ARTYKUŁY

POSTĘPY ASTRONOMII
Tom XXV (1977). Zeszyt 4

PROGRAMY KOMPUTEROWE DO SYMBOLICZNYCH PRZEKSZTAŁCEŃ ALGEBRAICZNYCH

Część I

JĘZYK PROGRAMOWANIA LISP

ANDRZEJ KRASIŃSKI

Centrum Astronomiczne im. M. Kopernika PAN (Warszawa)

MAREK PERKOWSKI

Instytut Automatyki Politechniki Warszawskiej

СИМВОЛИЧЕСКИЕ АЛГЕБРАИЧЕСКИЕ ПРОГРАММЫ ДЛЯ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

Часть I

ЯЗЫК ПРОГРАММИРОВАНИЯ ЛИСП

А. Красиньски, М. Пэрковски

Содержание

Статья описывает использование вычислительных машин для символических алгебраических вычислений в физике и астрономии. В настоящей первой части, как пример языка манипуляции символами, описан на элементарном уровне язык ЛИСП и его применение к алгебраическим вычислениям.

SYMBOLIC ALGEBRAIC COMPUTER PROGRAMS

Part I

THE LISP PROGRAMMING LANGUAGE

Summary

The paper describes the role of computers applied to symbolic algebraic formulae manipulation in physics and astronomy. In the present first part of the work the LISP programming language is described on an elementary level as an example of a symbol manipulation language. Its application to algebraic calculation is shortly described.

1. WSTĘP

Obliczenia dokonywane za pomocą komputerów przeszły ewolucję analogiczną do ewolucji matematyki uprawianej bezpośrednio przez ludzi. Zanim wynaleziono algebrę, ludzie umieli przez kilka tysięcy lat rachować jedynie na liczbach. Podobnie komputery elektroniczne przez pierwszych kilka lat swojego istnienia służyły wyłącznie do rachunków numerycznych. Powstałe od tamtej pory wyspecjalizowane języki rachunków numerycznych umożliwiają dziś pisanie bardzo pomysłowych, wyrafinowanych matematycznie programów, które, niestety, maszyna może realizować tylko po podstawieniu danych liczbowych pod wszystkie symbole. Jest to sytuacja analogiczna do tej, która panowała w matematyce późnego średniowiecza, gdy ludzie umieli już formułować pewne twierdzenia o treści algebraicznej, lecz czynili to za pomocą słownych opisów operacji, które należy wykonać na danych liczbowych. Wszyscy wiemy, ile zawdzięczają rachunkom numerycznym niektóre dziedziny fizyki i techniki i nikt rozsądny nie będzie tych osiągnięć negował, lecz do wielu celów rachunki numeryczne są niewystarczające i mogą dostarczyć zaledwie namiastki potrzebnego wyniku (przykład: przez numeryczne całkowanie równań ruchu Newtona dla dwu ciał oddziałujących grawitacyjnie nigdy nie dowiedzielibyśmy się, że ich orbity powinny być krzywymi zamkniętymi. Moglibyśmy tylko z coraz większą dokładnością stwierdzać, że w granicach błędu rachunku są one zamknięte. Dla celów „praktycznych" byłoby to może wystarczające, lecz o ile uboższa byłaby nasza wiedza, gdyby problem ten nie był rozwiązany analitycznie!).

Z drugiej strony w wielu dziedzinach nauki występują rachunki symboliczne, które są czysto algorytmiczne, tzn. przebiegają wg ściśle ustalonego schematu i nie wymagają twórczego myślenia, lecz tylko wielokrotnego stosowania tych samych reguł rachunkowych do coraz bardziej komplikujących się wyrażeń. Typowym przykładem takiego rachunku jest liczenie tensorów Riemanna i Ricciego w teorii względności. Jest to zajęcie bardzo czasochłonne i wymagające dużej niezawodności rachunkowej, ma więc wszystkie cechy zadania dla komputera.

