CHAPTER 6.

Appendix: HISTORICAL REMARKS ON COMPILER CONSTRUCTION

F. L. Bauer

Technical University of Munich

Munich, Germany

Historical Remarks on Compiler Construction

D. E. KNUTH [81] has observed (in 1962!) that the early history of cimpiler construction is difficult to assess. Maybe this, or maybe the general unhistorical attitude of our century is responsible for the widespread ignorance about the origins of compiler construction. In addition, the overwhelming lead of the USA in the general development of computers and their application, together with the language barrier, has in fact favoured negligence of early developments in Middle Europe and in the Soviet Union.

Far from being able to give a thorough and complete history of compiler writing - I hope to find one day the time for doing the immense reading and screening involved in such an enterprise - I will only try to give a few remarks together with bibliographical notes, that may help some interested readers to penetrate into the historical work

F. L. Bauer et al., *Compiler Construction*

and to do their own investigations. I should also give the warning that I may be biased: My own attitude to compilers has in the early 60's strongly favoured syntax-controlled analysis over syntax-directed analysis [1], faithful to the economic principle of doing things rather once for all than repeatedly. Moreover, I have been involved myself at some time with the development, and know some parts of it better than others.

Nevertheless, I would venture the hope that historical studies are still considered to be more than digging out material for patent courts and for the entertainment of students, that historical assessment is an indispensable part of science and in particular of a discipline that has keenly called itself "computer science" and is now compelled to live with the claim.

---

[1] In the sense of FLOYD [51]

1. Prehistorical 'Gedanken-compilers'

Translation is a conversion of sentences from one language to another, which preserves the meaning. A compiler translates descriptions of algorithms from a language suitable for human use to a more hardware-oriented language specifying somehow more elementary actions. In this sense, the problem of 'reading' a complicated arithmetical formula, particularly with the intention to do some calculation according to this prescription means actually a mechanical mental process (which is helped by some drill received at junior highschool). Thus, logicians, looking behind the curtain of obviousness in mathematics, were close to studying the processes of compiling arithmetical and other "algebraic" formulae, totally independent of the existence and use of computers.

The parenthesis-free or 'polish' notation which was introduced by J. LUKASIEWICZ in the late 20's [88], today frequently called prefix notation, was a perfect notation for the output of a compiler as defined above, and thus a step towards the actual mechanization and formulation of the compilation process.

A further step was made 1953 by BURKS, WARREN, WRIGHT [28] in specifying a right-to-left check algorithm for well-formed parenthesis-free Boolean formulae. However, H. ANGSTL, in 1950, had designed a mechanism that solved the problem, see [14].

Mere 'rules of spelling' for the parenthesis-free notation had been formalized and proved by MENGER 1932 and SCHRÖTER 1943 [91], [113] (for formulae with parenthesis in the most general case apparently 1934 by KLEENE [79] and 1941 by CHURCH [36]). In 1950, P. ROSENBLOOM's book [103] already contained a chapter on formal languages, based on work by POST [99] in 1943, and thus the tools for describing even the syntax of the class of arithmetic and other 'algebraic' formulae with parentheses. The aim, however, was mathematical proof theory and the mechanism heavy.

This was only changed when N. CHOMSKY in 1957 introduced very special classes of languages [32], [33], [34], [35]. Mention should also be made of a rather unsuccessful attempt by WELLS in 1947 [125]. All this happened in the realm of logic. An exception is K. ZUSE. Parallel to, but independent of, his pioneering work in hardware and functional design, ZUSE specified in 1945, immobilized by war events in a small bavarian village, an algorithm that determines whether an expression with parentheses is well-formed and tackles already simplification problems like double negation and removal of superflous parentheses. ZUSE writes a program, fully operational as an application example of his Plankalkül [129], [130]. But it only describes a check algorithm, the compilation itself is lacking; there is also no indication in ZUSE's work that he had considered mechanical processing in his Plankalkül, that he had even seen

the possibility. [1]

Thus, according to our knowledge, it was H. RUTISHAUSER who was (1951) the first to describe a compilation process both in its full operational detail and in its actual importance [104], [105], [106]. The Z4 computer accessible to RUTISHAUSER was actually not suited to do the compilation, RUTISHAUSER could only perform a Gedankenexperiment, that strongly influenced the design of the Swiss ERMETH [120]. RUTISHAUSER's method was to establish the parentheses level contour map (Klammergebirge) and then to work down level by level, see Fig. 1.

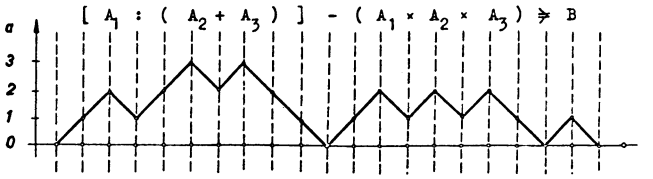

Fig. 1. Parentheses level contour map.
Facsimile from Rutishausers original paper.

2. The first compilers

RUTISHAUSER has used the term "Automatische Rechenplanfertigung". "Automatic programming" was a similar term, in usage since 1951 [127]. A compiler was originally a program that "compiled" subroutines [126]. When in 1954 the combination "algebraic compiler" came into use [66], [67], or rather into misuse, the meaning of the term had already shifted to the present one. In the german language area, "Übersetzer" was favoured, and language-conscious people like PERLIS [97], [7], used more properly "translator". But in the sloppy way so typical for a wide part of the computer community, compiler became the dominating term.

A number of early compilers in the more trivial sense was presented in 1954 at the US Office of Naval Research Symposium on Automatic Programming. Names to be mentioned are ADAMS and LANING (Comprehensive compiler, Summer Session Compiler) [2], RICE (APS III Comiler) [101], GOLDFINGER (NYU Compiler) [55], BACKUS and HERRICK (IBM 701

[1] KNUTH, in [81], simplifies history too much when merely writing "a complete history of compilers should mention the work of ZUSE ..."

