VOLUME ONE, NUMBER ONE — NOVEMBER, 1967.

PRICE: $1

THE AUSTRALIAN COMPUTER JOURNAL

PUBLISHED FOR **THE AUSTRALIAN COMPUTER SOCIETY**

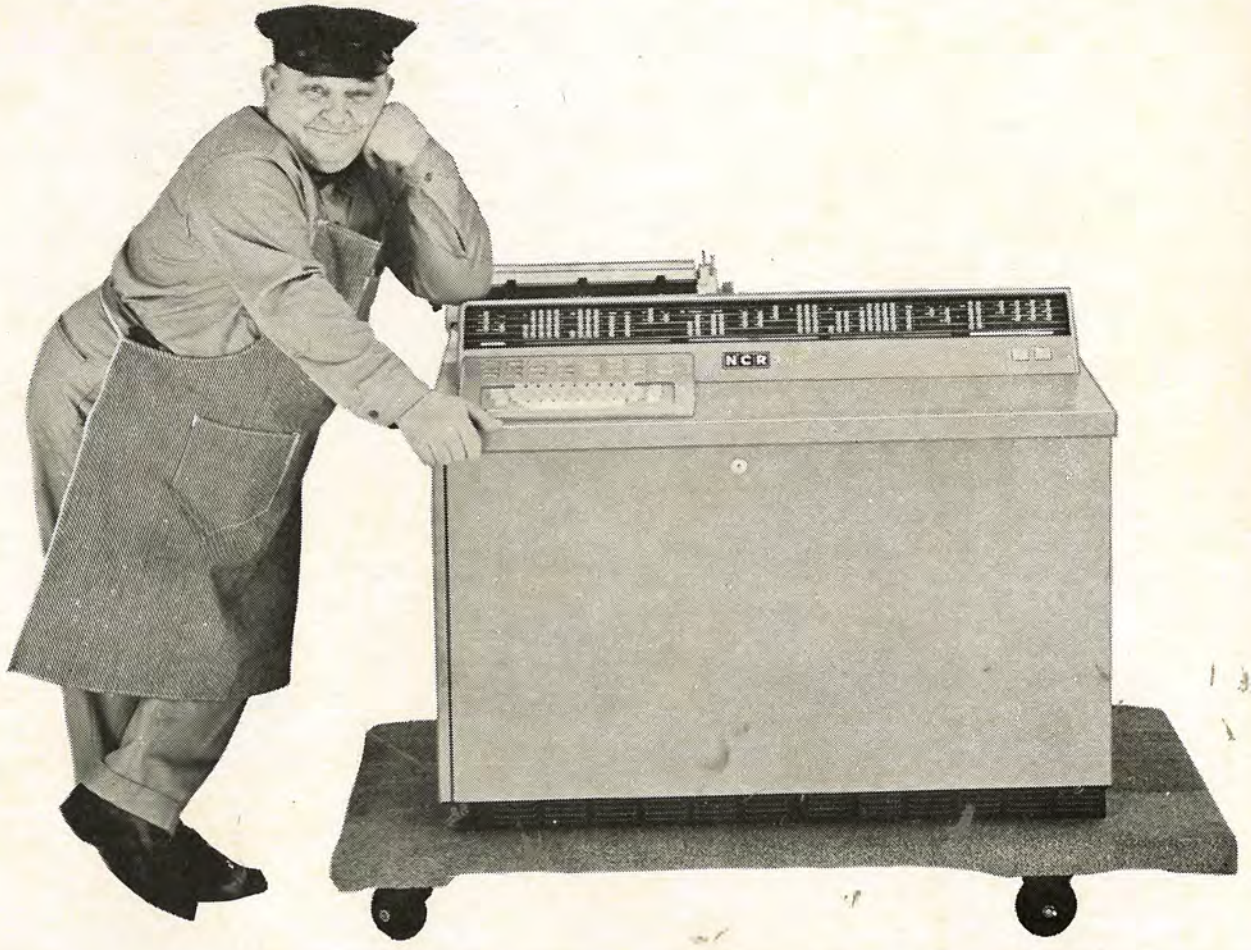
After we deliver a computer, we keep delivering.

What we deliver can save you time, headaches and money. Take off-the-shelf software for instance. Your NCR representative can deliver thoroughly documented and user-proven applied programs for business, scientific and engineering tasks right now. Software ranging from payroll to stepwise regression. From linear programming to bivariate correlation coefficient. (That's a sampling from the catalogue of management information tools in the NCR Library.)

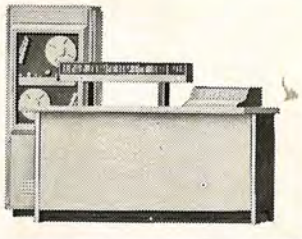
When we deliver an RMC (Rod Memory Computer), 315, or 315-100, we deliver the programming languages that are right for you. We're ready with Cobol, Neat, Best, RPG, Fortran IV, Fortran II and Neat Assembler. We're ready to deliver the right Sort/Merge routine to match your installation needs.

We're delivering an advanced operating system with automatic run-to-run control assuring the right file name and date version, and including dating flag options for any program in the system.

Software is a big reason why NCR is serving more and more customers every day. Give us a call. We deliver.



NCR



THE NATIONAL CASH REGISTER CO. PTY. LTD.

A. W. Goldsworthy.

Support....

with BASF service and know-how

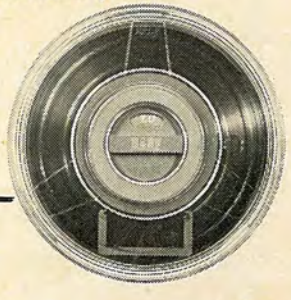
Usually, when you buy a reel of precision magnetic recording tape from somebody, they thank you very much and wish you lots of luck.
When it comes to using it, you're on your own.
Not so with BASF Computer Tape.
We have a selfish interest in making sure that BASF MAGNETIC COMPUTER TAPE gives you maximum performance in every application. That's why we have specialists available to give practical, technical advice to all BASF users.
BASF manufacture a complete range of Computer Tapes to the exacting specifications and requirements of all major computer manufacturers.

we support BASF users all the way.

There's a lot more we'd like to tell you about BASF Computer Tape, and the support you get with it, including the fact that

BASF OUTPERFORMS ALL OTHER COMPUTER TAPES AT ALL BIT DENSITIES INCLUDING 1600 bpi (3,200 fci).

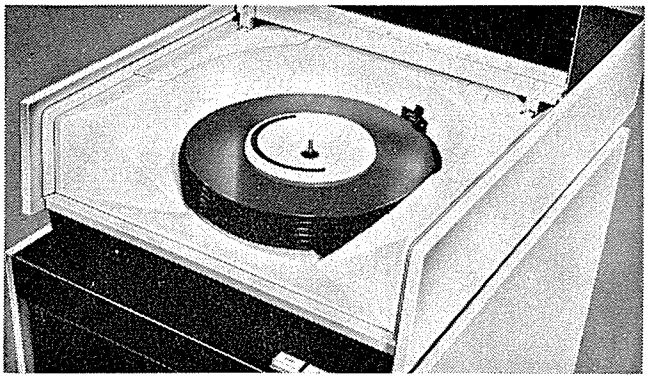
BASF®
COMPUTER TAPES



® *Registered Trade Mark of Badische-Anilin & Soda-Fabrik AG. West Germany.*

Write or 'phone the sole Australian distributors:

MAURICE CHAPMAN & CO. PTY. LTD.
210 Clarence Street, SYDNEY. 2000. 'Phone — 29 1704



I.C.T. Series 1900 exchangeable disc store

Quicker retrieval with direct access storage

Available with all I.C.T. 1900 Computers

The EDS (exchangeable disc store) has virtually unlimited storage, fast access time, is compatible with tape systems and is backed by full software facilities.

Major benefits of I.C.T. EDS:

1. **Almost immediate file reference,** with all relevant information presented for up-to-the-minute decisions.

2. **Increased computer power.** A new overlay technique stores on disc a large portion of your I.C.T. Executive program. This frees a greater share of core storage for working data; and also lets you have a very powerful Executive program to simplify your operations.

3. **Enormous capacity.** A single EDS pack holds up to 8 million characters, any item being accessed in under $^1/_{10}$ second. A library of these packs can be built up. The operator can interchange them in less than a minute.

4. **One-shot processing.** Input data need not be presorted or batched and are read only once by the computer, which posts them to one or more files as required.

5. **Extreme flexibility.** Discs make random access possible and may also be used sequentially like magnetic tape (but with much faster access time). Where both random and sequential techniques are used, you don't need tapes as well as discs, so configuration is reduced.

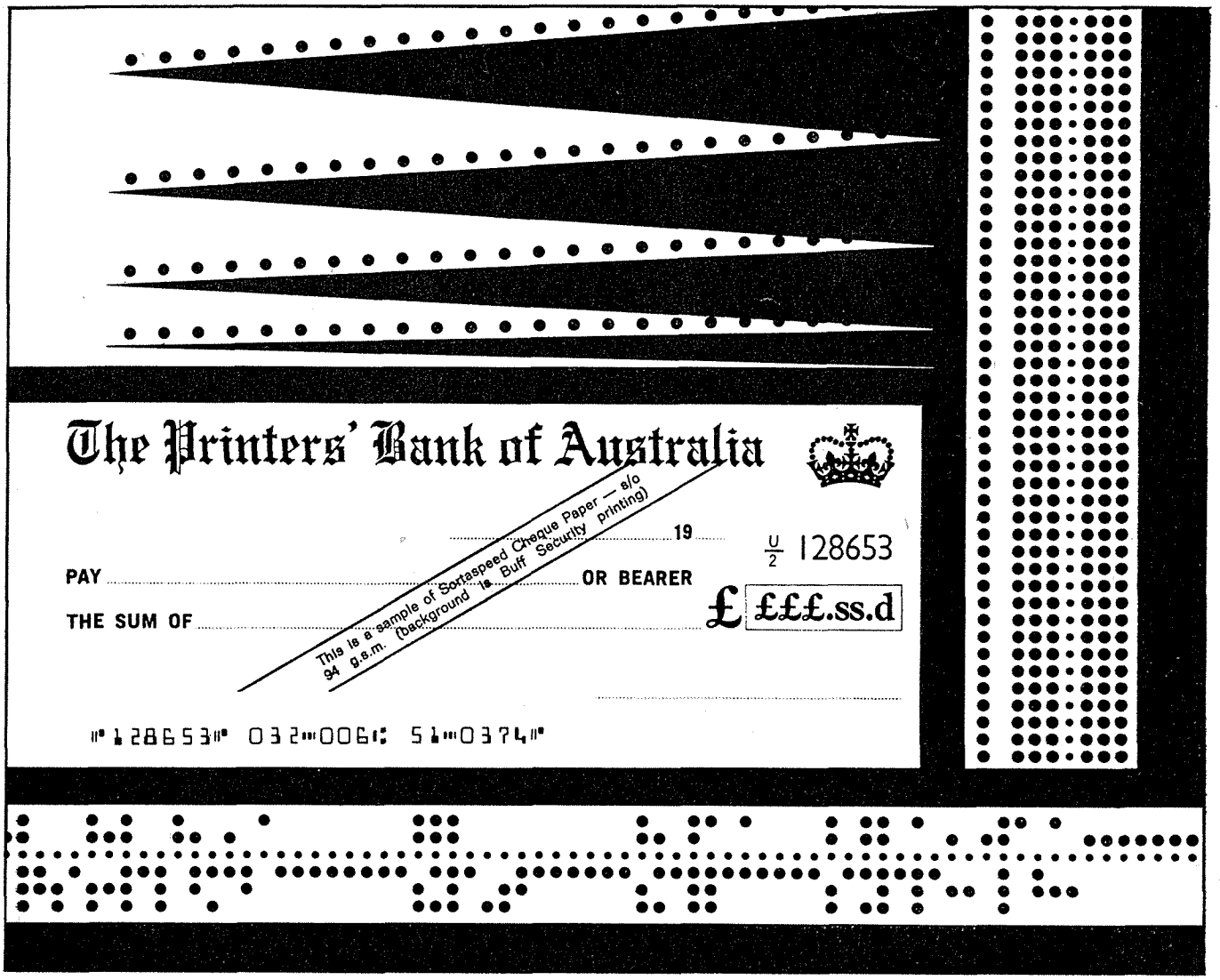
6. **Full software backing.** A wealth of experience and a vast library of software are available to help you use files and equipment to best advantage.

7. **Complete compatibility.** Existing taped data can be recompiled on to discs with ease, and there is full upward compatibility — no costly reprogramming when extending your computer configuration or changing computer.

I·C·T

International Computers & Tabulators (Australia) Pty. Ltd.
Sydney, 31-0871 — Canberra, 4-0291 — Hobart, 34-2416 — Perth, 21-9481—Melbourne, 51-0241—Brisbane, 2-3701—Adelaide, 23-3022

24.130



A new addition to Shoalhaven's range of outstanding Automation Papers

SHOALHAVEN COMPUTER PAPER TAPE

SPECIALLY DEVELOPED IN AUSTRALIA FOR HIGH-SPEED OPERATION AND PHOTO-ELECTRIC READERS.

Other automation papers in Shoalhaven's range:

SORTASPEED PAPER

— for large volume cheque sorting and other Automation applications

CONTINUOUS STATIONERY GRADES

— for single and multi-part sets

TELEGRAPHIC TAPE PAPER

— for low to medium speed mechanical readers and punchers

For further information ask your supplier or contact direct:

SHOALHAVEN PAPER MILL SALES PTY. LTD.

123 Euston Road, Alexandria, N.S.W. Phone: 51 0331 □ 5-9 Boundary Road, North Melbourne, Vic. Phone: 329 6277

SP277

THE AUSTRALIAN COMPUTER JOURNAL

Volume 1, Number 1 ● November, 1967

The Australian Computer Journal is the official publication of the Australian Computer Society.

Office-Bearers for 1967:

President: T. PEARCEY

Vice-President: P. M. MURTON

Secretary/Treasurer:
R. W. RUTLEDGE
Colonial Sugar Refining Company Ltd., 1 O'Connell St., Sydney 2000, N.S.W., Australia.
Tel.: 2-0515.

Editor: T. PEARCEY,
C.S.I.R.O., P.O. Box 109, Canberra City, A.C.T. 2600.
Tel.: 4-0455, Ext. 502.

Editorial Committee:

J. M. BENNETT, E. S. BURLEY,

E. H. GIBBS

Published by:

Australian Trade Publications Pty. Ltd., 28 Chippen St., Chippendale 2008, N.S.W. 69-6880.
Vic.: 386 Flinders Lane, Melbourne 3000. 61-3806.
England: 4a Bloomsbury Square, London, WC1. Holborn 3779.
U.S.A.: 1560 Broadway, New York, 36. LTI 3755.
Japan: 34 3-chome, Hatagaya, Shibuya-ku, Tokyo. 463-4665.

Printed by:

Publicity Press Ltd., 29-31 Meagher St., Chippendale, N.S.W. 2008

Contents:

7 *Comments On Some Legal Aspects Of Data Processing by Computer*
By K. S. POPE

15 *Mixed-Data Classificatory Programs I. Agglomerative Systems*
By Dr. G. N. LANCE and W. T. WILLIAMS

21 *A Basis For A Theory Of Programming Languages*
By J. G. SANDERSON

28 *On Intelligence, Intelligent Automata And Robots*
By D. L. OVERHEU

37 *Computer Science Education In Australian Universities*
By P. D. JONES

44 *A Computer Representation Of Plane Region Boundaries*
By B. G. COOK

51 *A Note On Nonlinear Regression Analysis*
By J. KOWALIK

54 *IFIP—Its Structure And Its Aims*
By Dr. A. P. SPSISER

Special features:

2. Notes On Submission Of Papers ● 4. Book Review ● 6. Editorial ● 36. Book Reviews ● 57. International Federation For Information Processing ● 57. Obituary ● 58. Standards Association of Australia ● 59. Australian Computer Society Branch Notes

REPRINTS: Fifty copies of reprints will be provided to authors. Additional reprints can be obtained, for which the scale of charges may be obtained from the publisher.

PRICE: The price of copies of the Australian Computer Journal to members of the Society and additional copies to members is $A1.00 per copy.

MEMBERSHIP: Membership of the Society is via a Branch. Branches are autonomous, and may charge different membership fees. Information can be obtained from the following Branch Secretaries:
Canberra: Mr. E. H. Gibbs, P.O. Box 258, Kingston, A.C.T. 2604. **N.S.W.:** Mr. J. E. Marr, Box B 157R, Royal Exchange P.O. 2001. **QLD.:** Mr. L. Olsen, P.O. Box 245, Brisbane 4001. **S.A.:** Mr. A. E. Norman, Institute of Technology, Adelaide 5000. **VIC.:** Mr. P. R. Masters, P.O. Box 104, South Melbourne 3205. **W.A.:** Mr. D. C. Carpenter, P.O. Box K/835, Perth 6001.

THE AUSTRALIAN COMPUTER JOURNAL

Notes on Submission of Papers

Communications. Papers submitted for publication should be sent to Mr. T. Pearcey, Editor, The Australian Computer Journal, P.O. Box 258, Kingston, A.C.T. 2604, Australia. All papers submitted will be subject to consideration by an Editorial Committee in consultation with referees authoritative in their particular fields. Authors will be notified as soon as possible whether the paper has been accepted for publication, of the date of the journal when it will probably appear, and of any modifications suggested by the referees.

Submissions should include the name and address of the person to whom the proof is to be sent, the name and address of the author's organisation or laboratory, and the author's title and full name. Honours and degrees may be included for editorial reference but these will not be published. Contributors who reside outside Australia should, wherever practicable, nominate a representative in Australia to whom their proofs may be sent for correction.

Eligibility of Papers. Papers submitted must be in English and they may be in the categories of review articles, exploratory articles, research articles or papers on advances in applications. Papers submitted should be certified by the authors as original and that they have not been previously published or are not being considered for publication in any other publication. If accepted for publication in The Australian Computer Journal a paper may not be published elsewhere in the same form, in English or any other language, without the consent of the Editor. Submission of a paper will be held to imply that it has been cleared for publication from military or commercial security standpoints, and no responsibility will be accepted by the Editor or the Australian Computer Society for any consequences arising from the failure by an author to obtain such a clearance.

Form of Papers. A high standard of preparation is expected of papers submitted for publication and those not deemed satisfactory may be returned to the authors for revision.

Papers should be in double-spaced typing on one side only of sheets of uniform size with wide margins. A progressive count of each hundred words of the paper should be shown in the left-hand margin. A top copy and one carbon copy of the paper should be submitted. The title of the paper should be as short as possible and should be set out clearly at the head of the first sheet. Each succeeding sheet should show a shortened version of the title suitable for a running title on each page of the published work. The sheets of the paper should be consecutively numbered in arabic numerals and firmly stapled together in the top left-hand corner. Each paper must be accompanied by a summary of about one hundred words which will be printed immediately below the title at the beginning of the paper. The first sheet of the paper should show the exact form of the author's name required for printing under the heading (last name, and given name or initials only) and the name and address of his organisation suitable for inclusion as a footnote.

Sub-headings. Sub-headings and paragraphs of papers should be clearly indicated. The hierarchy and required indentation of sub-headings and the sub-structures of paragraphs must be presented unambiguously.

Footnotes. Footnotes should be rarely used and brief. They should be typed immediately below the line to which they refer and the sheet should be ruled in ink or type for its full width immediately above and below the footnotes.

Tables. Each table should be numbered consecutively and should have a general but brief self-explanatory heading typed at the top. Column headings should be clearly shown, explicit and brief to facilitate type-setting. Tables should be carefully laid out to fit a two-column format which offers a printing area of $3\frac{1}{4}$ ins. x 9 ins. in each column. Exceptionally, tables may occupy the full printing width of 7 ins.

Tables should not normally be included in the text but should be typed on separate sheets. More than one table may be included in a single sheet, but tables should not be split between sheets. Their approximate position in the text should be indicated in the margin of the text.

Figures. Diagrams and flow-charts must be well drawn in indian ink and clearly lettered on stiff white paper or Bristol board. Directional arrows must be clearly indicated. **The diagram should be approximately twice the size of the finished block.** The size limits for finished blocks are as indicated for tables. Each diagram should be on a separate sheet, packed flat and have the author's name and figure number on the back. Where photographs are submitted, glossy prints are required, clips should not be used and heavy pressure should be avoided when writing on the back. The numbering, heading and positioning of figures should be shown as for tables.

Mathematical Formulae. Mathematical formulae must be clearly written and avoid symbols or arrangements that are difficult to set up. They should be consistent with the printing width available.

Appendices. If papers require the presentation of extensive mathematical treatments or supporting data (where quotation of references will not suffice) such detail should be included in appendices at the end of the paper. Appendices should be clearly identified as such, be consecutively numbered in arabic if the paper has more than one, and have self-explanatory headings. Subject matter should be prepared in accordance with the rules for papers.

References. References should be identified in the text thus: Curtis and Osborne (1966); (Cohen and Nguyen-Dinh, 1966). Where the paper cited has more than two authors, the names of all the authors should be given when reference is first made, e.g. (Fair, Flowerdew, Munro and Rowley, 1966). Subsequent citations of the same paper should appear as (Fair et al., 1966). Where more than one paper by the same authors has appeared in the one year the reference should be shown as follows: Lance and Williams (1966a); Lance and Williams (1966b); Lance and Williams (1966a, b); (Lance and Williams 1966a, 1967).

At the end of the paper, references should be given in alphabetical order according to the names of the first authors of the publications quoted, names and prefixes being entered under the prefix. The references should include the authors' initials, year of publication, title of paper, the name of the journal, volume and first page number. References to books and monographs should include year of publication, the title and edition, first page number of the section referenced, place of publication and the name of the publisher. Examples:—

BINGHAM, J. A. C. (1967): "An Improvement to Iterative Methods of Polynomial Factorization", Comm. Assoc. Comp. Mach., Vol. 10, p. 57.

BROOKS, F. P., Jr., and IVERSON, K. E. (1963): Automatic Data Processing, p .155. John Wiley and Sons, Inc., New York, London.

PAINE, R. M. (1966): "Preparation for Optical Character Recognition", The Computer Journal, Vol. 9, p. 221.

RICHARDS, R. K. (1955): Arithmetic Operations in Digital Computers. 10th Print. 1965, p. 314. D. Van Nostrand Company Inc., New York.

It is the responsibility of contributors to check the accuracy of references in their papers before submission.

Proofs. The authors are responsible for seeing that their type-scripts are in form for publication. Proofs are sent to the authors or their nominated representatives to ensure that the papers have been correctly set up in type and not for the addition of new material or amendment of texts. Excessive alterations may have to be disallowed or the cost charged against the author. The symbols used to indicate any corrections should be those laid down in the "Style Manual" published by the Commonwealth Government Printer, Canberra, pp 76-79.

Reprints. Fifty reprints will be supplied free of cost. A price list and order form will be provided by the printer, with the proofs, for notification of any additional reprints required.

Correspondence. All correspondence submitted for publication should be addressed to the Editor. Letters must be typed, well presented, dated and clearly indicate the reference to which discussion is directed. They must be signed and show the correspondent's name and address in typescript. Tables and mathematical formulae included should comply with the rules for papers.

Articles may be submitted dealing with the following or related subjects:—

1. ANALOG COMPUTERS
   - 1.0 General
   - 1.1 Applications
   - 1.2 Design and Construction
   - 1.3 Hybrid Systems
   - 1.4 Programming, Techniques
   - 1.9 Miscellaneous
2. APPLICATIONS—SCIENTIFIC AND RESEARCH
   - 2.0 General
   - 2.1 Natural Sciences
   - 2.2 Engineering
   - 2.3 Social and Behavioural Sciences
   - 2.4 Medical Sciences
   - 2.5 Humanities
   - 2.6 Artificial Intelligence
   - 2.7 Use of New Media, e.g., document, graphic, picture and acoustic media, etc.
   - 2.8 Information Acquisition Storage and Retrieval
   - 2.9 Real Time Systems
   - 2.10 Man-machine Systems
   - 2.11 Control Systems
   - 2.12 General Symbol Processing Systems and Techniques
   - 2.13 Operations Research
   - 2.19 Miscellaneous
3. APPLICATIONS—INDUSTRIAL
   - 3.0 Mining and Basic Metal Production
   - 3.1 Complex and Heavy Manufacturing
   - 3.2 Clothing and Textiles
   - 3.3 Food and Drink
   - 3.4 Pharmaceuticals
   - 3.5 Printing
   - 3.6 Other Chemicals (including Oil)
   - 3.7 Other Manufacturing
   - 3.8 General Production Line Processing and Control
   - 3.9 Miscellaneous
4. APPLICATIONS—UTILITIES
   - 4.0 General
   - 4.1 Electricity, Gas and Water, etc.

- 4.2 Transport
- 4.3 Communication Systems
- 4.9 Miscellaneous

5. APPLICATIONS—COMMERCE AND ADMINISTRATION
   - 5.0 General
   - 5.1 Finance
   - 5.2 Commerce
   - 5.3 Service Bureau
   - 5.4 Government (Commonwealth, State and Local)
   - 5.5 Hospital Administration
   - 5.9 Miscellaneous

6. COMPUTING MILIEU
   - 6.0 General
   - 6.1 Philosophical and Social Implications
   - 6.2 Professional Aspects
   - 6.3 Legislation, Regulations
   - 6.4 Administration of Computing Centres and Networks
   - 6.5 Management Implications
   - 6.9 Miscellaneous

7. DATA REPORTING AND TRANSMISSION
   - 7.0 General
   - 7.1 Code Systems, Reporting Media
   - 7.2 Programming for Data Transmission
   - 7.3 Design of Reporting and Transmission Systems, Data Collection
   - 7.4 Reliability, Error Detection and Correction
   - 7.5 Equipment, including Message Switching
   - 7.6 Message Switching Applications
   - 7.7 Organisation and Control of Reporting and Transmission Systems
   - 7.9 Miscellaneous

8. DESIGN AND CONSTRUCTION
   - 8.0 General
   - 8.1 Logical Design, Switching Theory
   - 8.2 Computer Systems
   - 8.3 Components and Circuits
   - 8.9 Miscellaneous

9. EDUCATION
   - 9.0 General
   - 9.1 Texts, Handbooks
   - 9.2 History, Biographics
   - 9.3 Introductory and Survey Articles
   - 9.4 Glossaries
   - 9.5 Education
   - 9.6 Automation of Teaching
   - 9.9 Miscellaneous

10. MATHEMATICS OF COMPUTATION
    - 10.0 General
    - 10.1 Numerical Analysis
    - 10.2 Theory of Computation
    - 10.3 Metatheory
    - 10.4 Combinational and Discrete Mathematics
    - 10.5 Non-numerical Mathematics
    - 10.6 Mathematical Programming
    - 10.7 Mathematical Statistics, Probability
    - 10.8 Information Theory
    - 10.9 Miscellaneous

11. MULTIPROCESSING AND MULTIUSER SYSTEMS
    - 11.0 General
    - 11.1 Multiuser and Time Sharing Systems Design
    - 11.2 Time Sharing Programming Systems
    - 11.3 Multiprocessor Design
    - 11.4 Multiprocessor Program Systems
    - 11.5 Computer Networks
    - 11.9 Miscellaneous

12. PROGRAMMING
    - 12.0 General
    - 12.1 Processors
    - 12.2 Programming Languages
    - 12.3 Supervisory Systems
    - 12.4 Utility Programs, Techniques
    - 12.5 Program and Data Structures
    - 12.9 Miscellaneous

13. SYSTEMS ANALYSIS AND DESIGN
    - 13.0 General
    - 13.1 The Structure of Information Systems
    - 13.2 Techniques of Analysis
    - 13.3 File Processing
    - 13.4 Control Techniques, Standards and Systems of Efficiency
    - 13.5 Computer Aided Design of Information Systems
    - 13.9 Miscellaneous

Book Review

Analogue Computer Applications, Ed. by A. McKenzie *(Isaac Pitman and Sons, London, 1967);* pp. VII + 96, $3.30A.