Pierwsze programy do obliczeń symbolicznych (programy różniczkujące) zostały stworzone, niezależnie od siebie, już w 1953 r. (Kahrimanian 1954; Nolan 1953). Były one pisane w językach wewnętrznych maszyn. Programy symboliczne mogą być również w zasadzie

pisane przy użyciu języka programowania numerycznego, np. FORTRANu, jest to jednak zadanie trudne i uciążliwe, toteż poczynając od połowy lat pięćdziesiątych powstało szereg języków przeznaczonych specjalnie do obliczeń symbolicznych. Największe spośród nich znaczenie ma język LISP (skrót od LISt Processor), opracowany przez Johna McCarthy'ego oraz implementowany pod jego kierownictwem w M.I.T. (McCarthy i inni 1962). W oparciu o język LISP powstały setki programów i prac teoretycznych, dotyczących teorii programowania, obliczeń symbolicznych w różnych dziedzinach nauki, badań nad językiem naturalnym, sztucznej inteligencji i robotyki, logiki matematycznej, automatycznego dowodzenia twierdzeń i grafiki komputerowej. Podstawową zaletą LISPu jest wielka uniwersalność, pozwalająca użytkownikowi na tworzenie w zasadzie dowolnego nowego języka, struktur danych i mechanizmów obliczeniowych, a przy tym jego względna prostota i podobieństwo niektórych konstrukcji do używanych w matematyce. Zalety te okupywane są stosunkowo niewielką efektywnością i nietypową notacją, toteż obecnie z LISPem konkuruje na polu obliczeń symbolicznych kilkanaście innych języków, m. in. opartych o FORTRAN.

W dalszym ciągu artykułu będziemy zajmować się tymi zastosowaniami programowania symbolicznego, które nazywane są obliczeniami algebraicznymi*. Zastosowania te mają obecnie największe znaczenie w fizyce i naukach pokrewnych i dotyczą wszelkiego rodzaju działań (symbolicznych i numerycznych) w klasach wyrażeń symbolicznych nad ciałem liczb wymiernych, liczb rzeczywistych lub zespolonych oraz funkcji i operatorów na tych ciałach. Nie wspominamy tu o wykorzystaniu programowania symbolicznego w samej matematyce: algebrze abstrakcyjnej (teorii grup, ciał, pierścieni itp.), geometrii wielowymiarowej, nieeuklidesowej, topologii, teorii liczb i logice matematycznej, ani o zastosowaniach algebraicznych dotyczących innych struktur algebraicznych (np. teorii grup w fizyce, ciał Galois w teorii sterowania, czy algebry Boole'a w teorii automatów). Czytelnika zainteresowanego tymi zagadnieniami odsyłamy do bibliografii opublikowanej przez Sammet (1968).

W tym samym czasie, gdy powstał LISP i wkrótce potem, pojawiła się seria nowych programów wykonujących obliczenia algebraiczne dla potrzeb elektrotechniki teoretycznej (Goldberg 1959), mechaniki nieba (Gerard i in. 1965), teorii względności (Walton 1965, Fletcher i in. 1967) i elektrodynamiki kwantowej (Hearn 1966).

Pierwsze programy były z wielu przyczyn kłopotliwe w użyciu. Niektóre z nich miały tę samą wadę, co niewprawny rachmistrz: nie umiały skutecznie wykonywać skracania przy mnożeniu i redukcji przy dodawaniu. Produkowały więc wyniki poprawne, ale niepotrzebnie skomplikowane, co w trudniejszych przypadkach prowadziło do „zatkania" całej pamięci maszyny. Często wymagały też wprowadzania danych zapisanych w specjalnej notacji języka programowania i w tej samej notacji wypłowadzały wyniki. Jedną z takich notacji przedstawimy w artykule; jest ona wprawdzie logiczniejsza od matematycznej, ale bardzo nieczytelna dla nieprzyzwyczajonego użytkownika. Nieczytelność, będąca wspólną cechą wszystkich notacji programowych, sprawia, że wyniki wymagają automatycznego przetłumaczenia lub przepisania na notację matematyczną. Przepisywanie bardzo złożonych

---

*Przyjęta już nazwa „algebraic computing" jest nieco myląca, gdyż obejmuje nie tylko operacje algebraiczne jak dodawanie i mnożenie, lecz również operacje takie jak różniczkowanie i całkowanie; natomiast jest stosowana tylko do działań w niektórych strukturach algebraicznych.

wyrażeń z jednej notacji na drugą było nadzwyczaj uciążliwe i stwarzało nie mniej okazji do popełnienia błędu niż sam rachunek. Z tego powodu korzystanie z pierwszych programów było nieraz ze strony użytkownika aktem poświęcenia dla przyszłego rozwoju tej dziedziny programowania.