Speedcoding) [8], somebody (Remington Rand A-2 Compiler) [1], BROWN and CARR III (MAGIC I) [27] and GORN [56].

Remarkable was in 1954 the first "algebraic compiler" written [2], [84] by ADAMS, LANING and ZIERLER for the Whirlwind and the proclamation of a universal, machine independent programming language by J. W. CARR III [27]; also a first practical attempt in this direction: APT, a common programming language proposed by B. RICH [102] which is essentially a prefix notation and thus escaped from the more subtle problems of arithmetic formulae.

Restricting the notation was obviously one way to ease the actual compilation process. Among the two trivial ways, suppressing operator precedence (as RUTISHAUSER had done) and requiring full parenthesation, the second one is of particular interest, since there is a simple mechanical way of establishing this form by inserting virtual parentheses (see Fig. 2), which was also in the logicians' folklore, according to KALMAR.

readily. An ingenious idea used in the first FORTRAN compiler was to surround binary operators with peculiar-looking parentheses:

+ and − were replaced by))) + (((and))) − (((

* and / were replaced by)) * ((and)) / ((

** was replaced by) ** (

and then an extra "(((" at the left end ")))" at the right were tacked on. The resulting formula is properly parenthesized, believe it or not. For example, if we consider "(X + Y) + W/Z," we obtain

((((X))) + (((Y)))) + (((W))/((Z)))

This is admittedly highly redundant, but extra parentheses need not affect the resulting machine language code. After the above replacements are

Fig. 2. The full parenthesation trick. Faksimile from Knuth's 1962 Survey paper [81].

C. BÖHM [19], in his 1952 Zurich thesis under RUTISHAUSER could therefore restrict his interest to the full parenthesized case; for this case he was first to show that a sequentially working process could be used in place of RUTISHAUSER's process, which worked on the whole formula (see Fig. 3).

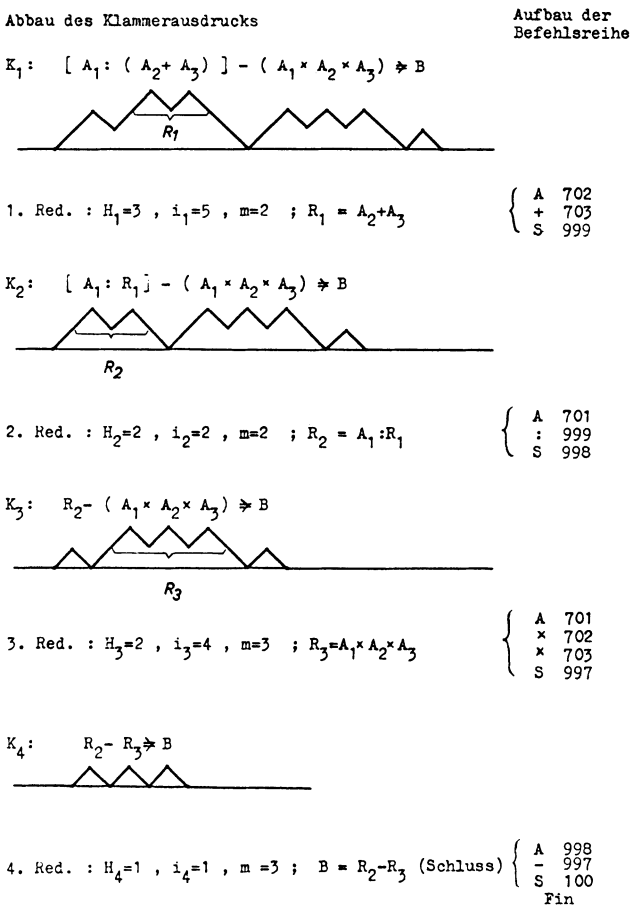

Fig. 3. RUTISHAUSER's method of layered reduction. Faksimile from [105].

Thus, BÖHM was ahead of the first FORTRAN compiler by SHERIDAN [116], which used the same technique. See also [10], [115]. The other way, suppressing operator precedence and thus requiring sufficient parenthesizing, was used in 1956 by A. J. PERLIS in the IT compiler [97]. He was also able to give a sequential algorithm. Again, BÖHM had already briefly discussed this case. Also, some early attempts by NAMUR in 1954 [92] are outdated by BÖHM.

In the Soviet Union, Ljapunov and Yanov [86], [75], [76] developed, without relation to compilation, in 1957 an algorithmic notation. KANTOROVIC showed in 1957 how a parsed formula can be represented with help of a tree [78] (see Fig. 4).

3. Sequentially working compilers

In the middle of the 50's, it had become evident that compilation of arithmetical formulae was only a prototype of the general problem of compiling a reasonable programming language. Moreover, machines were still slow and had restricted storage capacity so that effective compilation methods were sought for, in particular in the poverstricken European quarters, where rumours about the size of the FORTRAN system encouraged people particularly to do it better in order to do it at all. Thus compilation techniques were aimed at what would work sequentially for *unrestricted* arithmetic formulae and thus for the prototype of most general nested and agglomerated structures.

Described in terms of RUTISHAUSER's parentheses level, the simplest possibility is the following: to proceed until the first closing parentheses is found and then to work down from there. This was proposed by BOTTENBRUCH 1957 [20], was used by ADAMS and SCHLESINGER (1958) for the IBM 650 [3] and, as far as we could see, was also found by ERSHOV in 1958 [43], [41], [42] and used in the "Programming Program" for the BESM. The GAT Compiler of ARDEN and GRAHAM [5] works also in this way (and shows other striking parallels with the Russian approach). GARWICK also independently discovered this method, as mentioned in [18], and maybe others. Clearly this method may proceed farther than necessary, since for RUTISHAUSER's parenthesis levels, full parenthesation is not assumed.