This book contains the papers presented at a symposium held at the Paisley College of Technology. The purpose of the symposium was to give engineers in industry some idea of what an analogue computer might do for them. The first two papers cover analogue computer fundamentals and the other four describe applications of the analogue computer in mechanical, chemical, civil and marine engineering, respectively.

The papers on fundamentals are rather sketchy and readers who do not know what a "high-gain direct-coupled negative-feedback amplifier with drift compensation" is would be advised first to skim through a full-length text book such as that by Truitt and Rogers. The papers on applications, however, can be recommended as giving good, brief, descriptions illustrating how a computer can be made to simulate the dynamics of a variety of engineering systems. These include a hydraulically-coupled car suspension, a continuous flow stirred-tank chemical reactor, surge chambers in a hydro-electric scheme and the steam propulsion plant, roll stabilizer and automatic steering gear of a ship. There are some useful general remarks on the advantages and difficulties of computer simulation.

P. Benyon.

The Australian Computer Society

The Australian Computer Society was formed on 1st January, 1966, by the merging of five state computer societies. There are now six branches of the Society, viz., Canberra, New South Wales, Queensland, South Australia, Victoria and Western Australia.

The objects of the Australian Computer Society include the dissemination of information concerning the application of computer science among members of the profession and the public at large, encouragement of development of the science in all phases of the Australian scene and the development of codes of ethics within the profession.

In furtherance of these objects the Council of the Society, in view of the existing series and future probable shortage of trained personnel in the computing profession, has appointed a Training and Careers Committee. Also, in view of the future need for data transmission facilities, it has appointed a committee to review and advise in the field of data transmission.

Membership in the Society derives from membership in a branch. Members are elected by the Branch Executive Committees.

Membership comprises:—

Ordinary members, with voting rights, who have to satisfy the Branch Executive Committees that they have a good knowledge of the design or functioning or programming or application of computers;

Associate members, without voting rights, who are simply interested in any of the objects of the Society; and

Student members, without voting rights, who are full-time students, under the age of 26, interested in any of the objects of the Society.

The Council of the Society may also elect Honorary Members from time to time.

Some of the branches admit organisations and societies who are actively engaged in manufacturing, distributing or using computing equipment, or with objects similar to those of the Society, as **Corresponding Institutions.**

The branches, which are autonomous in most respects, hold regular meetings, usually monthly, and may run such additional activities as study groups and local conferences. Membership subscriptions vary from branch to branch.

Editorial

The Australian Computer Society, which was formerly constituted on January 1, 1966, from the societies in the various States, announced at that time its intention to produce a publication in the field of computing science. This is the first issue of that publication, "The Australian Computer Journal". It is intended that this Journal will be of professional standard and is being established to provide a vehicle for the interchange of knowledge of advances in the theory and practice of computing, data processing, automatic control, communications and allied matters in the computer field, with particular reference to the increasingly important role that computers are assuming in all fields of activity—commerce, industry and government, as well as in science and research. The Journal also aims to provide an informative and authoritative picture of Australian computing activity and development in those fields to the computer world at large.

The Australian Computer Journal will be published twice yearly with the eventual object of becoming a quarterly publication. The content and publication is under the control of an Editorial Committee directly responsible to the Council of the Australian Computer Society. All contributions submitted will be subject to consideration by this Committee in consultation with referees authoritative in their particular fields. A high standard will be required of selected contributions.

The Council of the Society believes there is a definite need for a Journal of high standard and that such a Journal can be maintained in Australia. However, the success of such a Journal depends not only upon its management by the Editors, but also the practical support of readers by their offering articles of sufficently high standard.

The contents of the Journal will include independently submitted and refereed papers in the following categories:

- review articles
- exploratory articles
- research articles
- advances in applications

and material of the following kinds:

- selected papers presented to branches of the Australian Computer Society
- abstracts of conferences and symposia papers
- reports by special interest groups
- notices of conferences and symposia
- book reviews
- notes on standards
- Branch notices
- editorial
- correspondence of a learned kind
- advertising, including staff advertising, of a limited and controlled nature.

The general scope of subject matter appropriate for inclusion in the Australian Computer Journal is set out elsewhere in this issue. Papers, articles, reports, reviews and correspondence relating to any of these topics will be welcomed by the Editorial Committee. Submissions from overseas authors will be welcomed.

Those working in the commercial, industrial and administrative fields of automatic data processing are particularly encouraged to submit articles. It is certain that a considerable number of ideas and their experience of practical application exist in those fields which are worth reporting. These areas are also those in which the majority of the profession work and in which the greater part of the total equipment is used.

The variety of subjects in which articles and material will be considered for publication is very wide. It is the intention of the Editors to maintain a suitable balance of material in each issue, although it is expected that issues may occasionally be devoted largely or entirely to particular subjects of topical interest. It is hoped that papers presented for publication will be of sufficient variety always to provide readers in their different areas of the computing profession with some material of interest.

The Council of the Society annually reviews the lectures given to the various branches and appoints a Lecturer of the Year and sponsors the delivery of the selected lecture to the various branches and it is intended that such papers will be published in this Journal.

This issue of the Journal contains an article by Dr. A. P. Speiser on the structure and aims of the International Federation for Information Processing (IFIP). This article, as well as being informative, is pertinent in the affairs of the Australian Computer Society at this time. The national Australian body that is currently affiliated with IFIP is the Australian National Committee on Computation and Automatic Control (ANCCAC), but negotiations are now in course for the functions of this body to be taken over by the Australian Computer Society. When this move is completed, the Australian Computer Society will be the body to be affiliated with IFIP and will be in the position to provide a strong, direct link between the computing field in Australia, the international organisation and its various technical committees and special interest groups.

The Journal will be issued free to members of the Australian Computer Society and will be available to registered subscribers and for general sale at $1 per copy.

The Council and the Editorial Committee of the Society gratefully acknowledge donations in support of the Journal from the Broken Hill Pty. Co. Ltd. and the Colonial Sugar Refining Co. Ltd.

T. PEARCEY

Comments On Some Legal Aspects Of Data Processing By Computer

By K. S. Pope*

Summary

This paper was selected by the Council for the Award of the A.C.S. Lectureship for 1967.

The present position concerning the recognition of computer output by the Courts is complex and unsatisfactory. Legal requirements in the field of criminal, civil and company law are examined and problems which arise are discussed in detail. Steps to be taken to safeguard the position of computer users are considered and current views as to the manner in which the validity of output may be demonstrated are criticised. Proposals are put forward as to possible lines of action to meet the needs which will emerge as a result of the developing use of computers within the community.

1. INTRODUCTION

1.1. The recognition of innovation by the Law

1.1.1. As each new technical innovation gains acceptance in our society, the law is presented with new problems to which new solutions have to be found. The process of legal recognition is necessarily slow. Since politicians tend to be conservative in office and understandably reluctant to impose statutory regulation of novel activities, the principles extant in the existing law have to be applied to the new situation arising. The fact that it is possible to do this, and to modify or extend those principles is typical of the inherent strength and flexibility of those legal systems which have their origin in English legal processes. The delay in affording legal recognition to the social consequences of technical innovations or changes in social behaviour is sometimes regrettable, even though the lawyers may not be to blame, and has sometimes caused injustice. Examples of this are not hard to find. Up to the passing of the Fatal Accidents Act, 1846, the recently invented railway engine could only be regarded as a 'deodand' in fatal accidents, so that while in theory it might be forfeited to the relatives of the deceased, in practice they could not sue for damages. Similarly, the prolongation of the 'last opportunity' rule in cases involving joint negligence up to the passing of the Law Reform (Contributory Negligence) Act, 1945, undoubtedly caused hardship to employees injured in modern industrial processes.

1.1.2. The slow response of the law to a changing situation is not by itself to be condemned, since the regulation of man's affairs is considered—at least by lawyers—to be a serious business, and one in which hasty or contrived solutions seldom provide adequate remedies.

1.1.3. By judicial standards the electronic digital computer is still very much an infant, and an un-named brat at that. And yet it is generally agreed that the computer will ultimately have a profound effect on social relationships and patterns of behaviour. So far as the use of computers as machines is concerned, however, the existing law relating to contract and tort will probably suffice. It is not likely, for example, that computers will bring about changes in the legal principles which govern contractual relationships, or will cause new kinds of harm that cannot be adequately dealt with by the existing law relating to damage and negligence. Revision of the laws of copyright to take account of computer programs would seem far more likely and, to some scientific programmers, a much more pressing need. But in the main the legal problems arising from the use or misuse of computers would not seem likely to call for new substantive law.

1.2. Computers and the laws of evidence

1.2.1. There are, however, other ways in which the law may be forced in the long run to recognise the special position of computers as data processing machines, because of difficulties arising in the application of the laws relating to evidence or performance of statutory duties. This is hardly surprising, because the main reason for the existence of computers is to produce output in one form or another and this output can be of considerable significance from the standpoint of the parties involved in any dispute. The regard the law will have (or even should have) for the output of computers is not clear, but it would seem that for some time to come the major problems confronting parties who wish to settle a dispute concerning performance of work by a computer is how to persuade the Court first that the output is fit to be received as evidence, and secondly that it has proper weight.

1.2.1. A further point in this regard concerns use of computers to perform mandatory duties under regulating legislation. Is the use of magnetic file processing equipment with the print-out of cyclic summarised reports only, a proper manner in which to keep books of account, for example? Could receipting machines be used on-line for the capture of input without producing hard copy of the transaction for retention as an accounting document? Could the storage of data

* *Formerly State Treasury, Queensland, now with I.C.T. Australia Pty. Ltd.* *Manuscript received March, 1967.*

on a disc file with retrieval by interrogating typewriter or visual display constitute a "statutory" index?

1.2.3. Questions of this kind, at the moment, are relatively trivial, but must assume greater significance. By their nature they are likely to arise mainly in the field of company law.

1.3. **Purpose of this paper**

1.3.1. The purpose of this paper is accordingly to comment on some of the main issues which arise in relation to the validity of computer output. No attempt is made to suggest legal solutions to the problems which are seen, and it is not claimed that the topics discussed cover the whole field of interest. Furthermore, the viewpoint adopted is necessarily that of the data processing rather than the legal practitioner.

2. COMPUTER OUTPUT AS EVIDENCE

2.1. The nature of documentary evidence

2.1.1. The law relating to the acceptance of documents as evidence is a wide subject which goes far outside the problems raised by computer output. There appears to be no precise definition of a document in English law, and the purpose for which the evidence is to be used may, in fact, influence the question whether the object constitutes a document. A document has been described as any substance on which writing, figures or other symbols are marked. The material on which the marking takes place, and the manner in which the marks are made, e.g., by hand, printing, typewriting, engraving, embossing or otherwise, seem to be of secondary concern provided that the result is capable of being interpreted in evidence.

2.1.2. Although the general description of a document is thus vague (unless specifically defined in a Statute for a particular purpose), extensions of the judicial view of what constitutes a document have been obtained to meet new circumstances. Thus photographs have been accepted and within recent times, tape recordings, although these are not documents within the ordinary sense of the word, and have sometimes been referred to as 'visual reproductions' or 'acoustic reproductions'. The more important considerations would seem to lie in (1) establishing the authenticity of the evidence, and (2) the weight to be given to it if accepted by the Court.

2.1.3. The problem of authenticity is considered later, but it is relevant to note here that the Courts have already indicated the conditions which non-written documents must satisfy before they can be accepted in evidence, at least as regards photographs and tape recordings. These turn upon proof that no opportunity for interference and falsification has arisen and been taken advantage of. With tape recordings it must also be established whether extraneous matters such as street noises or other conversations have resulted in relevant words being obscured or irrelevant words being introduced.

2.1.4. The grounds on which non-written evidence is to be admitted are of great importance to the Courts and it is to be expected that stringent safeguards will always be sought. It will be observed that as regards photographs and recordings the emphasis is upon the purity of the evidence, which from the aspect of computer output means that generally the evidence is what it purports to be, was produced at the time and in the manner which was appropriate to the creation of such evidence and that opportunity for alteration or falsification has not arisen. These are no small considerations for an organisation using computers to bear in mind.

2.1.5. We must also note in this context the general rule concerning written documents that the document itself must be produced. Copies of private documents are admissible only where the absence of the original can be satisfactorily accounted for according to strictly interpreted conditions which include statutory provisions, destruction, loss, impossibility to produce because of situation or inconvenience.

2.2. The nature of computer processing

2.2.1. Because of the emphasis on authenticity, it is desirable to summarise certain general characteristics of computer processing which those outside the field of data processing are likely to regard with suspicion or even—in some dusty legal offices—with downright amazement. These are—

(1) processing does not necessarily result in permanent records for physical inspection. Consider, for example, orthodox magnetic file processing. From cycle to cycle the master file records are liable to amendment, and the full content of a record or file is seldom printed out each cycle. Every effort is made, in fact, to **avoid** printing it out;

(2) the content of a file cannot be inspected, and recorded physical representation of it obtained, other than through the computer;

(3) there is not necessarily any direct relationship between the input documents and the physical output from the system at any one point in time. The results of the processing of any part of the input may not emerge in an identifiable form, recognisable by a human being, in the same processing cycle, or indeed in any cycle;

(4) the results of processing are not unique. Entire run sequences can be repeated, given the preservation of the transaction input and copies of the input magnetic files, without the least difference in the physical appearance of the output of the original run and the subsequent re-runs;

(5) falsification of the output can be achieved readily, physically, by the substitution of data which is illegal within the system but logically valid within the program.

2.2.2. None of these are new thoughts, although they may represent an unusual way of looking at processing. Systems designers are well aware of the difficulties encountered in presenting computer output in the proper form and at the proper time. Systems-wise, different techniques are adopted to overcome the problem. Audit trails are provided showing the amendment of master file data in relation to input transactions; periodic dumps of file content are made to provide a reference base for future cycles' processing; strict control is maintained over such mundane matters as file generation and retention periods.

2.2.3. The fact remains, however, that as systems become more sophisticated, as integrated information flow concepts using sequential or concurrent file processing techniques come more and more within the reach of the information processing specialist, so the practicability of reconciling input and output at any one point in time to an outside observer becomes more and more remote.

2.2.4. It has been suggested (George et al., 1965) that one (unsatisfactory) way out of this dilemma may be to produce by way of evidence a combination of the data from the computer and records of information obtained prior to being fed into the computer. This, however, is likely to present further problems — one is not likely to retain the basic input media (e.g., punched paper tape), and although the source documents may have been retained, the link between input and output would have to be established according to the program concerned. How many installations could produce exact documentation relating to the state of a program, at an instant of time, say, three years earlier? And why stop at the individual program —why not consider the compiler, too?

2.2.5. Problems such as these are not likely to appear in such an acute form at the outset, and many would be alleviated by the power of the Court to admit the evidence of experts on the use of computers in the particular environment. The more likely problem area initially will be acceptance of computer output as a document per se in evidence.

2.3. **Documentary evidence in civil cases**

2.3.1. The relevant law relating to the admissibility of documents in evidence in civil cases is based upon the United Kingdom Evidence Act of 1938. (This has been adopted in varying degrees by the States, e.g., Queensland Evidence Acts Amendment Act, 1962, and New South Wales Evidence Amendment Act, 1966). This Act provides for a number of situations in which a "document" may be admissible as evidence subject to certain conditions. The text of the Act is too long to quote, but the gist of the relevant sections (1, 2 and 6) is as follows:—

> where the fact to be established would be admissible if related by a witness directly to the Court, any statement made by a person in a document which tends to establish that fact is admissible in evidence also on production of the original document, provided that—
>
> (1) the maker of the statement had personal knowledge of the matter;
> (2) if he did not have personal knowledge, the statement was made as part of a continuous record, and in performance of a duty by the maker to record information supplied to him by a person who had, or might reasonably be supposed to have had, personal knowledge of the matter;
> (3) the maker of the statement is called as a witness (unless dead, physically or mentally ill, overseas, impracticable to secure his attendance or cannot be found).

It is also provided that in order to avoid undue delay or expense, the Court may accept such a document even if the maker of the statement is available but not called, **and even if the original of the document cannot be produced.** In the latter case, a copy of the original or part of it may be submitted, certified as to accuracy in such a way as the Court may decide.

2.3.2. Two other provisions are—

(1) the document or the material part of it must have been written, made or produced by the maker's own hand, or signed or initialled or otherwise identified in writing by him;
(2) in estimating the weight to be given to the statement, the Court must consider all the circumstances which go to substantiate the accuracy of the statement, particularly as regards whether the statement was made contemporaneously with the facts, and whether the maker had any incentive to conceal or misrepresent.

2.3.3. For the purposes of the Act the following definitions apply—

"Document" includes books, maps, plans, drawings and photographs.

"Statement" includes any representation of fact, whether made in words or otherwise.

These definitions are inclusive, not restrictive, of the items defined.

2.3.4. It will be readily appreciated that the above provisions are not of singular assistance so far as computer output is concerned. One can hardly envisage the installation manager solemnly initialling each page of the day's output. The position, however, is not all gloom since a Rule of the Supreme Court provides for the making of an order by a Court or Judge that evidence of any particular fact shall be given in such a manner as may be specified, and that particular use of this power extends to the production of documents or copies of documents. It has been suggested that all that is needed is legislation to extend the definitions of Section 6 of the Act of 1938, and the corresponding Acts, to include, specifically, computer print-out.

Would not this allusion to print-out itself be too restrictive, however, in view of the many possible ways in which output can now be generated?

2.3.5. A situation can be foreseen in a total information scheme where only magnetic file output emerged contemporaneously, e.g., production of labour costs from a payroll run by writing to an output file without ever printing the detail of the labour costs, or not printing it within the payroll cycle. Nevertheless, the tape contains a 'statement' which could be of legal significance, but one which could not emerge in printed form simultaneously with processing.

2.3.6. Considerations of this kind make it clear that if this approach is to be followed every effort must be made to seek as wide a definition as possible of output so as to include magnetic file recording.

2.4. **Documentary evidence in criminal cases**

2.4.1. When we turn to criminal law we must observe that the burden of proof is stricter than in civil cases. The provisions of the Evidence Act quoted earlier did not apply in criminal law, and the rule has been that statements in private documents are normally

inadmissible in evidence as to the truth of that which is asserted.

2.4.2. Recent developments in English law have considerably widened the earlier provisions. The Criminal Evidence Act, 1965, provides as follows:—

> In criminal proceedings where direct oral evidence would be admissible, a statement in a document is admissible if (a) the document is a record relating to any trade or business and compiled, in the course of that trade or business, from information supplied by persons with personal knowledge of the matter dealt with and (b) the person who supplied the information is dead or beyond the seas, or unfit through his bodily or mental condition to attend or cannot with reasonable diligence be identified or found or cannot reasonably be expected to have any recollection of the matters dealt with.

2.4.3. In this context "statement" includes any representation of fact whether made in words or otherwise, and "document" includes "any device for recording or storing information (Section 1 (4))."

2.4.4. Other relevant Sections of this Act are as follows:—

Section 1(2): For the purpose of deciding whether or not a statement is admissible the Court may draw any reasonable inference from the form or content of the document.

Section 1(3): In estimating the weight to be attached to an admissible statement regard must be had to all the circumstances from which any inference can reasonably be drawn as to its accuracy or otherwise, and, in particular to the question whether or not the information was supplied contemporaneously and whether or not there was any incentive to conceal or misrepresent the facts.

2.4.5. It will thus be seen that the provisions of this Act are apparently wider in their intent, and the reference to any device for recording or storing information would seem prima facie to include the use of a computer. In this the Section is wider than in the Evidence Act, 1938, which merely sets up an inclusive definition of a "document". The Section is thus much closer to the provisions of Section 369(1) of the Australian Uniform Companies Act of 1961 which provides that registers, etc., may be kept by making entries in a bound book "or by recording the matters in question in any other manner".

2.4.6. It is too soon to be able to state the view the English Courts will take of these provisions of the 1965 Act, but it is clear at least that the door has been opened to consideration of the use of computer output as evidence in criminal cases, particularly in view of the power given to the Court in Section 1(2) and 1(3). We may note in passing, however, the specific references to "accuracy" and "contemporaneously", which have been mentioned earlier in relation to the mechanics of computer processing.

2.4.7. Neither is it possible to state the view the Commonwealth States may take in regard to adoption of similar provisions in their criminal law, and the author has no knowledge as to the extent to which this has already taken place, or is likely to.

2.4.8. Finally, it may be noted that even if these provisions are included some difficulties will remain. The document must be a record relating to any "trade or business" and compiled in the course of that trade or business. Would this cover the case, for example, of falsification of the results of a public examination by manipulation of the input relating to marks gained?

3. COMPANY LAW ASPECTS

3.1. The Companies Act 1961

3.1.1. The uniform Companies Act of 1961 which applies in the various States is the main statute regulating the management of Companies. Those sections of the Act which are of most interest from the data processing aspect are set out below in the sequence which they follow in the Act—

1. **Definition of "Books"**

Section 5(1): "Books" includes accounts deeds writing and documents.

2. **Register of charges**

Section 107(2): Every company shall keep at the registered office of the company a register of charges and enter therein all charges specifically affecting property . . .

Section 107(3): The copies of instruments and the register of charges kept in pursuance of this section shall be open to inspection . . .

3. **Register and index of members**

Section 151(1): Every company shall keep a register of its members and enter therein:—

(a) the names and addresses of the members, and in the case of a company having a share capital a statement of the shares held by each member;
(b) the date at which the name of each person was entered in the register as a member;
(c) the date at which any person who ceased to be a member during the previous seven years so ceased to be a member; and
(d) in the case of a company having a share capital, the date of every allotment of shares to members and the number of shares comprised in each allotment.

Section 151(4): The register of members shall be **prima facie** evidence of any matters inserted therein . . .

Section 151(5): Every company having more than 50 members shall, unless the register of members is in such a form as to constitute in itself an index, keep an index in convenient form of the names of the members and shall, within 14 days after the date on which any alteration is made in the register of members, make any necessary alteration in the index.

4. **Mistakes in entering the register**

Section 155(1): If—

(a) the name of any person is without sufficient cause entered in or omitted from the register; or
(b) default is made or unnecessary delay takes place in entering the register the fact of any person having ceased to be a member,

the person aggrieved
or the company may apply to the
Court for rectification . . .

5. **Keeping of accounts**
Section 161(1): Every company and the directors and managers thereof shall cause to be kept in the English language such accounting and other records as will sufficiently explain the transactions and financial position of the company . . . and shall cause those records to be kept in such manner as to enable them to be conveniently and properly audited.
Section 161(2): The company shall retain the records referred to in subsection (1) of this section for seven years after completion of the transactions or operations to which they respectively relate.
6. **Manner of keeping accounts, etc.**
Section 369(1): Any register, index, minute book or book of account may be kept either by making entries in a bound book, or by recording the matters in question in any other manner.
7. **Reasonable precautions to be taken against falsification**
Section 369(2): Where any register, index, minute book or book of account required by this Act to be kept is not kept by making entries in a bound book, but by some other means, reasonable precautions shall be taken for guarding against falsification and for facilitating its discovery . . .
3.1.2. Attention is also drawn to Section 70 as to keeping a register of debenture holders (especially 70(5) as to supplying a copy) and Section 370 as to inspection and keeping of registers generally.

3.2. **The effect of Section 369**
3.2.1. Of the Sections quoted above, Section 369 is of general application and is probably the most far-reaching in its effect because of its wide extension of means of keeping registers and books to include "recording the matters in question in any other manner". This Section apparently ends the doubt which had prevailed earlier whether such registers could lawfully be kept on mechanised card systems, and would apparently authorise the use of records maintained by a computer system, notwithstanding the general provision of Section 161(1) that accounting records should be kept in the English language.
3.2.2. It would appear that there has so far been no judicial decision on the limits to be placed on the construction of the words "in any other manner" (Butts, 1966) and provided that the restrictions imposed in Section 369(2) concerning reasonable precautions for guarding against falsification can be observed, the use of computers for accounting purposes, share and debenture register records would appear to fall within the Section.
3.2.3. Doubt has been expressed in some quarters in England, however, whether the similar wording "in any manner" contained in the corresponding section of the United Kingdom Statute (Companies Act, 1948, Section 436(1)) can be deemed to cover records maintained by computers. This view is of great interest, but no conclusions can be derived from it at present.
3.2.4. The important question is—what are reasonable precautions? This is crucial not only because of the importance of the records from the Company's point of view, but also because of the penalty which is attached in Section 369(2) to failure to take reasonable precautions. This is a penalty of $100, and since it is a "default penalty" as defined in Section 380(1) it could continue each day after conviction until the situation had been remedied.

Furthermore, on a strict view it would seem that even if precautions of what were adjudged to be a reasonable standard were laid down for an installation in processing this work, any manager (or operator?) who "knowingly and wilfully" relaxed those precautions during processing could be guilty of an offence. (See Section 380(3) of the Act.)
3.2.5. The Companies Act of 1961 passed by the States is based upon the English Act of 1948 which revised the then existing law. It is interesting to note that the requirements of the section of the English statute concerning records not kept in a bound book seem more stringent than the Australian. Section 436(2) of the Act of 1948 states that "adequate" precautions shall be taken—which would imply that if any falsification took place the Company would inevitably be liable to penalty because its precautions however reasonable could not have been adequate. (Considerations of this kind prompt the reflection, would it be safe to allow lawyers to write programs? Probably, provided they were not allowed to use COBOL.)
3.2.6. The definition of reasonable precautions is a matter of some difficulty and it influences all data processing carried out on matters under the mantle of the Act of 1961. It has strong links with the general problem of convincing the Courts that the results of computer processing may be accepted as evidence, and detailed discussion of the problem is accordingly deferred until the latter part of this paper.

3.3. **Processing under Section 151 (register and index of members)**
3.3.1. The use of a computer for processing share registration transactions within the requirements of Section 151 is attractive because of the ability to integrate the various activities involved in recording transfers, balancing the register, calculating dividends and writing dividend cheques. The ability to carry out cheque reconciliation as a by-product is also obtained.
3.3.2. Various approaches may be adopted in processing, and all will utilise magnetic file media. It is possible to adopt techniques whereby the register and the index are combined so as to meet the requirements of Section 151; one device is to use an "expansible" master file on which computer records containing details of share transactions are accumulated from cycle to cycle, while ledger cards containing transaction data are printed out and added to an expanding index file. (Walker, 1966.)
3.3.3. Processing of this kind also falls under the cover of Section 369. In addition, however, other Sections apply in that Section 99(1) requires that the company shall complete and have ready for delivery all the appropriate certificates in connection with the allotment or transfer within one month of the date on which a transfer is lodged. The computer processing cycle must, therefore, be at less than monthly intervals to meet this requirement.
3.3.4. The situation is complicated by the require-

ments of Section 153 which provides for inspection of the register and production of a copy of any part of it within a period of twenty-one days after the inquiry.

