DAVID C. RINE, Editor

# COMPUTER SCIENCE AND MULTIPLE-VALUED LOGIC

THEORY AND APPLICATIONS

NORTH-HOLLAND

# computer science and multiple-valued logic

theory and applications

# computer science and multiple-valued logic

## theory and applications

edited by

david c. rine

1977

north-holland publishing company–amsterdam · new york · oxford

Library of Congress Catalog Card Number: 75-40064
North-Holland ISBN: 0 7204 0406 1

Published by:

North-Holland Publishing Company–Amsterdam, New York, Oxford

Sole distributors for the U.S.A. and Canada:

Elsevier/North-Holland Inc.
52 Vanderbilt Avenue
New York, NY 10017

**Library of Congress Cataloging in Publication Data**

Main entry under title:

Computer science and multiple valued logic.

1. Switching theory. 2. Logic circuits. 3. Threshold logic. 4. Many-valued logic. I. Rine, David C.
QA268.5.C65 511'.3 75-40064
ISBN 0-444-11056-6 (American Elsevier)

Printed in England

# contents

# list of contributors

George ABRAHAM, Naval Research Laboratory, Washington, DC, U.S.A.
C. Michael ALLEN, Dept. Electrical Engineering, University of North Carolina at Charlotte, Charlotte, NC 28213, U.S.A.
Robert C. BRADDOCK, ITT-Gilfillan Incorporated, 7821 Orion Avenue, Van Nuys, CA 91409, U.S.A.
Melvin A. BREUER, Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90007, U.S.A.
Peter T. CHEUNG, Packard Instrument Co., Downers Grove, IL, U.S.A.
Edgar DUCASSE, Dept. of Computer and Information Science, Brooklyn College, C.U.N.Y., Brooklyn, New York 11236, U.S.A.
George EPSTEIN, Indiana University, Bloomington, IN 47401, U.S.A.
Gideon FRIEDER, Scientific Center, IBM Israel Ltd., Technion City, Haifa, Israel.
Donald D. GIVONE, Dept. of Electrical Engineering, Parker Engineering Building, State University of New York at Buffalo, Buffalo, NY 14214, U.S.A.
V. C. HAMACHER, Depts. of Electrical Engineering & Computer Science, University of Toronto, Canada.
A. HORN, University of Southern California, Los Angeles, CA 90007, U.S.A.
Samuel C. LEE, Dept. Electrical Engineering, University of Oklahoma, Norman, OK 73069, U.S.A.
Marvin E. LIEBLER, Westinghouse, Buffalo, NY, U.S.A.
Ryszard S. MICHALSKI, Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, U.S.A.
Claudio MORAGA, Dept. of Informatics, University of Dortmund, 46 Dortmund 50, F.R.G.
David C. RINE, Computer Science, University of Texas, San Antonio, TX, U.S.A.
Robert P. ROESSER, Dept. of Electrical Engineering, Wayne State University, Detroit, MI 48202, U.S.A.
Ivo G. ROSENBERG, University of Montreal, C.P.6128 Montreal 101, Quebec, Canada.
K. C. SMITH, Depts. of Electrical Engineering & Computer Science, University of Toronto, Canada.
William R. SMITH, Dept. Electrical Engineering, University of Bridgeport, CT 06602, U.S.A.
Stephen Y. H. SU, Dept. of Electrical Engineering, Utah State University, Logan, UT 84322, U.S.A.
Stanislaw J. SURMA, Dept. of Logic, Jegiellonian University of Cracow, ul. Grodska 52, 31-044 Krakow, Poland.

T. TRACZYK, Koszkoma 75.m.27, 00-662 Warsawa, Poland.
Z. G. VRANESIC, Depts. of Electrical Engineering & Computer Science, University of Toronto, Canada.
Anthony S. WOJCIK, Dept. of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, U.S.A.
Hideki YAMANAKA, ITT-Gilfillan Incorporated, 7821 Orion Avenue, Van Nuys, CA 91409, U.S.A.

# preface

This preface briefly acquaints the reader to the book and its preparation, whereas the introductory chapter introduces the reader to the area of multiple-valued logic and how and why it has become related to computer science.

This book is aimed at those computer engineers, computer scientists, applied mathematicians and physicists who are not experts in the area of multiple-valued logic as the discipline pertains to computer engineering and computer science. Thus its primary goal is to convince these people that multiple-valued logic is indeed both intellectually challenging and practically applicable.

It was decided in conversations with a number of engineers, computer scientists and potentially interested people including applied mathematicians that several outstanding scientists, luminaries capable of explaining our basic concepts in an interesting way, should be asked to write an updated 'handbook' of ideas contained in the best papers. With the exception of a handful of good classical works by Epstein this expectation has been carried out.

Many new developments have recently taken place in computer science and multiple-valued logic. For example, fault analysis and software considerations are new and extremely important fields. In fact, there is a good chance that the impact of multiple-valued software considerations will be at least as important as hardware considerations on the use and design of fourth and fifth generation computing and data processing systems.

The organization of this documentary is top-down in its design in that the book is divided into five parts.

I Algebraic theory.
II Logic design and switching theory.
III Threshold logic design.
IV Physical components and implementations.
V Applications.

Each part has been broken down into several monographs or chapters, each chapter having been designed by one or two scientists. However the flow and sense from chapter to chapter is continuous. The flow between parts has been made continuous by means of introductory remarks and conclusions for each section.

The Introduction (Chapter 1) provides the new reader with interesting foundations, history and motivation for considering multiple-valued logic design and applications from both intellectual and practical points of view. No particular advanced technical background is assumed of the reader; likewise, the first portion of each monograph is written in such a way that important concepts or reasons for considering a given area can be grasped by each new reader. Since the book relates to computer science and engineering, many interesting considerations from ancient hisotry have not been mentioned due to lack of space. However, a conspectus on the history can be found in Rescher[41]. A short but precise historical summary of multiple-valued logic as it is related to computer science can be found in a paper by Epstein, Frieder, and Rine[14]. Papers which could date the beginning of multiple-valued logic and modern digital computer systems are those of Grosth[20] and Metze[31].

