Lecture Notes in Computer Science, 47

Methods of Algorithmic Language Implementation

Edited by
A. Ershov and C. H. A. Koster

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

47

Methods of Algorithmic Language Implementation

Edited by A. Ershov and C. H. A. Koster

Springer-Verlag
Berlin · Heidelberg · New York 1977

Editors
Prof. A. Ershov
Computing Center
Novosibirsk 630090/USSR

Prof. C. H. A. Koster
Faculteit der Wiskunde
en Natuurwetenschappen
Katholieke Universiteit
Toernooiveld
Nijmegen/The Netherlands

AMS Subject Classifications (1970): 68-02, 68A30, 90A05, 90A15
CR Subject Classifications (1974): 4.12, 4.2.

ISBN 3-540-08065-1 Springer-Verlag Berlin · Heidelberg · New York
ISBN 0-387-08065-1 Springer-Verlag New York · Heidelberg · Berlin

Printed in Germany
Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr. 43210

Preface to the English Edition

On September 10-13, 1975 Novosibirsk, USSR, a Symposium on Methods for the Implementation of Algorithmic Languages was held, attended by 58 Soviet delegates and 11 from outside the Soviet Union, including 5 Western delegates, invited by Prof. A.P. Ershov, who organised the Symposium.

The Russian edition of the proceedings of this Symposium contained 33 papers; 11 have not been offered for inclusion in the present English publication. Among the papers left out are some that are more tutorial in nature: for others, the reason for their exclusion may be in the technical problems in translating the paper into English and typing it in a camera-ready form. Those readers with a sufficient command of the Russian language are therefore referred the the Russian text of the proceedings, which was printed at the Computing Centre of the Siberian Branch of the Academy of Sciences in Novosibirsk.

The English edition of the proceedings allows an assessment of the state of the art in compiler writing in the Eastern countries, and of the main areas of interest (especially by the choice of Western invitees). I will not try to make such an assessment, but will just mention some of the points that struck me:

- Great interest in translator writing systems.
- Relatively large concern for syntax analysis methods (three papers).
- Unique interest in multi-language translators (four papers, including three on aspects of the BETA System, a combined implementation of PASCAL, PL/1, ALGOL 68 and SIMULAR 67: three more papers on the BETA systems appearing only in the Russian edition).
- Relatively little interest in MOHLLs and SILs (one paper in the proceedings on the Jarmo language).
- Interest in very high level languages.
- Relatively large stress of the fundamental and the formal, rather than a pragmatic approach.
- An earnest desire to form part of the international computing community and good awareness of western professional literature. (witness, e.g., references of papers; in the other direction, the information, or maybe the interest, is much smaller).
- A strong tradition in algorithmic languages (preponderance of ALGOL, leading through the ALPHA language to the BETA system. The FORTRAN problem hardly arises).
- An exemplary faithfulness to language standards (full implementation of internationally accepted language, rather than egotistic subsets, ameliorations or home brew).

A list of the contributors at this symposium is given on page VII.

It is hoped that these proceedings will allow insight in the work being done on compilers in the Soviet Union, and a view of the persons doing it, leading to responsible communication and collaboration between eastern and western scientists.

C.H.A. Koster.

Soviet Participants

Institute of Automation and Control, Wladywostok
A.S. Kleshchev

Research Institute of Appl. Math. and Cybernetics, Gorki
Yu.L. Ketkov

Kiev State University, Kiew
V.N. Red'ko

Institute of Cybernetics of the Ukrainian Ac.Sc., Kiew
I.V. Velbitsky
Yu.K.Kapitonova
G.E. Tseitlin

Kishenev State University, Kishenev
D.N. Todoroy

Institute of Math. and Comp. Center of the Moldavian Ac.Sci., Kishenev
M.G. Gontsa

Pavlov Institute of Physiology, Leningrad
V.L. Tyomov

Comp. Center of Leningrad State University, Leningrad
G.S. Tseitin
A.N. Terekhov

Leningrad Branch of Central Inst. of Mathematical Economics of the USSR Ac.Sci., Leningrad
I.V. Klokachev

Institute of Math. of the Byelorussian Ac.Sci., Minsk
G.K. Stolyarov
N.V. Shkut

All-Union Institute of Scientific and Technological Information, Moscow
A.N. Maslov