3.3.5. What constitutes the register, however? It would seem that under the computer schemes which produce expanding ledger card indices at any one point in time the register would consist of the ledger card file as at the last cycle updating of the master file together with such transfers as have been lodged meantime. To safeguard the inspection and copying procedures it would thus seem desirable for the processing cycle to be more frequent than once every three weeks, regardless of incidence and volume, and for the receipt of transfer data as input to be subject to the strictest clerical and computer controls. Since a default penalty of $40 per day could apply for failure to supply the correct information within the time specified, it would be as well to leave an adequate time margin for despatch after cycle updating so as to safeguard the position should program failure or breakdown arise.

3.3.6. Where in addition the index is separate from the register, but the two documents are integrated in processing, the further complication arises that under sub-section (5) of Section 151 alterations to the register must be made within the index within a period of fourteen days.

3.4. **Mistakes in processing**

3.4.1. Section 151(1) provides an opportunity for the Court to correct mistakes made in entering the register, on the application of an aggrieved person. It is to be hoped that the operation of this sub-section would be rare indeed, but it is worthwhile reflecting that it conveys the general implication (as do other sections of the Act relating to documents, etc.) that the records maintained are of a permanent and not a transient nature.

3.4.2. Can this really be said to be true of computer output? The results of processing can be repeated over and over again at any time, provided the necessary concurrence of input, master file and program data can be assured. Where this assurance can be achieved, it can even be duplicated on other compatible computers in different locations. It would seem unlikely, therefore, that the Courts would look kindly on a situation of this kind unless it could be demonstrated that steps were taken to identify the cycle output in some official manner at the conclusion of processing.

Extreme accuracy in operating would, therefore, be necessary to avoid the need to repeat processing after the identification had taken place. (Several attempts at producing this cycle's output would hardly be convincing, particularly if one or more of them took place after the "official" run.)

4. DEMONSTRATION OF THE VALIDITY OF COMPUTER OUTPUT

4.1. **Generally**

4.1.1. The possibility of the inclusion of computer output as evidence has been discussed earlier in relation to the various statutory provisions which govern the admissibility of evidence, or which regulate the manner in which records are to be kept. It is necessary always to keep in mind the two aspects of (1) authenticity and (2) weight to be given to the evidence once admitted. We must now consider how it could be demonstrated to the satisfaction of a Court that computer output is accurate and reliable, and a sure foundation on which to base a judgment or verdict. It will also be convenient in considering this aspect to deal with the method of establishing to the satisfaction of the Court that the provisions of Section 369(2) of the Companies Act, 1961, have been complied with as to the exercise of 'reasonable precautions'. This is for convenience only, however, since compliance with Section 369 is obligatory, and is in essence distinct from the more general questions of the admissibility and weight of output produced in compliance with the requirements of that Section. From the computer user's viewpoint the same considerations are likely to apply, however, whether it is compliance with Section 369 or the admissibility of evidence which is under examination.

4.1.2. It will be argued from discussion of these topics that the steps to be taken could include not only those procedures which a prudent installation manager would follow to secure the greatest efficiency in processing of any kind, but also other procedures which, if adopted, would constitute serious restrictions upon the development of systems within an organisation. After this viewpoint has been established, the final question to be discussed is the need to develop an approach to the problems envisaged in the years ahead which will meet the meticulous requirements of the law, and permit the free expansion of processing by computer.

4.2. **The likely requirements**

4.2.1. The extent to which a Court at this point in time would be prepared to accord weight to computer output as evidence is doubtful. It seems reasonable to suppose, however, that when the need arises it will be necessary to demonstrate that the output was produced (1) at a specific date (and even, perhaps, time), with acceptance of specific responsibility by some identifiable person (2) as part of a sequence of processing in which the relationship between input and output can be established logically, and (3) subject to such safeguards against error and falsification as would apply in good computer practice.

4.2.2. It has generally been accepted in data processing circles that great reliance will have to be placed on the management standards applying within the particular installation to meet these requirements. These standards are not uniform, and are peculiar to any installation in spite of the attempts by some computer manufacturers to impose standard conventions on their customers in the interests of efficiency. There are as yet no common standards in data processing, and until the pace of technological development in this field slows down there are not likely to be. Furthermore, although some computing associations have their eyes set on the prescription of standards in operating and performance, it will be a considerable time before the majority of commercial installations become subject to their influence.

4.2.3. The management standards required in processing are relatively well known and no explanation of them is called for here. They are primarily directed towards control—over systems analysis and design, over programming, over operating, the implementation of applications and over installation housekeeping systems. They exist to serve the computer installation, and in some instances go far beyond the requirements of other parties such as auditors. With the development of integrated processing, however, it has become necessary to co-ordinate the installation standards into a wider framework of management control, with the result that they now have reference generally to the following activities:—

The recording and design of systems in the application or total system concept.

Collection, movement and transcription of data.

Programming techniques and documentation.

Editing and validity testing of input data.

Control techniques in processing, including output validation.

Auditing of computer applications.

File generation, updating, security and retention.

Operating procedures including file dumps and file access techniques in a multiple on-line processing situation.

4.2.4. The items in the above list are not comprehensive. The extent of the procedures enforced under any one of them will vary between installations, and their very existence springs directly from the fact that there are no inherent technical safeguards to ensure that the right processing is done at the right time—a simple truth overlooked by some computer enthusiasts.

4.3. **The nature of the evidence**

4.3.1. It has been envisaged (Waller, 1966) that the nature of the evidence to be submitted must be on the following lines:—

(1) Production of source documents, authenticated as required, supported by computer print-out.
(2) Production of contemporaneous output, authenticated as may be required.
(3) Evidence of good computer practice and that an expert witness may be called by either side as to the efficient running of the installation.

4.4. **The retention and production of source documents**

4.4.1. The steps taken to retain application source documents are necessary in most instances where financial matters are dealt with for the observance of audit requirements or statutory requirements as under Section 161 of the Companies Act. Their validity, however, as evidence in support of the acceptance of specific computer output is open to question. Assuming that the source documents themselves are admissible there would seem to be many obstacles to establishing the link between source document and output, e.g., the document may have been completed partially by one person and subsequently some data fields coded by another; the document may have been produced as a hard copy from a machine providing paper tape as a by-product, and the original document may be "lost" to the computer system (as in the case of a receipt handed to a second party).

4.4.2. Furthermore, to what extent logically does a source document "tend to establish" a fact in relation to, or in support of the output? For some commercial applications it is likely that, even allowing for the conversion of the source document to the input medium, a direct connection can be established readily enough. In more complex applications it will be difficult to do so. Some of the reasons for this may be—

The input run may be widely separated from the file processing run in which used, not only in run sequence but also in time.

There may be no visual indication of the **results** of processing the particular transaction. (Suppose, for example, that daily receipts are applied to update a master file on a daily basis without a listing of transactions under moved records.)

A set of source documents (e.g., a batch) may have to be used to demonstrate that since the set was successfully applied it follows that the members of it were.

It can be seen that to prove the connection in some instances would require an exposition of computer processing techniques, and it is not likely that this would gain favour with any Court.

Even where the connection is made apparent, it is only demonstrated circumstantially, and since the programs through which the transaction records pass may edit, alter, add to or even suppress particular items of data, this demonstration may raise more doubts in the minds of outsiders that it settles.

4.4.3. Added complications arise in this regard where processing is carried out in a multi-processing or multi-programming environment.

4.4.4. It is thus considered that, even though present opinion seems to favour the retention and production of the source document to support proof of output, this reliance may be ill-founded. Whether the later widespread use of document reading devices will influence this view remains to be seen.

4.5. **Production of contemporaneous output, authenticated**

4.5.1. The production of the output document and the provisions which are likely to regulate its admissibility have been referred to earlier. As the law stands, conditions are not conducive to the use of the output as evidence and many technicalities have to be overcome. Some thought will have to be given to devices which provide safeguards for the installation. It is essential to bear in mind that, subject to any statutory limitations, proceedings may arise a considerable time after the occurrence of the events. To this end it has been suggested that an "installation identification" should be affixed to all stationery produced by an installation, whether pre-printed forms or standard listing paper. This by itself would not be sufficient, but serially numbered stationery used in conjunction with an installation diary recording the allocation of stationery sets to applications would assist. The diary would have to be maintained by a specific person.

4.5.2. How far must the installation go with this? It depends on (1) the importance of the processing,

and (2) the attitude of the Courts in cases yet to be heard and as yet no more than a glint in the litigant's eye. It seems reasonable to suppose, however, that as computer output becomes more and more widely known, the provision of a device on a high speed printer, automatically set to imprint a date, identification code and "engineer's" time independently of the program-controlled print line may become a necessity.

4.6. **Expert evidence as to observance of standards**

4.6.1. A situation may be envisaged in which a party to a dispute wishes to call evidence relating to the conduct of an installation in order to demonstrate that reliance should, or should not be placed upon its output. The extent to which the Court would allow evidence to be given on this aspect may be doubtful at present, but it seems certain development of this kind will take place.

4.6.2. Such evidence would be given by Counsel stating the practices followed, or more likely, by an expert witness. There are strict rules concerning the manner in which evidence may be given by such a witness, and the grounds on which his opinions are based, and the scope of his knowledge could be searchingly tested during cross-examination. Undoubtedly, however, the "management standards" in use (or not in use) within the installation will be major planks in his evidence.

4.7. **What will be the effects of expert evidence?**

4.7.1. This line of development could be unfortunate for the data processing fraternity. Lawyers are naturally cautious, and it could be assumed that a most conservative view would be taken of what constituted good standards, for fear that an expert's evidence could be challenged by omission of a detail. Furthermore, it is likely that in a commercial environment members of the accounting and audit professions with appropriate experience are likely to be called as expert witnesses; again professions which tend towards caution.

4.7.2. It is not difficult to envisage the kind of cross-examination of a data processing manager which could ensue—

Q. Was it the practice to print out master file amendments during the update run?
A. No, this was considered unnecessary in our new system.
Q. Would you agree that this practice is followed almost invariably by other installations processing similar work?
A. Yes.
Q. By the majority?
A. Yes.
Q. How many other installations are you familiar with in which this practice is not carried out?
A. Only two.
And so on . . .

4.7.3. The result could be that rigidity and formalism are encouraged in the application of standards to protect commercial installations, including service bureaux. Experimentation in systems design and application control could be stultified or at least retarded, and installations forced into a considerable amount of clerical and computer activity which has no other purpose than that of the "alibi record".

4.7.4. If this considered to be an extreme view consider the fettering effect of the following standards suggested recently (Waller, 1966) specifically to meet legal requirements—

Compare the reference fields in a record in full before processing any part of it.

Incorrect printed output should be replaced by reprocessing and not altered manually.

Running of diagnostic programs to find faults before breakdowns can occur.

Each program must be handled by more than one programmer and tested by a second.

More than one programmer must be assigned to any one application.

Programs not to be written by staff who are wholly employed by the user department and who would afterwards work on data preparation for those programs.

Support of all figures produced as output by the computer by reference to intermediate records, listings or source documents.

4.7.5. It should be made clear that this paper does not question the use of management standards generally. The above are examples of restrictive standards, some of which in the author's view do not display a sound, practical attitude to computer processing. They represent, however, the kind of thinking which could result from the innate conservativism of the "expert witness" approach.

4.8. **The problems of the future**

4.8.1. The problems raised above have all been considered in relation to the "single application single installation" environment. What, however, of the changes to be brought about in processing by the creation of multi-processing centres linked on an open-shop network basis with on-line access and display? For all users — scientific and commercial — the emergence of this facility could be of great significance. But their use presents problems in that the installation may have no control over, and no record of, the contents of programs submitted for processing, and that the user may be unaware of the specific computer on which his work has been carried out.

4.8.2. In the circumstances it might be desirable that the onus of proof should be to demonstrate that computer output is wrong, rather than prove that it is right. But so long as the computer and data processing remain mystifying and bewildering subjects, not rightly understood by ordinary mortals, this extreme view is hardly likely to gain support, except among computer practitioners.

References

1. GEORGE, M. A. R., WILLEY, E. L., BRAKEFIELD, A. V., and GREEN, C.: "Computers and the Law of Evidence; Interim Report of the B.C.S. Law Group", *The Computer Bulletin*, September 1965, Vol. 9, p. 57.
2. BUTTS, L. W. H.: "The Legality of Electronic Data Processing for Share and Debenture Records", *A.S.A. National Convention, 1966*, Conference papers.
3. WALKER, L. R.: "Computers in Share Registration Practice", *The Chartered Accountant*, July 1966, p. 21.
4. WALLER, R. R.: "Legal Requirements in Data Processing", *The Computer Bulletin*, March 1966.

Mixed-Data Classificatory Programs I. Agglomerative Systems

By G. N. Lance and W. T. Williams*

The raw data for classification are frequently "mixed", in that they may include qualitative, multi-state, ordered and/or continuous quantities in the same matrix, which may also contain a relatively high proportion of missing or "inapplicable" entries. The problems of processing such data are examined, and existing computer programs critically compared. A new system, based on information statistics, is described.

1. Introduction

The raw data for classification normally consist of a set of entities (variously referred to as elements, individuals, or O.T.U.'s) defined by a set of characteristics; the latter are commonly referred to as "attributes", the term being extended beyond its strict statistical usage to include variables, variates, and indeed any form of description which can be numerically coded. Examination of the many examples given in Sokal and Sneath (1963) will show that virtually all algorithms up to that date were designed to operate with a single form of attribute. Early workers in pure classification almost invariably confined their attention to qualitative (yes-or-no, presence-or-absence) data which could be coded 1/0, whereas workers using ordination techniques *(vide,* e.g. Jeffers and Black, 1963) or primarily concerned with statistical tests of allocation to an existing classification used only continuous variables. Nevertheless, data as commonly collected contain a mixture of attribute-types, and it was natural that attempts should be made to accommodate these within a single algorithm. The pioneer workers in this field (e.g., Rogers and Fleming, 1964) understandably concentrated on devising efficient methods of dichotomising other types of attribute, so that the entire process could be brought within range of existing 1/0 programs. This system is not without merit. It is already known (Lambert and Dale, 1964) that in the case of the matrix of continuous quantities with many zeros, dichotomising at the zero/non-zero boundary loses little information; and in the more general case we have ourselves shown (El-Gazzar *et al.,* in press) that, although information is undoubtedly lost by dichotomisation, the extent of loss is less than might intuitively be expected. Moreover, for very large problems there is still no practicable alternative. However, we are evidently not alone in believing that "the day should be over when it is necessary to distort a taxonomist's original data for the convenience of the computer" (Watson, Williams and Lance, 1967). There are now known to us, published or in course of publication, four mixed-data algorithms, due to Goodall (1966a), Lance and Williams (1966), Gower (1967) and Burr (in press). All are concerned with agglomerative intrinsic classifications. The intention of this communication is to make a comparative examination of these methods, and to introduce a fifth which shows considerable promise. In the second communication we shall deal with divisive systems, both intrinsic and extrinsic; in this field we know of no work other than our own.

2. Terminology

There is no generally-accepted terminology of attribute types. We shall adhere to the terms we have already defined (Watson, Williams and Lance, 1967), but to avoid confusion we now relate these to the terms used by other workers. We need to distinguish between the following three categories of attribute:—

(i) **Quantitative.** These are usually measurements or counts, and the values have absolute meaning. In the general case they may be signed, though our own earlier solution (see para. 4.1 (ii) below) is restricted to all-positive data. We originally called them **quantitative,** although in our program specifications we have used the term **numerical.** They are called **quantitative** by Gower, **metric** by Burr, and **metrical** by Goodall.

(ii) **Ordered Multistate.** These characters must be able to exist in more than two states, such that the states are ranked—i.e., the extreme states are regarded as being more different than any other pair. They are termed **ordered** by Goodall, but are not specially considered by Burr or Gower.

(iii) **Disordered Multistate.** The states are not ranked, so that none is considered intermediate between any of the others; in the general case there may be two or more states. We have ourselves restricted the term **multistate** to the case where there are more than 2 states; but Gower and Goodall use the term **qualitative,** and Burr the term **nominal,** to include all cases. Since 2-state attributes permit of special simplified input routines (such as that used for

* *C.S.I.R. Division of Computing Research, Canberra, A.C.T.* *Manuscript received August, 1967.*

association analysis—Lance and Williams, 1965) it may, nevertheless, be useful to keep them apart and to distinguish them within the general term; we, perhaps unfortunately, have always referred to them as **qualitative;** they are called **alternative** by Gower and **binary** by Goodall, and are not specifically distinguished by Burr.

There is a further possible distinction within the general class of disordered multistate. It is normally assumed that an individual specified by a multistate attribute can be in only one state at a time, so that one of the states must be present and the remainder all absent. If a group is under consideration, the *proportions* of elements must sum to unity. In a probabilistic context this is equivalent to saying that an attribute with m states is associated with $(m - 1)$ degrees of freedom; we designate such attributes as **exclusive multistate.** However, consider, for example, a plant which may bear m different types of hair, any one specimen of which may bear more than one type. There are now m degrees of freedom, and the usual strategy would be to regard these as m independent qualitative (i.e., 2-state) characters. A user may nevertheless regard this procedure as over-weighting the hair character, and may ask that the m possibilities be regarded as states of a single attribute. In this case an individual may be in more than one state, and the sum of group proportions may exceed unity. We designate such attributes as **non-exclusive multistate.**

Lastly the Gower program caters for two additional cases not covered by the others. First, the states of a 2-state attribute may be *symmetrical*, in that the states are equivalent and agreement in either state is equally meaningful, or they may be *asymmetrical*, in that one state (usually the presence of a character) is regarded as more important than the other. Some workers (e.g., Sneath, 1957, in his earlier work) have wished in such cases only to take into account agreements between the "positive" states, double-negatives being disregarded. Gower calls such characters **dichotomies** and makes special provision accordingly. Secondly, Gower's program makes fairly elaborate provision for weighting attributes, with particular reference to the system of hierarchical weighting suggested by Kendrick and Proctor (1964). We ourselves have met no demand for the use of asymmetrical qualitatives, and we incline to the view that weighting introduces an avoidable element of subjectivity; we shall therefore not consider these additional facilities further.

3. DESIRABLE PROPERTIES OF SIMILARITY-MEASURES

The measure of similarity chosen must, of course, always be definable between a pair of individuals; and if the sorting strategy in use is combinatorial (Lance and Williams, 1967a) this is all that is required. However, the strategy may demand individual/group or group/group measures, as in the centroid sorting of a measure for which no combinatorial solution exists. In such cases it is usual to define the centroid of qualitative and multistate attributes by the proportion of individuals in each state, so that after fusion all attributes become numerical; a detailed account of the procedure is given in Lance and Williams (1966).

There are, however, two further requirements which, though not essential, are highly desirable. The first is that the measure adopted should be additive over attributes. Users, given a classification, almost invariably wish to know which attributes have been mainly responsible for the definition of the classificatory boundaries; this information is required for the economical description of the groups, for the production of artificial keys, and for guidance in the selection of attributes to be used for further survey. The extraction of this information is very simple if the measure is additive over attributes, but very difficult if it is not. Secondly, in view of the growing demand for ordination as an aid in the interpretation of a classification, it is desirable that the inter-individual measure shall be suitable for an ordination procedure. As Gower (1966) has shown, this is equivalent to requiring that the symmetric matrix of measures, if necessary after some simple transformation, shall be positive semi-definite. It must be accepted that this property will be impaired if there are many missing values (Gower, 1967).

Some workers would maintain that it should be possible to associate a classificatory fusion with a probability that the two groups concerned could not have arisen as random samples from a single population. This aspect is controversial (for opposing views, *vide* Williams and Lance, 1965; Goodall, 1966b); but, if only for this reason, it would clearly be an advantage if the measure used were in some way associated with a probability. It will be convenient, in the discussion of measures which follows, to divide these into (i) those that are essentially metric in origin, and (ii) those that are primarily probabilistic.

4. METRIC MEASURES

We shall use x_{ik}, x_{jk} to denote the values (or states) taken by two individuals or groups (i), (j) for the kth of s attributes, and dij for the overall similarity measure between (i) and (j). All existing metric measures are then derivatives of the Minkowski metrics of general form,

$$l_p = \left(\sum_{k=1}^{s} \mid x_{ik} - x_{jk} \mid_p\right)^{1/p}$$

of which we shall be concerned only with l_1, the Manhattan metric, l_2, the Euclidean metric, and l_∞, the range, which we shall denote by w. Williams and Dale (1965) have argued that there are advantages in using a metric as a similarity measure; and the historical importance of l_1 in this context may well be due to the fact that it is both a metric and additive over attributes. In contrast, l_2 is a metric but not additive over attributes, whereas l_2^2 is additive over attributes but not a metric. Additivity itself is, however, of little value unless the contribution of each attribute can be made dimensionless and constrained in some simple way, and it is usual to require that the contribution of each attribute be constrained between 0 and 1. It is in the precise form of scaling adopted that the several Manhattan-metric programs primarily differ.

All metric programs share two undesirable properties. First, they are unable to make special provision for ordered multi-states; if the table entries are only ranks,

these are nevertheless regarded as genuine observations and handled as continuous attributes. Secondly, the final inter-individual measure is obtained by averaging the individual contributions over all known and applicable comparisons. Two pairs $(g)/(h)$ and $(i)/(j)$ may have equal overall similarity, but this may be based on very different amounts of information if the data have many unknown or inapplicable entries; no provision is made for discrimination on this basis in any existing metric program.

4.1 Manhattan-metric Systems

Four alternative forms of scaling have been suggested, and we deal with these in turn.

(i) Bray and Curtis (1957).

Though the symbolism is different, the measure is essentially of the form $(\sum_{k=1}^{s} |x_{ik} - x_{jk}|)/\{\sum_{k=1}^{s}(x_{ik} + x_{jk})\}$; it is the quantitative equivalent of the "coefficient of community" originally suggested by Jaccard (*vide*, e.g., Jaccard, 1912) for the all-qualitative case. It is not a metric; and, using it ourselves in our early qualitative programs, we designated it the "non-metric coefficient". It obviously requires that the attributes be everywhere positive, and scaled *before* calculation of the measure, and has therefore commonly been applied only to proportions or percentages; it is not additive over attributes. Despite these disadvantages, it has survived as a result of a single important property. In the multi-state case it is reasonable to require that the measure between two individuals or groups for an attribute shall be unity if there are no entries in corresponding states; this is equivalent to stating that double-negative matches between states of a single multi-state attribute are to be disregarded. In the inter-individual exclusive case this is easy to arrange; but it ceases to be so in the non-exclusive case, or in either case if the entries are to represent proportions. The Bray-Curtis measure, when used over the states of a single multi-state attribute, is the only measure in the literature known to us which possesses this essential property, and its use in this context is virtually unavoidable.

(ii) Lance and Williams (1966).

The measure here is $\sum_{k=1}^{s} \{(|x_{ik} - x_{jk}|)/(x_{ik} + x_{jk})\}$. By a false analogy with the Bray-Curtis measure, we continued in the paper to call this the "non-metric coefficient"; but it can be shown that it is in fact a metric, and we shall henceforth refer to it as the "Canberra metric". Its relevant properties are as follows:—

(a) The measure is a property solely of the individuals or groups being compared; it is unaffected by, e.g., the addition of new members to the population.
(b) It is sensitive to proportional, rather than absolute, differences; for example, the difference for 2 and 5 is much greater than that for 102 and 105.
(c) Apart from the trivial case of all-negative data, the data must not be signed, since if x_{ik} and x_{jk} are of opposite sign the 1/0 constraint will be lost.
(d) The measure has a singularity at zero; if either x_{ik} or x_{jk} is zero, then irrespective of the value taken by the other the measure becomes unity. (The measure is, of course, taken as zero if $x_{ik} = x_{jk} = 0$).
(e) We have examined, heuristically, the properties of the matrix $(1 - d_{ij})$. In every case we have so far examined this matrix has been positive semi-definite; but we have not yet been able to obtain a formal proof that this is necessarily so.

(iii) Adkins (*in litt.*)

Mr. B. L. Adkins has drawn our attention to the form $\sum_{k=1}^{s} \{(|x_{ik} - x_{jk}|)/(|x_{ik}| + |x_{jk}|)\}$. For all-positive data it is, of course, identical with the Canberra metric; but for signed data it has the interesting property of becoming unity whenever x_{ik} and x_{jk} are of opposite sign. It has proved useful in the special case where the signs represent differences in kind rather than in degree, so that it is desired to segregate the signs as far as is practicable consistent with the production of a good classification.

(iv) Gower (1967).

In this approach (used in the Rothamsted Orion program CLASP) the attributes are standardized by range; the measure thus takes the form $\sum_{k=1}^{s} \{(|x_{ik} - x_{jk}|)/w_k\}$. Its properties may usefully be summarized in a form which parallels those of the Canberra metric:—

(a) The measure is a property of the entire population under study; if, for example, an individual with an abnormally large, outlying, value is added to the population, all measures between existing pairs will be appreciably reduced as a result of the increase in range.
(b) It is sensitive to absolute, not proportional differences: the difference for 2 and 5 is the same as that for 102 and 105.
(c) There is no restriction on sign.
(d) There is no singularity at zero.
(e) Gower, in his account of the system, has proved that the matrix $(1 - d_{ij})$ is positive semi-definite in the absence of missing values.

Properties (c) and (d) make this a considerably more attractive measure than the Canberra metric; it is, however, at least conceivable that properties (a) and (b) of the Canberra metric might be preferable in the all-positive case. To permit the future investigation of this possibility we have now added the range facility to our own programs; but we have as yet had insufficient experience of the comparative use of the Gower and Canberra metrics to enable us to report on their relative merits.

4.2 The Euclidean Metric

A Euclidean system has many attractions. Not only is l_2^2 additive over attributes; it possesses useful combinatorial properties, notably in permitting a simple solution of centroid sorting, and its square root is ordinable by definition. The question of standardization remains to be considered. An obvious solution would be to standardize by w^2 (the square of the range) but this seems never to have been used. Previous workers in the all-continuous case have normally standardized by variance, and the origin of this practice is perhaps to be sought in its importance in component, and especially in factor, analysis; but its effects in the mixed-data case require examination.

For quantitative attributes the contribution of a single attribute will now be constrained between 0 and w^2/σ^2; and since for a population of n individuals the upper bound of this quantity is adequately defined by the relationship $w^2/\sigma^2 \sim (n + \frac{1}{2})$ (Plackett, 1947) the contributions of the different quantitative attributes will at least be comparable. For a qualitative attribute the upper bound is obviously $n^2/(n-1)$, so that in this case $w^2/\sigma^2 \sim (n + 1)$. It follows that the comparative standardization of attribute-contributions, though less exact than in the variants of the Manhattan metric, is acceptable; though in any diagnostic programs designed to investigate relative contributions it would probably be desirable to scale by, say, n to provide comparability between different analyses.

The only Euclidean system known to us is the program TAXAN of Burr (in press). Quantitative attributes are conventionally regarded as orthogonal axes; standardization is by variance, but ingeniously avoids a separate calculation by use of the identity.

$$\frac{\sum\limits_{j \neq i} (x_i - x_j)^2}{2n(n-1)} = \frac{\sum\limits_i (x_i)^2 - \frac{1}{n}\left(\sum\limits_i x_i\right)^2}{(n-1)}$$

so that standardization by half the average of all single-attribute distance contributions is equivalent to standardization by the "sample" variance. Multistate attributes are accommodated by the provision of a separate axis for each state; these axes are regarded as independent, so that the variance of the whole attribute can be taken as the sum of the sample variances of the separate states. Non-exclusive multistates cannot, of course, be permitted in this system, since the variance algorithm would fail. The program incorporates a variety of sorting strategies, including a novel combinatorial strategy which minimizes the increment in within-group sums of squares. At the time of writing the program exists only in machine orders for the IBM 1620; when a FORTRAN program is available, a comparison of TAXAN with the Manhattan-metric programs will be highly desirable.