A more sophisticated method proceeds in the parentheses contour map until the first pair of relative maxima is found, and works down from there (see Fig. 4, the numbering indicates the order). For dyadic operators, this is intuitively the most effective method. It has been found by SAMELSON in 1955 [107] and was used by BAUER and SAMELSON in 1957 [16] in designing the hardware of a machine accepting 'algebraic code'.[1)]

[1)] Another early attempt in the direction of 'high level language computers' was made in the SEAC design, see [117]. In the Burroughs B 5000 [29] and in Ferranti KDF9 [64] design, as well as in microprogrammed machines, to some extent a revival of the idea of high level language computers can be found.

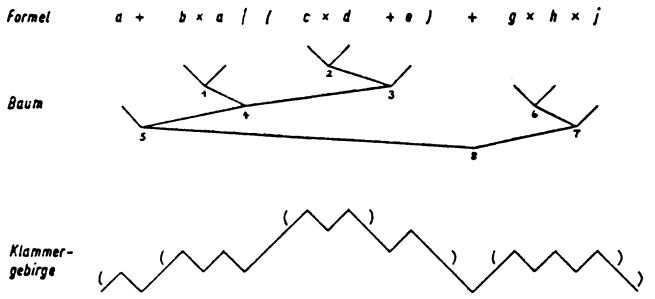

Bild 11. Darstellung eines Ausdrucks als Formel, als Baum (nach *Kantorovich*) und als Klammergebirge (nach *Rutishauser*)

Fig. 4. RUTISHAUSER's parentheses level contour map, KANTOROVIC's tree and the sequential method of SAMELSON-BAUER. Facsimile from the BAUER-SAMELSON 1961 survey paper [18].

It was the basis of the work preparing the GAMM proposals [53], [15] for the Zurich 1958 ALGOL conference and subsequently of the ALCOR compilers for ALGOL, the prototype and the first of which was written 1958 (working in spring 1959) for ALGOL 58 by PAUL for the ZUSE Z22, a small machine with a drum memory of 2000 cells. It was followed in the compilers for ALGOL 60 by PAUL for the Z22, by SCHWARZ for the ERMETH, by SEEGMÜLLER for the PERM, by HILL and LANGMAACK for the Siemens 2002, see [109] and [111]. WEGSTEIN (1959) independently found [124] essentially the same solution, which was characterized by SAMELSON and BAUER as "left-to-right (or right-to-left) scanning, deferring symbols just as long as necessary and evoking action as soon as possible" [16].

The outbreak of ALGOL stimulated independent, more or less systematic work on a number of other early compilers, among which we mention from 1959 in the WEGSTEIN line KANNER [77], in the PERLIS line KNUTH with RUNCIBLE [80],then GORN and INGERMAN [57], [70]; from 1960 and later FLOYD [46], [47], HUSKEY and WATTENBURG ([68], [69], [123]) with the NELIAC adherents (see [65]). Apparently, MAUCHLY, HOPPER and Anatol W. HOLT with the Remington Rand people [118], [45], [119], were also on this road. The Mercury Autocode people around BROOKER [21], [22], [23], [24], [25], [26], and some others will be mentioned later at appropriate places. [1)]

[1)] Further material that might be relevant, concerning work by DAHN and by BARTON and TURNER [12] seems to be difficult to get at.

Almost all the efficient sequential compilation methods [1] had used for intermediate storage of not yet processed parts a 'last in - first out' store, also called 'stack', 'cellar' [2] (from German 'Keller'), 'pushdown store'. RUTISHAUSER had stored intermediate results in a stack-like fashion, (see Fig. 3), but had not stressed at all this point. In fact, such a last in - first out store had already been used by ANGSTL and BAUER in the logical (relay-) computer STANISLAUS designed in 1951 [4], [14], and this experience was the starting point for SAMELSON (1955), to use a "cellar" not only for deferred intermediate results, as needed even with parenthesis-free notation, but also for deferred operations. The principle was independently found elsewhere and other intuitive imaginations were involved, for example a railyard shunt served for DIJKSTRA [38]. General aspects were also taken into account in 1961 by ARDEN, GALLER and GRAHAM [6] describing the MAD compiler, and in 1963 by GREIBACH [61].

In 1959 [109] (an English translation [110] appeared in 1960) [3] BAUER and SAMELSON summarized their technique as applied to a full language and worded it in terms of a transition table with input characters and the state-determining top position of a cellar. In a colloquium with logicians 1960 in Berlin (see [82]), the interpretation as a pushdown automaton, a generalization of a finite state automaton was particularly stressed. This coincided with A. G. OETTINGER's (1961) independent introduction of pushdown automata [93], which originated from earlier considerations of mechanical translation of natural languages. Following OETTINGER's approach, P. C. FISCHER [44] had reached in 1959 about the same mastery of languages with parentheses that may be omitted either in the presence of associative operators or in the presence of operators with hierarchical difference, as the ALCOR group. Pushdown automata were further investigated by SCHÜTZENBERGER [114], the design of the transition table was studied by CONWAY [37] and later by GRIES [62a]. All methods, so far, had been "bottom-up" in modern terminology. Except for OETTINGER's, who apparently had used a top-down parsing ('predictive analysis' of the linguistics) and HUSKEY's. This line will be taken up again in 6.

---

1) It seems that the NELIAC compilers had not used explicitely a stack (see GRIES, Compiler Construction for Digital Computers. Wiley 1971, p. 153.) but recursive descent (see 6.) instead.

2) The expression 'cellar' came from a particular set-up of wire lines in the design drawing of STANISLAUS, see below.