Doświadczenia zebrane w ciągu kilkunastu lat sprawiły, że obecnie działa kilkadziesiąt wygodnych w użyciu programów, wykonujących rozmaite wyspecjalizowane rachunki algebraiczne w matematyce i naukach przyrodniczych, oraz kilka systemów obliczeń algebraicznych ogólnego przeznaczenia. Ich rozpowszechnienie napotyka jednak pewne przeszkody techniczne. Programy te wymagają maszyn o dużej pamięci i szybkości, a ponadto często nie mogą być w prosty sposób przenoszone z maszyny na maszynę, gdyż wymagają specjalnych języków bazowych, które bywają niedostępne na innych maszynach niż ta, dla której program był pisany, lub dostępne w odmiennym wariancie. Systemy oparte o FORTRAN są stosunkowo łatwe do przenoszenia, gdyż wiele maszyn posiada translator tego języka, lecz napisanie programów symbolicznych w LISPie jest mniej pracochłonne niż utworzenie najpierw języka programowania symbolicznego opartego na FORTRANie czy ALGOLu, a następnie napisanie w tym języku programów wyspecjalizowanych. Obydwa rozwiązania pociągają za sobą kilka do kilkunastu miesięcy wytężonej pracy i niezbyt wielu ludzi ma ochotę podejmować ten wysiłek.

W Polsce, poczynając od 1968 r., opracowano kilka translatorów języka LISP: na Uniwersytecie Warszawskim, Wojskowej Akademii Technicznej i Politechnice Poznańskiej. Nie dysponowano jednak maszyną o wystarczającej szybkości i pamięci, zaś w środowisku potencjalnych użytkowników brakowało informacji. Te przyczyny sprawiły, że bardzo pożyteczne narzędzie pracy nie zostało wykorzystane w praktyce do obliczeń algebraicznych.

Nowa szansa pojawiła się w roku 1973, gdy uruchomioną w Instytucie Badań Jądrowych w Świerku maszynę CDC CYBER 73, dostępną dla bardzo szerokiego grona użytkowników, wyposażono w język LISP. W następstwie tego powstała mała grupka ludzi zajmujących się nim do różnych celów zawodowych. Szczęśliwym zbiegiem okoliczności jeden ze specjalistów od LISPu, współautor niniejszego artykułu (M.P.) był szkolnym kolegą jednego z relatywistów, drugiego autora (A.K.). W tych warunkach nawiązanie współpracy nie przedstawiało większych trudności. Podjęliśmy działalność w dwóch kierunkach. Dowiedzieliśmy się, że od roku 1969 działa na podobnej maszynie CDC na Uniwersytecie w Londynie program CLAM, napisany przez d'Inverno i Russell-Clarka (d'Inverno 1975), liczący lewą stronę równań Einsteina. Ponieważ autorzy wyrazili w swoich publikacjach gotowość wysłania programu nagranego na taśmie, poprosiliśmy ich o to i w maju 1977 r. otrzymaliśmy nagranie od dra Johna Packera z Computer Unit przy King's College w Londynie*. Program ten działa już na maszynie CDC w systemie CYFRONET. Osoby, które chciałyby wykorzystać program CLAM mogą otrzymać od autorów niniejszego artykułu instrukcję korzystania z niego (d'Inverno i Russell-Clark, brw). Równocześnie postanowiliśmy napisać w języku LISP nasz własny program, wyliczający tensory krzywizny wg algorytmu stosującego rachunek form zewnętrznych. Program ten, o

*Zdajemy sobie sprawę z tego, że pp. d'Inverno i Packer nie przeczytają niniejszego artykułu, pragniemy jednak wyrazić im naszą wdzięczność, ponieważ stali się dobroczyńcami „polskiej szkoły teorii względności".

nazwie ORTOCARTAN, został już uruchomiony, zaś obecnie znajduje się w fazie optymalizacji. Ma on pewne dodatkowe możliwości w stosunku do programu CLAM. Być może napiszemy o nim dokładnie przy innej okazji.