Computing Center of the USSR Ac.Sci., Moscow
V.M. Kurochkin

Computing Center of Moscow State University, Moscow
E.A. Zhogolev
V.Sh.Kaufman

Institute of Applied Mathematics, USSR Ac.Sci., Moscow
Yu.M.Bayakovsky
E.Z. Lubimsky

Institute of Precise Mech. and Comp. Machinery of the USSR Ac.Sci., Moscow
D.B. Podshivalov

Institute of Electronic Control Computers, Moscow
L.A. Kalinichenko

Research Center of Electronic Computing Machinery, Moscow
A.S. Markov

Central Institute of Mathematical Economics, Moscow
M.R. Levinson

Computing Center of Rostov State University, Rostov-on-Don
A.L. Fuksman
S.P. Kritsky

Research and Development Institute of Technology, Tallin
E.H. Tyugu

Institute of Math. of the Siberian Branch of the USSR Ac.Sci., Novosibirsk
V.N. Agafonov
L.T. Petrova

Computing Center of the Siberian Branch of the USSR Ac.Sci., Novosibirsk
V.G. Bekasov
A.A. Baers
L.V. Gorodnyaya
V.V. Grushetsky
A.P. Ershov
L.L. Zmievskaya
V.N. Kasyanov
V.L. Katkov
S.K. Kozhukhina
L.A. Korneva
S.E. Kozlovsky
V.E. Kotov
D.Ya.Levin
P.K. Leonov
R.D. Mishkovich
A.S. Narinyani
V.A. Nepomnyashchy
O.N. Ochakovskaya
Yu.A.Pervin
S.B. Pokrovsky
I.V. Pottosin
A.F. Rar
V.K. Sabelfeld
F.G. Svetlakova
G.G. Stepanov
M.B. Trakhtenbrot
B.G. Cheblakov
L.B. Efros
T.S. Yanchuk

Foreign Participants

GDR
I. Kerner
J. Lehmann
R. Strobel

Poland
J. Borowiec
Z. Pawlak

Czechoslovakia
J. Kral

Great Britain
B. Marks

Berlin (West)
C.H.A. Koster

USA
J. MacCarthy
D. Schwartz

France
B. Lorho

Contents

Problem-oriented Languages and DEPOT Implementing System

N. Joachim Lehmann
Technische Universität Dresden
Sektion Mathematik

1. Trends of Programming Languages

The large scale application of electronic information processing would have been unthinkable without the supply of suitable programming languages. It was thus possible to include specialists of various fields immediately in program preparation of computer installations without having to subject them to unhandy programming with machine commands. FORTRAN marked a successful start, whereas ALGOL 60 was the first mathematically sophisticated conception of an algorithmic language with a formalised syntax description. Both languages were highly oriented to numerical activities.

After almost 20 years of programming language development, two contradictory trends currently exist:

- a differentiation in numerous (smaller) special languages with limited application, but especially problem-friendliness and easy learnability and handling;
- an integration towards developing only a few extensive universal languages with expanded application. Proven language conceptions and practical data structures are standardized and compacted.

Both trends are objectively founded and will continue to develop; in natural languages these include special languages problem-oriented to parts of objective reality as highly specialised tools of intellectual work; whereas common languages supply the foundation for their definition and offer crosslinks between special disciplines. In computer application this would seem to represent the relationship of special and universal programming languages. However, the thus implied linkage of initially contradictory developmental tendencies must be implemented in practice first of all. This has essential consequences on translator engineering, the theory of higher programming languages

as well as their application.

The significance of programming languages is thus expressed in a manner as unnoticed in the original intent. With increasing certainty of handling, most users change to "their programming language". It is no longer a tool, but moreover becomes guideline of user thinking and presentation manner. The formation of a methodically built and well structured programming language which promotes logical thinking and acting must be emphatically stressed. Only then can special languages be sensibly created.

2. Special Languages and Language Systems [8]

Universality, availability of all important language conceptions and a good logical structure are the advantages of a properly constructed universal programming language. This entails however a few shortcomings: the extent makes handling and learning difficult. In any case, missing problem orientation must be compensated by a certain breadth of presentation and by utilizing expansion mechanisms at the expense of the language. The high expressability of the universal language is only poorly exploited in each individual case.