Part I on algebraic theory gives the applied mathematical foundations for much of multiple-valued logic as related to computer science. The reader who is either partially familiar with these interesting relations or who is more immediately interested in only logic design and switching theory may wish to turn directly to Part II. Modern axioms and properties of Post algebras are nicely summarized by Traczyk[54], a Polish scientist; while Surma[53] presents a general algorithm for axiomatizing every finite logic. Relationships between Post and Boolean algebras have been studied by Wojcik and Metze[59,60]. Completeness properties of multiple-valued logic and switching theory have been documented by Rosenberg[45] and Patt[35]. Abstractions and extensions of Post algebras have recently been introduced by Epstein[11,12,14], one interesting concept being that of a P-algebra; Epstein has, also, given an important equational axiomatization for the disjoint system of Post algebras[13]. Interestingly enough, Nutter, Swartwout, and Rine[37] have shown that many multiple-valued logic algebras are isomorphically equivalent to Post algebras.

Part II of this book covers logic design and switching theory. This section gives important models for computer subsystems and evaluates multiple-valued logic design criteria directly related to cost. Initially, applications of multiple-valued logic systems to circuits were studied by Metze[31] and Yoeli and Rosenfeld[61]. Since various algorithms for minimizing Boolean logic systems, for example the Quine–McCluskey technique, have been of interest, it must be noted that much research has also been directed toward the "minimizing" of multiple-valued logic systems. While this general problem is still open and is far more difficult for the multiple-valued logic cases and presents itself as a "can-of-worms", computer minimization techniques have seen partial success through the ongoing work of Allen and Givone[3], Su and Cheung[52], W. R. Smith[50], and Ostapko, Cain, and Hong[38]. Hazards and cost analyses of hazard-free logic systems have always been an important topic of investigation in computer engineering. DuCasse and Metze[10] and others[24] are studying this problem. Also, fault detection using multiple-valued test machines has been studied by Sheppard and Vranesic[40], while a multilevel balanced code with redundant digits was investigated by Asabe and Tezuka[4]. Treatment of error signals was reported by Breuer and Epstein[7]. Finally, multiple-valued logic symmetric functions and their properties were well-noted by S. C. and E. T. Lee[28] and also by Cheung and Su[8].

Threshold Logic Design, covered in Part III, is an extremely important area of computer engineering and logic design in its own right. Multiple-valued variable-threshold logic is being investigated by Aibara, Takamatsu, and Murakami, while applications of multiple-valued threshold logic to digital computer arithmetic were discovered by the Finnish computer engineers, Koukkunen and Ojala[26]. On the other hand, Druzeta and Sedra[9] have made good progress by using multiple-threshold circuits in the design of multi-state storage elements. Finally, it must be mentioned that Moraga has achieved excellent results in the study of non-linear[33], minimal[37], and periodic[34] ternary threshold logic systems. Another good summary paper is that of Vranesic and Hamacher on threshold logic in fast ternary multipliers[50]. Along the same lines is the work on multifunction threshold gates by Hampel[23].

It goes without saying from a commercial-industrial point of view that Part IV, physical components and implementations, is very important for computer engineering. Historically, despite relatively little effort being put into the design of electronic circuits for multiple-valued logic, there has been a very encouraging evolutionary trend towards many more practical realizations. In the past several years implementations have suffered from insufficient utilization of integrated circuit techniques. In the past, for example, stray capacitance in complex discrete circuits has led to slow operating characteristics. However, additional components and the miniaturization possible with integrated circuit technology is now beginning to lead to a much improved speed performance. Once the speed problems are overcome, the natural and very valuable advantages of multiple-valued logic techniques in the reduction of wiring complexity, a big cost factor in building present day computers, will assert themselves. The computer engineering group at the University of Toronto has made elegant strides in constructing and implementing multiple-valued logic physical components; I mention the work of Vranesic and Smith[56,57] and Sebastian and Vranesic[47]. One must also mention the independent work on multiple-valued integrated circuits by Abraham[1] at the Naval Research Laboratory. A new concept for ternary logic elements has been reported by Etiemble and Israel[16], while Mouftah and Jordan[35] have reported success on integrated circuits for ternary logic. Shiva and Nagle[49] at Auburn University have advanced technology on multiple-valued memory elements, while Strasilla[57] in Switzerland has made excellent headway on a multiple-valued logic memory system using capacitor storage. Also in the area of physical components Irving and Nagle[25] have invented an approach to multiple-valued sequential logic. Historically, Braddock, Epstein, and Yamanaka[6] hold an original patent on a multiple-valued logic design and application in binary computers. The work of Metze[31] also deserves mention in this section.

Part V, applications, is one of the most important because it covers the ways in which people use multiple-valued logic systems in problem solving. This section describes both hardware and software applications in the very new area of multiple-valued logic and programming. This area will see a major expansion within the next few years. Microprogramming has made it possible sensibly to study the emulation of multiple-valued logic computer systems; it is often better to build an emulator than an actual prototype of the system. It has been said that "emulation is the bridge between hardware and software". Frieder, Fong, Chao, and Luk[18,19] have used emulation to study a balanced ternary computer, particularly the arithmetic unit. On the other hand, Halpern and Yoeli[21] and Vranesic and Hamacher[22,55] consider the actual hardware prototype of the arithmetic unit in their studies. Another good German reference is [5]. Of importance to testing in computer engineering, Rowe[46] has researched the generation by multiple-valued logic of pseudorandom noise, while Lam and Vranesic[27] reported the multiple-valued logic generation of pseudorandom numbers, potentially better than binary generation. Leibler and Roesser[29] describe multiple-real-valued Walsh functions, while McDonald and Singh[30] have extensively researched the extensions of the Weiner-Smith algorithm for multiple-valued logic synchronous sequential circuit design. Asabe and Tezuka[4] reported on the multilevel balanced code with redundant digits. Returning to software applications, Rine[42,43] has mentioned applications of multiple-valued logic to programming languages and also to statistical decision theory[44]. Of great interest to computer science, diagnostic pattern recognition and multiple-valued logic systems has been the variable-valued logic systems VL1 Project under the direction of R. S. Michalski[32]. Also of interest has been the recent work of Rasiowa[40] on a logical structure of mix-valued programs.

## References

[1] G. Abraham, Variable radix multistable integrated circuits: Impact of electron physics on multiple-valued logic design, 1974 *Intern. Symp. on Multiple-Valued Logic*, and *Computer*, September 1974.