3) See also [17].

4. "Syntax controlled compilers"

It was obvious that the transition tables of the BAUER-SAMELSON and of the OETTINGER pushdown automaton reflected the structure of the language. The problem of deriving them mechanically from a structural language description, was given to PAUL and solved in his Mainz thesis of 1962 [94], see also [96] and [95] as well as [108]. The generality, however, in which PAUL had attacked the problem, resulted in a complicated process. Nevertheless, later independent attempts to solve this problem for restricted types of language, for example for operator languages with precedence properties of the kind PERLIS had used, by FLOYD in 1963 [48], would have been corollaries of PAUL's results, had those been known. Work in this direction was interpreted as refining the grammar in order to cut off dead-end roads and thus defining deterministic transitions. Results by EICKEL et al. [40] in 1963 were generalized by FLOYD [49], GRAHAM [58], and IRONS [74] in 1964, introducing bounded context grammars. The first paper that actually showed how to construct tables for a (m,n) left-to-right bounded context recognizer was by EICKEL [39][1]. The most general class LR(k) of grammars that could be treated with a deterministic pushdown automaton, which is allowed to look to all the symbols in the stack and the k symbols to the right, was specified (and shown how to construct the recognizer) by KNUTH. Still, the class is at present somewhat too general to be used unrestrictedly in practice. Here we reach the modern and recent development, as discussed by HORNING in this course.

5. Compilers based on precedence

The idea to use precedence between adjacent symbols in controlling the recognizer, introduced intuitively by PERLIS 1956 and used also by others, was generalized in the NELIAC compilers and elsewhere to the use of precedence between neighbouring pairs of operators. It led FLOYD in 1963 to introduce precedence grammars, for which a recognizer was easily constructed [48]. FLOYD's operator precedence grammar was paralleled by the simple precedence grammar of WIRTH and WEBER [128], the (1,1)-case of the more general (m,n)-grammars which were also introduced by WIRTH and WEBER, but, according to GRIES, 'the definition was wrong' and was corrected later by him [62].
In 1966, McKEEMAN found a generalization of simple precedence, the (1,2)(2,1) precedence. Again, we reach here the material of this course.

---

1) GRIES writes, 'the paper is not only hard to read, but also hard to obtain'. The second is not true, several hundred copies have been sent out on request, and copies can still be obtained from the Math. Inst., Techn. Univ. Munich; therefore anyone who wants may find out for himself whether the first assertion is true.

The further development seems to approach (1,1) bounded context and thus to converge. There are in any case deeper connections between: Use of precedence for controlling their pushdown transitions was discussed by BAUER and SAMELSON in 1959 [17]. In 1961, M. PAUL and C. A. PETRI (see [111]) independently found that parts of the ALCOR pushdown automaton transition tables reflect a simple linear ordering of the entry pairs. This was later used in the ALCOR ILLINOIS compiler by GRIES, PAUL and WIEHLE [63].

6. Syntax-directed compilers

Syntax-directed compilation was in its pure form first proposed by GLENNIE in 1960 [54]. He already treated the algebra of recognition graphs and used it intuitively in order to avoid backup. Parsing by explicite recursive descent was proposed by LUCAS in 1961 [87], describing a simplified ALGOL 60 compiler by a set of recursive subroutines that closely followed the BNF syntax. Clearly, this was top-down analysis, the call-and-return stack of the recursive procedure implementation mechanism [1] serving as a pushdown store, and it was without backup because it contained appropriate tests. It was somehow similar to GRAU's [60], [59] compiler in 1961, using recursive procedures, which is not surprising since both were based on the transition tables for a deterministic pushdown automaton that were in current use in the ALCOR group. As a result, it was more clearly seen in the ALCOR group how pushdown state symbols reflected syntactic states, and this stimulated work on syntax-controlled processor generators. HUSKEY [68], [69] also used recursive descent. Then, in 1961, IRONS described a parsing algorithm [72], [73] driven by tables which described the syntax, but his method was a mixture of top-down and bottom-up. His method needed backup and so did BROOKER's [25], [26], [23]. The idea was fascinating (many papers: WARSHALL [121], REYNOLDS [100], LEDLEY and WILSON [85], INGERMANN [71], CHEATHAM and SATTLEY [31], BASTIAN [13], BARNETT and FUTRELLE [11], WARSHALL and SHAPIRO [122], CHEATHAM [30], SCHORRE [112] followed within a short time) but its practical use was restricted to places that had abundant computer time. KUNO and OETTINGER, in natural language translation, also favoured a 'multi-path syntactic analysis'[83]; a similar idea was used in COGENT by REYNOLDS in 1965 [62].

Several compilers in practical use used top-down parsing with recursive descent: The META compilers of SCHORRE [62], the Burroughs extended ALGOL compiler and the SHARE 7090 ALGOL compiler. The elegance and convenience for the compiler writer was paid for in compile time by the user. While GLENNIE had estimated that 1000 instructions were to be executed in order to produce one machine instruction, PAUL in his ALGOL 60 compiler for the Z22 did it with about 50.

---

[1] A subroutine return stack was proposed in 1952 by W. L. van der Poel [98].

The situation changed, when one learned to do syntax-directed (top-down) parsing without backup. In 1965, FOSTER found the general rule for mechanically transforming a grammar into what was called later LL(1). The publication was delayed until 1968 [52]. Independently, KNUTH, in the Copenhagen summer school 1967, gave a definition of LL(1) - the publication was delayed until 1971. So, techniques for achieving this systematically were first published in 1968 by UNGER [62], while LEWIS and STEARNS [62] in 1968, ROSENKRANTZ and STEARNS [62] in 1969 treated the class LL(k) of grammars that can be parsed top-down without backup by examining at each step all the symbols processed so far and k symbols more to the right. Clearly, LL(k), like LR(k), is an efficient method, and the discrepancy between bottom-up syntax-controlled and top-down syntax-directed methods is no longer of more than historical importance, it has been replaced by competition. Again, we have reached the actualities of this field.