This is where problem-oriented programming languages come in. They employ the respective application termini and consider "pet" ways of thinking and working. In operations, formulas and procedures the basic algorithms for the behavior of complete subsystems of the respective work object and associated linking relations are included; the language is oriented to task structure. Thus decisive relations and influences are highlighted and the creative work of man is undivided on the essence of his task. The limitation to a special field and low redundance make them relatively easy to learn and handle for the individual specialists. The significance is demonstrated by the vivid saying of H. Scholz, a logic specialist, who 20 years ago stated [11]:

"It is always amazing to see what good symbolics can do. They show the structure symbolized by them to the intellectual eye as X-rays produce the skeleton of man to the naked eye."

Subject-oriented programming languages are thus matched to parts

of infinitely varied reality and cannot be replaced by universal languages without loss of efficiency.

In practice special languages are drafted and determined under varying premises:

- few simple language constructions are used to make problem-oriented descriptions of very restricted job classes. Easy learnability and handling without previous knowledge dominate. Formations of this kind – often hardly worthy of the term language – are frequently used in operating systems to describe editing instructions and similar.
- precondition is familiarity with a universal programming language, the orientation of which to special job classes is obtained by extensions and additional elements. SIMULA based on ALGOL 60 for implementing simulation jobs is a known example.

Whereas in the second case the question of interdisciplinary cross-links between various special languages seems solvable if proceeding from a standardized basic language, the first case makes no notice of it.

In implementing many special languages translators (or interpreters) are required to transform each program directly or indirectly through an intermediate language with compiler into the machine command list. The intermediate language can be selected as very elementary and machine-oriented or on a higher level. Approach method decisions can be made from the standpoint of translator or as dictated by communication practice. Here the latter, application-oriented standpoint is brought to the fore, which surely would lead to a technically proper solution.

3. Basic Languages and Pretranslator Principle

Working with a multitude of special languages definitely requires the use of universal compilers or compiler-writing systems. Single manual or computer-aided production must in our day be limited to individual cases in which either highly effective, optimizing compilers are required or other special features must be considered.

Precondition for each language transformation is a description of syntax and semantics of the respective special language. Instead of going into too elementary description forms – Backus-Naur – or 2 stage Wijngaarden grammars only as examples–let reference be made to an appropriate high-level programming language as problem-oriented tool. This would permit describing semantics in a form matched to the example of common language; in addition the relationship between several special languages is maintained and work with language families is possible. Furthermore the definition of special languages by universal ones relieves the latter by constructions which should be kept available for unhampered language extensions. Thus the whole system includes not only the advantages of universal languages but also those of special description means, giving in the field of programming a working method equal to the relationship of special and common language. It confirms as well the well known pretranslator principle: (Fig. 1).

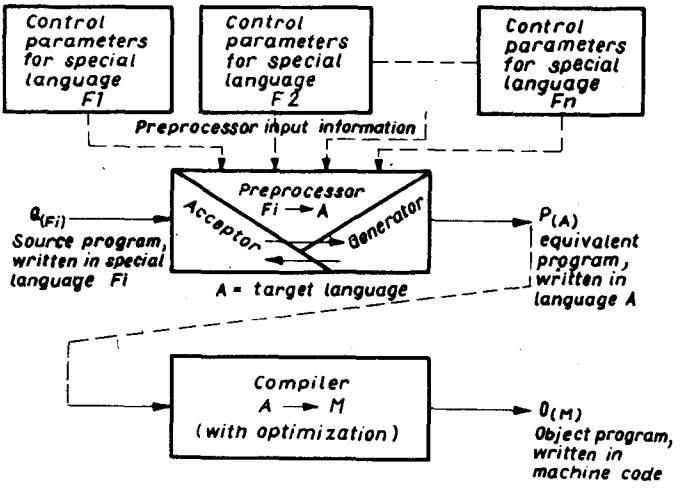

Fig. 1 Pretranslator principle

The basic-target language of the system (called "basic language" in the following) must be very logical, well structured and universal. Its level should be a "common language", but in no manner a conven-

tional assembler – though the term basic language is still often used in this sense. Regardless of considerable shortcomings, PASCAL, ALGOL 68 and PL/1 are currently best applicable for this purpose.

4. DEPOT Special Language System

At the Mathematics Department of the University of Technology of Dresden in 1973 the DEPOT (DrEsden PrOgram Transformation) special language system was finished as fully automatic implementing system according to the pretranslator principle. Not only the special languages to be transformed, but also basic ones are mostly free selectable. In Dresden often BESM 6/ALGOL is used as latter. This offers the possibility of character string manipulation, library work as well as access to FORTRAN input-output statements, and includes a well optimized compiler. The use of machine code as target language is permissible, however with limitations in problem-orientation and simple handling.