[2] T. Aibara, Y. Takamatsu, and K. Murakami, Multiple-valued variable-threshold logic and its application to the realization of the set of boolean functions, *Proc.* 1973 *Intern. Symp. on Multiple-Valued Logic*, Toronto, Canada.

[3] C. M. Allen and D. D. Givone, A minimization technique for multiple-valued logic systems, *IEEE Trans. Computers*, **C-17** (1968) 182–184.

[4] T. Asabe and Y. Tezuka, Multilevel balanced code with redundant digits, *Proc.* 1972 *Symp. Theory and Applications of Multiple-Valued Logic*, SUNY, Buffalo, N.Y.

[5] Arithmetisch Operationen Mit Binär Codierten Ternarzahlen, *AGEN*, **11** (1970) 55.

[6] R. Braddock, G. Epstein, and H. Yamanaka, Multiple-valued logic design and applications in binary computers, *Proc.* 1971 *Symp. on Theory and Applications of Multiple-Valued Logic*, SUNY, Buffalo, N.Y.

[7] M. A. Breuer and G. Epstein, The smallest many-valued logic for treatment of complemented and uncomplemented error signals, *Proc.* 1973 *Intern. Symp. on Multiple-Valued Logic*, Toronto, Canada.

[8] P. Cheung and S. Su, Algebraic properties of multi-valued symmetric switching functions, *Proc.* 1971 *Symp. on Theory and Applications of Multiple-Valued Logic*, SUNY, Buffalo, N.Y.

[9] A. Druzeta and A. S. Sedra, Multi-threshold circuits in the design of multi-state storage elements, *Proc.* 1973 *Intern. Symp. on Multiple-Valued Logic*, Toronto, Canada.

[10] E. G. DuCasse and G. A. Metze, Hazard-free realizations of boolean functions using post functions (Also, A cost analysis of hazard-free networks synthesized from monotone jump

functions), *Proc.* 1974 *Intern. Symp. on Multiple-Valued Logic*, West Virginia University, Morgantown, W.Va.

[11] G. Epstein, On multiple-valued signal processing with limiting, *Proc.* 1971 *Symp. on Theory and Applications of Multiple-Valued Logic*, SUNY, Buffalo, N.Y.

[12] G. Epstein, P-Algebras, an abstraction from Post algebras, Tech. Rep., U.C.L.A. and The Indiana University, Bloomington, (1973).

[13] G. Epstein, An equational axiomatization for the disjoint system of Post algebras, *IEEE Trans. Computers* **C-22** (1973) 422–423.

[14] G. Epstein, G. Frieder and D. Rine, The development of multiple-valued logic as related to computer science: A historical summary, *Computer*, September 1974. (And, a brief in *Proc.* 1974 *Intern. Symp. on Multiple-Valued Logic*, West Virginia University, Morgantown, W. Va.).

[15] G. Epstein and A. Horn, Chain based lattices, *Pac. J. Math* (1975).

[16] D. Etiemble and M. Israel, A new concept for ternary logic elements, *Proc.* 1974 *Intern. Symp. Multiple-Valued Logic*, West Virginia University, Morgantown, W. Va.

[17] Final report on design of multiple-valued logic systems, The Digital Systems Labs., SUNY at Buffalo, N.Y. (June, 1970).

[18] G. Frieder, A. Fong and C. Y. Chao, A balanced ternary computer, *Proc.* 1973 *Intern. Symp. on Multiple-Valued Logic*, Toronto, Canada.

[19] G. Frieder and C. Luk, Algorithms for binary coded balanced and ordinary ternary operations, SUNY Computer Science Report, Buffalo, N.Y. (1974).

[20] H. R. Grosth, Signed ternary arithmetic, Digital Computer Lab., MIT, Cambridge, Mass., Memorandum M-1496 (May 1954).

[21] I. Halpern and M. Yoeli, Ternary Arithmetic Unit, *Proc. IEEE*, **115** (1968) 1385–1388.

[22] V. C. Hamacher and Z. Vranesic, Multivalued versus binary high speed multipliers, *Proc.* 1971 *Symp. on Theory and Application of Multiple-Valued Logic*, SUNY, Buffalo, N.Y.

[23] D. Hampel, Multifunction Threshold Gates, *IEEE Trans. Computers*, **C-22** (1973) 197–203.

[24] Hazard Detection in Combinational and Sequential Switching Circuits, *IBM J. Res. Develop.*, **9** (1965) 90–99.

[25] T. A. Irving and H. T. Nagle, An approach to multi-valued sequential logic, *Proc.* 1973 *Intern. Symp. on Multiple-Valued Logic*, Toronto, Canada.

[26] H, Koukkunen and L. Ojala, Some applications of many-valued threshold logic in digital arithmetic, *Proc.* 1972 *Symp. on Theory and Applications of Multiple-Valued Logic*, SUNY, Buffalo, N.Y.

[27] C. L. Lam and Z. G. Vranesic, Multiple-valued pseudorandom number generators, *Proc.* 1974 *Intern. Symp. on Multiple-Valued Logic*, West Virginia University, Morgantown, W. VA.

[28] S. C. Lee and E. T. Lee, On Multivalued Symmetric Functions, *IEEE Trans. Computers*, March 1972.

[29] M. Leibler and R. Roesser, Multiple-real-valued Walsh functions, *Proc.* 1971 *Symp. on Theory and Applications of Multiple-Valued Logic*, SUNY, Buffalo, N.Y.

[30] J. F. McDonald and I. Singh, Extensions of the Weiner–Smith algorithm for $m$-ary synchronous sequential circuit design, *Proc.* 1974 *Intern. Symp. on Multiple-Valued Logic*, West Virginia University, Morgantown, W. Va.

[31] G. Metze, An Application of Multivalued Logic Systems to Circuits, *Proc. Symp. on Circuit Analysis*, University of Illinois, (1955) pp. 11-1–11-14.

[32] R. S. Michalski, Variable-valued logic systems-VL1, *Proc.* 1974 *Intern. Symp. on Multiple-Valued Logic*, West Virginia University, Morgantown, W. Va.

[33] C. Moraga, Non-linear ternary threshold logic, *Proc.* 1972 *Symp. on Theory and Applications of Multiple-Valued Logic*, SUNY, Buffalo, N.Y.

