również przeprowadzone próby dla innych układów równań pozwalają wyciągnąć następujące wnioski:

Przy $\alpha = 1$, czyli przy iteracji Seidla, liczba iteracji jest największa. Zwiększenie współczynnika nadrelaksacji powoduje zmniejszenie liczby iteracji, jednak nie ma praktycznie kryteriów na przyjęcie początkowej wartości współczynnika nadrelaksacji. Próby przeprowadzone przy rozwiązywaniu układów równań metodą nadrelaksacji z jednoczesnym stosowaniem procesu Aitkena pozwalają wyciągnąć wnioski świadczące korzystnie o tej metodzie.

Jak widać z tablicy I, rząd liczby iteracji przy stosowaniu procesu Aitkena nie zmienia się przy zmianie współczynnika nadrelaksacji (w granicach 1.125 — 1.750). Jest to fakt pozytywny; gdyż wskazuje na niezależność zbieżności procesu iteracyjnego od startowej wartości współczynnika nadrelaksacji α, wartości trudnej do wyboru.

Drugim ważnym zjawiskiem jest to, że liczba iteracji przy stosowaniu metody nadrelaksacji w połączeniu z procesem Aitkena jest znacznie mniejsza niż przy stosowaniu zwykłej metody nadrelaksacji.

BIBLIOGRAFIA

[1] J. Szmelter — Opis programu „Nadrelaksacja" — Biblioteka Programów Katedry Mechaniki Technicznej Politechniki Łódzkiej

[2] D. K. Faddiejew, N. W. Faddiejewa — Wyczislitielnyje mietody liniejnoj algiebry — Fiz-Mat, Moskwa 1963

[3] Nowoczesne metody numeryczne PWN Warszawa 1965.

MOC OBLICZENIOWA WYŻSZYCH UCZELNI NRF

Gdy ośrodek obliczeniowy w Darmstadt (NRF) otrzymał w roku 1963 dużą elektroniczną maszynę cyfrową IBM-7090, a większość uniwersytetów i politechnik — mniejsze jednostki, wydawało się, że zapotrzebowanie na moc obliczeniową dla celów naukowych i szkolnictwa wyższego zostanie zaspokojona co najmniej na 10 lat. Wtedy nikt nie przypuszczał, że komputer IBM-7090 już po dwóch latach będzie użytkowany przez 24 godziny na dobę. Trudności nie zostały też zlikwidowane po rozbudowie zestawu, przez co zwiększono o 50% zdolność przerobową.

Moc obliczeniowa jest artykułem deficytowym nauki zachodnioniemieckiej i pozostanie nim przez co najmniej 5 lat, choć podaż mocy obliczeniowej dla nauki szybko wzrasta.

Jak duży jest brak elektronicznych maszyn cyfrowych do celów naukowych w NRF i o ile pogorszyła się sytuacja w ciągu ostatnich lat widać z opracowania wykonanego przez ośrodek obliczeniowy w Darmstadt w listopadzie 1967 roku. W roku 1963 można było mówić o umiarkowanym wyposażeniu uczelni wyższych NRF w sprzęt obliczeniowy w porównaniu do USA. Stosunek ten wynosił 1 : 7. W roku 1965 stosunek ten pogorszył się i wynosił 1 : 12. W roku 1967 uległ dalszemu pogorszeniu i wynosił 1 : 23 przy stosunku ludności 1 : 4.

Jeszcze bardziej niekorzystny stosunek był w roku 1969, ponieważ siedem zainstalowanych w r. 1968 jednostek obliczeniowych na wyższych uczelniach USA podwoiło co najmniej posiadaną przez nie uprzednio moc obliczeniową.

Zapotrzebowanie mocy obliczeniowych dużych ośrodków naukowych w Europie co roku podwaja się. Stwierdzono, że na wyższych uczelniach i w ośrodkach badawczych USA zachodzą podobne zjawiska, jak w Europie. W zachodnioniemieckim ośrodku obliczeniowym w Darmstadt coroczne zapotrzebowanie mocy obliczeniowej wzrastało trzykrotnie od chwili uruchomienia ośrodka w roku 1962.

Podwajanie się w ciągu każdego roku zapotrzebowania mocy obliczeniowej do celów naukowych wydaje się początkowo niewiarygodne. Oznacza to, że zapotrzebowanie mocy obliczeniowej wzrasta w ciągu pięciu lat trzydzieści razy, a w ciągu sześciu lat — sześćdziesiąt razy. Przyczyna tego lawinowego wzrostu zapotrzebowania mocy obliczeniowej leży w tym, że coraz więcej dziedzin nauki wykorzystuje możliwości dawane przez komputery. Badania naukowe w szerszym zakresie były możliwe dopiero od czasu postawienia do ich dyspozycji komputerów o dużych możliwościach obliczeniowych. W początkowym okresie, komputery wykorzystywała fizyka, cybernetyka i statystyka; ostatnio chemia kwantowa, technika przewodowa, psychologia, sztuka i literatura otworzyły nowy rynek dla nienumerycznego przetwarzania danych. Specjaliści są zdania, że trend ten będzie się utrzymywać jeszcze przez kilka lat.