5. PROBABILISTIC SYSTEMS

5.1 A New Information-Statistic Strategy

In the all-qualitative case the use of an information statistic has proved remarkably successful (Williams, Lambert and Lance, 1966; Lambert and Williams, 1966), and is the basis of our program CENTCLAS. We now describe a generalization of this system to deal with the mixed-data situation, and three features require special mention:

(i) Standardization in the metric sense is impossible, since the additive properties of the statistic would be lost. Instead, the number of degrees of freedom associated with each type of attribute must be defined. We first consider a disordered multistate attribute of m states. Since the information gain associated with this attribute for a given fusion must be obtained from a $2 \times m$ contingency table, it follows that the attribute will contribute $(m-1)$ degrees of freedom towards the total available; a qualitative attribute will thus as usual contribute a single degree of freedom. It also follows that non-exclusive multistates cannot be permitted, and must be replaced by a set of m qualitative attributes.

(ii) An information statistic cannot be defined for a continuous variable unless its distribution is known, or postulated by non-parametric methods. We therefore propose that, whereas in metric systems ordered multistates are treated as quantitatives, in the information-statistic case quantitatives be treated as ordered multistates, which we permit to contribute only a single degree of freedom.

(iii) Missing and inapplicable values can now be handled more rigorously, since identical information gains may be associated with different degrees of freedom. To compare such quantities we make use of two approximations: (a) that, for a given number, ν, of degrees of freedom $2\Delta I \sim \chi^2\nu$, and (b) that $\sqrt{(2\chi^2)}$ tends to be normally distributed about mean $\sqrt{(2\nu - 1)}$ to order $\nu^{-3/2}$, with variance which is unity to order ν^{-1} (*vide*, e.g., Kendall and Stuart, 1963, p. 372). Since in mixed-data problems it is unusual for ν to be very small, we consider that the quantity $2\sqrt{(\Delta I)} - \sqrt{(2\nu - 1)}$ is an acceptable approximation to a normal deviate with zero mean and unit variance, and hence can be used for comparing information gains of different ν.

It will be clear that two algorithms are required, for disordered and ordered multistate (including quantitative) attributes; these are as follows:—

Disordered Multistate

Let there be an attribute with m states, and of two groups (i) and (j) let the numbers of individuals in the hth state be x_{ih} and x_{jh} respectively. We write $lg(a)$ for $a \log_e a$ and consider only summations from $h = 1$ to $h = m$; we then define an information gain for the fusion of (i) and (j) such that

$$\Delta I = lg(\Sigma x_{ih} + \Sigma x_{jh}) - lg(\Sigma x_{ih}) - lg(\Sigma x_{jh}) - \Sigma\{lg(x_{ih} + x_{jh})\} + \Sigma\{lg(x_{ih})\} + \Sigma\{lg(x_{jh})\}$$

This expression may appear cumbersome; it is, however, simple to compute since an array of $lg(x),(x = 0,n)$ can be set up at the start of the analysis and directly referenced thereafter.

Ordered Multistate and Quantitative

An m-state ordered attribute will again generate a $2 \times m$ contingency table; but we now partition the degrees of freedom and pick out only that associated with the largest information gain. By dichotomizing at each state in turn and accumulating the frequencies on both sides of the dichotomy, a set of $(m-1)$ 2×2 contingency tables is defined, representing successively the configurations:

$(1)/(2 + .. + m)$; $(1 + 2)/(3 + .. + m)$; $\ldots (1 + ..(m-1))/(m)$.

The ΔI for each table is calculated and the largest value selected to represent the contribution of this attribute to the fusion. It should be noted that the concept of order implied here is novel. It does not imply, as it would in the metric case, that differences between adjacent states are less important than those between remote states; the contribution for a single attribute in the initial stages (when all comparisons are between single individuals) is always either zero or $2\log_e 2$. The concept of order arises in that the states are regarded as defining only a single degree of freedom, but that the portion of the range which contributes this is optimal. Weighting for remoteness of states is not permissible,

since the ΔI transformation would then fail; and our present justification for the procedure rests on the excellent results it gives in practice.

Continuous attributes are converted into ordered multistates by dividing the range into m equal sub-ranges, making an entry of unity in the state appropriate to the value actually taken by the individual, and proceeding as before; but a decision is required as to the value of m. It is obvious that an increase in m increases the discrimination but also increases computing time; in our existing program we have taken $m = 8$, which appears to give acceptable discrimination without unduly extensive computation, though it is inevitable that large numbers of quantitative attributes slow down the analysis. It would, of course, be possible still further to refine the system by permitting cyclic ordering; but this would involve $\frac{1}{2}m(m - 1)$ possibilities, which we regard as computationally unacceptable.

Given the resulting accumulated ΔI values, two considerations remain: monotonicity of ΔI over successive fusions and the possibility of compatible ordination. In the all-qualitative case with no missing values, monotonicity appears to be perfect; although no formal demonstration that this must be so is known, no failure has yet been reported. With the mixed-data version, however, occasional failures occur. This may well be due to a shift in the position of dichotomy of ordered variables as the analysis proceeds. However, all sets of data we have so far processed have contained missing values; and it is possible that these, too, may impair monotonicity. So far we have encountered reversals only infrequently, and always at very low hierarchical levels; they have caused no inconvenience comparable to that experienced, for example, with centroid or median sorting of the Euclidean metric.

We have previously pointed out (Lance and Williams, 1966) that, again in the all-qualitative case, ΔI can be regarded as a constant multiple of a squared Euclidean distance, so that ΔI is ordinable by definition. We have therefore heuristically examined the properties of the inter-individual matrix defined by $1 - \Delta I/(\Delta I)_{max}$; our experience is as yet necessarily limited, but in every case we have so far encountered the matrix has been positive semi-definite.

5.2 The Goodall Non-Parametric System

This system stands apart from those previously discussed. It is not hierarchical, and is intended for use with a pure clustering strategy. However, the clustering strategy itself is not yet fully developed, so that in its present published form (Goodall, 1966a) it is confined to the calculation of inter-individual measures. A small-scale worked example of the clustering strategy under consideration is, however, given in Clifford and Goodall (in press). The measures are complements of probabilities; for each attribute these are obtained by the usual non-parametric postulate that the proportion of values within a given range can be taken as an estimate of the probability that they will fall within that range. The separate probabilities are then combined into a joint probability of their simultaneous occurrence: the system is therefore not additive over attributes and its ordination properties are unknown.

The computation of the joint drobabilities is very lengthy compared with that of the measures previously discussed, and it seems inevitable that the system will be confined to relatively small-scale problems. We ourselves have little experience of its use, but we include it here for completeness; in particular, it is the only mixed-data strategy known to us which makes a rigorous distinction between quantitative and ordered-multistate attributes. It is most desirable that the probabilities it generates should be compared with those obtained from the previous information-statistic program by means of the approximate χ^2 equivalence, since the latter are obtained very much more easily.

6. THE CANBERRA HIERARCHICAL PROGRAMS

Three programs are currently available on the Control Data 3600 computer at Canberra:—

MULTCLAS

This provides the Canberra and Gower metrics as alternatives. The strategy is entirely combinatorial, permitting nearest- and furthest- neighbour, median, group-average and "flexible" sorting; centroid is not permitted, since there is no combinatorial solution for derivatives of the Manhattan metric (Lance and Williams, 1967a). In practice, flexible sorting with $\beta = -0{\cdot}25$ is most commonly employed. Inter-individual measures can be punched out if desired, in a form suitable for processing by the ordination program GOWER; they will normally be entered into a matrix with unities in the diagonal and the complements of the measures as off-diagonal entries.

The data may include any or all of the types quantitative, disordered multistate and (as a special case of the latter) qualitative; missing values are catered for. The number of individuals may not exceed 179. The required metric is computed for each pair of individuals once and for all, and the entire symmetric matrix of measures is then stored on the drum. From this point onwards the data are no longer required; at each stage after the minimum remaining measure, say d_{ij}, has been found, one new column of measures **h** is computed from the old columns **i** and **j**. This column replaces **i** or **j** and the order of the matrix is reduced by one. The scanning for minima is so arranged that only the remaining relevant columns are considered at each stage.

MULTBET

This provides three alternatives: (i) Canberra metric with exact (i.e., non-combinatorial) centroid sorting, (ii) Gower metric with exact centroid sorting, and (iii) the information-statistic option of para. 5.1. above. A punching facility for inter-individual measures is not provided for the metric options, since these can be more easily obtained from MULTCLAS; for the information-statistic option the inter-individual information-gains can be punched out. Our present ordination practice is to standardize these by division by the largest, and then to treat them in the same way as the metric measures.

The form of data is the same as for MULTCLAS with the additional facility of the separate declaration of ordered multistates. In this program the data must be retained throughout the computation, since the exact centroid sorting strategy requires a knowledge of the composition of the groups at each stage. As a result, storage space is at a premium, and two special pro-

cedures are introduced to reduce storage requirements as far as practicable. First, the data are stored throughout in compact form—e.g., qualitatives are held as binary digits—and the expanded definition of a group is synthesised from the compressed original data only immediately before it is needed for computation; this strategy was originally suggested to us by Mr. Jacob Tharu. Secondly, only the minimum measure in each column is retained. Clearly, the overall minimum is readily available from the column minima; suppose it to be d_{ij} as before, and that columns **i** and **j** therefore combine to form **h**. The new column of values, **h**, relating the group (*h*) to all others, replaces column **j**; column **i** is no longer relevant and so can be removed from consideration. However, if (*i*) or (*j*) provide the minimum value for any other column then that entire column of measures must be recomputed in order to find the new, relevant, minimum measure. Both these devices, and particularly the second, involve additional computation; but this is the price that must be paid for the accommodation of reasonable-sized problems, and MULTBET will, in fact, handle problems considerably larger than those practicable in MULTCLAS.

A problem of choice obviously arises as between MULTCLAS and MULTBET. The main comparisons so far undertaken have been between MULTCLAS (with Canberra metric and flexible sorting at $\beta = -0{\cdot}25$) and the information-statistic version of MULTBET. In every case in which a user has been able to apply a quasi-objective test of predictive efficiency, MULTBET has been clearly superior. Its programming strategy, too, is such that it can accommodate larger problems; but it involves considerably more calculation and is therefore slower-running. Preliminary comparisons between the Gower and Canberra metrics have been inconclusive, so that the information-statistic strategy appears to be superior to both. However, comparisons between mixed-data strategies are necessarily sensitive to the precise configuration of the data, and we shall need considerably more experience over a wide range of data before we feel able to make authoritative comparative statements. As a result of our limited existing experience we advise users to employ the information-statistic MULTBET strategy unless computer time needs to be very strictly limited.

LINKED

This provides the Gower and Canberra metrics with exact centroid sorting; its special feature is the inclusion of the facility for processing multi-level quantitative attributes described in Lance and Williams (1967b). The inter-individual measures can be punched out for use either in an ordination program or for classification by the flexible strategy in the program CLASS (Lance and Williams, 1967a).

The computation strategy for this program follows that of MULTBET, except that the inclusion of the multi-level facility necessarily complicates the calculation.

References

BRAY, J. R., and CURTIS, J. T. (1957): "An ordination of the upland forest communities of southern Wisconsin", *Ecol. Monogr.*, Vol. 27, p. 325.

BURR, E. J. (Univ. New England, Armidale, N.S.W.): "Cluster sorting with mixed character types."

CLIFFORD, H. T., and GOODALL, D. W. (1967): "A numerical contribution to the classification of the Poaceae", *Aust. J. Bot.* (in press).

EL-GAZZAR, A., WATSON, L., WILLIAMS, W. T., and LANCE, G. N.: "The taxonomy of *Salvia*: a test of two radically different numerical methods", *Proc. Linn. Soc.* (in press).

GOODALL, D. W. (1966a): "A new similarity index based on probability", *Biometrics*, Vol. 22, p. 882.

GOODALL, D. W. (1966b): "Classification, probability and utility", *Nature*, Vol. 211, p. 53.

GOWER, J. C. (1966): "Some distance properties of latent root and vector methods used in multivariate analysis", *Biometrika*, Vol. 53, p. 325.

GOWER, J. C. (1967): "A general coefficient of similarity and some of its properties", *Biometrics* (in press).

JACCARD, P. (1912): "The distribution of the flora in the alpine zone", *New Phytol.*, Vol. 11, p. 37.

JEFFERS, J. N. R., and BLACK, T. M. (1963): "An analysis of variability in *Pinus contorta*", *Forestry*, Vol. 36, p. 199.

KENDALL, M. G., and STUART, A. (1963): *The Advanced Theory of Statistics* (Vol. 1 of 3-volume Edn., 2nd Edn.). Griffin: London.

KENDRICK, W. B., and PROCTOR, J. R. (1964): "Computer taxonomy in the fungi imperfecti", *Canad. J. Bot.*, Vol. 42, p. 65.

LAMBERT, J. M., and DALE, M. B. (1964): "The use of statistics in phytosociology", *Adv. Ecol. Res.*, Vol. 2, p. 59.

LAMBERT, J. M., and WILLIAMS, W. T. (1967): "Multivariate methods in plant ecology, VI.", *J. Ecol.*, Vol. 54, p. 635.

LANCE, G. N., and WILLIAMS, W. T. (1965): "Computer Programs for monothetic classification ('Association analysis')", *Comput. J.*, Vol. 8, p. 246.

LANCE, G. N., and WILLIAMS, W. T. (1966): "Computer programs for classification", *Proc. ANCCAC Conference, Canberra*, May 1966, Paper 12/3.

LANCE, G. N., and WILLIAMS, W. T. (1967a): "A general theory of classificatory sorting strategies, I.", *Comput. J.*, Vol. 9, p. 373.

LANCE, G. N., and WILLIAMS, W. T. (1967b): "Note on the classification of multi-level data", *Comput. J.*, Vol. 9, p. 381.

PLACKETT, R. L. (1947): "Limits of the ratio of mean range to standard deviation", *Biometrika*, Vol. 34, p. 120.

ROGERS, D. J., and FLEMING, H. (1964): "A computer program for classifying plants. II.", *BioScience*, Vol. 14, p. 15.

SNEATH, P. H. A. (1957): "The application of computers to taxonomy", *J. Gen. Microbiol.*, Vol. 17, p. 201.

SOKAL, R. R., and SNEATH, P. H. A. (1963): *Principles of Numerical Taxonomy*. Freeman: San Francisco and London.

WATSON, L., WILLIAMS, W. T., and LANCE, G. N. (1967): "A mixed-data numerical approach to Angiosperm taxonomy", *Proc. Linn. Soc.*, Vol. 178, p. 25.

WILLIAMS, W. T., and DALE, M. B. (1965): "Fundamental problems in numerical taxonomy", *Adv. Bot. Res.*, Vol. 2, p. 35.

WILLIAMS, W. T., LAMBERT, J. M., and LANCE, G. N. (1966): "Multivariate methods in plant ecology. V.", *J. Ecol.*, Vol. 54, p. 427.

WILLIAMS, W. T., and LANCE, G. N. (1965): "Logic of computer-based intrinsic classifications", *Nature*, Vol. 207, p. 159.

A Basis For A Theory Of Programming Languages

By J. G. Sanderson*

Summary

Four conditions are considered in order to arrive at a mathematical model of a programming language: the model must represent the fact that the result of a computation is fully determined by its program and data; it must be able to represent the processes of simulation and translation of languages; it must be able to define the syntax and semantics of a language; and it must agree with the theory of computability. A model, the computable language, is derived which meets these conditions. Simulation, translation, semantics, syntax and a number of related concepts are defined in terms of this model and their properties are discussed. The merits of the model as a possible basis for a theory of programming languages are considered.

Introduction

It is normal in science for a theory to evolve by passing through a series of approximations, each giving way to the next as a result of critical assessment.

The task of assessing a theory which deals with something purely artificial like a programming language may be quite different from the corresponding task in natural science. Since a programming language may be defined completely, if necessary by an assembly listing of a compiler, it may be possible to prove directly the validity in principle of a mathematical model. In natural science a model must be tested by experimentation. According to the usual view, this can disprove, but never prove its validity.

This does not mean that a final theory of programming languages may be set up at one stroke. The existence of one model which is valid in principle does not exclude the existence of others, also valid in principle and more useful in practice. Thus the approach used in this paper is to pose several criteria which must be met by any theory of programming languages and, by considering them in turn, to derive successive approximations to the model.

The first criterion states a basic property of all languages (excluding certain cases resulting from hardware or program faults), namely, that the result of a computation is uniquely determined by its program and data. This leads to a very general model which is called the type-A language.

The type-A language fails immediately on the second criterion, that the language operations of translation and simulation should be representable. Thus the second approximation, the type-B language, is introduced, and its ability to represent these operations illustrated by a discussion of the technique of bootstrapping used in preparing compilers and in transferring them from one machine to another.

The third criterion requires that the terms "syntax", "semantics" and "pragmatics" should be definable within the theory. The type-B model is examined in some detail in the light of this test.

The fourth and last requirement is that the theory should agree with the theory of computability. It is shown that certain type-B languages could not be realised in practice, so that a new model, the computable language, is defined to exclude these cases. This model meets all of the four criteria.

1. The Program, Data and Result of a Computation

If a computation in a particular language is run with a given set of data the result produced is uniquely determined, so that it may be re-run any number of times, always giving the same result. There are two exceptions to this rule. Firstly, a hardware fault may lead to incorrect results, and secondly the program may use the value of a variable which it has not previously set. Clearly, only the second exception is relevant to a theory of programming languages.

An analogous situation exists in numerical work where, in some circumstances, a division by zero or an overflow may occur and give a spurious result without any warning to the user. We do not make special provision for these cases in numerical analysis. Instead, we regard them as due to a defect in the system, and seek to modify it so that the user will be warned that his results may be incorrect. Similarly, in constructing a theory of programming languages, we will not make special provision for the case where an unset variable has been used, but will regard the result of the computation as being undefined. If existing systems do not give a diagnostic for this error, the fault lies rather with the systems than with the theory. Therefore this theory must meet—

Criterion 1. The result of a computation in a particular language is completely determined by the program and data comprising the computation.

* *Department of Computing Science, University of Adelaide, South Australia.* *Manuscript received May, 1967.*

In set theoretic terms, this suggests that a programming language may be regarded as a function mapping the ordered pair (program, data) into the element 'result'. McCarthy (1963) used this idea when he described the operation of an ALGOL program π on a set of variables combined in a state vector ξ in the equation $\xi' = \text{algol}(\pi, \xi)$. In the same way, we may consider the function L to be a model of a language where

$$r = L(p, d)$$

and r is the result of a computation having p as program and d as data.

Before defining this model formally we must look at the sets from which r, p and d are drawn. First of all, we may assume, without any loss of generality, that they all come from a single set S, for if they come from distinct sets we simply take S to be their union.

Secondly, we may assume that S is countable, since in practice it can never be non-denumerable. For example, in the case of a language like FORTRAN where r, p and d are symbol strings, we know that the set of finite strings on a finite alphabet is denumerable. Alternatively, we might regard r, p and d as being states of a computer store. But in practice the number of possible states is always finite.

At this stage it is best not to place any other restrictions on S, since they may have the effect of limiting the generality of the model unnecessarily.

Definition 1. Let S be a countable set. Then any function L which maps a subset of $S \times S$ into S is a **type-A language.** In the relation

$$r = L(p, d)$$

r, p and d are respectively the **result, program** and **data** of the **computation** (p, d).

In the sections which follow, as we discuss various conditions which must be met by a theory of programming languages, we shall introduce two other similar models. However, it would be well to meet one objection immediately, namely, that these models are too simple to do justice to the sequential character of actual computation.

We make three points in reply:—

(1) The models do not exclude the possibility that the result of a computation may be obtained by applying a series of elementary operators. This may be made explicit in specifying the function L.

(2) By using these models we will be able to define a number of important terms such as "translator", "semantics" and "equivalence of programs". These definitions would be made unnecessarily complex and obscure if the sequential nature of computation were recognised explicitly at this stage.

(3) There has, in the past, been too much emphasis on the sequence of operations. As more use is made of multiprocessor machines in which different parts of the same program are executed simultaneously by different processors, we will have to get away from the idea that a computation is effected by applying certain operators in a fixed sequence.

2. Simulation and Translation

By a **translator** we understand any program which takes a program in one language as its data, and produces an equivalent program in another language as its result. We class compilers, assemblers and loaders as translators.

A **simulator** is any program which allows a computer to execute a program in some language other than its own machine language without first translating it into machine language. Thus the term includes interpretive systems, microprogram control units and emulators. In the last two we regard the microprogram language as "machine language" and the microprogram, which may be held in a read-only store, as the simulator.

Criterion 2. A theory of programming languages must be able to represent the simulation and translation processes, as well as more complex processes in which several simulators and translators are combined.

Faced with this condition, the type-A model immediately runs into serious difficulties because it implies a clear and permanent distinction between program and data. Such a distinction cannot be made in the case of a simulator. Consider, for example, the case of an interpretive system where there are three elements present: the interpreting system s, the interpreted program p, and the data d. While s is clearly program, and d data, p may be validly regarded either as data (for example, when one is writing the interpreter) or as program (when one is using the system to perform a calculation). The same trouble occurs in representing a computer with microprogrammed control, or in the simulation or emulation of one computer by another. In each case we have an entity involved which is program in one context and data in another, so that it cannot adequately be represented in the type-A model.

Theory of Programming Languages

To avoid this difficulty, we introduce a model in which the distinction between program and data is one of degree rather than of kind. This is achieved by representing a programming language as a function of one argument only. The program (or levels of program) and data will be combined by means of a suitable operator to give this argument. Taking the operator to be *, we would represent the example of the previous paragraph by

$$r = L(s * p * d).$$

If the argument is written as $(s * p) * (d)$ then p is regarded as program; in $(s) * (p * d)$ it is regarded as data.[1]

Before giving a formal definition of the new model, it is necessary to discuss the order in which the data and various levels of program are combined in a computation. There does not seem to be any rule which could be applied mechanically, but the following informal rule appears to be adequate: program and data

[1] *We assume here that the operator * is associative. This will be made explicit in definition 2.*

elements must be combined in such an order that the meaning of any element is determined only by the elements on its left.

In the interpretive system $L(s * p * d)$ the meaning of the interpreter depends on neither p nor d, so that it may appear in the left-most position. The significance of p is determined by the program s which interprets it, so that it must be placed to the right of s. And suppose the data contained the string "123, 456, 000". This might represent, among other things, three decimal numbers, three octal numbers, or a single large number divided into triplets for legibility. The particular interpretation depends on s and p, so that d must appear to the right of these. The argument is therefore uniquely determined to be $s * p * d$.

The various levels of program in the interpretive system do not lose their identity as a result of being adjoined. Thus the factorization of $q = p_1 * p_2 * \ldots * p_n * d$ must be unique. Suppose, for example, $p * d$ is implemented by concatenating the three strings p, e and d, where e is an end-of-file marker not used in p or d. Then $p * d$ is a single string but yet is uniquely decomposable.

Definition 2. Let S be a non-empty set and * an operator, such that

(1) S is countable;
(2) if $x, y \epsilon S$ then $x * y \epsilon S$;
(3) $(x * y) * z = x * (y * z)$; and
(4) If $x \epsilon S$ then either x is prime, or is uniquely expressible as finite a product of primes
$$x = x_1 * x_2 * \ldots * x_n, \quad n > 1,$$
where an element $p \epsilon S$ is said to be **prime** if it cannot be expressed in the form $p = p_1 * p_2$.

Then S is called a **text-set** and its elements **texts.**

The text-set is a much less general concept than the "countable set, S" of Definition 1. However, the loss of generality is not significant since Definition 2 merely expresses in algebraic form the fact that several files or records may be adjoined without losing their identity.

At this point we introduce a useful notation.

Definition 3. Let a and b be elements of S. Then the relation
$$a \simeq b$$
holds if and only if either

(1) a and b are both defined and $a = b$, or
(2) a and b are both undefined.

It is clear that $\simeq$ is reflexive, symmetric and transitive, and so is an equivalence relation.

In the next definition we set out a second language model intended to meet Criterion 2.

Definition 4. Let S be a text-set. Then any function which maps a subset of non-prime elements of S into S is a **type-B language.** Furthermore, if L is a type-B language and
$$r \simeq L(p * d)$$
for $p, d, r \epsilon S$, we say that r is the **result** of the **computation** $p * d$, and that p is **a program** and d **a datum** of this computation.

The foregoing discussion of simulation may now be summed up concisely:—

Definition 5. The program s is a **simulator** for language L in language M if and only if
$$L(q) \simeq M(s * q) \quad \text{for all } q \epsilon S.$$

As an example, consider the emulation of an IBM 1401 on a System 360 machine. In this case, L would denote the 1401 machine language, and p and d the program and data in that language (where $q = p * d$). M would denote the language in which the System 360 microprograms are written, and s the microprogrammed emulator.

At the beginning of this section we described a translator as producing, from a program in one language, an equivalent program in another. Equivalence of two programs implies that they always produce the same result (or no result at all) from the same data. In terms of the type-B model this leads to

Definition 6. The programs a and b in languages L and M respectively are equivalent if and only if
$$L(a * d) \simeq M(b * d) \quad \text{for all } d \epsilon S.$$

Now suppose that t is a translator from language L to M, written in N. If p is a program in language L, then $N(t * p)$ is the corresponding program in M. Thus our initial description of "translator" becomes:—

Definition 7. A program t is a **translator** from L to M in language N if and only if
$$L(p * d) \simeq M(N(t * p) * d) \quad \text{for all } p, d \epsilon S \ldots \qquad \text{(a)}$$

Notice that the nesting of $N(t * p)$ in (a) implies that $p_1 = N(t * p)$ is evaluated before $M(p_1 * d)$. This corresponds to the distinction commonly made between "compile time" and "run time" operations.

Suppose a compiler is written for the CDC 6600 in ASCENT to translate FORTRAN into machine code. Then (a) represents the compilation and execution of a FORTRAN program, where L, M and N denote FORTRAN, 6000 Series machine code and ASCENT respectively. t is the compiler, p the FORTRAN source program, and p_1 the corresponding object program.

Criterion 2 requires that combinations of translators and simulators, such as occur in a bootstrapping process for example, should be representable within the theory. In the example which follows we shall see that the apparatus we have already set up is sufficient to solve a bootstrapping problem, and to prove the correctness of the solution.

The language F (FORTRAN) has already been bootstrapped into a computer (machine language X) to give an F to X compiler, t_1, in X. In the process of doing this, a compiler t_2 was written in F to translate F into an intermediate language I. Show that an F to Y compiler in Y may be prepared for a computer (machine language Y) which is not yet available, by using an I to Y translator (t_3) in language Y and a simulator (s_1) of Y in X.