[34] C. Moraga and J. Gutierret, Multithreshold periodic ternary threshold logic, *Proc.* 1974 *Intern. Symp. on Multiple-Valued Logic*, West Virginia University, Morgantown, W. Va.

[35] H. T. Mouftah and I. B. Jordan, Integrated circuits for ternary logic, *Proc.* 1974 *Intern. Symp. on Multiple-Valued Logic*, West Virginia University, Morgantown, W. Va.

[36] J. Nazarala and C. Moraga, Minimal realization of ternary threshold functions, *Proc.* 1974 *Intern. Symp. on Multiple-Valued Logic*, West Virginia University, Morgantown, W. Va.
[37] R. Nutter, R. Swartwout and D. Rine, Equivalence and transformations for Post multivalued algebras, *IEEE Trans. Computers* **C-23** (1974) 294–300.
[38] D. L. Ostapko, R. G. Cain and S. J. Hong, A practical approach to two-level minimization of multivalued logic, *Proc.* 1974 *Intern. Symp. on Multiple-Valued Logic*, West Virginia University, Morgantown, W. Va.
[39] Y. N. Pratt, Toward a characterization of logically complete switching functions in $K$-valued logic, *Proc.* 1972 *Symp. on Theory and Applications of Multiple-Valued Logic*, SUNY, Buffalo, N.Y.
[40] H. Rasiowa, On a logical structure of mix-valued programs and the $\omega^+$-valued algorithmic logic, *Bull. Acad. Polon. Sci. Ser. Sci. Math. Astronom. Phys.* 21 (1973).
[41] N. Rescher, *A Historical Conspectus of Many-Valued Logics*, (McGraw-Hill, N.Y., 1969).
[42] D. C. Rine, A proposed multi-valued extension to ALGOL 68, *Kybernetes* **2** (1973) 107–111.
[43] D. C. Rine, Conditional and Post algebra expressions, *Discrete Math.* **10** (1974) 309–324.
[44] D. C. Rine, Basic concepts of multiple-discrete valued logic and decision theory, *Information and Control* **22** (1973) 320–352.
[45] I. G. Rosenberg, Completeness, closed classes and relations in multiple-valued logics, *Proc.* 1974 *Intern. Symp. on Multiple-Valued Logic*, West Virginia University, Morgantown, W. Va.
[46] I. Rowe, The generation of Gaussian-distributed pseudorandom noise sequences from multiple-valued logic, *Proc.* 1972 *Symp. on Theory and Applications of Multiple-Valued Logic*, SUNY, Buffalo, N.Y.
[47] P. Sebastian and Z. G. Vranesic, Ternary logic in arithmetic units, *Proc.* 1972 *Symp. on Theory and Applications of Multiple-Valued Logic*, SUNY, Buffalo, N.Y.
[48] D. A. Sheppard and Z. G. Vranesic, Fault detection of binary sequential machines using $R$-valued test machines, *IEEE Trans. Computers* **C-23** (1974) 352–358.
[49] S. G. Shiva and H. T. Nagle, On multiple-valued memory elements, *Proc.* 1974 *Intern. Symp. on Multiple-Valued Logic*, West Virginia University, Morgantown, W. Va.
[50] W. R. Smith, Minimization of multivalued functions, *Proc.* 1974 *Intern. Symp. on Multiple-Valued Logic*, West Virginia University, Morgantown, W. Va.
[51] U. J. Strasilla, A multiple-valued logic memory system using capacitor storage, *Proc.* 1974 *Intern. Symp. on Multiple-Valued Logic*, West Virginia University, Morgantown, W. Va.
[52] S. Y. H. Su and P. T. Cheung, Computer minimization of multivalued switching functions, *IEEE Trans. Computers* **C-21** (1972) 995–1003.
[53] S. J. Surma, An algorithm for axiomatizing every finite logic, *Proc.* 1974 *Intern. Symp. on Multiple-Valued Logic*, West Virginia University, Morgantown, W. Va. (and in *Reports on Mathematical Logic* no. 3 (1974) p. 57.).
[54] T. Traczyk, Axioms and some properties of Post algebras, *Colloq. Math.* **10** (1963) 193–209.
[55] Z. G. Vranesic and V. C. Hamacher, Ternary logic in Parallel multipliers, *Comput. J.* **15** (1972) 254–258.
[56] Z. G. Vranesic and K. C. Smith, An electronic implementation of multiple-valued logic, *Proc.* 1974 *Intern. Symp. on Multiple-Valued Logic*, West Virginia University, Morgantown, W. Va.
[57] Z. G. Vranesic and K. C. Smith, Engineering aspects of multi-valued logic systems, *Computer*, September 1974.
[58] Z. G. Vranesic and V. C. Hamacher, Threshold logic in fast ternary multipliers, *Proc.* 1975 *Intern. Symp. on Multiple-Valued Logic*, University of Indiana, Bloomington, Ind.
[59] A. Wojcik, The minimization of higher-order Boolean functions, *Proc.* 1972 *Symp. on Theory and Applications of Multiple-Valued Logic*, SUNY, Buffalo, N.Y.
[60] A. Wojcik and G. Metze, Some relationships between Post and Boolean algebras, *Proc.* 1971 *Symp. on Theory and Applications of Multiple-Valued Logic*, SUNY, Buffalo, New York.
[61] M. Yoeli and G. Rosenfeld, Logical design of ternary switching circuits, *IEEE Trans. Computers*, **EC-14** (1965) 19–29.

# introduction

chapter 1

# an introduction to multiple-valued logic

david c. rine

Computer scientists, computer engineers, applied mathematicians and physicists are familiar with options in which there are no middle choices between true and false. Statisticians are familiar with the soft logic of probability, and physicists are familiar with the fuzzy logic of uncertainty. The lack of such choices is inconvenient and even critical when trying to determine whether or not the status of a computer system is go, wait or no-go. Multiple-valued logic is concerned with these intermediate choices.

The development of multiple-valued logic as viewed by the authors mentioned within this book is contemporaneous with the computer age and is clearly related to computer science, where there is by default a well established connection of 2-valued logic to computer structures and programs. The emphasis herein is on the development of multiple-valued logic as interrelating with theoretical and applied work in areas of computer science such as switching theory, programming and architecture. Parts I and II of this book discuss switching theory and theories of algebra and logic, while Part V treats multiple-valued aspects of programming, Part III describes full scale implementations, emulated and actual and Part IV gives further developments concerning special implementations or applications as they arise in threshold logic.