Możliwości zastosowań programów algebraicznych w fizyce i naukach pokrewnych są niemal nieograniczone, lecz w całym świecie zbyt mało znane i wykorzystywane. Chcielibyśmy w naszym artykule zaznajomić wszystkich potencjalnych użytkowników takich programów z językiem LISP oraz zwrócić ich uwagę na korzyści wynikające z nowego, obok rachunków numerycznych i sterowania aparaturą, zastosowania komputerów w nauce. Pierwsza część artykułu będzie poświęcona tym elementom języka LISP, które bywają przydatne przy korzystaniu z programów obliczeń symbolicznych, zaś w części drugiej omówimy przykłady zastosowań (już dokonanych oraz możliwych przyszłych) języków programowania symbolicznego w fizyce i astronomii.

2. JĘZYK PROGRAMOWANIA LISP*

Język ten posługuje się dwoma podstawowymi pojęciami: atomu i listy.

A t o m e m nazywamy liczbę lub dowolny ciąg znaków drukarskich nie zaczynający się od cyfry i nie zawierający wewnątrz odstępów, okrągłych nawiasów, kropek, przecinków ani znaków $ (z przyczyn technicznych atom nie może przekraczać pewnej krytycznej długości, dla CDC CYBER 73 wynosi ona 30 znaków). Atom jest najmniejszym, w zasadzie niepodzielnym, elementem programu w LISPie. Jego długość i rodzaj użytych znaków, jeśli spełniają podane ograniczenia, są wyłącznie sprawą użytkownika, maszyna bowiem w ten sam sposób potraktuje np. symbol A, co symbol BARDZODLUGIATOM i symbol TROCHE38CYFR53W73SRODKU, jeśli będą użyte w trzech różnych programach w tym samym znaczeniu.

Przykłady atomów: A, X2, IS1IN2, A*B**C, 12, ATOM.

Przykłady wyrażeń nie będących atomami: (ATOM), 3DELTA, 1/2, DOLLAR$.

L i s t ą nazywamy ciąg zaczynający się lewym okrągłym nawiasem, zawierający atomy porozdzielane odstępami lub pojedynczymi przecinkami, i zamknięty prawym okrągłym nawiasem. Listą jest też ciąg list ujęty w nawiasy. Szczególnym przypadkiem listy jest lista bez elementów, czyli lista pusta ( ), która jest równocześnie atomem (z definicji). Dla wygody programisty można ją oznaczać przez ( ), albo, gdy chcemy podkreślić raczej jej atomowy aspekt, przez NIL. Przed i za nawiasem listy odstępy nie są konieczne (ale są dozwolone), zaś jeden odstęp ma dla programu to samo znaczenie, co wiele kolejnych odstępów.

Przykłady list: (ATOM), (DWA ATOMY), (BARDZO(((SKOMPLIKOWANA))(LISTA))), (1 / 2), (( )), (5 A), (3 5), (A).

Atomy i listy są razem nazywane wyrażeniami symbolicznymi (w skrócie: S - w y r a ż e n i a m i). Język LISP służy do wykonywania odwzorowań jednych S-wyrażeń na inne. Programy napisane w LISPie posługują się do tego celu funkcjami standardowymi,

---

*Język LISP został dokładnie opisany w cytowanej książce M c C a r t h y' e g o, ogólne omówienie zawiera też, łatwiej dostępna, książka T u r s k i e g o (1973). Bardzo dobrym podręcznikiem LISPu jest S i k l o s s y (1976).

umieszczonymi na stałe w translatorze LISPu, oraz funkcjami zdefiniowanymi przez samego użytkownika dla potrzeb jego programu.