The translators and simulator which are available satisfy the following relations[1]:

$$F(p * d) \simeq X(X(t_1 * p) * d) \qquad \text{(a)}$$
$$F(p * d) \simeq I(F(t_2 * p) * d) \qquad \text{(b)}$$
$$I(p * d) \simeq Y(Y(t_3 * p) * d) \qquad \text{(c)}$$
$$Y(p * d) \simeq X(s_1 * p * d) \qquad \text{(d)}$$

[1] *We will omit the clause "for all p, dεS", which is taken to follow all relations.*

Then the required bootstrapping process is represented by a transformation of a FORTRAN computation $F(p * d)$ into an expression in which the only function letter is Y.

$$
\begin{aligned}
&F(p * d)\\
&\quad \simeq I(F(t_2 * p) * d) && \text{from (b),}\\
&\quad \simeq I(X(X(t_1 * t_2) * p) * d) && \text{from (a),}\\
&\quad \simeq I(X(t_4 * p) * d) && \text{where } t_4 = X(t_1 * t_2). \quad \text{(e)}\\
&\quad \simeq I(F(t_2 * p) * d) && \text{from (b),}\\
&\quad \simeq I(I(X(t_4 * t_2) * p) * d) && \text{from (e),}\\
&\quad \simeq I(I(t_5 * p) * d) && \text{where } t_5 = X(t_4 * t_2),\\
&\quad \simeq I(Y(Y(t_3 * t_5) * p) * d) && \text{from (c),}\\
&\quad \simeq I(Y(X(s_1 * t_3 * t_5) * p) * d) && \text{from (d)}\\
&\quad \simeq I(Y(t_6 * p) * d) && \text{where } t_6 = X(s_1 * t_3 * t_5),\\
&\quad \simeq Y(Y(t_3 * Y(t_6 * p))) * d) && \text{from (c)} \qquad \text{(f)}
\end{aligned}
$$

The programs t_1, t_2, t_3 and s_1 may be regarded as constants in the sense that they are available at the time at which the bootstrapping operation is performed. Also, since a computer is available which executes language X, any appearance of X with a constant argument may be replaced by a single constant. Thus the replacement of $X(t_1 * t_2)$ by t_4 in the third step above indicates that t_1 has been used to compile t_2 from FORTRAN into X, giving an F to I compiler in X — equation (e).

The compiler just produced is again applied to t_2 to give t_5. From the expression in which t_5 first appears it may be seen that it is a FORTRAN to I compiler in I. t_5 is then translated by means of t_3 to give an F to I compiler in Y. Notice that $Y(t_3 * t_5)$ cannot be evaluated immediately, since there is no machine available to execute Y. However, an application of the simulator s_1 transforms this into $X(s_1 * t_3 * t_5)$ which may be evaluated.

The FORTRAN to I compiler in Y may now be coupled with an I to Y translator in Y (program t_3) to give a two-pass FORTRAN to Y compiler, as shown in (f). The two passes might be linked manually to give a single pass compiler by producing a program t_7 which satisfies

$$Y(t_7 * q) \simeq Y(t_3 * Y(t_6 * q)). \qquad \text{(g)}$$

Then

$$
\begin{aligned}
&F(p * d)\\
&\quad \simeq Y(Y(t_3 * Y(t_6 * p))) * d) && \text{from (f),}\\
&\quad \simeq Y(Y(t_7 * p) * d) && \text{from (g).}
\end{aligned}
$$

Thus t_7, which is obtained simply by sending the result from t_6 as data to t_3, is the required F to Y translator in Y.

3. Syntax and Semantics

> **Criterion 3.** Any theory of programming languages must define the terms "syntax", "semantics" and "pragmatics". In addition it must provide means of formally specifying the syntax and semantics of particular languages.

The distinctions between syntax, semantics and pragmatics were first made by Morris (1938) in connection with what he called "semiotics", the science of signs. Here, we are concerned only with these terms as they are used in the computing community. The meanings of the first two are well illustrated in the ALGOL Report (1963) where each new feature of the language is introduced with a set of formulae in Backus–Naur form specifying the grammar of the feature. Taken together, these formulae enable one to decide whether any program is syntactically correct, and, if it is, to parse it. Following the syntax is a verbal statement of the semantics of the feature from which, hopefully, one can deduce the effects of any program which uses it.

We consider first how "semantics" may be defined. In the relation $r = L(p * d)$ if L and p are held constant r is a function of d. We call this "the function defined by p". In the terminology of the previous paragraph, the "effect" of the program p is to compute this function, so that we may say that the semantics of a language is a rule or set of rules which specify the function defined be every program in the language. This definition may be translated directly into mathematical terms.

Definition 8. Let L be a type-B language. Then

(1) every program, p, **defines the function** f_p given by $f_p(d) \simeq L(p * d)$ for all d; and
(2) the **semantics,** $\mathcal{L}$, of L is the function given by $\mathcal{L}(p) = f_p$ for all $p \epsilon S$.

The semantic function, $\mathcal{L}$, has S as its domain. It maps every program in S into the function defined by that program. Now it may happen that for some text $p \epsilon S$, $L(p * d)$ is not defined for any $d \epsilon S$.[1] At the risk of some abuse of language, we still call p a program, and say that $\mathcal{L}(p) = Z$, where Z is the **null function** for which $Z(d)$ is always undefined.

In the following theorems, which indicate the relation between a language and its semantics, we use the notation $[\mathcal{L}(p)](d)$ to denote the application of the function $\mathcal{L}(p)$ to d.

Theorem 1. Every type-B language L defines a unique semantics $\mathcal{L}$. Furthermore, if $a * b = c * d$,

$$[\mathcal{L}(a)](b) \simeq [\mathcal{L}(c)](d).$$

The uniqueness of $\mathcal{L}$ is evident from Definition 8. Also $[\mathcal{L}(a)](b) \simeq L(a * b)$, so that $a * b = c * d$ implies $[\mathcal{L}(a)](b) \simeq [\mathcal{L}(c)](d)$ as stated.

Theorem 2. Every function $\mathcal{L}$ which maps of S into a subset of the unary functions on S, and which satisfies $[\mathcal{L}(a)](b) \simeq [\mathcal{L}(c)](d)$ whenever $a * b = c * d$, is the semantics of exactly one type-B language.

Let P be the set of prime elements of S. Define the function L by

$$
\begin{aligned}
&L(q) \text{ is undefined if } q \epsilon P;\\
&L(q) \simeq [\mathcal{L}(p)](d) \quad \text{otherwise,}
\end{aligned}
$$

where $q = p * d$, and $p \epsilon P$. L maps a subset of S–P into S, and is therefore a type-B language by Definition 4, so that there is at least one language having $\mathcal{L}$ as semantics. Suppose that there is another, L'. By Definition 4, $L'(q)$ is undefined if $q \epsilon P$, and, by Definition 8,

$$L'(a * b) \simeq [\mathcal{L}(a)](b) \simeq [\mathcal{L}(p)](d) \simeq L(p * d),$$

where $p * d = a * b$ and $p \epsilon P$. Thus $L' = L$.

We have already noticed that syntax serves the double purpose of allowing one to decide the syntactic correct-

[1] *In practice this would mean that the program is either rejected by the compiler or always goes into a loop.*

ness of a program, and, if it is correct, to parse it. In order to arrive at a definition of syntax we will examine the first of these two functions.

The errors which result in a program defining the null function may be divided into three classes.

(1) Errors of syntax, for example, missing commas or parentheses.
(2) Errors which will be picked up by the compiler, but which cannot be specified syntactically — at least by using Backus–Naur form. Double definition of an identifier is an error of this type.
(3) Errors which will pass the compiler. For example, few ALGOL compilers check to see that the first statement of the program does not have the form

A: **go to** A.

We see that the set of programs which define non-null functions — "good programs" in ordinary terminology — is a subset of the set of syntactically correct programs.

Definition 9. The **semantic domain,** $\delta(L)$, of a type-B language is given by

$$\delta(L) = \{p \mid \mathcal{L}(p) \neq Z\}.$$

$\delta(L)$ is the set of all programs which give a defined result for at least one datum. Its complement, $S - \delta(L)$, will consist of all programs having one or more of the errors mentioned above.

In order to introduce the idea of syntax we must define "grammar" and "metalanguage". To avoid placing restrictions on the set S this will be done in such a way that a grammar simply specifies a set of syntactically correct programs, without any implication of syntactic structure.

Definition 10. Let G be a set of elements, which we will call **grammars.** Then any function, Γ, which maps G into the set of all subsets of S, is a **metalanguage.**

As an example, consider the case where S is the set of strings on an alphabet A, and Γ is the metalanguage, Backus–Naur form. Then G is a subset of the sets of strings on $A \cup \{::=, <, >, |\}$, namely the set of all such strings conforming to BNF. For an arbitrary grammar $g \epsilon G$, $\Gamma(g)$ will be the set of all strings in S which may be generated from g.

Definition 11. Any grammar $g \epsilon G$ is a **syntax** of L in the metalanguage Γ if

$$\delta(L) \subset \Gamma(g).$$

This definition is much less specific than might have been desired. Nevertheless, it conveys one of the functions of syntax accurately, namely, its use in detecting a class of program errors, and it does not exclude the other functions of syntax analysis. In order to give a full definition of "syntax", including its connotation of program structure, we would have to introduce more structure into our data set S, or, in other words, make it less general. We have seen that Backus–Naur form is a metalanguage according to Definition 10, and the same is true of the various other formal grammar systems in use, including those for which syntax analysers have been implemented. In other words, Definitions 10 and 11 are the links which connect this theory to the theory of formal languages, and allow results of the latter to be taken over.

Since it is easier to pass from the general to the particular than to go in the reverse direction, it is preferable to develop the theory in its present form first, and attempt to introduce more complete and specialized definitions later. For the same reason we must be content to do without a definition of "pragmatics" at this stage. This term refers to the relations between the language and its user — and more particularly its processor. For reasons that were given in Section 1 the concept of sequential operation has been excluded from the models, so that the idea of a processor does not arise. Nevertheless, just as we are free to postulate more structure in S at a later stage, so we are free to define the language function L in terms of a processor.

We conclude this section with a brief study of the problems of specifying semantics. In principle, one may specify the semantics of a language by specifying either of the functions L or $\mathcal{L}$. But how may this be done?

One consequence of Definition 8 is that we may regard a programming language as a device for defining functions. Now suppose that B is some language with known semantics[1] and that s is a program in B which defines the function L. Then we have

$$L(q) \simeq B(s * q) \quad \text{for all } q \epsilon S,$$

so that, by Definition 5, s is an interpreter for L in B.

The alternative of specifying $\mathcal{L}$ is less straightforward. One approach is to say that if the translator t in language C is applied to a program p in L, the result, $C(t * p)$, defines the same function in another language D as p does in L. That is,

$$\mathcal{L}(p) = \mathcal{D}(C(t * p)) \quad \text{for all } p \epsilon S \qquad \text{(a)}$$

Both sides of this equation are unary functions on a subset of S. From definition 8 we see that the result of applying an expression of the form $\mathcal{L}(p)$ to d is to give $L(p * d)$. Thus, if both sides of (a) are applied to an arbitrary $d \epsilon S$ we have

$$L(p * d) \simeq D(C(t * p) * d) \text{ for all } p, d \epsilon S.$$

t is therefore a translator from L to D in C.

In addition to using a simulator or translator to define the semantics of a language, we may use configurations involving several simulators or translators. The problem then relates to the problem of bootstrapping, which was discussed in Section 2, with this difference. Instead of attempting to minimise the amount of software preparation, the aim will be to produce a system whose components perform distinct, easily intelligible functions. In this way the human user will gain the maximum advantage from the semantic specification.

We see from this discussion that the theory proposed in this paper does not add anything to the known techniques of specifying semantics. However, it does provide a conceptual framework which might be used to unify more detailed studies.

4. Computability

A number of devices have been used by mathematicians to define a set of computable functions. The fact that in each case the same set of functions has resulted has strongly reinforced the claims that the various devices contain everything necessary to perform any computation. Thus it is commonly held that a function is computable in practice if and only if it is

[1] *This does not involve a vicious circle, for it is possible to define the semantics of a very simple language in English without ambiguity. Two such languages are discussed in Sanderson (1966).*

a computable partial function in the sense of, say, Turing.[1]

Criterion 4. A theory of programming languages must agree with the theory of computability.

Suppose we call the set of computable partial functions C.[2] Then it is required that a function should be definable by a program if and only if it is in C. At this point it is convenient to introduce

Definition 12. The **semantic range,** $\rho(L)$, of a type-B language L is given by

$$\rho(L) = \mathscr{L}(S).$$

Thus $\rho(L)$ is the set of function definable by programs in L. Notice that it is not quite the complementary concept to semantic domain, for whereas $\rho(L)$ is the range of $\mathscr{L}$, $\delta(L)$ is not the domain of $\mathscr{L}$.

In the new terminology, we ask that $\rho(L) \subset C$ for every L. The following theorem shows that type-B languages do not meet this condition.

Theorem 3. Every unary function on S is defined by a program in some type-B language.

Let f be any unary function on S, and e any element, $e \epsilon S$. Define the function L as follows: if q has the form $q = e * d$, $L(q) \simeq f(d)$; otherwise $L(q)$ is undefined. Then the program e in language L defines the function f.

At first sight it appears that we might correct this situation by saying that L is, say, a type-C language only if $\rho(L) \subset C$. But this condition is not strong enough, for theoretically it is quite possible for a language which satisfies this condition to require the evaluation of a non-computable function in its compilation. This may happen in the commonly occurring case where S contains a denumerable infinity of primes.

Theorem 4. Suppose that S contains an infinity of primes, and that M is a type-B language satisfying $\rho(M) = C$. Then there is a type-B language L, $\rho(L) = C$, such that, for no $t \epsilon S$,

$$L(p * d) \simeq M(M(t * p) * d) \quad \text{for all } p, d \epsilon S.$$[3]

Let the set of primes be P. Since P and C are both infinite, the set of functions from P onto C is non-denumerable. Corresponding to every such function g we may define a type-B language L in such a way that different functions give different languages:—

$$L(q) \text{ is undefined if } q \epsilon P;$$
$$L(q) \simeq [g(p)](d) \quad \text{otherwise,}$$

where $q = p * d$, and $p \epsilon P$. In the language L each prime program p defines the function $g(p)$, so that

$$\rho(L) = g(P) = C.$$

Thus there is a non-denumerable infinity of type-B languages satisfying $\rho(L) = C$. But since $t \epsilon S$, and S is denumerable, the relation

$$L(p * d) \simeq M(M(t * p) * d) \quad \text{for all } p, d \epsilon S,$$

can be satisfied only by a denumerable subset of these languages, and the theorem is proved.

To overcome the difficulties posed by Theorems 3 and 4 we must define a new language model.

Definition 13. Let S be a text-set in which the operation $*$ is computable.[1] Then any computable function which maps a subset of non-prime elements of S into S is a **computable language.**

Since every computable language is also a type-B language, it is not necessary to re-draft the Definitions 5 to 12. In fact, the entire discussion of Sections 2 and 3 applies to computable languages, except that Theorem 2 is no longer true if we replace "type-B language" by "computable language" in its statement.

Theorem 5. If L is a computable language, $\rho(L) \subset C$.

An arbitrary program, p, defines the function f_p given by $f_p(d) \simeq L(p * d)$ for all $d \epsilon S$. The operator $*$ and the function L are computable, and therefore their composition, f_p, is computable. Thus $\rho(L) \subset C$.

Theorem 6. If L and M are computable languages, $\rho(M) = C$ and $\rho(L) \subset C$, then there is a translator $t \epsilon S$ from L to M in M.

Since $\rho(M)$ contains every computable function and L is computable, there is a program s such that

$$L(q) \simeq M(s * q) \quad \text{for all } q \epsilon S.$$

Thus s is a simulator for L in M. We go on to convert this to a (degenerate) translator. The function which maps every p into $s * p$ is computable, since it is obtained by composition of the constant function whose value is s and the $*$ operator, both of which are computable. Let t define this function in M.

Then $M(t * p) \simeq s * p$, and

$$M(M(t * p) * d) \simeq M(s * p * d)$$
$$\simeq L(p * d) \quad \text{for all } p, d \epsilon S,$$

so that t is the required translator.

These two theorems show that a computable language can never define a non-computable function and that it does not require one for its compilation. This leads to one of the questions which will be discussed in the next section: Is every practical language a computable language?

Concluding Remarks

In the introduction we saw that the conventional method of testing a theory by experimental verification may not be appropriate for a theory of programming languages. Since a programming language is purely artificial it may be possible to decide the correctness of a mathematical model directly, without resorting to experiment. Even if this can be done, however, it is still necessary to look for the most accurate and useful model. We therefore ask, first of all: are the languages used in practice computable languages? Secondly, having

[1] *The theory of computability is treated in Davis* (1958).

[2] *Computability is usually thought of as applying to functions on the set of positive integers. However, it is reasonable to define a mapping from the set of symbol strings appearing on the tape of a Turing machine onto the set S used in this paper, instead of the set of positive integers. If this is done, the Turing machine defines computability of functions on S; it is this set of computable partial functions that we call C.*

[3] *Despite an apparent similarity, Theorems 4 and 6 have no connection with the Theorem 3.2 of Ginsburg and Rose (1963), which may be interpreted as asserting that "there is no finite procedure for deciding whether or not one Algol-like language can be translated into another in a non-trivial way" (ibid. p. 40). An Algol-like language is, of course, defined in a completely different way from a type-B or a computable language.*

[1] *This operation is binary, whereas we have previously considered computability as applying only to unary functions. We shall assume that the term can be extended to binary functions, that * is computable, that every constant function is computable, and that the composition of computable functions is computable. These assumptions will always be fulfilled in practice, although it would take us too far afield to justify them theoretically here.*

shown that the model is adequate in principle, we attempt to assess its practical adequacy.

In order to be a computable language, a programming language must meet three conditions.

(1) The combined program and data, and also the result, must be texts in the sense of Definition 2. That is to say, they must be composed of files which can be adjoined without losing their identity. Now it is common practice to load the various program and data files of a computation onto a magnetic tape or an equivalent bulk storage medium before execution. Similarly, the results of the computation will be written on tape by the operating system before being transferred to the various output devices involved. Here we see the texts $p * d$ and r realized in physical form. The fact that this can be done guarantees that the first condition is met.

(2) The second condition is that the result of a computation is uniquely determined by its program and data. The exceptions to this rule, involving machine and program faults, were discussed in Section 1.

(3) Thirdly, the language function, L, must be computable. If the common view is true, that only computable functions are effectively calculable, this condition must always be satisfied.

We conclude that the languages used in practice are computable languages, so that any theory which can be built on the computable language model will be applicable to them. Can we make any assessment of the value of such a theory at this stage? One requirement we can place on the theory is that it should fit harmoniously with other, related theories.

In the case of the theory of formal languages this presents no difficulty. The definitions of "metalanguage" and "grammar" given in Section 3 are sufficiently general to ensure that the corresponding concepts in the theory of formal languages are included as special cases. Thus it will be possible to take over results of that theory concerning matters such as ambiguity and syntactic analysis, and, of course, to give a satisfactory treatment of syntax.

Only one aspect of the theory of computability was considered in Section 4. Quite extensive translation may be required before other detailed results can be taken over. For example, in place of a theorem concerning the halting problem for Turing machines, one would have a theorem about the recursiveness of the set $S - \delta(L)$.

From the point of view adopted in this paper, the theory of mathematical machines would be considered rather as the theory of language processors. One method which may be used to define a language function L is to set up a processor which executes programs in L. Work along these lines has been reported in Elgot and Robinson (1964), for example, and as this area is explored we may expect to see a theory of pragmatics elaborated.

References

1. Davis, M. (1958). Computability and unsolvability. Mc-Graw-Hill, New York.
2. Elgot, C. C. and Robinson, A. (1964). Random-access stored-program machines, an approach to programming languages. J. ACM *11*, 4, 365–399.
3. Ginsburg, S. and Rose, G. F. (1963). Some recursively unsolvable problems in ALGOL-like languages. J. ACM *10*, 1, 29–47.
4. McCarthy, J. (1963). Towards a mathematical science of computation. Proc. IFIP Congress 62, 21–28. North-Holland Co., Amsterdam.
5. Morris, C. (1938). Foundations of the theory of signs. In *International Encyclopedia of Unified Science*, *1*, 2. University of Chicago Press, Chicago.
6. Naur, P. (1963). Revised report on the algorithmic language ALGOL 60. Comm. ACM *6*, *1*, 1–17.
7. Sanderson, J. G. (1966). A contribution to the theory of programming languages. Ph.D. Thesis. University of Adelaide, South Australia.

On Intelligence, Intelligent Automata And Robots

By D. L. Overheu *

Summary

It is postulated that the essence of 'intelligence' is the ability to select hypotheses about an environment which give a high probability for predicting future events within the environment. Deductive and inductive reasoning are then seen only as tools in the application of intelligence and not specifically as evidence of intelligence. On this basis, it is possible to postulate the existence of active finite state probabalistic automata as intelligent mechanisms inter-acting with their environment. For historical reasons it is proposed that these machines should be called robots. The discussion is intended to be informal but definitive, and is aimed at encouraging the use of the term 'robot' for a particular class of intelligent machines before its misuse relegates it to describing certain automata which perform physical activities.

Introduction

What we seek to determine is whether or not there exists a class of mechanical devices or 'machines' which are sufficiently different within the class of mechanical devices called 'automata' to justify calling these devices by a new name. For historical reasons we would use the term 'robot' for such devices. If, for some reason, a sufficient difference cannot be found then we must conclude that machines with the properties sought are not to be found.

The term 'automaton' (plural 'automata' or 'automatons') derives from a Greek neuter adjective meaning 'acting of itself'. In English it means variously 'thing endued with spontaneous motion; living being viewed materially; piece of mechanism with concealed motive power; living being whose actions are involuntary or without active intelligence'. In computing theory an 'automaton' is usually somewhat more specifically a mechanical device (i.e., piece of mechanism) for performing certain operations on abstract entities to produce other abstract entities, although there is no reason for excluding concrete entities from consideration. In particular, the possible steps available to an automaton in performing such operations are known a priori even though the operations performed may continue indefinitely, or be selected according to some probability rule, or terminate after a finite number of steps. In general, however, automata theories are not concerned so much with useful activities but with existence of classes of automata.

In the Polish language we find the word 'robotnik' for 'workman' and a similar word 'rabotnik' for 'worker' or 'workman' is found in Russian. Carol Capek (1923) used the term 'robot' in his play 'R.U.R.' as a name for apparently intelligent but, as it turned out, not so obedient synthetically constructed human-like slaves. In English it means variously 'an apparently human automaton; an intelligent and obedient but impersonal machine; a machine-like person'. It has also been unfortunately used to mean 'automatic traffic signals' and 'flying bomb'. Because the term 'robot' exists with some sort of meaning in the body of the English language, there is no particular reason for supposing that we can define a class of mechanical devices to which it would be reasonable to give the name 'robot' within the class of automata. It may be that the term relates to a physically unrealisable mechanism no matter how clever we become. However, it is clear that the distinction between an 'automaton' and a 'robot' in English is connected with the rather vague concept of 'intelligence'. That is 'automata' are 'unintelligent' mechanical devices whereas 'robots' are 'intelligent' mechanical devices. Such things as 'automatic traffic signals' and 'flying bombs' (at least as we understand such devices at the present time) must obviously be rejected from being accorded the name 'robot'. It is thus necessary to understand what is meant by 'intelligence' if we are to find a meaningful distinction. However, it will be intuitively felt that mere possession of 'intelligence' by a mechanical device may not be sufficient to distinguish it within the general class of automata. There are, for example, various programs for digital computers purporting to demonstrate certain human activities which might be thought 'intelligent', although they are now more usually referred to as examples of 'artificial intelligence'. The body of opinion tends to avoid at this time a definition of 'intelligence'. If the historical connotation of 'robot' is to be kept we must consider, therefore, as well 'active' mechanical devices. By this we will mean 'devices which can perform operations on concrete objects, as distinct from 'passive' mechanical devices which we understand as 'devices for performing operations on abstract entities'. The term 'robot' has tended to be associated more with the former than with the latter. If it is possible to find a suitable definition for the term 'robot', then it will be argued that such devices exist. In any case, we must conclude

* *Department of Defence, Canberra.* *Manuscript received June, 1967.*

that those properties we seek, and which lead us to believe that 'robots' might exist, are demonstrated by particular applications of automata, if a plausible definition can be found for 'active, intelligent mechanical devices'.

Automata

It is somewhat easier to describe the general rules which assign certain mechanical devices to the set of objects which we call 'automata' than to attempt a formal definition in the first instant, although this term is by no means used consistently. An 'automaton' is characterised by the following—

(i) it is a mechanical device;
(ii) it has a means for sensing its environment;
(iii) it has, possibly, a means for storing objects or representations of objects selected from its environment;
(iv) it has, possibly, a means for altering its environment;
(v) it has a means for controlling its activities.

As a concrete example of a very simple automaton, consider a moving conveyor belt carrying objects of a certain size and weight. If an object whose weight is less than a predetermined value is detected, it is removed from the belt. As a quasi-abstract example, consider an electrical signal having some sequence of positive and negative fluctuations. If a particular predetermined pattern of electrical signals is detected by a sensing device, an alarm is sounded, otherwise no alarm occurs. Finally, consider the abstract concept of a tape of indefinite length marked into squares in each of which is written a symbol '1' or '0'. If a particular sequence of ones and zeros occurs the device sensing the tape stops it at the end of the sequence, otherwise it never stops.

These examples are particular instances of what are usually called 'strictly finite state' automata. They can sense their environment (i.e., the objects presented to them). They cannot alter their environment but some can answer 'yes' or 'no' to the question 'is this environment acceptable by the device'. They cannot store objects relating to their environment but they all must have a control mechanism for what they are able to perform.

If we accept the idea that any concrete example of an automaton such as the conveyor belt weight selecting device can be represented abstractly by a tape automaton, then we can give a more formal mathematical definition for a general automaton after the style of Davis (1958) or Fischer (1966) as follows.

D1. An 'internal configuration' or 'internal state' is 'a condition of stability, i.e., when no action is taking place in the mechanical device'. Each internal state is designated by a symbol 'q_i' ($i = 0, 1, 2 \ldots$).

D2. An 'alphabet' is 'a finite set of graphic signs (symbols)'.

We designate the i^{th} symbol of an alphabet by 'S_i', and it is assumed that symbols can be recorded in some way on a tape. The i^{th} symbol of an alphabet recorded on the k^{th} tape for a system with more than one tape is designated by '$S^k{}_i$'. It is further assumed that the mechanical system has n scanning heads which can scan symbols on all tapes in the system simultaneously. In order to describe what action is to be taken when the device is in state q_i scanning an n-tuple of symbols $S = (S^1{}_{i_1}, S^2{}_{i_2}, \ldots\ldots S^n{}_{i_m})$ on the tapes we have the following,

D3. A 'quadruple' is an expression having one of the following forms—

(i) $q_i\ S\ S^k{}_i\ q_j$ (write S_i on the k^{th} tape)
(ii) $q_i\ S\ R^k\ q_j$ (move the head for the k^{th} tape one place to the right)
(iii) $q_i\ S\ L^k\ q_j$ (move the head for the k^{th} tape one place to the left)

D4. An 'automaton' is 'a finite (non empty) set of quadruples'.

The automaton is 'deterministic' if no two quadruples have the same first internal state symbol and n-tuple, otherwise it is 'non-deterministic'. Thus, the set describing the actions of the automaton also describes the physical structure of the automaton in an abstract sense. The set of quadruples is understood as follows. If the automaton is in internal state q_i and the n heads are scanning an n-tuple $S = (S^1{}_{i_1}, S^2{}_{i_2}, \ldots. S^n{}_{i_m})$ of symbols recorded on the tapes, then it will perform the action appropriate to the third symbol of the quadruple and enter internal state q_j. Next, we have that —

D5. A 'tape expression' is 'a sequence composed entirely of the symbols $S^k{}_i$ for a given tape'.