The relation between Boolean algebra and computer science and computer engineering began in the early 1940's. This relation was established because of the construction of actual computers and an increasing number of models comparing logical, computing and living systems by engineers, physicists, applied mathematicians and scientists from all disciplines.

The development of multiple-valued logic in the twentieth century can be dated at around 1920. However, the first algebra of $n$-valued truth-functionally complete logic corresponding to the work of Post was not formulated until the early 1940's.

This relation between the development of multiple-valued logic in its modern era and computer science is not surprising in consideration of the above connection. Early investigators directed their work toward disjunctive normal forms since these forms correspond in an obvious way with Boolean logic truth-tables and some of the associated minimization techniques. It is not surprising therefore that a number of computer scientists in the eight years following 1955 described the normal form theorem for Post functions, and this led to the identification of certain operations occuring in other non-classical logics which are of interest in computer programming and computer engineering.

Some early research reported the change of state in relays and relay circuits. The use of 4-valued logic in order to analyze transients or faults was considered in the early 1950's. But since the intermediate values of transients need not be specified in certain transient or fault analyses, other approaches which do not specify intermediate values were also investigated.

In 1958 the first full scale 3-valued computer was completed at Moscow State University in the Soviet Union. While this encouraged work on subsystems such as arithmetic units, the lower order programming language devised for this Russian computer was so difficult to use that there was little insight into 3-valued logic by users.

Although higher order languages such as Fortran and Algol were conceived in the mid-1950's, these languages incorporate only 2-valued logic explicitly, and so disguise the use of concepts from multiple-valued logic when they occur within programs using these languages. The only early exception is the "arithmetic if" statement of early Fortran, which was a three-way branch, and the "computed go to" statement. Besides the familiar 2-way branch conceived in 2-valued terms, the multiple-valued $n$-way branch, where $n \geqslant 2$, may be simply devised in $n$-valued terms, but in 2-valued terms only through consecutive levels of nested 2-way branches. A large number of such levels of nested branches increases psychological complexity which may be faced by some programmers when writing programs. Further, the arguments in logico-arithmetic expressions used in branching are multiple-valued due to the multiple arithmetic constants of the computer, while the range of such expressions is 2-valued due to the two logical constants of the computer.

A higher order programming language with multiple-valued branching was IBM's Mathematical Formula Translating System of 1954, Fortran I. This language contained a control statement called the "computed go to" which allowed unrestricted branching to any of a number of multiple program statements depending on the value of a positive integer valued argument. The Darnsdtadt Meeting of 1955 resulted in an international programming language known as Algol 58, an algorithmic language. This language has the same kind of branching capability as the computed go to. Although the computed go to was an improvement over the arithmetic if in Fortran and other statements yielding only 3-valued or 2-valued branching, it very often led through poor software design to terrific program complexities and intolerable bother. Due to these complexities, flowcharting projects could probably achieve only average code-design rates of two to three statements per man-day, because of the time wasted on debugging and recoding.

There were a number of improvements in software design during the five years following 1962. Multiple-valued branching was provided in other languages such as Lisp. Also, the go to concept was replaced with the idea of the "case clause" by the Algol 68 international committee and in the language Algol 68. This was repeated later with the language Pascal. The consequence of using these multiple-valued branching control statements is often a significant reduction in program and flowchart complexity, because multiple-valued branching occurs in a linear or sequential fashion from top to bottom or bottom to top. The difference for the programmer or problem solver is the important difference between working through a maze by a systematic method as opposed to a hit-or-miss method.

The first emulation of a full scale ternary computer was completed in 1973 at the State University of New York at Buffalo, Amherst, New York.

Full scale implementations for the ternary case (see Part IV) take advantage of arithmetic operations of addition and multiplication which are admitted in Post algebras of order $p$, when $p$ is any fixed prime integer greater than 0.

There are many other applications apart from arithmetic units which use this fact, such as linear feedback shift registers.

Further developments are not limited to Post algebras of order $n$ and fixed-radix $n$. When the radix is allowed to vary without exceeding an integer $n$, then the resulting system is said to be of "mixed-radix $n$". Details on mixed-radix number systems are given in a well-known volume by Donald Knuth. The corresponding algebras for systems of mixed-radix $n$ are the $P_2$-lattices of order $n$ described in Part I of this book.

Generalizations of Post algebras beyond the case when $n$ is finite can be dated at around 1961.

The natural inclination to regard multiple-valued logic in a probabilistic setting has led to connections with physics and philosophy. Recent work on decision-making in a fuzzy environment is due to Richard Bellman and L. Zadeh.

By knowing that there are $2^v$ switching functions of $v$ binary variables which form a Boolean algebra it is easy to see that there are $n^v$ switching functions of $v$ $n$-ary variables where $n$ is a fixed integer greater than or equal to 2. One can easily show that these switching functions form a Post algebra of order $n$. Arrays of lattice diagrams exist for Post algebras, where the lattice diagram at row $a$ and column $b$ is a Post algebra of order $a + 1$ having $(i + 1)^j$ elements; it is not difficult to show that $2^j$ of these elements are complemented. Row $a = 1$ contains Boolean algebras which are the Post algebras of order 2. Column $b = 1$ contains Post chains which are the linearly ordered Post algebras. The reader can turn to Part I for further descriptions of these.

A major drawback to overcomplicated flowcharts developed by computer programmers is the difficulty with which they are checked, corrected or modified. It is this situation which suggests a structured design approach, where a structured flowchart or well designed program is built up to an adequate level of detail according to definite rules from a small, simple and sufficient set of elemental blocks or primitives.

Being concerned herein with multiple-valued logic as related to computer science, one notes that an important difference between the usage of hardware logic and software logic is that the hardware logic is fixed while it is actually desirable for the software logic to be variable, such that each appearance of a given software logic within a program reflects in a transparent way the semantics of the program. The semantics of the flowchart corresponding to the program should be recognizable by humans or automatically in a minimal number of steps.

There are certain desirable properties for programs to possess in order that their flowcharts may be mathematically or psychologically analyzed. First, control paths should be as simple as possible. Second, the lengths of subroutines in a program should be limited to a manageable size which in practice may be either a single displayable unit, one CRT screenful, one printer page or one page of standard flowchart logic.