Działanie programu w LISPie opiszemy przy użyciu przykładu. W tym celu musimy poznać trzy podstawowe funkcje języka LISP: CAR, CDR i CONS.

Argumentem funkcji CAR jest dowolna niepusta lista, zaś jej wartością – pierwszy element listy, np.:

CAR ((OJ NIEDOBRZE NIEDOBRZE)) da wynik OJ,
CAR ((((A) (JEDNAK)) (CORAZ LEPIEJ))) da wynik ((A)(JEDNAK)),
CAR ((A)) da wynik A.

Argumentem funkcji CDR jest również dowolna niepusta lista, zaś wartością CDR jest ta sama lista z pominiętym pierwszym elementem – np.:

CDR ((JAK DLUGO JESZCZE)) da wynik (DLUGO JESZCZE),
CDR (((((JUTRO))(MOZE)) BEDZIE (JESZCZE LEPIEJ))) da wynik
(BEDZIE (JESZCZE LEPIEJ)).
CDR ((A)) da wynik ( ), czyli listę pustą.

Funkcja CONS ma dwa argumenty. Pierwszym może być dowolne S-wyrażenie, drugim dowolna lista. Wynikiem CONS jest dopisanie pierwszego argumentu na pierwszym miejscu listy, która była drugim argumentem CONS. Przykłady:

CONS ((DZIS TAK) (JUTRO INACZEJ)) da wynik ((DZIS TAK) JUTRO INACZEJ),
CONS ((RAZEM) (((PRZYJACIELE)))) da wynik ((RAZEM) ((PRZYJACIELE))),
CONS (A ( )) da wynik (A).

Za pomocą tych trzech funkcji napiszemy teraz program, który będzie przekształcał zdanie (MOJE SLOWA TO PRAWDA) w zdanie (PRAWDA TO MOJE SLOWA). Przedtem jednak musimy zapoznać się z ogólną zasadą wywoływania funkcji w programach. Istnieją dwa sposoby wywoływania:

1) Jeżeli wywoływana funkcja jest wystawiona bezpośrednio pod działanie programu (tzn. jest wywoływana na poziomie zewnętrznym), wywołanie składa się z nazwy funkcji, po której następuje l i s t a (w sensie języka LISP) jej argumentów. Przykłady takich wywołań podaliśmy wyżej.

2) Jeśli wartość wywołanej funkcji ma stać się argumentem innej funkcji (tzn. funkcja jest wywoływana na poziomie wewnętrznym), wywołanie jest listą, której pierwszy element jest nazwą wywoływanej funkcji, zaś następne – s y m b o l a m i kolejnych j e j a r g u m e n t ó w. W LISPie rozróżnia się symbole zmiennych i ich wartości na tej samej zasadzie, co w algebrze i analizie (wartości zmiennych wcale nie muszą być liczbami, np. wartością symbolu A może być atom B, lista (A B C D) lub jakiekolwiek inne S-wyrażenie). Wywołania takie pojawiają się głównie w definicjach funkcji wprowadzanych przez użytkownika.

Oznaczmy listę (MOJE SLOWA TO PRAWDA) symbolem L. Program wygląda następująco:
(CONS (CAR(CDR(CDR (CDR L)))) (CONS (CAR(CDR(CDR L))) (CONS (CAR I)(CONS (CAR(CDR L)) NIL)))).

Wypisanie takiego S-wyrażenia w definicji funkcji spowoduje, że program zapamięta schemat postępowania, a przy wywołaniu tak zdefiniowanej funkcji zastosuje ów schemat do S-wyrażenia, które będzie aktualną wartością zmiennej L. Nie podajemy sposobu definiowania nowych funkcji, ponieważ zaprowadziłoby nas to zbyt głęboko w szczegóły.

Przykład ten ukazuje dwie cechy charakterystyczne LISPu:

1) Po kilku minutach nauki można napisać pierwszy program.