The basic structure of the system is shown in Fig. 2 .

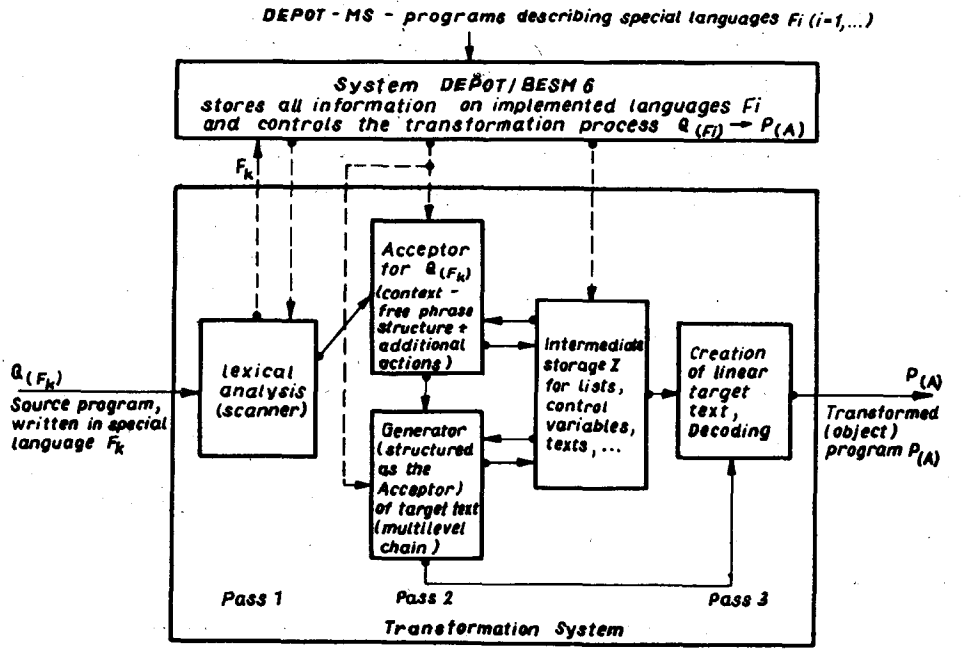

Fig. 2 DEPOT special language system

DEPOT-MS is a problem-oriented meta-language to describe syntax and semantics of special languages Fi relative to the significance of basic language A. The semantics are thus defined through the connected compiler for A by the machine language. Each DEPOT-MS program describes a special language to which a scanner (to describe lexical elementary units), a program acceptor (with context-free basic structure), such a language generator and a decoding program for the final structure of the object program $P_{(A)}$ are automatically produced. These subprograms are then available at random in DEPOT operating and library system. As soon as a program $Q_{(F_K)}$, described in special language F_K, is run in the associated acceptor becomes active. It controls the language generator which starts building an equivalent program in the target language. Both procedures need not be synchronized. The acceptor as well as generator can refer to mutually stored lists and stacks, thus picking up and generating widely independent text ranges. The significance of actual texts is thus referred to the semantics of the basic language, considered to be known.

The function of the acceptor analogous to the generator is described by an especially dynamic grammar which acts as context-free at all times but considers additionally list contents and variables reservations of an intermediate storage Z as values of syntactical or control variables. Z is filled up or transformed by actions scattered in the grammar production rules during accepting or generation. These actions are here about 10 standardized instructions which can be imbedded and executed in production rules if the syntactical analysis has penetrated to them. This covers all previously – quite extended – required context dependencies. (An appropriate proof of completeness in reference to sufficiently large language class is still missing.)

The DEPOT program system is currently implemented on the BESM 6 computer and written in BESM 6/ALGOL. A transformation to PASCAL which offers about the same range of action is executable without difficulty. According to the availability of these languages systems portability is ensured.

Program range of the whole system is marked by 5,000 ALGOL instructions. A special operating system supports work, especially

error information and if required print outs, setting up libraries for standard procedures and many more are intended to ease implementing special languages and translating special language programs.

5. Problem-oriented Language Description

It is decisive for the easy handling of DEPOT special language system that the description of required grammars does not depend immediately on Backus-Naur or similar schemes. It seems to be a contradiction in itself if the description of problem-oriented languages is only possible with elementary production rules of not sufficient transparence in their effect. Who would want to execute in numerical practice a root calculation on the basis of Markow's algorithm. This basic philosophy determines the essential differences to many other language description methods, e.g. [3], [10]. A comparable solution to DEPOT represents CDL [6] to a certain degree. The DEPOT-MS meta-language in language description proceeds on problem-oriented word structure functions, the structure of which is also algebraically founded.