Finally, recorded observations give some empirical evidence that the limit to the number of decision nodes which can be grasped by the programmer sitting in front of a page or screen is around 16 to 25, and the actual value of this limit will be determined by the number of such node flowchart symbols that can be comfortably diagrammed per page or screen. However, without any initial insights into algebraically structured or user compatible ways of handling the program branchings, the psychological limit to this number of nodes for the programmer would be closer to 7 or 9.

It is empirically known that proper software engineering designs may sometimes reduce average coding rates to one detected error per 10,000 lines of coding, or one detected error per man-year; although most case studies to date record a figure slightly larger. However, little is known as to how structured programming works at either the psychological or mathematical level. Applied mathematicians and computer scientists have observed that in order to design a large program successfully one must combine together many smaller parts of programs. The rules for these combinations or operations allow iterated combinations of simple elements into a large totality defining some "decision algebra". In general one concludes that (1) programming involves the discovery of practical algebraic principles, varying with applications, and allowing the iterated combination of elements into compound objects representing useful processes, and (2) programming involves the discovery of useful logical functions and entities possessing simple external descriptions for the selecting, routing, controling, branching and classifying of these processes.

Structure variations depend upon the program applications. An application dealing with a multiple-choice selection will use Post algebra structures of order 3 with symbols such as "negative, zero, positive", for example. On the other hand, event control will often use Boolean algebra structures, since events and their meanings often project or translate into go, no-go features. One is not only looking at the bottom nodes of the structures but also the way in which they are reached through the branches.

It has been noted that the lower order programming language of an early ternary computer was not user oriented, and programming was difficult, One notes that the higher order source code statement "IF(I) 9, 10, 11" becomes simpler when translated into a balanced ternary digit $-1$, 0, $+1$.

In general, the argument I may have any data value representable within the computer. Branching may, also, depend on "logico-arithmetic expressions" such as u.LE.v, which assume only the values .TRUE. or .FALSE., depending on whether multiple-valued $u$, $v$ satisfy $u \leqslant v$ or not.

In order to motivate the idea of logico-arithmetic expressions from current programming languages, let us consider the following assignment from the APL programming language:

$$\text{RESULT} \leftarrow ((0+1)\vee(5-5))\times(1\wedge 1)$$

The right hand side of the arrow is a logico-arithmetic expression since logic function "v" and arithmetic function "×" are seen to operate on either logic or arithmetic terms. However, APL expressions are only partially logico-arithmetic since the following assignment is not defined:

$$\text{RESULT} \leftarrow 5\vee 5$$

One should conclude that any kind of decision algebra for the programmer must reflect to some degree the programmer's actual tools and system framework. A feature of the previously mentioned operation .LE. is that it interfaces and corresponds well with similar operations in applications such as real-time process control, air traffic control and probability where logic and arithmetic constants blend together. This is discussed in Part I of this book.

Since the applications programmer often wants to extend the logic used within his or her programs from a basic "core-set" of logic primitives given in the programming language definition any selection of Post functors, for example, for this core-set must allow for the selection of still different multiple-valued functors. The ability to declare, protect and use a particular set of functors will allow the programmer to start from the logic types of a core-set and define new types from this core-set depending on his or her application. The core-set might contain logico-arithmetic constants such as .FALSE., arithmetic constants representable in the computer, .TRUE. and the logic operations .AND., .OR. and .LE. mentioned earlier. In this example the core-set is functionally complete, so that any other multiple-valued functor may be obtained.

The applications programmer may wish to generalize the logic used within his or her programs to infinite-valued logics. This is not at all impractical since some modern computers deal with numbers such as $2^{\pm 256}$ or $2^{\pm 60}$ which are so finitely quantized that the extension from a finite-valued logic to an infinite-valued logic seems a natural one. A fourth generation computer called Staran will be soon using a 2048-bit word. Progressing from the finite-valued logics to the infinite-valued logics is consistent with the fact that input signals themselves may be continuous signals such as voltages or frequencies prior to threshold detection or analog-to-digital conversion.

Programmers who can conceive a long series of statements invoked by a case statement or its equivalent in terms of a structured system such as multiple-valued logic, introduce this structure, or at least have this structure reflected, in their final program. There is a clear advantage in using the case statement as opposed to the computed go to or if-then-else statements.

Similar to the case statement of Algol 68 is the "select statement" of Lisp 1.5. It has the form

$$\mathrm{SELECT}(Q; (Q_1\,E_1); (Q_2\,E_2); \ldots; (Q_n\,E_n); E)$$

where the statement is evaluated by choosing the first $E_i$ whose $Q_i$ matches $Q$, otherwise $E$ is chosen if no such matching occurs. The connections in Lisp and search logic to Post Algebras and multiple-valued logic have been studied by D. Rine.

Prior to the writing of this book there had been no major programming languages based on multiple-valued rather than 2-valued logic. This may be not so much because of the reflection of computer hardware structures on computer programming languages as because of the newness of higher order programming languages coupled with the very old seemingly 2-valued characteristics of some natural languages. However, different number systems have been observed throughout the study of modern man. The Babylonians used a base sixty number system; and base three systems for computation were discussed in the eighteen hundreds. At any rate, two ternary computer implementations have already occurred, one actual and one emulated; at least two others are being considered by industry. The first two are described in Parts IV and V of this book.

It seems that the earliest document seriously proposing a full scale ternary computer was written by Grosch in 1952. Grosch advocated the implementation of a balanced ternary arithmetic unit for the Whirlwind II computer which was then in design. However, there was no discussion in this document of algebraic or logic operations.

In 1958 the first full scale implementation of a ternary computer was completed by a Russian team at the Computing Center of Moscow State University, and named Setun'. It was used for some time, but both poor hardware reliability and inadequate software hampered its usage.

By 1972 it seemed that the various efforts in multiple-valued logic were mainly in the areas of logic and switching theory, on the one hand, and electronic technology and circuitry, on the other hand. No attempt had been made to research the relative merits of ternary versus binary computers. The decreasing prices of binary components made such an attempt very desirable. Therefore, in 1972 G. Frieder proposed the design and implementation of a ternary computer aimed not for the advancement of ternary technology, but for the evaluation of arithmetic and logic operations. A computer emulation named Ternac was implemented in 1973 at the State University of New York, Amherst, New York. The first version of this implementation has proved that both the speed and price are on the order of binary computers.