7. Concluding remarks

The development in the Soviet Union and its neighbours has progressed quite independently from the Western side, and vice versa. An attempt to arrive at a complete and comparative coverage would be very desirable, but I am not even able to give a first approximation in the framework of this course. I would, however, venture the hope that in co-operation with Soviet colleagues such an aim can be approached. Quite generally, I would appreciate to receive additional material, from whatever part of the world, on the history of compilers, and may combine with the thanks to potential contributors the thanks to my colleagues and friends, in particular the lecturers of this course, for their help.

References

[1] The A-2 Compiler System: Automatic Programming. Computers and Automation 4, 25-31 (Sept. 1955) and 4, 15-23 (Oct. 1955)

[2] Adams, Ch. W., Laning, J. H. jr.: The MIT systems of automatic coding: Comprehensive, Summer Session, and Algebraic. Symposium Automatic Programming Digital Computers, Office of Naval Research, PB 111 6o7. May 13-14, 1954, p. 3o-33

[3] Adams, E. S., Schlesinger, S. I.: Simple automatic coding systems. Comm. ACM 1:7, 5-9 (1958)

[4] Angstl, H.: Seminar über Logistik. Prof. W. Britzelmayr, Universität München, 195o

[5] Arden, B. W., Graham, R. M.: On GAT and the construction of translators. Comm. ACM 2:7, 24-26 (1959)

[6] Arden, B. W., Galler, B. A., Graham, R. M.: The internal organization of the MAD translator. Comm. ACM 4, 28-31 (1961)

[7] Automatic Programming: The IT translator. In: E. M. Grabbe et al. (eds.): Handbook of Automation, Computation, and Control 2, 2·2oo-2·228. New York: John Wiley and Sons 1959

[8] Backus, J. W., Herrick, H.: IBM 701 speedcoding and other automatic programming systems. Symposium Automatic Programming Digital Computers, Office of Naval Research, PB 111 6o7. May 13-14, 1954, p. 1o6-113

[9] Backus, J. W.: Automatic Programming: Properties and performance of FORTRAN Systems I and II. Symposium Mechanisation of Thought Processes, National Physical Lab., Teddington, November 24-27, 1958, p. 231-255

[10] Backus, J. W. et al.: The FORTRAN automatic coding system. Proc. AFIPS 1957 WJCC 11, p. 188-198

[11] Barnett, M. P., Futrelle, R. P.: Syntactic analysis by digital computer. Comm. ACM 5, 515-526 (1962)

[12] Barton, R. S.: Another (nameless) compiler for the Burroughs 220. Comm. ACM 4, A 11 (1961)

[13] Bastian, A. L. jr.: A phrase-structure language translator. Air Force Cambridge Res. Labs., Hanscom Field (Mass.), Rep. No. AFCRL-69-549, August 1962

[14] Bauer, F. L.: The formula-controlled logical computer "Stanislaus". Mathematics of Computation (MTAC) 14, 64-67 (1960)

[15] Bauer, F. L., Bottenbruch, H., Rutishauser, H., Samelson, K.: Proposal for a universal language for the description of computing processes. Zürich, Mainz, München, Darmstadt (ZMMD-Projekt), April 1958

[16] Bauer, F. L., Samelson, K.: Verfahren zur automatischen Verarbeitung von kodierten Daten und Rechenmaschine zur Ausübung des Verfahrens. Deutsche Patentauslegeschrift 1094019. Anm.: 30. März 1957; Bek.: 1. Dez. 1960

[17] Bauer, F. L., Samelson, K.: The cellar principle for formula translation. Proc. ICIP Paris 1959, p. 154

[18] Bauer, F. L., Samelson, K.: Maschinelle Verarbeitung von Programmsprachen. In: Hoffmann, W. (Hrsg.): Digitale Informationswandler. Braunschweig: Vieweg 1962

[19] Böhm, C.: Calculatrices digitales. Du dêchiffrage de formules logico-mathématiques par la machine même dans la conception du programme (Dissertation, Zürich 1952). Annali Mathematica pura applicata, Ser. 4, 37, 5-47 (1954)

[20] Bottenbruch, H.: Einige Überlegungen zur Übersetzung einer algorithmischen Sprache in Maschinenprogramme. Manuskript, Institut für Praktische Mathematik (IPM) der Techn. Hochschule Darmstadt, 1957

[21] Brooker, R. A.: Some technical features of the Manchester Mercury Autocode Programme. Symposium Mechanisation of Thought Processes, National Physical Lab., Teddington, November 24-27, 1958, p. 2o1-229

[22] Brooker, R. A., Morris, D.: An assembly program for a phrase structure language. Computer J. 3, 168-174 (1960)

[23] Brooker, R. A., Morris, D.: Some proposals for the realization of a certain assembly program. Computer J. 3, 22o-231 (1961)

[24] Brooker, R. A., Morris, D.: A description of Mercury Autocode in terms of a phrase structure language. In: Goodman, R. (ed.): Second Annual Review of Automatic Programming. New York: Pergamon 1961

[25] Brooker, R. A., Morris, D.: A general translation program for phrase-structure languages. J. ACM 9, 1-1o (1962)

[26] Brooker, R. A. Morris, D.: A compiler for a self-defining phrase structure language. University of Manchester, England (undated)

[27] Brown, J. H. Carr III, J. W.: Automatic programming and its development on the MIDAC. Symposium Automatic Programming Digital Computers, Office of Naval Research, PB 111 6o7. May 13-14, 1954, p. 84-97