D6. An 'instantaneous description' or 'present state' is 'a tape expression containing exactly one special symbol 'm_k' not in the alphabet'.

By convention, this indicates that the automaton is scanning the symbol on the k^{th} tape immediately to the right of this special symbol.

D7. A 'total present state' is 'an $(n + 1)$-tuple $(q_i, T_i \ldots . T_n)$ where q_i is an internal state and each T_i is a present state.

The general automaton as defined above may be assumed to have tapes of infinite length, in which case it is called a 'non-finite state' or 'infinite state' automaton otherwise it is a 'finite state' automaton. This will be clear from the fact that both the number of internal states and number of present states is finite so that the number of present total states is finite. In the special case when only one tape of infinite length is used the automaton is a 'Turing machine'. In the case where only the 2nd or 3rd quadruples are allowed and there is one tape of finite length we have the special case of a 'strictly finite state' automaton. By varying the number of tapes and specifying finite or infinite tapes a variety of other special cases can be distinguished. For an example, a 'finite state transducer' has two tapes and allows quadruples (i) and (ii) to apply to the second tape. A 'push down store automaton' with output has three tapes. It allows quadruple (ii) to apply to one tape, quadruples (i) and (ii) to a second tape, and quadruples (i), (ii) and (iii) to a third tape in a particular way. Finally, a non-deterministic automaton is sometimes referred to

as a 'probabilistic automaton' if some probability rule exists for choosing between two quadruples whose first two expressions are the same. Such a rule would be considered to be part of the device's control.

These various types of automata are used to show that certain classes of calculations can be performed and that there exist other classes of calculations which cannot be performed by one type of automata but can be performed by more complex automata. However, no matter how many tapes are used, there are no automata which can perform classes of calculation which cannot be performed by a Turing machine. On the other hand there exists so-called 'well formed' functions (i.e., the function is properly constructed according to the rules for constructing functions) which cannot be evaluated by a Turing machine (Davis, 1958), or any known machine.

In theories of computability it is usual to say that the problem of finding an algorithm for a calculation is either solvable or unsolvable and hence that the function to be evaluated is computable or non-computable. If a function is computable, it can be evaluated by a Turing machine. Correspondingly, if the problem is to determine whether or not certain objects belong to a given set of objects, the problem is said to be decidable or undecidable. Again, in recursive function theory the foregoing types of problems are called recursively solvable or recursively unsolvable.

Some care is needed in interpreting these various concepts. For example, Ginsburg and Rose (1963) have shown that the problem of constructing a translator from one language to another in the case of a certain class of formal languages is recursively unsolvable. This does not mean that in particular cases it cannot be done but only that there is no general algorithm which suits all possible cases. Again, the fact that a supposedly 'well formed' function is non-computable must be interpreted with caution. The rules which determine a 'well formed' condition for expressions say nothing about the interpretation of expressions. The theory of the relationship between symbols and their interpreters (particularly mechanical interpreters) is called 'pragmatics'. It is the view of Gorn (1962) that all instances of antinomies in logic are instances of pragmatic ambiguities. That is to say the interpretation of certain expressions changes according to whether the expression is interpreted in a descriptive sense or a control sense. For example, the Turing machine whose alphabet is (0, 1) and given by

$$q_1\ 0\ 0\ q_1$$
$$q_1\ 1\ 1\ q_1$$

clearly has a non-stopping computation. It is however, a pragmatically ambiguous machine. There is a continual shift in interpretation between the use of the alphabet symbols as descriptions of what is recorded on the tape and as control symbols instructing the machine to write these symbols on the tape. Thus, to say that something cannot be computed does not necessarily mean that it could not be computed in some cases. It may suggest that we have a case of pragmatic ambiguity which can sometimes by resolved by construction of an appropriate interpretation. The instances of non-computability or undecidability in computing theory are concerned mainly with the question of deciding in all possible cases whether or not a given proposition is valid and not with determining what cases will make the proposition valid. Conversely, however, if it is known that a problem is solvable, then there must be an algorithm for evaluating it.

The description of an automaton in the abstract manner of computing theory can be extended to define a 'universal automaton'. This device is an automaton in which at least one tape has a description of the automaton for performing a given calculation and whose control unit can interpret this description. In the case of the 'universal Turing machine', its single tape has both a description of the calculation to be performed and a tape expression, and its control unit can interpret the description on the tape. This automaton has been used as a model for the automatic digital computer in the sense that any computation performable by a computer will be performable by a universal Turing machine. Although it is sometimes said that the steps of a Turing machine are too small for such a purpose, it should not be overlooked that any suitable set of operators can be chosen such that compositions of these operators explain the most complex calculations in a simple manner. It is only required that these operators should be definable by Turing machines or by compositions of conceptually simpler operators which are definable by Turing machines.

Minsky (1961) remarked that examination of computer programs simulating artificial intelligence seem to be meaningless and do not tell one what 'intelligence' is. In a like manner we would not be able to determine what 'intelligence' is simply by an examination of the structure of the human brain. It is therefore not proposed to suggest later a mechanical definition of 'intelligence'. However, the Turing machine equivalence with a digital computer will allow us to assume that computer programs for artificial intelligence have equivalent Turing machines as counterparts.

The properties of self-reproducing machines have recently been studied again by Arbib (1966) who describes a simple self-reproducing automaton using a combination of a Turing machine and a 'constructing machine'. This latter device is an expression of the ability of an automaton to actively alter its immediate environment by following a description of an automaton on the tape of a universal Turing machine. The important properties of self reproduction are the ability to make a copy of both the structure to be copied and the mechanism for making the copy. In a more abstract sense, a universal Turing machine which can replicate its tape is a self-reproducing automaton. However, Arbib's self-reproducing automaton introduces the concept of physical construction as well. Burkes (1963) has proposed that the tape of an automaton should be regarded as a 'passive' automaton and its control unit is an 'active' automaton. In Arbib's case, the constructing mechanism is not part of the control unit but is controlled by the control unit. It seems to be preferable, therefore, to define

a 'passive automaton' as an automaton which cannot alter its physical environment. An 'active automaton' would then be defined as one which is able to alter its physical environment, and these are the definitions suggested earlier. Thus, a Turing machine is a passive automaton and a self-reproducing machine like Arbib's machine is an active automaton. It is then quite clear that any device which is called an 'active automaton' has to interact in some way with the physical world, whereas a 'passive automaton' cannot do so. The suggested definitions for active and passive automata do not prevent these terms being used to describe control units and tapes of automata, and it provides concepts which would seem to be more in accord with the way we think about automata.

Intelligence

It has been said by Minsky (1963) that 'intelligence' seems to denote little more than the complex of performances which we happen to respect but do not understand. Armer (1960) in defence of the 'intelligent machines' concept noted with some degree of scorn that Meszar (1953) would have us believe 'perhaps the most flexible concept is that any mental process which can be adequately reproduced by automatic systems is not thinking'. Clearly, we must avoid this type of automatic redefining of the lower bound below which a behaviour is no longer intelligent. On the other hand we must surely attempt to go further than suggested by Minsky.

As long ago as 1950 Turing suggested an idealised test for the 'intelligent machine' based on a question and answer game in which the human questioner had to determine whether or not the unseen respondent was a machine. Conversely, it may be better to ask an 'intelligent machine' to decide whether a respondent is a human being or another machine. If such a device could solve this decision problem we would presumably have no doubt that it was acting intelligently. Turing's test regards the machine as acting intelligently if the human being cannot tell that he is talking to a machine. In the absence of a strict measure for 'intelligence' there could be some doubt that the human being was acting intelligently, or at least argument about the capacity of particular human beings. The main point at issue is that if two devices are given the same input and they produce the same output in all cases, then they are equivalent. Hence, if one device is a human being, and human beings are intelligent, then the other device must also be intelligent even if it is a machine. However, this tends to demand that the machine should be not less than a genius, whereas we still think of human beings as intelligent even when their capabilities are well below the genius level.

'Intelligence' is said to be 'intellect' or 'understanding' and 'intellect' is the 'faculty of knowing and reasoning'. Unfortunately, if we pursue English explanations of such an abstract concept we are likely to find a circular chain. One approach to a discovery of the concept of 'intelligence' has been discussed by Uttley (1950) on the basis of the concept of 'information'. This latter term has been strictly defined for the purpose of developing a theory about 'information'. We can define first a 'representation' as 'any structure, whether abstract or concrete, of which the features purport to symbolise or correspond in some sense with those of some other structure' (Mackay, 1950). 'Information' is now defined as 'that which adds to a representation'. If also we mean by 'deductive reasoning' the 'proof of the particular from what is generally known' and by 'inductive reasoning' the 'proof of the general from what is known in particular' then Uttley points out that neither process produces new information. He assumes that information is computable and hence deductive and inductive reasoning is realisable mechanically. Uttley concludes, however, that 'intelligence' is associated with the creation of new information and that 'there may be an unknown cause operating in man which we have not, and perhaps cannot, introduce into a machine'.

These views, which give a mystique to 'intelligence', are often hotly contested. We may well feel also that deductive and inductive reasoning are too narrowly defined. It does appear to be the case, however, that if we accept such definitions, then something is left over which can often only be expressed as the creation of new information. The doubt that exists arises because we cannot be absolutely certain that new information has been created and it may only appear to be so. The relevance to the present discussion is clear. If we accept Uttley's thesis, then we can construct machines for performing deductive and inductive reasoning but the 'intelligent machine' does not exist, unless we can demonstrate the existence of a machine which creates information.

The question of the status of deductive and inductive reasoning relative to 'intelligence' is worth exploring. Although it now seems unpopular to suggest that deductive reasoning is a sign of 'intelligence', it is difficult to believe that this was always so or should be so now. Our difficulty is that once something becomes known and understood we feel we must have been incredibly stupid not to have understood this thing before. This view is strengthened if the procedure leading to the result is shown to be algorithmic. Thus, while Newell et al. (1957) sought to find ways to use a computer to simulate human theorem proving in the propositional calculus (an activity which was thought at one time to demonstrate 'intelligence'), Wang (1960) demonstrated an algorithmic method which did not depend on a truth table evaluation procedure. More generally, it is now clear that for those areas of logic which are decidable (i.e., for which the decision problem is solvable), algorithmic theorem proving is possible. Further, Robinson (1965) has developed an algorithmic theorem proving method containing an inference principle which, in the psychological sense, he claims 'condones single inferences which are often beyond the ability of the human to grasp (other than discursively)'. The success of the Wang or Robinson methods depends on the peculiar properties of digital computers for rapid scanning of large amount of data as compared with the human pattern recognition facility which is apparent in the simulation attempt by Newell et al. who were seeking a possible general

computer procedure for deductive reasoning. Clearly, if deductive reasoning were to be admitted as an example of 'intelligence', it would be necessary to distinguish, if possible, that part which might be non-algorithmic, otherwise it would be necessary to recognise that machines have already surpassed human ability in this area.

It is possible that the symbolic integration program described by Slagle (1963) provides an example of a non-deductive reasoning system and thus suggests an approach to a concept of 'intelligence'. Slagle concludes that 'a machine can manifest . . . behaviour which, if performed by people, would be called intelligent'. This program uses algorithmic methods, semi-algorithmic methods (by choice of an integrand transformation which is nearly always appropriate), and 'heuristic' methods. (An 'heuristic' is 'a process that may solve a given problem but offers no guarantee of doing so'—Newell et al. (1957).) It seems that the triology of algorithm, semi-algorithm, and heuristic, is essential to any process which appears to exhibit 'intelligent behaviour' in the sense that the process simulates an activity which may be considered in human being to be 'intelligent'. It is unfortunately, not clear whether this triology is anything more than a form of deductive reasoning. Even assuming that this latter is the case, then the semi-algorithmic and heuristic methods appear to be only a means for selecting a small number of possible solutions from a very large set of possible solutions which could be found by algorithmic methods if it were physically possible. Hence, it is difficult to find a specific example about which it could be said that this demonstrates clearly a non-algorithmic deductive reasoning rather than just the use of an heuristic to solve an algorithmic problem which would be impractical to solve otherwise.

It is tempting to assume that deductive reasoning has algorithmic and non-algorithmic components and to exclude algorithms alone as evidence of intelligent behaviour but we might then be guilty of adopting Meszar's proposal. We cannot possibly determine at this time what processes, which now seem to us to be non-algorithmic and thus possible evidence of human 'intelligence', will later be explained by algorithms. On the other hand, by admitting algorithms into any definition of 'intelligence' we allow many processes to be so described which are clearly not what we would call 'intelligent behaviour'. What we might ask, therefore, is whether the schemes of Newell et al. or Slagle have any other features that are simulating human 'intelligence' or whether they are, in fact, only examples of partial algorithms (i.e., descriptions of an evaluation of partial recursive functions). Even if we accept the hypothesis that these schemes are only simulating deductive reasoning, it is possible that we might find some clear non-algorithmic component in what has been called heuristic processes. Unfortunately, merely because the correct answer cannot always be found, an heuristic process is not evidence that the underlying process is non-algorithmic, although an heuristic process would be a necessary part of any non-algorithmic deductive reasoning. Evidently those people who have studied this problem of simulating aspects of human activity by means of computers, and have employed heuristic processes, also have doubts that their results demonstrate 'intelligence'. It is now more usual to refer to work of this nature as in the field of 'artificial intelligence', i.e., something which seems to simulate what we presently think might be the process of human 'intelligence'. The only area in which a non-algorithmic process might be called deductive reasoning is that of theorem proving in the case of non-decidable problems. It would appear that the success of human intellect in this case is related to the ability to select appropriate particularisations at certain steps in the deduction. The author's belief is that this involves a certain amount of inductive reasoning so that it is very likely one may regard 'pure' deductive reasoning as always basically algorithmic. Further, it has not yet been shown that deductive reasoning does have any special or unusual feature that requires a human being to have a special insight in Uttley's sense. It seems in any case that if we attempt to exclude all processes which seem to be algorithmic in deductive reasoning we will be faced with the Meszar problem. On the other hand, if we accept deductive reasoning as an element in a definition of 'intelligence' then all automata will be 'intelligent machines' because algorithmic processes are a part of deductive reasoning. The considerations on deductive reasoning have lead to an impasse which suggests that we should follow Uttley's view and exclude this form of reasoning from any definition of 'intelligence'.

In a similar way, we may analyse what occurs if we attempt to simulate inductive reasoning. The results of such simulation studies are to be found, for example, in Hunt and Hoveland (1961) on concept learning, or in studies on pattern recognition such as Kovalevsky (1965). Minsky (1961) quotes the work of Solomonoff on 'grammatical induction' (i.e., the discovery of grammars for languages) as among the most interesting in this area. Hunt and Hoveland quote Church's definition of the term 'concept' as follows. 'Any symbol or 'name' can be attached to the members of a set of objects. For any arbitrary object there exists a rule concerning the description of the object, a rule which can be used to decide whether or not the object is a member of the set of objects to which the name applies. The decision rule is the concept of the name, the set of objects is the denotation of the name'. Thus, concept formulation involves an inductive reasoning process in which a concept is to be formed from instances of denotations of its name. In fact, this is precisely what we are attempting in this paper in relation to the name 'robot'. We are attempting to find a rule which will allow us to assign objects in the form of certain machines or mechanical devices to a set of similar objects and we propose giving the name 'robot' to members of this set.

We are now confronted with something of a dilemma for it seems that to claim inductive reasoning is 'unintelligent' implies that this paper could have been written by an automaton, whereas to claim that inductive reasoning is 'intelligent' implies that no

algorithmic processes are involved. There is a parallel in the classical Spartan's dilemma in logic but in our case we escape the dilemma on the plea that we still have to resolve the definition of 'intelligence'. If we look at the processes involved in computer programs for simulation of concept formulation or pattern recognition, we find algorithmic, semi-algorithmic, and heuristic methods again being employed. The only difference is the way in which these methods are used. Hence, we would find it difficult to claim that deductive reasoning was not an example of 'intelligence' but that inductive reasoning was.

Cohen and Nagel (1934) writing on the 'scientific method' note that we establish hypotheses from observation following which we deduce various facts which we then test by observation in order to fortify our belief in the hypotheses. That is to say, scientific method involves inductive reasoning, deductive reasoning and experimentation. They are careful to point out 'that the method of science does not seek to impose the desires and hopes of men upon the flux of things in a capricious manner . . . its successful use depends upon seeking . . . to recognise, as well as take advantage of, the structure which the flux possess'. (ibid p391). Thus they precede Uttley by some distance in suggesting that inductive reasoning does not create new information.

There is, however, one facet of both deductive and inductive reasoning touched on by Cohen and Nagel and also by Wang (1960). There does not seem to be any way for deciding which hypotheses or theorems out of a very large number of possible hypotheses for a theory, or theorems within a theory, that are the best to select. That is to say, how does a human being, for example, select just those things which have greatest value in co-ordinating, explaining, or clarifying some part of human knowledge so that he can predict with greater certainty the happening of future events? Whether this implies creation of new information or not is debatable. New hypotheses about the physical world are often suggested by observation of what must be existing structure and new information is only created if new hypotheses create new additions to representation of structure. It is highly probable that new information is created in this case since a representation exists which did not previously exist. In the case of useful theorems, however, it is difficult to believe that anything new has been created in Uttley's sense. However, there appears to be some facility at work which guides the selection in either case and which uses both deductive and inductive reasoning as tools in making the selection. Minsky (1965) has proposed that this whole process is one of constructing suitable models of the environment. Such models are modified by experience (i.e., by continual comparison with the actual environment) and thus are representations to which new features are continually added. If this is the case, and we must begin with only the means for constructing such models (i.e., deductive and inductive reasoning), then creation of a model is creation of new information.

By assigning deductive and inductive reasoning to the role of tools used to get an end point we remove at once all the contradictions these terms bring if they are retained as elements in 'intelligence'. It seems reasonable, therefore, that we should define 'intelligence' as 'the ability to select or choose those hypotheses and theorems about a given environment which increase the probability of correctly predicting future events within the environment relative to the body of available relevant knowledge about the environment'. This definition has a number of advantages. In relation to Meszar's proposal it does not demand that everything that has intelligence must of necessity be a genius or that only a biological machine can attain such a state. The definition is not inconsistent with Minsky's ideas on models nor Uttley's ideas on creation of new information. Intelligence, by this definition, is a perfectly natural, everyday phenomena which at present is exhibited best by human beings carrying out everyday activities. If the selected hypotheses happen to be tautologies or are selected as such we use deductive reasoning to prove theorems (i.e., other tautologies) and we can predict future events completely but we must still select in some way those theorems to be proved that will be most useful. If the hypotheses are constructed by observation and inductive reasoning, the probability of correctly predicting future events is low or high depending on the soundness of the selection. Again, the definition does not exclude those selections which, in the general opinion, exhibit only a "low" intelligence value and includes those which would be considered to have "high" intelligence value.

It is possible that this definition is too wide for it appears to be only a re-statement of a definition for an heuristic. There is no doubt that heuristic methods are employed in intelligence but through deductive and inductive reasoning. It is the way in which reasoning is applied that is the essence of intelligence not the way in which reasoning is implemented. The definition does exclude much of the work in what is currently called 'artificial intelligence'. For example, we would not agree that Slagle's integration program was an example of intelligent behaviour unless it could be shown that its heuristic procedures lead to the selection of hypotheses about the next step to be taken and these hypotheses are remembered for future use. On the other hand, it would not exclude the Hunt and Hoveland concept construction system if this is capable of selecting useful concepts for aiding in prediction of future events. It is possible that the definition could include animals and this might raise various objections. However, it requires to be shown that an animal can or cannot construct hypotheses about its environment that enables it to predict future events. Most animal activities which appear at first to be in accord with the definition seem to be finally resolved as no more than conditioned reflex activity. It is difficult, in any case, to see why we should object to calling other animals intelligent, although we would agree that some animals were more intelligent than others. Since ability to predict future events carries with it ability to alter the environment consequent upon such prediction, one measure of intelligence might be the extent to which the intelligent animal alters its environment to its own advantage. In this

case human beings would still head any list of intelligent animals.

Intelligent machines

Clearly, we have a definition for an automaton and a means for classifying automata. It is also not difficult to show by the construction of suitable programs that a digital computer can simulate any automata (for example see Curtis, 1965). This does not imply that a computer can evaluate functions which cannot be evaluated by some automaton. In particular, and according to theories on computability (Davis, 1958), if a function is computable it can be evaluated by a Turing machine. Further, although it does not appear to have been carried out in detail, it is intuitively clear that a universal Turing machine could be constructed so as to evaluate a certain basic set of operators for a computer such that these operators are themselves sufficient to evaluate all operators defined by hardware for the computer in question. If we accept this intuitive view without proof in order to shorten the discussion, then an automaton with artificial intelligence must exist because we have seen that a computer can exhibit this property. It is, of course, not essential that the argument should rest on the requirement for a universal Turing machine, because a Turing machine, T, with a sufficiently complex control unit for evaluating functions describing acts of artificial intelligence will behave as a universal Turing machine which has a description of T on its tape.

If we turn again to the work of Hunt and Hoveland, we see in it, in addition to the simulation of inductive reasoning, a suggestion of the use of a computer for exhibiting intelligence in the sense of the definition given earlier. That is to say, an ability to construct concepts is an ability to determine rules which enable a decision to be made about an object relative to a concept, and this is implied by our definition. However, such rules could be no more than hypotheses formulated on the basis of observation of events in an environment and evidence in favour of an hypothesis may be partial or incomplete. The question is whether or not they are selected by a person so as to increase the probability of predicting future events relative to available relevant knowledge about the environment. It must surely be agreed that they are, otherwise they would have no value as decision rules. In the Hunt and Hoveland study they reported a reasonable degree of success by comparison with human subjects and suggest directions of attack for improvements. It appears, then, that to confine intelligent behaviour to non-machines would be to suppose that a significant area of human activity is unintelligent; thus implying also that only a very few gifted people are intelligent. Such an argument would be very difficult to support. It therefore appears that we have to accept as an intelligent automaton an appropriately constructed Turing machine at most. However, a Turing machine may have an infinite tape whereas we must surely regard the human brain as having only finite storage capacity. Under our definition of intelligence, human beings are intelligent. Hence, we must accept that there exists an intelligent finite state automaton.

The argument advanced above for the existence of an intelligent automaton does not disregard Uttley's suggestion that intelligence is the act of creating new information. It is implicit in our definition of intelligence that there is such a possibility, for to increase the probability in the way proposed in the definition may require creation of new additions to representation as demanded by the definition of information. This type of activity is exemplified by concept learning and, as seen above, there is nothing to suggest that this cannot be carried out by a machine. However, it does rule out the suggestion that there is something mystical in creating new information. This suggestion by Uttley is too close to Meszar's views to be seriously considered. Finally, if we admit that most human activity is intelligent as suggested earlier, then we must also agree with the existence of intelligent automata for we cannot avoid the definition of intelligence that has been proposed.

There are several questions which remain unanswered. Firstly, can a strictly finite automaton exhibit intelligence and secondly, how much storage is needed if the automaton must be at least finite state? Finally, must it be non-deterministic and, if so, will it be probabilistic? Some of these questions appear to have answers. Consider first whether an intelligent automaton could be deterministic. On the definition we have given for 'intelligence' the possibility of an algorithmic procedure for selecting hypotheses and theorems is certainly not excluded. Further, the evaluation of heuristic procedures does not exclude use of an underlying algorithmic procedure and hence a deterministic automaton. On the other hand, it can be argued that the selection of the next state in the case of an heuristic procedure is according to some probability rule, even though there would have been a deterministic evaluation of the probability. It is this selection of a state according to a probability rule that distinguishes a probabilistic automaton from other automata.

The difficulty that arises at this point is due to having no constraint of time in the definition of 'intelligence'. We know that in a practical sense it is not much use having an algorithmic procedure which takes too long to give an answer. However, to introduce the constraint of time into the definition of intelligence only introduces a further area for argument with regard to the longest interval of time that should be stated. The only way out of this dilemma is to accept that an intelligent automaton could be a deterministic automaton. This approach does not effect our definition of intelligence nor make it harder to decide that a strictly finite automaton cannot be an intelligent automaton. This latter device cannot alter its environment and it cannot remember individual past events. It can only remember restricted relationships between past events and the present event. Clearly, then, it cannot form any hypotheses to enable it to predict the future. For example, a strictly finite automaton can recognise a sequence of k occurrences of the symbol '1' followed by '0' followed by j

occurrences of the symbol '1', but it cannot recognise as a different sequence one similar to the preceding form when 'k = j' is always the case. That is to say, it is unable to decide that when the two sequences of '1's' on either side of the central '0' are the same in number then it has a different pattern on its tape to the case when they are not the same in number (Rabin and Scott, 1959). Hence, we can exclude strictly finite automata as intelligent automata.

Finally, the question of the amount of finite storage needed by an intelligent automata cannot be answered at this time. We would clearly expect that it would be quite large, in general, but equally it is to be expected that some intelligent actions may not require very much storage. We therefore cannot find any help in this direction for a definition of a 'robot'.

Robots

We are not bound by the English meaning of 'robot' nor are we forced in any way to justify this meaning. However, one of the properties which might be considered sufficiently distinctive is possession of intelligence by a machine. It has been seen above that by accepting a reasonable definition of intelligence we are lead to conclude that an intelligent automaton could be deterministic and finite state. It will undoubtedly be felt that this places an intelligent automaton at either too low a level in the classification of automata or at an impractical level from the viewpoint of the time required to complete an evaluation. There could not be any real objections to requiring that an intelligent automaton should also have the property of being probabilistic. If it turns out that some intelligent activity is possible with a deterministic automaton we have only to note that this is the same as having a probabilistic automaton whose states are all selected with a probability of unity.

The idea of an active device as expressed earlier in the introduction is that the values of arguments of functions it evaluates can be concrete objects but a passive device can only have abstract entities as values of the arguments. We have in mind as an active machine one which is able to examine its environment, take actions consequent upon this examination, or physically alter this environment such as is the case for the self-reproducing automaton. The mechanical devices of Gray-Walter (1950) were also very simple active automata and they exhibited a wide range of behavioural patterns but demonstrated that these behavioural patterns do not involve intelligence. On the other hand, as a property associated with the property of intelligence, we could expect to be able to distinguish a new class of machines.

It is obvious that an intelligent automaton could be either passive or active. However, it will undoubtedly be felt that those devices which deserve the special name 'robot' should be active machines because of the association of this name with the idea of doing work. It is therefore proposed that we should define a 'robot' as follows:—

> A 'robot' is 'an intelligent automaton such that it is at least an active, finite state, probabilistic automaton'.

This definition has the disadvantage of attempting to wrap up in one machine what would undoubtedly be best described as a collection of machines. On the other hand it has the advantage of avoiding a need to define inter-relations between machines. For example, we would at least have to define a robot otherwise as 'a collection of inter-related finite state active and passive automata of which at least one is an intelligent automata which is at least a passive finite state probabilistic automata controlling the remaining automata in the collection'. The question is now to decide what is meant by 'inter-related' and what is meant by 'controlling'. That is to say, we have to determine some hierarchical structuring of various automata in which the apex of the hierarchy has an intelligent machine in the form of some finite state probabilistic automata. While it may well be true that the ultimate realisation of a robot would have such a form, a collection of automata and a single automaton will become conceptually identical simply by naming the collection by the name of the single automaton. That is to say, if 'Z' is the single automaton, and 'Z_1' 'Z_2' . . . . Z_n the automata in the collection, then the two definitions are identical when 'Z' is a union of the 'Z_1' for $1 \leqslant i \leqslant n$. However, the matter of how the inter-relations and control are effected remain to be defined if we talk about a collection of automata, and this is avoided if we refer only to a single automaton as in the case of the first definition for a robot above. The advantage in demanding a probabilistic automaton as a property of a robot is that it avoids the problem of time to complete a task as mentioned earlier. For example, the game of chess has an algorithmic solution but in an impossibly long time. Application of an heuristic based on 'most probable move' in a given situation enables a computer program for chess playing to be built.