The reader can acquire more information about Setun' and Ternac from Part I of this book.

In order to stimulate the reader further in these introductory remarks let us consider a very simple but interesting application of multiple-valued logic. There are some simple and rather obvious non-numerical applications of multiple-valued logic to computer programming.

As a first simple application let us consider some functions for the design of a digital image data processing system:

(1) The rate at which digitized imagery data must flow through the system, determined by the nature of the digital encoding of the image data.

(2) The complexity of the image transformation.

(3) A digitized image is broken down into frames or windows, and each frame is broken down into the basic primitive picture cells or pixels.

(4) The encoding of a pixel may be binary, ternary, octal, decimal, hexadecimal or whatever.

As a software implementation of these functions one may consider Pax II, the emulator for Illiac III, which uses stacks of bit planes for integer multiple-valued pixels; there would, for example, be $4 \cdot 8 \cdot 8 = 256$ binary digits for the encoding of an 8 by 8 frame for 16 levels of gray. Each of the four bit-matrix planes is stored individually on the host computer.

Comparatively, only 64 hexadecimal digits and one hex-matrix plane would be needed on a base 16 memory computer, an implied reduction in storage (see Fig. 1.).

Storage of one bit-matrix on an IBM System 360 would require 64 contiguous packed bits or two full words of storage. Four such bit strings would be required to store the 8 by 8 frame, whereas only one hexadecimal string would be necessary to store the frame on a corresponding base 16 memory computer.

Programming complexity can also be reduced. Consider, for example, an associative/parallel processing computer similar in architecture to the Goodyear Aerospace Corporation's Staran-S.

Referring to Fig. 2 the first bit-matrix is stored in the first bit of every four bits; and, if the program is an algorithm to search the associative memory for all frames matching the frame in the comparand register, then the complete iterative step of the program must be carried out by masking fourth bit patterns four times.

For a binary machine the search algorithm is as follows:

(1) Load CR.

(2) Load MR.

(3) Search I–set response bits.

(4) Reload MR.

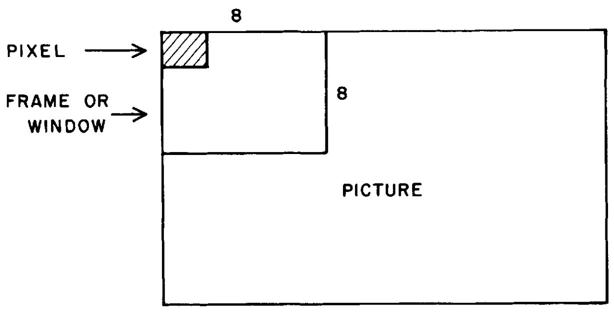
(5) Search II–considering only bits sets set in (3).

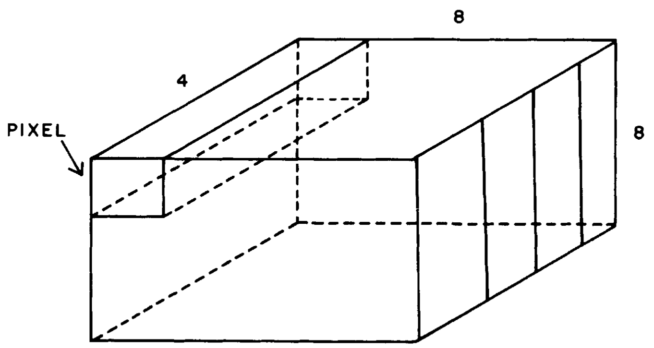
(6) Reload MR.

(7) Search III–.

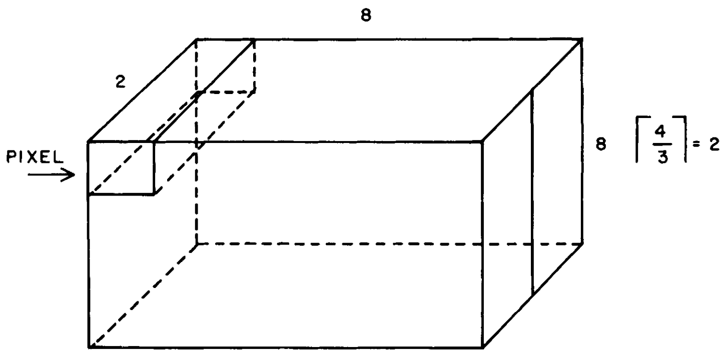
(8) Reload MR.

(9) Search IV–.





BINARY CODED, 16 LEVELS OF GRAY, 0, 1, ..., F; $16 = 2^4$ – 4 PLANES.



OCTAL CODED, 16 LEVELS OF GRAY, 0, 1, ..., F; $16 = 8^{4/3}$ – 2 PLANES.

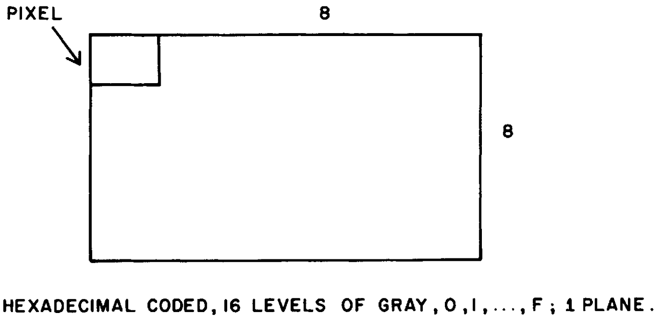

PIXEL
8
8
HEXADECIMAL CODED, 16 LEVELS OF GRAY, 0, 1, ..., F ; 1 PLANE.

Fig. 1. Comparative Storage.