2) Czytelność programów w LISPie jest bardzo utrudniona przez dużą liczbę nawiasów pojawiających się w wyrażeniach. Liczenie nawiasów i sprawdzanie ich prawidłowego ustawienia urasta często do osobnego problemu (z tego powodu ktoś wymyślił kiedyś złośliwe objaśnienie skrótu LISP: Lots of Irritating Single Parentheses). Jest to utrudnienie, ale nie ma na to rady – taka już jest wewnętrzna logika LISPu.

Podane przykłady pozwalają uzmysłowić sobie potencjalne możliwości języka LISP. Atomami i listami są (lub, przez wzięcie w nawiasy, łatwo mogą być uczynione) wyrazy, zdania i teksty języków naturalnych, zatem LISP może służyć do sporządzania katalogów i manipulowania zawartymi w nich informacjami oraz do automatycznego tłumaczenia tekstów z jednego języka naturalnego na inny. S-wyrażeniami i ciągami S-wyrażeń są równania matematyczne, skąd wynika możliwość stosowania LISPu do rachunków „algebraicznych". Listy mogą reprezentować zbiory w sensie teorii mnogości, a więc programy w LISPie mogą wykonywać operacje na zbiorach. Listami są też programy w LISPie, zatem jedne programy mogą być modyfikowane przez inne programy, a także istnieją możliwości pisania programów samomodyfikujących się oraz programów piszących programy. Duża liczba zastosowań wynika też z dogodnej reprezentacji grafów*, którą daje LISP. LISP może służyć do kodowania pewnych informacji o świecie zewnętrznym i wybierania najwłaściwszej reakcji na dany zespół informacji, a więc do pisania programów sterujących robotami. Dla nas najbardziej interesujące będą zastosowania LISPu do rachunków algebraicznych, toteż poświęcimy im nieco więcej uwagi.

3. REPREZENTACJA DZIAŁAŃ ALGEBRAICZNYCH W LISPIE

Podana w poprzednim paragrafie zasada wywoływania funkcji dotyczy wszystkich bez wyjątku funkcji i odwzorowań, a więc również dodawania, mnożenia i potęgowania (ograniczymy się do przykładów wywołań na poziomie wewnętrznym). Na przykład sumę A + B + C w konwencji LISPu pisze się jako (PLUS A B C), iloczyn $5 \cdot X \cdot Y$ pisze się jako (TIMES 5 X Y), zaś potęgowanie X^A zapisuje się w postaci (EXPT X A). Konwencja ta wydaje się na pierwszy rzut oka niezręczna i nieczytelna, lecz w rzeczywistości jest o wiele bardziej logiczna od matematycznej. W matematyce również z reguły pisze się najpierw nazwę funkcji lub odwzorowania, a dopiero potem jego argumenty, tylko dla dodawania, mnożenia i potęgowania zrobione są wyjątki od tej ogólnej zasady. Symbole + i • (które są przecież symbolami odwzorowań) pisze się nie przed, lecz między argumentami, wielokrotnie powtórzone, zaś symbol potęgowania wcale nie istnieje, zastąpiony specjalnym ustawieniem argumentów względem linii druku W istocie właśnie to jest niezręczne i niekonsekwentne, lecz nie zauważamy tego faktu, bo zbyt mocno przyzwyczailiśmy się to takiego zapisu.

O zaletach notacji LISPu (zwanej notacją polską** lub prefixową, w odróżnieniu od matematycznej: infixowej) można łatwo się przekonać, próbując napisać program wykonujący

---

*Grafem nazywamy uporządkowaną parę $G = \langle N, S \rangle$ gdzie N jest zbiorem węzłów zaś S zbiorem jego strzałek, czyli trójelementowych list $s = (n, d, m)$ gdzie $n, m \in N$ zaś d jest opisem strzałki s. Węzły (reprezentowane przez S-wyrażenia) mogą odpowiadać dowolnym obiektom zaś strzałki relacjom pomiędzy nimi. Wypływają stąd zastosowania grafów do reprezentacji różnego typu struktur, układów dynamicznych, języków, obrazów, modeli i innych złożonych systemów.