Conclusion

It is believed that two things have been demonstrated. Firstly, there is no reason to believe that there exists an intelligent machine which is different from any automata for if this were so it would contradict the basis of computing theory which postulates that anything which is computable is computable by some automaton. It cannot be objected that intelligence demands the application of non-computable functions because our quite reasonable definition of intelligence would then have to be abandoned. As far as we can tell, both deductive and inductive reasoning are the only tools required to achieve what we call intelligence and both these procedures are realisable by computable functions. Further, we reject the idea that creation of new information occurs in some mystical way and claim that it is a result of applying inductive and deductive reasoning in some computable way, although the method for doing this is not clear at present.

Secondly, it is believed that there is a reasonable definition for a particular class of automata which would exhibit intelligent behaviour and be able to interact with their physical environment. If this definition is accepted, then there exists a class of machines

to which we should give the name 'robot' in order to formalise this as a concept in the body of computing theory rather than continue with its rather loose English connotation. Clearly, the confirmation that a robot exists must wait until the construction of such a device, either as a collection of inter-related and co-ordinated theoretically acceptable automata, or as a physical fact, is demonstrated.

Finally, the question of 'learning machines' that is, devices that are self adaptive to their environment in some way has not been considered here because it has been assumed that any intelligent machine that qualifies to be called a robot would be capable of learning. This is inherent in the proposed definition of intelligence which demands that the selected hypotheses or theorems must benefit the selecting mechanism by enabling it to adapt to its environment through better prediction of future events in its environment.

References

1. ARBIB, M. A. (1966): Information and Control *9*, 177.
2. ARMER, P. (1960), in 'Symposium on Bionics', p. 13, Wadd Technical Report, 60, 600.
3. BURKES, A. W. (1963), in 'Computer Programming and Formal Systems', p. 100; North Holland Publishing Co., Amsterdam.
4. CAPEK, C. (1923): 'The R.U.R.'; Doubleday, Oxford University.
5. CHURCH, A. (1956): 'Introduction to Mathematical Logic'; Princeton University Press, Princeton, N.J.
6. COHEN, M. R., and NAGEL, E. (1934): 'An Introduction to Logic and Scientific Method', 1st ed. reprint 1961, p. 391; Routledge and Regan Paul, London.
7. CURTIS, M. W. (1965): Journ. A.C.M., *12*, 1.
8. DAVIS, M. (1958): 'Computability and Unsolvability'; McGraw-Hill, New York.
9. FISCHER, P. C. (1966): Information and Control, *9*, 364.
10. GINSBERG, S., and ROSE, G. F. (1963): Journ. A.C.M., *10*, 29.
11. GORN, S. (1962): Proc. Symp. Pure Mathematics, *5*, 201.
12. GRAY-WALTER, W. (1950): Symposium on Information Theory, p. 134; Ministry of Supply, London.
13. HUNT, E. B., and HOVELAND, C. I. (1961): Proc. Western Joint Computer Conference, Vol. 19, p. 145.
14. KOVALEVESKY, U. N. (1965): Proc. IFIPS Congress 65, Vol. 1, p. 37; Spartan, New York.
15. MACKAY, D. M. (1950): Symposium on Information Theory, p. 9; Ministry of Supply, London.
16. MESZAR, J. (1953): Bell Telephone Laboratories Record, February.
17. MINSKY, M. L. (1961): Proc. I.R.E., *49*, 8.
18. ——— (1965): Proc. IFIPS Congress 65, Vol. 1, p. 45.
19. NEWELL, A., SHAW, J. C., and SIMON, H. (1957): Proc. Western Joint Computer Conference, Vol. 15, p. 218.
20. RABEN, M. O., and SCOTT, D. (1959): I.B.M. Journ. Res. & Dev., *3*, 114.
21. ROBINSON, J. A.: (1965): Journ. A.C.M., *12*, 23.
22. SLAGLE, J. L. (1963): Journ. A.C.M., *10*, 507.
23. SOLOMONOFF, R. J. (1959): ASTIA Document AD 210390.
24. TURING, A. M. (1950): Mind, *59*, 433.
25. UTTLEY, A. M. (1950): Symposium on Information Theory, p. 143; Ministry of Supply, London.
26. WANG, H. (1960): I.B.M. Journ. Res. & Dev. *4*, 2.

Special Note: The papers of Armer, Hunt and Hoveland, Minsky (1961), Newell et al., Slagle, and Turing are collected together in 'Computers and Thought', McGraw-Hill, 1963.

Book Reviews

Selecting the Computer System, D. N. Chorafas, *Gee & Co. Ltd., 1967*, pp. XV + 336, $10.90A.

This book will be of assistance to any organization which is about to automate or consider automating its information system. Apart from its subject matter, much of the value of the book lies in the practical and commonsense approach the author makes to the problems he considers.

The book starts with an outline of essential attitudes to the basic problem of how to (and whether to) automate an information system. It then moves on to a general discussion of what can be achieved and, what is most welcome, a recognition that the computer is but a component of the information system it serves.

The author presents an interesting and informative approach to the task of matching available hardware facilities to the requirements of the information system being specified. This leads on to some useful ideas about how to at least mitigate the difficulty of estimating the costs of a large programming job.

Part II of the book develops in some detail the important question of how to select a computer. There are examples of comparison studies and, later in the book, an illustrative case study of a failure that takes its point not from the fact that the wrong computer was chosen but from the fact that a computer was not chosen at all.

Much of what is in this book will not be new to those who have experienced the pangs of introducing an EDP system but it meets a heavily felt need for a readable and straightforward synthesis of such experience. Although the book professes to be designed and written to assist the executive in his choice of a computer system, it will be most helpful also to EDP staff generally who are setting out on the road.

An Introduction to Critical Path Analysis, K. G. Lockyer, *Pitman and Sons Ltd., 1964*, pp. X + 134, $3.30A.

This book is a very simple and elementary treatment of the topic, easy to read and understand and is a compact, useful book purely for the non-specialist who wants to know what it is all about.

This is mainly what the book intends, although the claim that it shows "how it (i.e., CPA) can be put into practice" is not substantiated.

For instance, there is no worthwhile discussion of what steps an organization must take in order to undertake CPA effectively, nor is any sense of the scale of such an operation portrayed or the amount of time and effort required.

The basic rules of network construction are well presented, as also are the calculations of activity floats and the finding of the critical path. The importance of checking a time plan for resource demands is not overlooked and there is a chapter on using CPA for control. The chapter on the use of the computer provides a summary of input requirements and the output sorts normally available and lets it be known that resource totalling and levelling programs also commonly exist.

For the would-be specialist, however, this book is perhaps not as useful a starting point as others covering similar ground rather more comprehensively. It is well suited for the man in an organization who comes into direct contact with those responsible for actually carrying out CPA.

M. Anson.

Computer Science Education In Australian Universities

By P. D. Jones*

Abstract

Formal undergraduate and graduate computer science courses are just beginning to take shape in most Australian universities. It appears immensely important in these early formative years to promote as much wide-scale interest and discussion in order to help formulate the best possible courses and define the role of the computer science department, particularly in its relationship to other bodies. The paper proposes a possible course structure suited to the Australian scene and discusses important related issues. In the past computer teaching and research have proceeded independently in the various universities, with little impact on matters outside the university sphere of activity; ways for improving this state of affairs are suggested in the paper.

Course Structure and Associated Matters

Scope for Computer Science Teaching

There are four main areas where computing science in some form or other is likely to be taught in the university.

(1) Formal courses at graduate and undergraduate levels leading to degrees in computing science. The aim of these courses is to produce what might be called 'professional computer people'. The responsibility for these courses is clearly that of the computer science department, but there is no reason why this department should not make as much use as possible of suitable course modules provided by other schools and departments within the university. The structure of these formal courses is the main consideration of this paper.

(2) Introductory courses in computing for first year students. These courses most likely would be the responsibility of the computer science department, but, if not, the department or school responsible for them should certainly work in close collaboration with the computer science department.

(3) Courses in different aspects of computer science taken by students who are studying for degrees in subjects other than computer science. Such courses might be data processing by commerce students, numerical analysis by mathematics students and digital machine design by electrical engineering students. Some of these courses might be provided by the computer science department, others by the school or department responsible for the course. Clearly, it is neither possible nor desirable for the computer science department to organise and run all computing courses within the university. As time goes by, courses of this type will become increasingly common on the university campus. Again, the only basic rule is close collaboration between the various schools and departments concerned.

(4) Service courses to educate computer users within the university. These would normally be the responsibility of the university computer centre who might call on the computer science department for assistance in running the courses.

Publications on Computer Science Education

There are numerous excellent reports on computer science education (References 1 to 10) which should be studied by those interested in this subject and who may have the responsibility of formulating courses. One of the most valuable aspects of these reports, apart from the computer science curriculum itself, is the long list of references they contain on computer science subjects. Perhaps the two most comprehensive reports are the A.C.M. (Association for Computing Machinery) report (Reference 1), and Professor Forsythe's paper (Reference 5) which quotes extensively from the Computer Science course at Stanford University. One great value of the A.C.M. report is that it consists of preliminary recommendations and it is to be followed by other reports along the same lines (Reference 8).

Until recently very little appeared on computer education in the publications of the British Computer Society (Reference 13 is one paper). Our main indebtedness to the U.K. seemed to be for the Flower's Report (Reference 11) and another paper (Reference 12) on the functions of a National Computing Authority. In June of this year (1967), however, the British Computer Society Education Committee published their first annual educational review (Reference 23). This is a comprehensive survey including reports not only on computer science education in universities, but also on computer education in schools and management, training of systems analysts, programmed learning in computer training and computer training aids. The specimen computer science courses put forward are much the same as those recommended by A.C.M. but they do include three courses on different aspects of data processing.

* *University of New South Wales, Kensington, N.S.W.* *Manuscript received August, 1967.*

Virtually nothing was said about computer science education at either the 1962 or 1965 IFIP's congresses but the subject received some comment at the Australian Computer Conference in May, 1966 (References 16 and 17).

Two excellent reports which have only recently come to hand are those of the President's Advisory Committee on computers in higher education (Reference 21) and an Australian Vice-Chancellors' Committee survey of university computing activities (Reference 22).

Course Structure

The basic course structure in computer science suggested by this paper for use in Australian universities is that recommended by the A.C.M. (Reference 1). A table summarising the curriculum is given in Figure 1. There is little point in going into the details of the course here, since they are clearly defined in Reference 1. It will suffice to make a few general comments and say why this particular course is recommended.

Perhaps the most important aspect of the A.C.M. proposal is its comprehensiveness, covering all the recognised computer science subjects and producing a well balanced course structure. This all-round nature of the curriculum is often the very thing which has been lacking in a number of computer science courses. The fact that it is recommended by A.C.M., the most formidable computer organisation of its kind in the world, is sufficient reason for giving it careful consideration. The A.C.M. curriculum committee on computer science consisted of a number of outstanding computer persons who considered the problems involved for three years before publishing the report. The committee did not consider their proposals to be final, but rather a point of departure; universities using this structure as a standard will have the benefit of knowing the course is continually being improved and modified to adapt to changing conditions. The report has also had the benefit of critical review by a large number of computer scientists.

These remarks are not meant to suggest that we should slavishly follow the course in every detail, but

Figure 1:

PRELIMINARY RECOMMENDATIONS OF THE CURRICULUM COMMITTEE OF A.C.M. FOR MAJORS IN COMPUTER SCIENCE

Courses / Recommendations	Computer Science				Supporting
	Basic Courses	**Theory Courses**	**Numerical Algorithms**	**Computer Models Applications**	
Requires	1. Introduction Algorithmic Processes 2. Computer Organisation and Programming 4. Information Structures	5. Algorithmic Languages and Compilers	3. Numerical Calculus (or Course 7)		Beginning Analysis Linear Algebra
Highly Recommended Electives	6. Logic Design and Switching Theory 9. Computer and Programming Systems		7. Numerical Analysis I 8. Numerical Analysis II		Algebraic Structures Statistical Methods Differential Equations Advanced Calculus Physics
Other Electives	10. Combinatorics and Graph Theory	13. Constructive Logic 14. Introduction to Automata Theory 15. Formal Languages		11. Systems Simulations 12. Mathematical Optimisation Techniques 16. Heuristic Programming	Analog Computers Electronics Probability and Statistics Theory Linguistics Logic Philosophy and Philosophy of Science

that the suggested curriculum should provide a standard for all computer science courses in Australia.

Course Duration and Module Size

The combined total of the required courses and the highly recommended electives in the A.C.M. curriculum amount to approximately 54 semester credit hours. For those not familiar with the semester credit system, there are two semesters each of 15 weeks per year. A semester credit hour is usually one lecture or two laboratory classes per week for a semester. The average student undertakes around 15 credit hours per semester and graduates on obtaining 120 semester credit hours.

Leaving aside the supporting subjects, the A.C.M. total is 8 course modules each of 3 credit hours which is the approximate equivalent of a third year science subject or half the full third year course in Australia. To fit the Australian schedule, the A.C.M. curriculum is easily modified from 15 week semester to 10 week term modules which consist of 3 to 4 hours lectures and tutorials per week. The value of tutorials and group activity cannot be over emphasised. Each computer science student should at least gain experience in writing a compiler and operating system, designing a simple computer, operating a computer under data centre conditions and programming a wide variety of problems, commercial and scientific, in both high and low level languages.

One danger of having computer science purely as a third year subject is that so few students at the present time have any background in computing, and one year is just too short a span of time for them to absorb all the new ideas. It is clear that students must do an introduction to computing course module in first year and be encouraged to use the computer during their first two years. Intending computer science students should also be encouraged to undertake a reading course, attend seminars and seek vacation employment related to computing before reaching third year. It is worth noting that at Manchester University Computer Science is a full three year course.

So far nothing has been said on honours and graduate courses in computer science. The 1964 A.C.M. recommendations only cater for the undergraduate curriculum but a similar report is expected soon on the graduate curriculum. In Australia it is to be expected that universities will naturally provide a fourth year honours course as it done for other science subjects.

Staff Requirements

There is no computer science department in Australia at present with enough staff to properly conduct the course curriculum recommended in the A.C.M. Report. Each of the four main course streams — hardware, system software, applications and numerical analysis—require at least one senior man (professor-associate professor) and an assistant (senior lecturer-lecturer). Thus, handling of the main course requires at least eight persons on the staff. To this number might be added another four to cater for extra lecturing commitments such as introductory computing lectures to first years and the special research interests of the department. A staff of one dozen therefore appears to be of the right order if the A.C.M. curriculum is adopted. This assumes, of course, the staff are absolutely free of any computing centre commitments. As a comparison, it is understood that the staff to handle the Computer Science course at Manchester University numbers approximately 20.

Many things could be said of the type of staff required. The person to lead each course stream should have had considerable experience in his own field, preferably experience at home and overseas, both in universities and outside in industry and commerce. There are quite a few Australians in leading computer positions overseas and they should be attracted back, together with new migrants from Europe and America. There should be sufficient bright research students at home who, on completion of their post-graduate studies, could fill the lecturer positions.

It need hardly be stated that the day of the talented, all-round computer amateur has virtually reached its end. Whilst there is always a coveted place for the extremely talented person with wide interests, the norm is surely for the professionally trained person who confines his interests to a particular field in order to make significant and worthwhile contributions.

Finally, it might be remarked that it is always good to have a proper balance between purely theoretical and practical members of staff. Each should be allowed to pursue his own interests without pressure to conform to the other side. It is most important that students see computing science from the theoretical and formal viewpoint as well as the practical side.

Introductory Courses in Computing

There is a strong argument for making all first year university students in Australia undertake some kind of introductory course in computing science. The computer is now a recognised general purpose tool in society and it is likely that most university graduates will have cause either to use a computer or evaluate results processed by a computer. Even when computing is introduced in high school courses, as it inevitably must be, there may still be need for refresher courses of university standard in computing science.

The introductory course need not follow any strict syllabus or be of any standard duration; however, the course structure shown in Figure 2 might be considered a good standard for comparison. The factor which dominates the organisation of such courses is the number of students involved; in most universities the numbers will be in the thousands. If we agree there is no substitute for students having direct contact with the computer, this clearly means several things—the average student program must run for no more than a few seconds; the error diagnostics must be excellent in order to avoid students swamping the lecturing staff with enquiries and adequate program preparation devices must be made available. Whilst batch processing student jobs, either on cards or paper tape, will be hard to beat in the immediate future, the advantages of the online display terminal should not be overlooked.

There is no doubt that when individual display terminals become the normal means for man-machine communication we shall look back in horror to the present period. Today's programs are usually punched on cards or paper tape and taken to the computer, after a lapse of hours and sometimes days a response is received which more often than not states the program is in error. To be fair the case against batch processing often looks stronger than it really is. This is due mainly to computer overloading and the fact that many students spend too high a proportion of their time grappling with the practicabilities of particular data preparation devices when it is clear that professional punching staff should be used.

A computing course of this kind is ideally suited to the application of computer assisted learning techniques. One can imagine, in the not too distant future, an introductory computing course involving lectures, films and computer centre tours being given once a term and the students then directed to use the computer terminals. A dozen terminals set up in a room near the centre of the campus and available 12 hours a day, six days a week, could easily cater for 1,500 students, allowing each student half an hour on the terminal per week. Assuming a course lasted one term, each student would spend five hours on the terminal; with a three term year 4,500 students could be handled. The essence of teaching programming is practice on a wide range of sample problems and there is no doubt that the online terminal is by far the best from this viewpoint as the average student might have 10 runs on the computer in a half hour session. Before allowing the student to write his own programs he should be asked to correct, improve and modify existing programs to gain experience and confidence.

Both the hardware and software are presently available from most computer manufacturers and really the only likely barrier to moving ahead is the economics of the situation. However, this is not as bad as most people suppose. For an analysis on the subject of time-shared terminals versus batch processing techniques the reader is referred to Reference 18. Each student program run, whether it be under batch processing or terminal mode, will occupy the computer's processing unit for only a second or two. The average student in a half hour terminal session would probably use about 10 seconds actual computing time. If the computer uses portable disc packs (around $500 each), one such disc pack could easily hold all teaching texts, sample programs and student records at an average access time of approximately a tenth of a second. The system overhead running terminal mode is usually greater than when running batch mode, but in a properly organised system the difference is not significant.

The remaining large item to be considered is the cost of the terminal; this is assumed to be of the display type rather than typewriter and teletype which are too slow and inflexible. A set of twelve displays plus controller bought over the counter from a manufacturer at today's prices is likely to cost around $60,000, or rent for $1,500 per month. (Work presently in progress at the University of New South Wales on using commercial television sets as displays looks like reducing the cost price per terminal station down to between $2,000 and $2,500. Annual rental costs of $18,000 may seem high, but the cost of cards, line printer paper and hire of card punching equipment would probably exceed this in catering for thousands of students.

Whichever method, batch processing or terminal mode, the costs are going to be high and it is worth quoting Professor C. E. Forsythe, Head of the Stanford University Computer Science Department (Reference 5)—"the total cost for handling student jobs in computer science courses has now reached $84,000(U.S.) per year, and it is essential to receive special funding for this work." On the general question of budgeting equality with the university library has been suggested (Reference 21) and sets the scale of the whole operation; a figure of $50 per year per student for computing appears to be of the right order.

Research Areas

Every computer science department should be strong in research in at least one of the main course activities —computer design, programming system and language design or applications and numerical analysis. In fact, the major research areas in the computer science field should be covered by the various departments in Australia. Rather than work solely on individual research, there appears great merit in a department tackling at least one large research job which requires

Figure 2:
Introductory Course in Computer Science

Week	First Hour	Second Hour	Third Hour
1	History of Computing	Computer Centre Tour	Films
2	Information Structures	Machine Code	Tutorial
3	Logic Design	Machine Code	Tutorial
4	Algorithmic Procedures	Assembler Language	Tutorial
5	Computer Organisation	Assembler Language	Tutorial
6	Programming Systems	High Level Language	Tutorial
7	Languages & Compilers	(PL1, Algol or Fortran)	Tutorial
8	Problem Solving	(PL1, Algol or Fortran)	Tutorial
9	Data Processing	(PL1, Algol or Fortran)	Tutorial
10	Numerical Analysis	(PL1, Algol or Fortran)	Tutorial

the co-operation and effort of a number of department members. The task might be the design and production of an actual machine, implementation of an operating system, design and development of a new language, or some large application job. Of course, the opinion that except in very special circumstances the computing industry has passed beyond the point where it is a feasible activity for universities to build their own computer, is widely held. This author believes, however, that the basic building blocks (memories, circuits, peripherals, etc.) are readily available for computer design and production and that the benefits to be gained from such an exercise are great.

Rather than just critiqueing what others are doing, there appears no substitute for actually launching out as a department on some large creative venture. This type of research usually leads to most valuable first hand experience, a sense of real department purpose, and often a breeding of new research projects out of the main one. Whilst the task should be large it should not be too large in that it completely swamps all the members of the department for a number of years. In a university context it is also important that the individual research student should acquire sufficient thesis (i.e., publishable) material within his mandatory period of "residence".

The Relationship Between the Computer Science Department and Other Bodies

The Computer Science Department and the University Computer Centre

Whilst at the moment in Australia it is normal for the Computer Science Department to control the running of the University Computer Centre, it is becoming increasingly common in the United States for the two to be separate. The only common link is that the Head of the Computer Science Department is also Director of the Computer Centre. This is usually regarded as a sign of maturity and coming of age in the computer world where research is often no longer closely allied to solving the many practical problems associated with an efficient computer centre operation. This trend must certainly come as a relief to some departments whose research and teaching has been swamped by computer centre problems. On the other hand, some university installations may also be glad in that large amounts of time, previously devoted to the development of programming systems and compilers of dubious value, now become available to their normal users.

Despite the trend in the United States, it appears unlikely that the situation will change rapidly in Australia. This is probably a good thing as there is no reason why the computer science department and the computer centre should not exist alongside each other in a healthy relationship bringing mutual benefit to both sides. The critical figure is the proportion of the time of academic staff which is spent running the computing installation and this undoubtedly is decreasing. A good computer manager and working system appear to be the key factors; the good manager will handle the majority of problems himself and need call upon the director only when questions of policy arise. He will also discipline the use of the computer for computer research on the one hand, yet offer the benefits of research to his users on the other.

The computer science department running the university computer centre has been likened to the civil engineering department building the roads around the university. The analogy may or may not be fair, but it might be argued that it is a fair test of the civil engineering department to see what kind of roads they build.

The Computer Science Department and the Rest of the University

Where should Computer Science be found — Electrical Engineering, Physics, Mathematics or as an independent department? Historically, computer science departments have grown up as part of a school—the School of Electrical Engineering at the University of New South Wales and the School of Physics at Sydney University, for example. Whilst the trend in the United States is for the computer science department to become independent, there is still much to be said for it remaining part of an older school or discipline — at least until the new courses in computer science have had time to settle down.

No one can really say to which school Computer Science has its closest attachments; there are equal arguments for attaching it to Physics, Electrical Engineering, Mathematics, Philosophy and Commerce, to mention a few. The effect of the older disciplines has been to bring character and a sense of direction to the computer science departments leading one towards hardware design, another to numerical analysis or operations research, another to commercial applications and yet another to language studies. It would appear to be a pity if all the computer science departments in Australia were suddenly to shake off their old attachments and seek for independence; the general danger is that they might all become too similar. Of course, there may be certain departments who rightly feel the advantages of independence far outweigh any possible disadvantages. A useful parallel can be found in the early history in older universities of most disciplines which have achieved independent status since the turn of the century — statistics, psychology, electrical engineering, for example. They have all started as sub-departments of some other discipline.

Whether the computer science department is part of an older school or not, there is still occasion for it to seek collaboration with other departments in conducting its courses and possibly also in research. The A.C.M. clearly foresees the possibility of parts of the course being given by other departments and of certain course modules being of value to students whose prime interest is in another area of study. Certainly there is considerable benefit to be gained from mutual co-operation between computer science and other departments of the university. One cannot but help comment on the tragedy that most university courses in Australia are not usually taught in standard sized modules. The great advantage of modular courses is that it encourages and enables sharing and participation in the various courses of different disciplines by

students to a much greater degree than is presently possible. Perhaps in older universities the difficulty arises mainly from faculty structure rather than a lack of modularity in courses.

Relationship of the Australian Computer Science Departments to Each Other

One of the very obvious shortcomings on the Australian scene is the lack of co-operation between the various computer science departments. It should be added that this lack of co-operation reflects the inevitable effect of staff shortages and geographical separation rather than a deliberate unwillingness to co-operate. It is recognised that much co-operation throughout the computer world falls by the wayside and is of no great profit. This does not mean that all co-operation is of this type but simply that the type of co-operative venture to be undertaken is a matter for careful study and consideration. Whatever may be said about the excessive number and quality of computer conferences in the United States, without doubt the sharing of information and experiences at these conferences has contributed significantly to the computer boom in research, new applications, teaching and manufacture in that country.

If the A.C.M. proposal is basically adopted by the computer science departments in this country then those strong in a given area might provide lecture notes and tutorial problems on certain course modules for the other departments. Certainly there should be some degree of co-operation in order to avoid duplication of research interests, service programs and course texts. It is suggested that an Australian National Computing Authority is the right body to co-ordinate computing activities in the universities.

The Computer Science Department and the Outside World

Like any other university department, computer science has much to gain from contact with outside research, commerce and industry. Every possible area of contact should be encouraged — seminars and lectures from persons outside the university, tours of computer centres, participation in computer society activities, collaboration on research projects with outside interests and consultancy to firms. Close contact should be maintained with all leading computer manufacturers in order to keep abreast with latest developments and products.

Perhaps the most important aspect is to maintain constant feedback on the types of graduates needed by industry, commerce and research establishments. There is always the strong temptation for universities to become removed from reality and produce graduates whose training is of little direct value to industry. This does not mean a purely practical computer course, nor does it mean the university is unduly influenced by outside concerns, but it does mean the university is aware of the ever changing needs of the community outside its own campus.

The Computer Science Department and a National Computing Authority

From an economic viewpoint it appears inevitable that the important centres (Canberra, Sydney, Melbourne, etc.) in Australia will have large central powerhouse type computers to serve whole regions. It is natural enough to suppose that these centres should be run by an Australian National Computing Authority (A.N.C.A.). Universities, government research organisations and, indeed, anyone willing to pay the rent could go online to the large computer (References 19, 20 and 24).