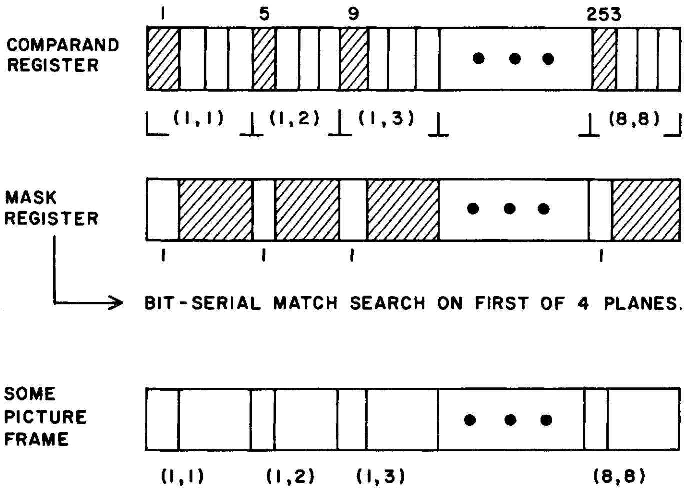


Fig. 2. 256 bit memory registers.

For a hexadecimal machine needing a shorter word length of only 64 digits, which also shortens the digit serial operation, the search algorithm is as follows:

(1) Load CR.

(2) Search I–set response bits.

Alternatively, a frame could be stored in four consecutive words (see Fig. 3.) with the first bit-matrix in the first word of each group. A flag or flip-flop could be set for each group, indicating the active bit-matrices involved in a given iterative step. For the binary machine and the same algorithm, the Load CR (or alternatively the Load CR, shift and mask) plus setting and resetting of flags would have

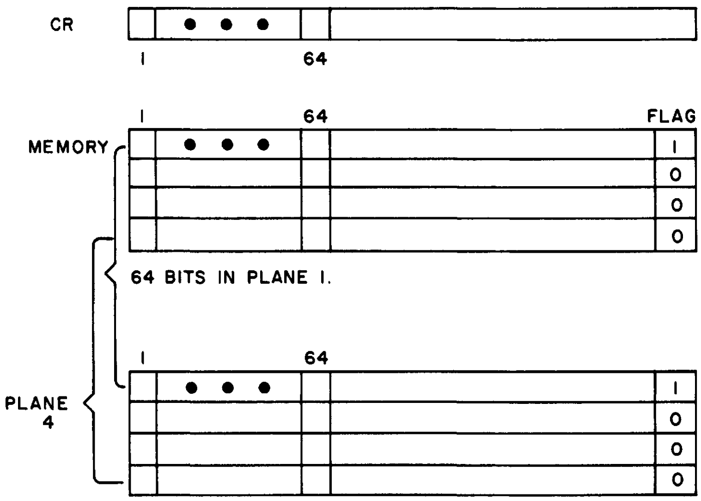


Fig. 3. Four words per frame.

to be carried out four times. On the other hand, for the hexadecimal machine the Load CR with no flags would be carried out just once.

Consider the following picture applications:

(1) Analysis of aerial photo scenes–defense application.
(2) Search of malignant lesions in chest X-ray images–radiology application.
(3) Analysis of skull X-ray images where brain growth determines skull growth and shape–anatomy application.

Digital operations common to (1)–(3) are as follows:
(a) Image subtraction.
(b) Ranking of a sequence of photos.

Question: Why is ranked output needed?
Answers:

(A) It is needed to determine the enlargement pattern of chest growth.
(B) It is needed to determine the enlargement pattern of skull growth.

Two distinct problems to be considered are as follows:

(p) Rating or simple comparing by taking differences (image subtraction).
(q) Ranking by extended binary to multiple-valued, for example by the Extended Seeber-Lindquist Algorithm due to D. Rine.

Let "EXOR" denote the exclusive-or logic function extended to bit-matrices. Image subtraction has the following logic interpretation. Let $P_1$, $P_2$ be two picture frames and define $\mathrm{EXOR}(P_1, P_2) = P_1 \oplus P_2$ coordinate-wise. Although EXOR can be used for detecting differences, it cannot be used for ranking $P$'s since symmetrical $\mathrm{EXOR}(P_1, P_2) = \mathrm{EXOR}(P_2, P_1)$ implies that $P_1 \leqslant P_2$ whenever $P_2 \leqslant P_1$. EXOR is efficiently extendable to multiple-valued logic as follows: $\mathrm{MEXOR}(P_1, P_2) = |P_1 - P_2|$, non-negative integer absolute value coordinate-wise.

For example, a possible design for a 3-valued disjoint system of algebra is

$$\mathrm{MEXOR}(x, y) = 1\cdot[C_0(x)\cdot C_1(y) \vee C_1(x)\cdot C_0(y) \vee C_1(x)\cdot C_2(y) \vee C_2(x)\cdot C_1(y)] \vee [C_0(x)\cdot C_2(y) \vee C_2(x)\cdot C_0(y)].$$

One can extend the use of MEXOR to the ranking problem by defining

$$R(a, b) = \Big|\big||a - b| - a\big| - b\Big| = \begin{cases} 0, a \geqslant b \\ 2a, b \geqslant 2a \\ 2(b - a), b < 2a. \end{cases}$$

Then, $P_1 > P_2$ whenever, at each coordination, $R(P_1, P_2) = \mathrm{MEXOR}(\mathrm{MEXOR}(\mathrm{MEXOR}(P_1, P_2), P_1), P_2) = 0$, assuming that $P_1 \neq 0$ (else $2a = 0$, $0 = a \in P_1$). Programs using R and MEXOR are less complicated and require fewer iterations than their binary counterparts.

Conclusions are as follows:

Multiple-valued logic leads to a

(1) decrease in storage size for digital encoding of the image, thus taking advantage of high density storage; and a

(2) reduction in complexity of the image transformation due to reduction in the number of pixel iterations.

And,

(3) Staran-like associative memories/processors architecture adapts well to new advances in integrated circuits, if the following unknowns are satisfied satisfactorily:

(a) existence of multiple-valued integrated circuits for array or long word memory construction;

(b) reliability of such circuits;

(c) "significant" reductions in memory logic wiring complexities.

Further developments in multiple-valued logic as related to computer science include a widening range of disciplines in which comparisons to multiple-valued logic and computer science are being made, such as neural science and ethology. Information may, also, be obtained from the references of this book and the proceedings of the 1971 and 1972 International Symposia on Multiple-Valued Logic at Buffalo, New York, the 1973 Symposium at Toronto, Canada, the 1974 International Symposium at Morgantown, West Virginia, the 1975 Symposium at Bloomington, Indiana, U.S.A., the 1976 Symposium at Utah State University, Logan, U.S.A. and the 1977 Symposium at North Carolina, U.S.A.

# part i

# algebraic theory