**Jej twórcą był polski logik Jan Łukasiewicz (Łukasiewicz 1929).

tłumaczenia z jednego zapisu na drugi. Każde wyrażenie w notacji LISPu ma jednoznaczny sens, natomiast w notacji matematycznej istnieje sporo umownych, trądycyjnych elementów nie uwidocznionych w zapisie, które program musi zapamiętać, a następnie mozolnie wyszukać przed przystąpieniem do właściwej pracy. Są to np.: zwyczajowa hierarchia działań (najpierw potęgowania, potem kolejno: mnożenia, dzielenia oraz dodawania z odejmowaniem), możliwość napisania lub opuszczenia plusa przed pierwszym składnikiem sumy, możliwość pomijania znaku mnożenia, pewna dowolność w użyciu nawiasów (nawiasy nie zmieniające hierarchii działań nie zmieniają sensu zapisu), ten sam symbol „–" używany dla odejmowania i dla operacji zmiany znaku itd. LISP ma tylko jedną regułę: najpierw wyliczyć wyrażenia w nawiasach.

Nie zamierzamy namawiać użytkowników matematyki, aby porzucili notację infixową; chcemy tylko podkreślić, że używanie notacji prefixowej jest bardzo istotnym ułatwieniem przy pisaniu wszelkich programów algebraicznych, np. upraszczających, różniczkujących itd.

Dobrze napisany program nie wymaga jednak od użytkownika nawet tego elementu wiedzy o LISPie. Użytkownik powinien znać jedynie pojęcia atomu i listy, aby zrozumieć rygory, którym podlega pisanie danych. Istnieje bowiem możliwość włączenia do programu procedur wykonujących tłumaczenie z notacji LISPu na dowolną inną i odwrotnie. W związku z tym można urządzić program tak, że na wejściu użytkownik będzie pisał dane w notacji FORTRANu – bardzo podobnej do matematycznej, z tą tylko różnicą, że potęgowanie jest zapisywane jako <podstawa potęgi> ** <wykładnik> (urządzenia wejściowe na ogół nie pozwalają pisać w kilku równoległych liniach). Wydruk można zorganizować jeszcze wygodniej – w normalnym formacie matematycznym, z potęgami i górnymi wskaźnikami nad linią druku, z dolnymi wskaźnikami pod linią. Tak właśnie jest zorganizowany wydruk w programie CLAM*.

Opisane poprzednio funkcje arytmetyczne PLUS, TIMES i EXPT, wraz z kilkoma innymi (np. MINUS – funkcja zmieniająca znak, DIFFERENCE – dwuargumentowa funkcja wyliczająca różnicę, QUOTIENT – część całkowita ilorazu dwu liczb całkowitych, REMAINDER – reszta z podzielenia) są umieszczone na stałe w translatorze LISPu, lecz umieją działać automatycznie tylko na liczbach. Jedna ze standardowych funkcji LISPu, DEFINE (wspominana już pośrednio) umożliwia użytkownikowi definiowanie zupełnie dowolnych operacji na dowolnych listach. Definicje te są dołączone do systemu LISP na czas działania programu. W ten sposób można rozszerzyć działanie niektórych funkcji arytmetycznych z liczb na symbole. Dla prostoty rozszerzonym funkcjom (teraz już algebraicznym) nadaje się zwykle te same nazwy, co odpowiednim funkcjom arytmetycznym. Rozszerzenie to nie jest wcale proste i w każdym programie obliczeń symbolicznych stanowi jego najważniejszą i najtrudniejszą część. Kłopot polega na tym, że rozszerzone w ten sposób funkcje nie potrafią automatycznie wykonywać upraszczania (tzn. skracania i redukcji wyrazów podobnych). Źle napisany program nie pozna, że $ab/a - b = 0$, i będzie ciągnął wyrażenie $(ab/a - b)$ dalej przez wszystkie rachunki. Skuteczny algorytm upraszczania musi być ułożony przez samego programistę i włączony do definicji działań algebraicznych. Poprawność tego algorytmu jest sprawą wielkiej wagi. Na przykład w CLAMie zdarza się, że w