5.1 Context-free Syntax Representation with Word Functions [1], [4], [5], [7]

Let Σ denote a finite alphabet, Σ^* the set of words over Σ with ε as empty word. The power set $\mathbb{P} = \mathrm{Pot}(\Sigma^*)$ of Σ^* is a lattice in reference to set theory operations $\cup$, $\cap$ with the inclusion as associated order relation; related to concatenation (symbol $\circ$ receives highest priority and may be left out)

$$\alpha \circ \beta = \alpha\beta = \{ab \mid \forall a \in \alpha, \forall b \in \beta\}\ , \qquad \alpha,\beta \in \mathbb{P}$$

and the union result in a semiring.
For calculations in algebra $(\mathbb{P}, \circ, \cup)$ various elementary functions have proven purposeful. Only the simplest are listed which are useful in the following example. (In the following all elements of $\mathbb{P}$ which have only one letter (terminals) of Σ are termed by the same. For the union we have the simplification $|.$)

With α, α_i, $\beta_i \in \mathbb{P}$ and $J = \{0,1,2,\ldots\}$ are defined:

a) conditional identity

(1) $\langle\alpha\rangle_p := \begin{cases} \alpha & \text{in case } p = \text{true} \\ \varepsilon & \text{otherwise} \end{cases}$

with p as a predicate independent of α (e.g.: $\langle\alpha\rangle_{i>0} = \alpha$ if $i > 0$, $= \varepsilon$ otherwise).

b) selection function

(2) $\langle\alpha_1,\ldots,\alpha_n\rangle_i := \begin{cases} \alpha_i & \text{for } i \in \{1,\ldots,n\} \\ \varepsilon & \text{else} , \end{cases}$

c) power function

(3) $\alpha^m := \begin{cases} \varepsilon & \text{for } m = 0 \\ \underbrace{\alpha\circ\alpha\circ\ldots\circ\alpha}_{m \text{ time}} & \text{for } m \geq 1 , \end{cases} \quad m \in J ,$

d) product

(4) $\prod_{i=n}^{m} \alpha_i := \begin{cases} \varepsilon & \text{if } m < n \\ \alpha_n \circ \alpha_{n+1} \circ \ldots \circ \alpha_m & \text{if } m \geq n \end{cases} , \quad m,n \in J ,$

e) list product

(5) $\prod_{i=n}^{m} \langle \alpha_i \beta_i \rangle := \begin{cases} \varepsilon & \text{for } m < n \\ \alpha_n \beta_n \circ \ldots \circ \alpha_{m-1} \beta_{m-1} \alpha_m & \text{for } m \geq n . \end{cases}$

The sets of numbers and of simple arithmetic expressions are used as application examples presented by ALGOL 60.
To avoid confusion terminals are occasionally placed in brackets { }, | serves as separation mark.
With $\delta := \{0|1|2|3|4|5|6|7|8|9\}$, $\sigma := \{+|-|\varepsilon\}$ follows the number presentation

(6) $\{\text{numbers}\} := \sigma\langle \bigcup_{i,j,k\in J} \delta^i \langle\cdot\rangle_{j>0} \delta^j \langle 10^{\sigma}\rangle_{k>0} \delta^k \rangle$ and $i+j+k \geq 1$.

From $\omega := \{+|-|*|/|\uparrow|\div\}$ and $\Phi := \{\text{numbers}\} \cup \{\text{variables}\} \cup \{\text{functions}\} \cup \langle\{(\underline{\text{if}})\ldots\{)\}\rangle$

we receive for arithmetic expressions

(7) $\left\{\begin{matrix}\text{simple}\\ \text{arith.}\\ \text{expr.}\end{matrix}\right\} := \sigma\langle \bigcup_{p_i,q_i\in J} \prod_{i=1}^{j} \langle\langle \{(\} \sigma\rangle^{p_i} \Phi \{)\}^{q_i} \underline{\omega}\rangle$

with $j \in \{1,2,\ldots\}$ and $\varkappa(r) = \sum_{i=1}^{r} (p_i - q_i) \begin{cases} \geq 0 \text{ for } r < j \\ = 0 \text{ if } r = j \end{cases}$
as context conditions.