[28] Burks, A. W., Warren, D. W., Wright, J. B.: An analysis of a logical machine using paranthesis-free notation. Math. Tables and Other Aids to Computation (MTAC) 8:46, 53-57 (1954)

[29] Burroughs B5000. In: Data Processing Encyclopedia. Detroit 1961, p. 5o-55

[30] Cheatham, T. E.: The TGS-II translator-generator system. Proc. IFIP Congr. 65, New York. Washington (D.C.): Spartan 1965

[31] Cheatham, T. E., Sattley, K.: Syntax-directed compiling. Proc. AFIPS 1964 SJCC 25. Washington (D.C.): Spartan 1964, p. 31-57

[32] Chomsky, N.: Syntactic structures. Den Haag: Mouton 1957

[33] Chomsky, N.: On certain formal properties of grammars. Information and Control 2, 137-167 (1959). Addendum: A note on phrase structure grammars. Information and Control 2, 393-395 (1959)

[34] Chomsky, N.: Formal properties of grammars. In: Luce, R. D., Bush, R., Galanter, E. (eds.): Handbook of Mathematical Psychology, Vol. 2. New York: John Wiley and Sons 1963, p. 323-418

[35] Chomsky, N., Schützenberger, M. P.: The algebraic theory of context-free languages. In: Braffort, P., Hirschberg, D. (eds.): Computer Programming and Formal Systems. Amsterdam: North-Holland 1963, p. 118-161

[36] Church, A.: The calculi of lambda-conversion. In: Annals of Mathematics Studies No. 6. Princeton (N.J.): Princeton University Press 1941

[37] Conway, M. E.: Design of a seperable transition-diagram compiler. Comm. ACM 6, 396-4o8 (1963)

[38] Dijkstra, E. W.: Making a translator for ALGOL 6O. Annual Review in Automatic Programming 3, 347-356 (1963)

[39] Eickel, J.: Generation of parsing algorithms for Chomsky 2 - type languages. Mathematisches Institut der Technischen Universität München, Bericht Nr. 6401, 1964

[40] Eickel, J., Paul, M., Bauer, F. L., Samelson, K.: A syntax-controlled generator of formal language processors. Comm. ACM 6, 451-455 (1963)

[41] Ershov, A. P.: On programming of arithmetic operations (Russ.). Doklady AN SSSR 118:3, 427-43o (1958). (Engl. Transl. in: Comm. ACM 1:8, 3-6 (1958))

[42] Ershov, A. P.: The work of the computing centre of the Academy of Sciences of the USSR in the field of automatic programming. Symposium Mechanisation of Thought Processes. National Physical Lab., Teddington, November 24-27, 1958, p. 257-278

[43] Ershov, A. P.: Programming Programme for the BESM Computer (Russ.). Moskau: Verlag der Akademie der Wissenschaften der UdSSR 1958 und London: Pergamon Press 1960

[44] Fischer, P. C.: A proposal for a term project for applied mathematics 205. Manuscript, 1959

[45] Flow-Matic Programming System. Remington-Rand Univac Div. of Sperry Rand Corp. New York 1958

[46] Floyd, R. W.: An algorithm defining ALGOL statement analysis with validity checking. Comm. ACM 3, 418-419 (1960)

[47] Floyd, R. W.: An algorithm for coding efficient arithmetic operations. Comm. ACM 4, 42-51 (1961)

[48] Floyd, R. W.: Syntactic analysis and operator precedence. J. ACM 1o, 316-333 (1963)

[49] Floyd, R. W.:Bounded context syntactic analysis. Comm. ACM 7, 62-67 (1964)

[50] Floyd, R. W.: Syntactic analysis and operator precedence. In: Pollack, B.W. (ed.): Compiler Techniques. Princeton (N.J.): Auerbach Publishers 1972

[51] Floyd, R. W.: The syntax of programming languages - a survey. In: Pollack, B. W. (ed.): Compiler Techniques. Princeton (N.J.): Auerbach Publishers 1972

[52] Foster, J. M.: A syntax improving device. Computer J. 11, 31-34 (1968)

[53] GAMM Fachausschuß Programmieren (Hrsg.): Vorschläge für eine algorithmische Schreibweise zur Formelübersetzung. Zürich, Mainz, München, Darmstadt, Oktober 1957

[54] Glennie, A.: On the syntax machine and the construction of a universal compiler. Carnegie-Mellon University, Techn. Report No. 2 (AD-24o512), July 1960

[55] Goldfinger, R.: New York University Compiler System. Symposium Automatic Programming Digital Computers, Office of Naval Research, PB 111 6o7. May 13-14, 1954, p. 3o-33

[56] Gorn, S.: Planning universal semi-automatic coding. Symposium Automatic Programming Digital Computers, Office of Naval Research, PB 111 6o7. May 13-14, 1954, p. 74-83

[57] Gorn, S.: On the logical design of formal mixed languages. Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, 1959

[58] Graham, R. M.: Bounded context translation. Proc. AFIPS 1964 SJCC 25. Baltimore (Md.): Spartan 1964, p. 17-29

[59] Grau, A. A.: Recursive processes and ALGOL translation. Comm. ACM 4, 1o-15 (1961)

[60] Grau, A. A.: The structure of an ALGOL translator. Oak Ridge Nat. Lab., Oak Ridge (Tenn.), Report No. ORNL-3o54, February 9, 1961

[61] Greibach, S. A.: Inverses of phrase structure generators. Harvard University, Cambridge (Mass.), Ph. D. dissertation, June 1963

[62] Gries. D.: Compiler construction for digital computers. New York: John Wiley and Sons 1971

[62a] Gries, D.: Use of transition matrices in compiling. Comm. ACM 11, 26-34 (1968)

[63] Gries, D. et al.: Some techniques used in the ALCOR-Illinois 7090. Comm. ACM 8, 496-5oo (1965)