Jak wykazuje porównanie z USA, nauka zachodnioniemiecka wyposażona jest w elektroniczne maszyny cyfrowe w stopniu absolutnie niedostatecznym. Za bazę porównawczą przyjmujemy najbardziej swego czasu rozpowszechniony komputer IBM 7090.

W roku 1963 moc obliczeniowa wyższych uczelni USA wynosiła 28 × „7090" jednostek, w roku 1969 już 92 × „7090" jednostek, a w r. 1967 — 510 × „7090" jednostek!

Moc obliczeniowa wzrosła w ciągu czterech lat 18,2-krotnie, a więc rocznie nieco ponad dwukrotnie. Politechniki i wyższe uczelnie NRF posiadały w swych instytutach obliczeniowych w r. 1963 moc obliczeniową 4 × „7090" jednostek, w roku 1965 ponad 7,5 × „7090" jednostek, a w roku 1967 ponad 23 × „7090" jednostek. Deficyt wynosił więc w roku 1967 73 × „7090" jednostek i w roku 1968 wzrósł do 120 × „7090" jednostek.

W NRF główny ciężar obliczeń naukowych spoczywa na Ośrodku Obliczeniowym w Darmstadt. W r. 1965 na swej maszynie IBM-7090 liczył on 7000 godzin i maszyna była całkowicie obciążona. Dla zaspokojenia potrzeb nauki w NRF musiałby liczyć około 15 000 godzin, a w roku 1968 — około 60 000 godzin. Posiada on jednak tę samą, tylko nieco rozbudowaną maszynę i w praktyce w r. 1967 liczył niewiele więcej, niż w roku 1965.

Nowa maszyna TELEFUNKEN TR-440, oddana do eksploatacji w roku 1969 ma moc obliczeniową 4- do 5-krotnie większą, niż IBM-7090. Gdyby ta moc była zainstalowana w roku 1966 prawdopodobnie byłaby już wykorzystana w roku 1967.

Pewna rezerwa istnieje w nie wykorzystanej jeszcze całkowicie mocy obliczeniowej czterech komputerów, zainstalowanych w ośrodku Synchrotrona Elektronowego w Hamburgu w r. 1967: IBM 360/75 (o czterokrotnie większej szybkości, niż IBM 7090). To samo dotyczy podobnej maszyny zainstalowanej w Ośrodku Badań Jądrowych w Jurich 3800 (5 × „7090") w Instytucie Meteorologicznym w Offenbach i CD 6400 (3 × „7090") na Politechnice w Aachen. Jeśli instytucje te mają jeszcze wolną moc obliczeniową — to z pewnością ich pełne obciążenie dla potrzeb własnych osiągnięte będzie w roku 1970.

W celu przezwyciężenia braku mocy obliczeniowej w wyższych uczelniach NRF, przewiduje się wyposażenie ich w większym stopniu w maszyny IBM 7090 lub 7094. Przewidywany koszt maszyny wynosi, 3,5 do 4 mln DM. Nie zlikwiduje to jednak deficytu mocy obliczeniowej. Szczytowe zapotrzebowanie mocy obliczeniowej uczelni wyższych w NRF nie będzie mogło być pokryte przez własne ośrodki obliczeniowe. Przewiduje się pokryć je przez duże ośrodki regionalne. Plan Związku Pracowników Naukowych NRF (Forschungsgemeinschaft) przewiduje zorganizowanie ośmiu takich ośrodków, m. in. w Stutgarcie, Monachium, Hamburgu i Hanowerze. Ośrodki te mają być wyposażone w duże maszyny w cenie 18—15 mln DM. Planowany koszt budynku i instalacji dla jednej maszyny wynosi 3—4 mln DM. Pierwszy ośrodek regionalny został uruchomiony na Uniwersytecie w Stutgarcie. Wyposażony on jest w komputer CD 6600. Przewiduje się, że budynki dla dalszych ośrodków obliczeniowych zostaną wykonane do końca 1970 r. Do uzyskania pełnej mocy obliczeniowej ośrodka regionalnego i przygotowania się do obsługiwania swych klientów ze wszystkich kierunków badań, łącznie z przetwarzaniem danych, upłyną co najmniej dalsze 3 lata. Ośrodek wyposażony w maszynę w roku 1969 osiągnie pełną zdolność przerobową dopiero w roku 1972. Plan ośrodków regionalnych (jeśli będzie w tym tempie realizowany) zostanie wykonany w pełni dopiero za dziesięć lat. Nie można więc liczyć, aby w okresie najbliższych 5 lat wzrost mocy obliczeniowej dla nauki NRF był choćby w przybliżeniu równy wzrostowi zapotrzebowania.

„INFORMATIONEN", nr 01-69
Wyd. „Institut für Datenverarneitung" Dresden

Opracował
Tadeusz Wróblewski