*Na świecie istnieją już eksperymentalne systemy, w których zarówno wprowadzanie jak i wyprowadzanie danych dokonywane jest w normalnym zapisie matematycznym, łącznie z ułamkami, całkami itd. Przykładem takiego systemu jest radziecki system ANALITYK (Bugrij, 1977).

dodawaniu pojawia się dwukrotnie ten sam iloczyn, raz ze znakiem +, drugi raz ze znakiem –, lecz za każdym razem inaczej uporządkowany. CLAM nie potrafi automatycznie zredukować takich wyrażeń; redukcja wymaga ingerencji użytkownika w przebieg programu. Bez niej następuje katastrofalny rozrost nieuproszczonych wyników pośrednich, prowadzący do przekroczenia zakresu pamięci maszyny i wyrzucenia programu przed końcem realizacji.

W Polsce dysponujemy już w tej chwili skutecznym programem upraszczania, opracowanym przez Wardęckiego (1973) oraz udoskonalonym i uruchomionym na maszynie przez autorów. Może on być bez większych zmian włączany jako część składowa do wielu rozmaitych programów obliczeń algebraicznych.

W ten sposób doszliśmy do problemu tworzenia nowych programów w oparciu o fragmenty już istniejących. Zajmiemy się nim bliżej w następnej części artykułu.

LITERATURA

Bugrij, A. 1977, inf. prywatna.

d'Inverno R. A., 1975, Gen. Rel. Grav., 6, 567.

d'Inverno R. A., Russell-Clark R. A.,bez roku wyd., *CLAM–CDC LISP Algebraic Manipulator*, part 1+2.

Fletcher J. G., Clemens R. W., Matzner R. A., Thorne K. S., Zimmerman B. A., 1967, Astrophys. J. Lett,. 148, 91.

Gerard J. M., Izsak I. G., Barnett M. P., 1965, Commun, Comp. Mach. 8, Assoc. 27.

Goldberg S. M., 1959, *Solution of an electrical network using a digital computer.* M.S. Thesis, M.I.T.

Hearn A. C., 1966, Commun. Assoc. Comp. Mach. 9, 573.

Kahrimanian H. G., 1954, *Symp. on Aut. Progr. for Digital Comp.*, Office of Naval Research, Dept. of the Navy, Washington D.C., 6.

Łukasiewicz J., 1929, *O znaczeniu i potrzebach logiki matematycznej*, Nauka Polska, Tom X.

Mc Carthy J., Abrahams P.W., Edwards D.J., Hart T.P., Levin M.I., 1962, *LISP 1.5 Programmer's Manual*, MIT Press.

Nolan J., 1953, *Analytical Differentiation on a Digital Computer* (MA Thesis) M.I.T. Cambridge, Massachusetts.

Sammet J. E., 1968, *Proc. of the IFIP Working Conference on Symbol Manipulation Languages*, Bobrow, D.C. (ed.), North Holland Publ. Comp. Amsterdam, 358.

Siklossy L., 1976, *Let's talk LISP*, Prentice Hall, Englewood Cliffs, N. J.

Turski W., 1973, *Podstawy użytkowania maszyn cyfrowych*, Wyd. 2 zmienione, PWN, Warszawa.

Walton J. J., 1965, *Formula manipulation code for the calculation of Christoffel symbols*, UCRL–14177 Univ. of Calif. Lawrence Rad. Lab., Livermore, Calif.

Wardęcki M., 1973, *Pewne heurystyczne metody całkowania symbolicznego*, Praca magisterska wykonana na Wydziale Matematyki i Mechaniki UW.

Przypis dodany w czasie korekty:

Program autorów ORTOCARTAN jest już ukończony i posiada wydruk w formacie matematycznym. Po dokładnym przetestowaniu zostanie on skomplikowany, czyli przetłumaczony na kod wewnętrzny maszyny CDC, i udostępniony użytkownikom.