[64] Haley, A. C. D.: The KDF 9 computer system. Proc. AFIPS 1962 FJCC 22. Washington (D.C.): Spartan 1962, p. 1o8-12o

[65] Halstead, M. H.: Machine independent computer programming. Washington (D.C.): Spartan 1962, p. 37 ff

[66] Hopper, G. M.: Automatic programming, definitions. Symposium Automatic Programming Digital Computers, Office of Naval Research, PB 111 6o7. May 13-14, 1954, p. 1-5

[67] Hopper, G. M.: First glossary of programming terminology. Report to the Association for Computing Machinery (ACM), June 1954

[68] Huskey, H. D.: Compiling techniques for algebraic expressions. Computer J. 4, 1o-19 (1961)

[69] Huskey, H. D., Wattenburg, W. H.: A basic compiler for arithmetic expressions. Comm. ACM 4, 3-9 (1961)

[70] Ingerman, P. Z.: A new algorithm for algebraic translation. Reprints of papers presented at the 14th National Meeting of the Association for Computing Machinery, 1959, p. 22·1-22·2

[71] Ingerman, P. Z.: A syntax oriented compiler... . Moore School of Electr. Engineering. University of Pennsylvania, Philadelphia, April 1963

[72] Irons, E. T.: A syntax directed compiler for ALGOL 60. Comm. ACM 4, 51-55 (1961)

[73] Irons, E. T.: The structure and use of the syntax-directed compiler. In: Annual Review in Automatic Programming 3, 2o7-227 (1963)

[74] Irons, E. T.: 'Structural connections' in formal languages. Comm. ACM 7, 67-72 (1964)

[75] Janov, Y. J.: On the equivalence and transformation of program schemes (Russ.). Doklady AN SSSR 113:1, 39-42 (1957). Engl. transl. in: Comm. ACM 1:1o, 8-12 (1958)

[76] Janov, Y. J.: On matrix program schemes (Russ.). Doklady AN SSSR 113:2, 283-286 (1957). Engl. transl. in: Comm. ACM 1:12, 3-6 (1958)

[77] Kanner, J.: An algebraic translator. Comm. ACM 2:1o, 19-22 (1959)

[78] Kantorovich, L. V.: On a mathematical symbolism convenient for performing machine calculations (Russ.). Doklady AN SSSR 113:4, 738-741 (1957)

[79] Kleene, S. C.: Proof by cases in formal logic. Annals of Mathematics 35, 529-544 (1934)

[80] Knuth, D. E.: Runcible: Algebraic translation on a limited computer. Comm. ACM 2, 18-21 (1959)

[81] Knuth, D. E.: A history of writing compilers. Computers and Automation 11, 8-14 (1962)

[82] Kolloquium über Sprachen und Algorithmen. Berlin, 8. - 11. Juni 1960. Math. Logik 7, 299-3o8, 1961

[83] Kuno, S., Oettinger, A. G.: Multiple-path syntactic analyzer. Proc. IFIP Congr. 62, Munich. Amsterdam: North-Holland 1962, p. 3o6-312

[84] Laning, J. H., Zierler, N.: A program for translation of mathematical equations for Whirlwind I. Massachusetts Institute of Technology, Cambridge (Mass.), Engineering Memorandum E-364, January 1954

[85] Ledley, R. S., Wilson, J. B.: Autmatic-programming-language translation through syntactical analysis. Comm. ACM 5, 145-155 (1962)

[86] Ljapunov, A. A.: On logical schemes of programming (Russ.) Problemi Kibernetiki 1, 46-74 (1958). Deutsche Übers. in: Ljapunov, A. A. (Hrsg.): Probleme der Kybernetik, Bd.1. Berlin: Akademie Verlag 1962, p. 53-86

[87] Lucas, P.: The structure of formula-translators. Mailüfterl, Vianna, Austria. ALGOL Bulletin Suppl. No. 16, September 1961 und Elektronische Rechenanlagen 3, 159-166 (1961)

[88] Lukasiewicz, J.: O znaczeniu i potrzebach logiki matematycznej (On the importance and needs of mathematical logic). Nauka Polska 1o, 6o4-62o (1929)

[89] Lukasiewicz, J.: Elementy logiki matematycznej (Elements of mathematical logic). Lecture Notes, 2nd edition (1929) - Warszawa, 1958, PWN, p. 4o

[89a] Lukasiewicz, J.: Elements of mathematical logic. Oxford: Pergamon Press 1963

[90] Lukasiewicz, J., Tarski, A.: Untersuchungen über den Aussagenkalkül. C.R.Soc. Sci. Lett. Varsovie, Ch. III, 23, 31 (1930)

[91] Menger, K.: Eine elementare Bemerkung über die Struktur logischer Formeln. In: Menger, K. (Hrsg.): Ergebnisse eines mathematischen Kolloquiums 3, (1932). Leipzig und Wien: Deutige 1935, p. 22-23

[92] Namur, P.: Entwurf eines Hochgeschwindigkeits-Rechenautomaten mit Leuchtpunktabtastung als Grundelement. Technische Hochschule Darmstadt, Dissertation, November 1964

[93] Oettinger, A. G.: Automatic syntactic analysis and the push-down store. Proc. Symp. Appl. Math. 12, Providence (R.I.): Amer. Math. Soc. 1961, p. 1o4-129

[94] Paul, M.: Zur Struktur formaler Sprachen. Universität Mainz, Dissertation D77, 1962

[95] Paul, M.: A general processor for certain formal languages. Proc. Symp. Symbolic Languages in Data Processing, Rome, 1962. New York - London: Gordon and Breach 1962, p. 65-74

[96] Paul, M.: ALGOL 60 processors and a processor generator. Proc. IFIP Congr. 62, Munich. Amsterdam: North-Holland 1962, p. 493-497

[97] Perlis, A. J. et al.: Internal translator (IT), a compiler for the 650. Carnegie Institute of Technology, Computation Center, Pittsburgh 1956. Reproduced by Lincoln Lab. Div. 6, Document 6D-327

[98] van der Poel, W. L.: Dead programmes for a magnetic drum automatic computer. Applied Scientific Research (B) 3, 19o-198 (1953)

[99] Post, E. L.: Formal reduction of the general combinatorial decision problem. Amer. J. Math. 65, 197-215 (1943)

[100] Reynolds, J. C.: A compiler and generalized translator. Applied Math. Div., Argonne Natl. Lab., Argonne, Ill., 1962

[101] Rice, H. G.: The APS III compiler for the Datatron 204. Westinghouse Research Lab., Pittsburgh, Manuscript, 1957

[102] Rich, B.: APT common computer language. Manuscript, Appl. Phys. Lab. Johns Hopkins University, Baltimore (Md.) 1957 and Annual Review in Automatic Programming 2, 141-159 (1961)

[103] Rosenbloom, P. C.: The elements of mathematical logic. New York: Dover 1950

[104] Rutishauser, H.: Über automatische Rechenplananfertigung bei programmgesteuerten Rechenmaschinen. Z. angew. Math. Mech. 31, 255 (1951)

[105] Rutishauser, H.: Automatische Rechenplananfertigung bei programmgesteuerten Rechenmaschinen. Inst. f. Angew. Mathematik ETH Zürich, Mitteil. Nr. 3. Basel: Verlag Birkhäuser 1952

[106] Rutishauser, H.: Automatische Rechenplananfertigung bei programmgesteuerten Rechenmaschinen. Z. angew. Math. Mech. 32, 312-313 (1952)

[107] Samelson, K.: Probleme der Programmierungstechnik. Aktuelle Probleme der Rechentechnik. Ber. Internat. Mathematiker-Kolloquium, Dresden November 22-27, 1955. Berlin: VEB Deutscher Verlag der Wissenschaften 1957, p. 61-68

[108] Samelson, K.: Programming languages and their processors. Proc. IFIP Congr. 62, Munich. Amsterdam: North-Holland 1963, p. 487-492

[109] Samelson, K., Bauer, F. L.: Sequentielle Formelübersetzung. Elektron. Rechenanlagen 1, 176-182 (1959)

[110] Samelson, K., Bauer, F. L.: Sequential formula translation. Comm. ACM 3, 76-83 (1960)

[111] Samelson, K., Bauer, F. L.: The ALCOR project. Proc. Symp. Symbolic Languages in Data Processing, Rome, 1962. New York - London: Gordon and Breach 1962, p. 2o7-217

[112] Schorre, D. V.: A syntax oriented compiler writing language. Proc. 19th ACM Conf. 1964, Philadelphia, D1.3-1-D1.3-11

[113] Schröter, K.: Axiomatisierung der Frege'schen Aussagenkalküle. Forschungen zur Logik und zur Gundlegung der exakten Wissenschaften. Neue Serie Bd. 8, Leipzig 1943

[114] Schützenberger, M. P.: On context-free languages and pushdown automata. Information and Control 6, 246-264 (1963)

[115] Share Assembly Program (SAP). In: Grappe, E. M. et al. (eds.): Handbook of Automation, Computation and Control Vol. 2. New York: John Wiley and Sons 1959, p. 2·165-2·167

[116] Sheridan, P. B.: The arithmetic translator-compiler of the IBM FORTRAN automatic coding system. Comm. ACM 2:3, 9-21 (1959)

[117] Slutz, R. J.: Engineering experience with the SEAC. Proc.AFIPS 1961 EJCC 1. New York: Amer. Inst. Electr. Engineers 1951, p. 9o-93

[118] UNIVAC Generalized Programming. Remington Rand Univac Div. of Sperry Rand Corp., New York 1957

[119] UNIVAC Math-Matic Programming System. Remington Rand Univac Div. of Sperry Rand Corp., New York 1958

[120] Waldburger, H.: Gebrauchsanweisung für die ERMETH. Institut für Angew. Math. an der ETH Zürich, 1958

[121] Warshall, S.: A syntax-directed generator. Proc. AFIPS 1961 EJCC 2o. Baltimore (Md.): Spartan 1961, p. 295-3o5

[122] Warshall, S., Shapiro, R. M.: A general table-driven compiler. Proc. AFIPS 1964 SJCC 25. Washington (D.C.): Spartan 1964, p. 59-65

[123] Watt, J. B., Wattenburg, W. H.: A NELIAC-generated 7090-1401 compiler. Comm. ACM 5, 1o1-1o2 (1962)

[124] Wegstein, J. H.: From formulas to computer oriented language. Comm.ACM 2:3, 6-8 (1959)

[125] Wells, R.: Immediate constituents. Language 23, 81-117 (1947)

[126] Wilkes, M. V.: The use of a floating address system for orders in an automatic digital computer. Proc. Cambridge Philos. Soc. 49, 84-89 (1953)

[127] Wilkes, M. V. et al.: The preparation of programs for an electronic digital computer. Cambridge (Mass.): Addison-Wesley 1951 (1957)

[128] Wirth, N., Weber, H.: EULER: A generalization of ALGOL and its formal definition, Part I. Comm. ACM 9, 13-25 (1966)

[129] Zuse, K.: Über den allgemeinen Plankalkül als Mittel zur Formalisierung schematisch-kombinatorischer Aufgaben. Archiv der Math. 1:6, 441-449 (1948/49)

[130] Zuse, K.: Über den Plankalkül. Elektron. Rechenanlagen 1:2, 68-71 (1959)