The large central computer certainly does not do away with the need for a powerful central computer within each university, nor with special purpose small computers and display terminals in the various schools, departments and colleges. The four levels of computer system organisation are shown in simplified form in Figure 3. The central computer will, in fact, not

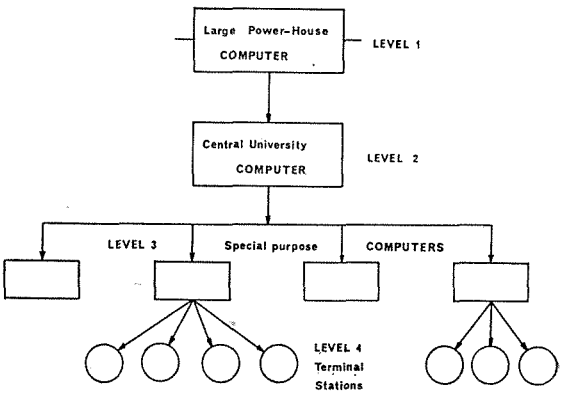


Fig. 3—University Computer Network

make a great deal of difference, except to the small percentage of users who have very large (either in time or storage requirement) programs. The average user will receive better service only because more time becomes available on the university computer when the big jobs are off-loaded onto the A.N.C.A. machine. Though sheer horsepower will still be important for the university's computer, because of the backup of the A.N.C.A. computer speed will not be the dominant consideration. Other features which increase the all round performance of the university computing system such as large memories, terminal equipment, special purpose computers and peripherals, will become increasingly important.

Of course, the functions of a National Computing Authority are far wider and more important than simply running a few computer centres (Reference 12). Certainly, A.N.C.A. must work in close collaboration with the universities, serving as a co-ordinating influence between the various computer science departments. Free interchange of both information and staff between A.N.C.A. and the universities is an absolute necessity. Whatever the area—encouraging research,

setting standards, sponsoring overseas visitors, making scholarships available—A.N.C.A. has a great contribution to make, providing it is organised along the right lines.

Dr. K. V. Roberts in stressing the need for a National Computing Authority in England writes (page 36, Reference 12)—"the author believes that in order to make the most of our computing opportunities a single unified autonomous authority is required, organised on a professional-engineering rather than an administrative basis, and certainly not by part-time committees as at present . . . The U.S. space program and the British atomic energy program have both been outstandingly successful, and it is worth studying these models if an equivalent computing project is to be set up. They each have the same characteristics; far-reaching plans accompanied by a minute attention to detail". An important and necessary first step on the Australian scene would be the establishment of a committee with representatives from government bodies, universities, industry and commerce, computer manufacturers and computer societies, to draw up a charter for A.N.C.A. and report on some of the more important administrative problems that such an organisation is likely to meet (staffing, etc.).

Conclusions

1. Whilst acknowledging the importance for each computer science department to remain independent and pursue its own interests, a basically common undergraduate computer science course is recommended for all Australian universities. The curriculum proposed is that put forward by the A.C.M. (Reference 1, Figure 1), which is characterised by its modular structure and comprehensive coverage.
2. The majority of all Australian university students should undertake an introductory computing course (Figure 2) in their first year. Computer-assisted learning techniques should be invoked to automate the course using display terminals for student-machine communication.
3. Mutual co-operation between the various computer science departments and their collaboration with other university departments and outside organisations will benefit all concerned.
4. The formation of an Australian National Computing Authority is envisaged to control power-house computers in the large centres and co-ordinate the country's computer interests. To a large extent the success of this body will depend on its close association with the universities.

Acknowledgements

The author is indebted to his colleagues at the University of New South Wales for many helpful discussions, and in particular to Professor M. W. Allen who organised the University's Computer Science Course. His thanks also go to Professor J. M. Bennett who made many helpful suggestions.

References

(1) An Undergraduate Program in Computer Science—Preliminary Recommendations. A.C.M. Curriculum Committee Report on Computer Science.
Comm. A.C.M.8 (Sept. 1965), 543-552.

(2) Mathematical Association of America, Committee on the Undergraduate Program in Mathematics.
Recommendations on the Undergraduate Mathematics Program for Work in Computing.
The Committee, P.O. 1024, Berkeley, Calif., May, 1964.

(3) FORSYTHE, C. E.: An undergraduate curriculum in numerical analysis.
Comm. A.C.M. 7 (April, 1964) 214-215.

(4) FORSYTHE, C. E.: University education in computer science.
Proc. Foll Joint Computer Conference, San Francisco, calif., November, 1964.

(5) FORSYTHE, C. E.: A University's Educational Program in Computer Science.
Comm. A.C.M. 10 (January, 1967) 3-11.

(6) A Report of Two Symposia on the Impact of Computing on Undergraduate Mathematics Instruction.
Comm. A.C.M. 9 (September, 1966) 662-670.

(7) FINERMAN, A.: Computer Capabilities at Western European Universities.
Comm. A.C.M. 9 (December, 1964) 840-844.

(8) OETTINGER, A. G.: President's Letter to the A.C.M. Membership.
Comm. A.C.M. 9 (December, 1966) 838-839.

(9) ORDEN, A.: The Emergence of a Profession.
Comm. A.C.M. 10 (March, 1967) 145-147.

(10) FULKERSON, L., and ATCHISON, W. F.: Letters to the Editor.
Comm. A.C.M. 10 (March, 1967) 148.

(11) A Report of a Joint Working Group on Computers for Research, chaired by Prof. B. H. Flowers, Cmd. 2883, HM30 1966.

(12) ROBERTS, K. V.: The Functions of a National Computing Authority.
The Computer Bulletin 10 (December, 1966) 30-37.

(13) BROOKER, R. A., and ROHL, J. S.: Experience with a first-year Computer Science Course.
The Computer Bulletin 10 (December, 1966) 42-44.

(14) SCHATZOFF, M., TSAO, R., and WIIG, R.: An Experimental Comparison of Time Sharing and Batch Processing.
Comm. A.C.M. 10 (May, 1967) 261-265.

(15) WALTER, E. S., and WALLACE, V. L.: Further Analysis of a Computing Centre Environment.
Comm. A.C.M. 10 (May, 1967) 266-272.

(16) GRAINGER, K. E.: Recruitment, Education and Utilisation of Staff.
Proc. Third Australian Comp. Conf. (May, 1966), 411-417.

(17) FINDLER, N. V.: On the Status of Computer Science in U.S., Canadian and Mexican Universities.
Proc. Third Australian Comp. Conf. (May, 1966) 501-504.

(18) SCHATZOFF, M., TSAO, R., and WIIG, R.: An Experimental. Comparison of Time Sharing and Batch Processing.
Comm. A.C.M. 10 (May, 1967) 261-265.

(19) PARKHILL, D. F.: The Challenge of the Computer Utility.
Addison-Wesley (1966).

(20) BENNETT, J. M.: Some Thoughts on Computer Utilities.
Basser Comp. Dept. Tech. Report 45 (February, 1967) and A.C.S. (N.S.W. Branch) Bulletin (April, 1967).

(21) Computers in Higher Education.
Report of the President's Science Advisory Committee. The White House, Washington, D.C. (February. 1967).

(22) Report of a Survey of University Computing Activities.
Australian Vice-Chancellors' Committee, Report C67/9 (February, 1967).

(23) Review of Education.
The Computer Bulletin 11 (June, 1967) 3-80.

(24) BENNETT, J. M., and PEARCEY, T.: Networks of Computers.
(I.E.E.E. (Aust.) to be published.)

A Computer Representation Of Plane Region Boundaries

By B. G. Cook*

Summary

A data structure for the storage of representations of plane region boundaries is described which attempts to exhibit their pictorial structure. The reasons for adopting this representation are discussed in some detail.

1. Introduction

There are many situations where data can be presented economically by mapping an area into regions on the basis of common attributes. These regions are commonly distinguished by giving each a colour which differs from the colour of its neighbouring regions. We are concerned here with a computer representation of such a map.

Throughout this paper, **region** means a connected plane area bounded by one or more closed curves.

A map of plane region boundaries contains both positional and structural information. The positional information defines the shape and position of the boundary curves. The structural information defines the pictorial relationships of regions and their boundaries; for example, in Fig. 1, region A is surrounded by region B, region C is adjacent to region D, regions E, F and I meet at a common boundary junction.

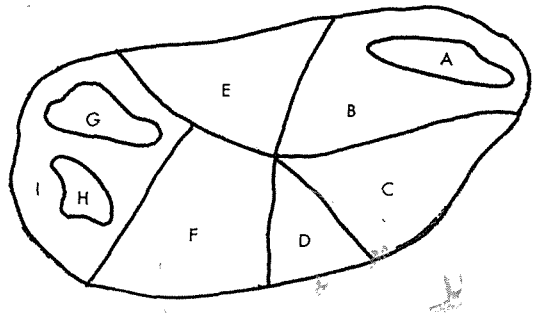


Figure 1. A map of region boundaries.

Any computer representation of region boundaries must contain the positional information defining the boundary curves. The pictorial structure can always be recovered from this positional information, but only with considerable difficulty unless the representation facilitates its recovery.

While considerable attention has been paid to the development of schemes to compact the positional information in region boundaries, existing representations vary in the extent to which the structure of a map is exhibited. Pfaltz, Snively and Rosenfeld (1967), in an essentially unstructured representation, attach to each region block lists of other regions containing or contained in the region, and suggest possible extensions to include other relationships. Loomis (1965) attaches to each section of a boundary the labels of the regions on either side, thereby achieving reasonable facility in extracting structural information from a partially structured system. The system described by Tomlinson (1967) uses a similar representation. In the wider field of general purpose graphics, the highly structured Sketchpad system (Sutherland, 1963) displays relationships between the elements of a drawing by gathering all references to a common element into a "ring" structure. All elements of the same type are also gathered together on a ring. Such a system could undoubtedly be applied successfully to the storage of region maps, but it seems an unnecessarily complicated solution to this particular problem.

In planning a computer based information storage system for land resources data, an uncomplicated representation for region boundaries was sought which would allow the context of a region to be readily recovered.

The results of land resources surveys are commonly presented by describing land units—regions of substantially homogeneous topography, soil, vegetation, etc.—and mapping the boundaries of land systems—regions having a recurring pattern of interrelated land units. (Christian and Stewart, 1953). The boundaries of land systems are usually fixed by interpretation of air photographs, subject to limited ground checking.

We have two reasons for wanting a representation which exhibits the context of a region. Firstly, in the evaluation of land potential for agricultural, engineering or other purposes, the context of a land region is usually of considerable importance. For example, ease of access or likelihood of flooding or erosion may depend more on the nature of neighbouring regions than on the nature of the region concerned. Secondly, the concept of a land system assumes contextual relationships between land units, and it is hoped that such a representation may be of assistance in possible future attempts to use a computer to recognise and map land units and systems.

Existing representations of region boundaries which

* *Division of Land Research, C.S.I.R.O., Canberra.* *Manuscript received August, 1967.*

store positional information in an unstructured form, and associate contextual attributes with regions or sections of their boundaries, facilitate the recovery of only a limited range of structural information. Since we wish to be able to recover a wide range of contextual information, a representation which is itself structured in a manner similar to the pictorial structure of a region map should have advantages. The data structure described is an attempt to develop such a representation.

2. Preliminary definitions

A **region** (R) is a connected area bounded by one or more closed curves (see Fig. 2). An area within one region cannot belong also to another region of the same map. Any grouping of disconnected areas into classes on the basis of common attributes (e.g., areas which would be shown with the same map colour) must be done outside this structure.

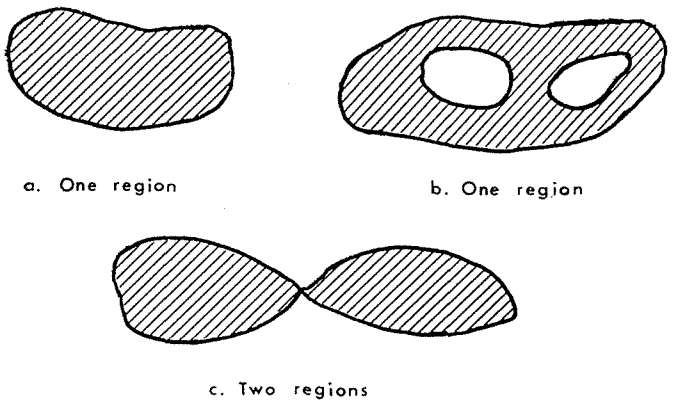


Figure 2. Definition of a region.

Region boundaries are represented as polygons, curved boundaries being approximated by polygons with arbitrarily large numbers of sides. This is a practical restriction, compatible with the usual methods of input and output of computer graphics, although not achieving the storage economies possible with, for example, the piecewise fitting of polynomial functions.

A **point** (P) is a vertex belonging to only two polygons, with two adjacent boundary vertices. A **junction** (J) is a vertex belonging to three or more polygons, with three or more adjacent boundary vertices. An **edge** (E) is a straight line joining two adjacent boundary vertices. A **boundary part** (BP) is that part of a boundary between adjacent junctions. (See Fig. 3.)

3. Region structure

Fig. 4 illustrates in lattice form, the relationships between the parts (as defined above) of the region R_1 of Fig. 3. Region R_1 can be broken down into three boundary parts BP_1, BP_2, BP_3; BP_3 comprises edges E_5, E_6, E_7; edges E_5 and E_6 share a common vertex P_3; etc. By convention, J_3 is shown twice. We have not attempted to show the interrelations between this region and the adjacent regions R_2, R_3 and R_4, because

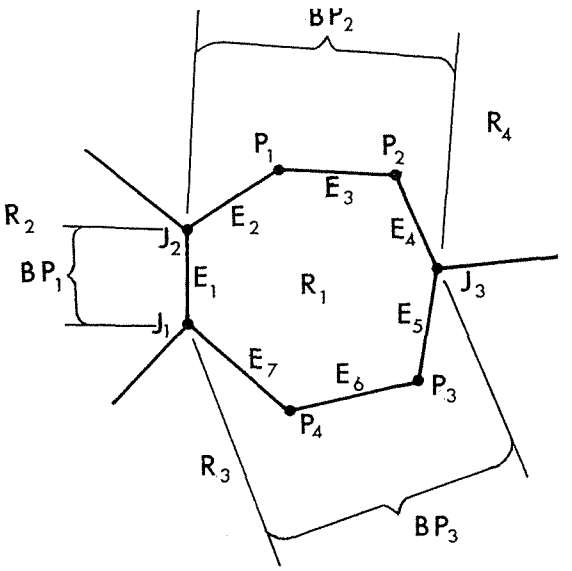


Figure 3. Definition of region parts.

of the difficulties of displaying these in two dimensions : each boundary part, edge and point is common to two regions, and each junction to three or more. This complex of interrelations between region lattices, reflecting the complexities of a region map, suggests that normal hierarchical list structures are not suitable for representing the structure of a region map, as was concluded by Pfaltz et al. (1967). A structure such as the "plex" of Ross (1961) appears more appropriate, and this is the type of structure adopted.

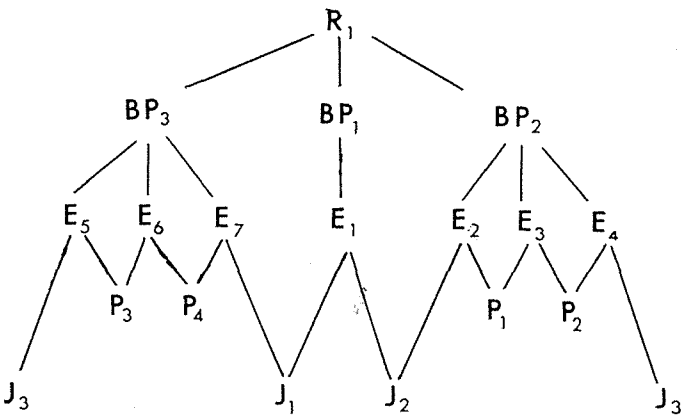


Figure 4. Relationships between region parts.

In developing such a plex structure to represent region boundaries, we must decide which parts should be explicitly represented as elements of the data structure, and to which of these the positional information should be attached.

The positions of the points and junctions must be defined, if only as attributes of other elements, and here we retain them explicitly in the system. Since most edges and boundary parts are completely defined by ordering the points between adjacent junctions, they are omitted from the structure and adjacent points linked into lists, terminating at adjacent junctions (see Fig. 5). The special case of the boundary part which consists of only one edge, i.e., two adjacent junctions with no intervening points, must still be provided for. (The exclusion of edges and boundary parts from the lattice representation considered earlier would not have appreciably lessened the complexities discussed.)

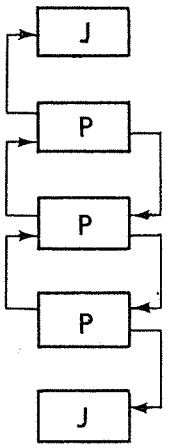


Figure 5. A list of points.

In traversing the boundary of a region we can now move without difficulty along a list of points to a junction, but we need a way of choosing the correct path beyond a junction. In Fig. 6, on arriving at J_1 via P_4, we must choose between P_5 and J_2 as the next vertex of the polygon being traversed. If we are moving around a region boundary in a clockwise sense we must keep to the right, i.e., look anti-clockwise around J_1 from P_4 for the next vertex J_2. If we are moving in an anti-clockise sense we keep to the left, i.e., look clockwise around J_1 for the next vertex P_5. Therefore, the three points P_4, J_2, P_5 adjacent to J_1 must be cyclically ordered to provide a means of looking around the junction and distinguishing the next clockwise or anti-clockwise path. This can be done by linking the vertices adjacent to the junction on a cyclic list, or ring, as in Fig. 7.

If we are to see the context of a region we must be able to identify the regions adjacent to it. Linking the several regions meeting at a junction into a ring (Fig. 8) would enable us to identify pairs of adjacent regions, but to be able to decide which of the vertices P_4, P_5, J_2 is common to a given pair of adjacent regions we must tie at least one ring member to one of the vertices on its boundary, e.g., R_1 to P_4 (Fig. 9).

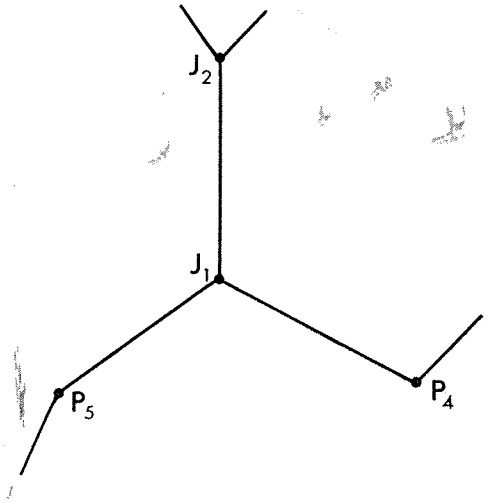


Figure 6. A junction.

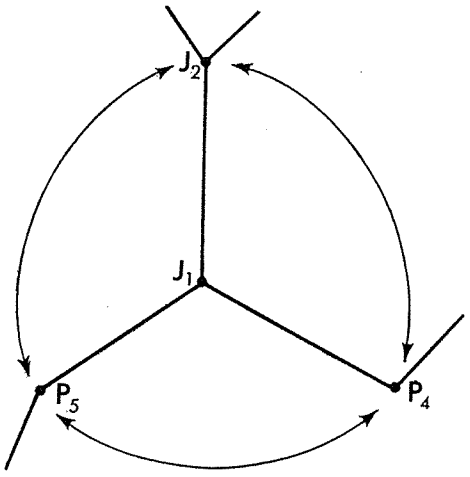


Figure 7. Possible structure—the vertex ring.

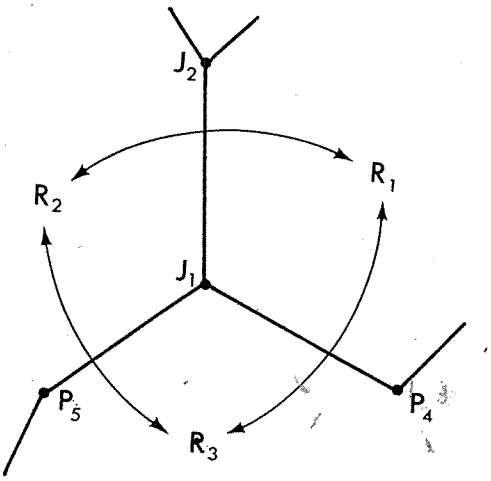


Figure 8. Possible structure—the region ring.

Both the ring structures suggested above raise difficulties. A point may be adjacent to a second junction, and so we have three types of points: points on two rings, points on one ring, and points belonging to no ring, i.e., not adjacent to a junction. A junction can be adjacent to any number of other junctions and so can belong to any number of vertex rings. A region may have any number of junctions on its boundary, and so belong to any number of region rings. Since an element of the structure will need within it at least one pointer for each ring to which it belongs, we would need to provide for variable length elements of the structure, i.e., elements of the structure would themselves be lists. These complications are unacceptable.

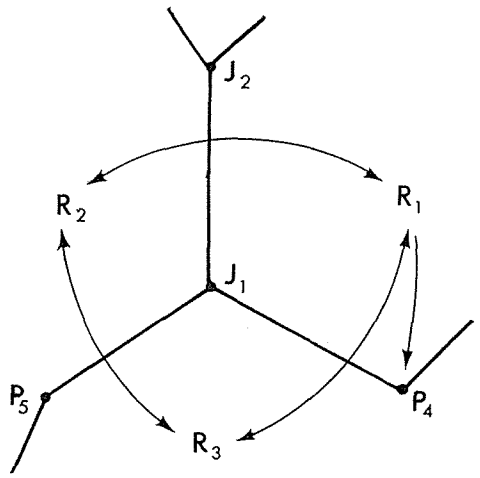


Figure 9. Possible structure—relating the region ring to the vertices.

There is, however, an alternative. We can introduce for each region a set of new elements, called **angles** (A), one to each junction. (Angle is merely a convenient name for this element; it does not have a value in the normal geometric sense.) We form the angles around each junction into a ring and have each angle point to its region, and to one of the vertices adjacent to the junction on its region boundary (Fig. 10). This structure provides the facilities of both the rings previously suggested without the need to introduce variable length elements.

4. Data structure

The data structure devised has four element types: region, angle, junction and point, of which only the latter two have explicit positional attributes. Fig. 11 illustrates the structure about the junction J_1 of Fig. 3.

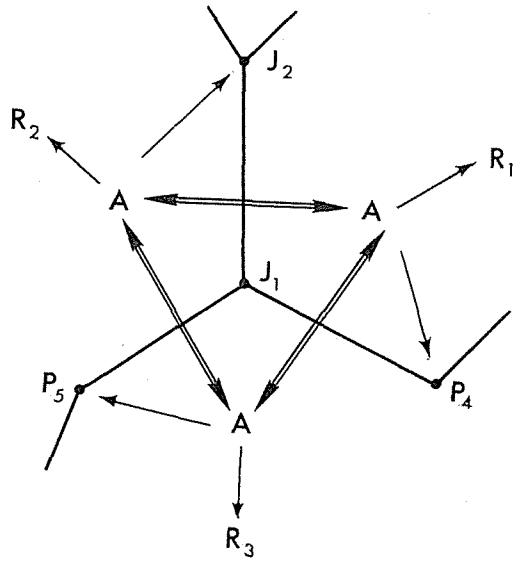


Figure 10. Adopted structure—the angle ring.

Each region points to one (any one) of the junctions on its boundary. This allows us, given a region, to locate a junction on its boundary, and hence locate the other junctions and points by traversing the boundary. (It has been suggested to me that there would be advantages in being able to locate all the junctions of a region without having to traverse the boundary. This could be accomplished by placing each angle on a second ring linking all the angles of a region, and having the region point to this ring instead of to one of its junctions. This feature has not been incorporated in the structure described, but may well be included at a later date.)

Each junction has positional data (x, y) associated with it, and points to one (any one) of its ring of angles.

Each angle points to its region, and to one of the two points or junctions adjacent to its junction on the region boundary (the left-hand one facing the junction). An angle also points to the two adjacent angles around the junction (the **previous angle** (pA) is clockwise and the **next angle** (nA) is anti-clockwise).

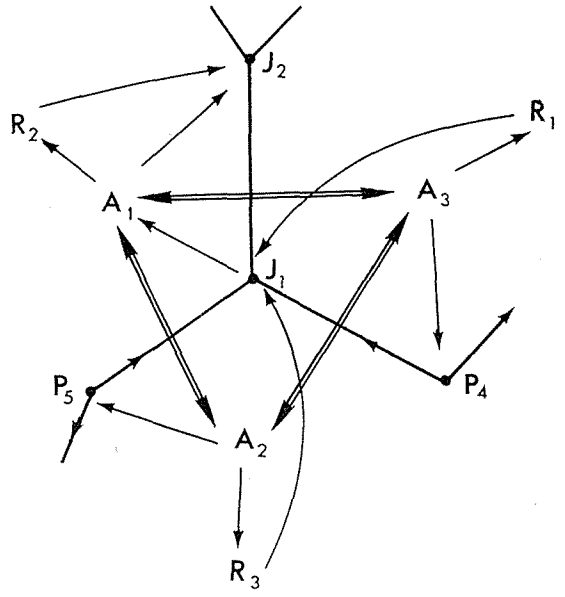


Figure 11. The complete structure at a junction.

Each point has positional data and points to its two adjacent points (or junctions) (the **previous point** (pP) and the **next point** (nP), the sense being arbitrary but consistent between junctions).

Thus each element of the structure contains a number of data, which can be positional data (x, y co-ordinates) or structural data (pointers to, i.e., addresses of, other elements), see Fig. 12. The interconnections between the elements of the structure of Fig. 11 are diagrammed in Fig. 13; outward pointers are shown leaving the appropriate field of an element, but all inward pointers refer to an element block as a whole, not to a particular field within it.

The data structure outlined so far allows simple

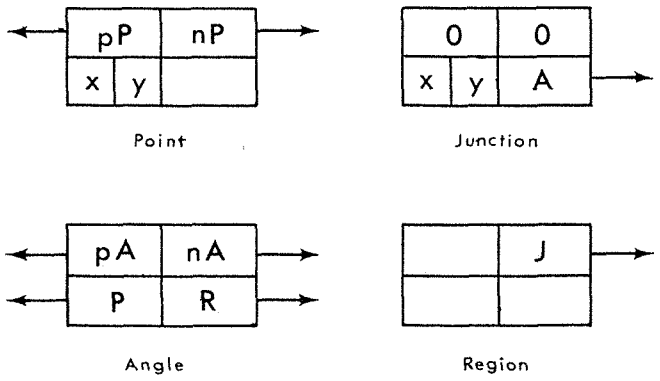


Figure 12. Elements of the data structure.

polygonal boundaries to be traversed without difficulty. A procedure for choosing the next point on a boundary past a junction, and determining the direction along the new point list is given in Fig. 14. **(Notation:** the function z(W), where z is the name of a field of an element of the structure, has the value of the z field of the element at address W. For example, "nextpoint (P)" denotes the value in the "nextpoint" (nP) field of the point element at address P. P_2 and P_1 are on entry the addresses of junction and previous vertex, and on exit the addresses of the next vertex and junction respectively.)

However, the enclosure of one region within another, and hence the possible existence of internal boundaries of a region, is not exhibited by the system as it stands. It is difficult, for example, to traverse the entire boundary of the shaded region of Fig. 2b.

This difficulty can be overcome by introducing **virtual edges** interconnecting the closed polygons which together form a region boundary. This is done by linking one junction on each inner boundary with a junction on the outer boundary, junctions being especially created from points where necessary. These virtual edges are distinguished from **real edges** by having the same region on either side, which is exhibited by adjacent angles in a ring referencing a single region. This is illustrated in Figs. 15 and 16, and the interconnections between elements of the structure in the neighbourhood of the virtual edge are diagrammed in Fig. 17. The entire boundary of any region can now be traversed. We can determine whether the edge beyond a junction is real or virtual by the procedure given in Fig. 18. A complete procedure for plotting the boundary of a region is given in the Appendix.

To complete the structure an outer region is defined which includes all that area within the available coordinate frame (fixed by the size of the x, y fields) not included in other regions. Each inside boundary

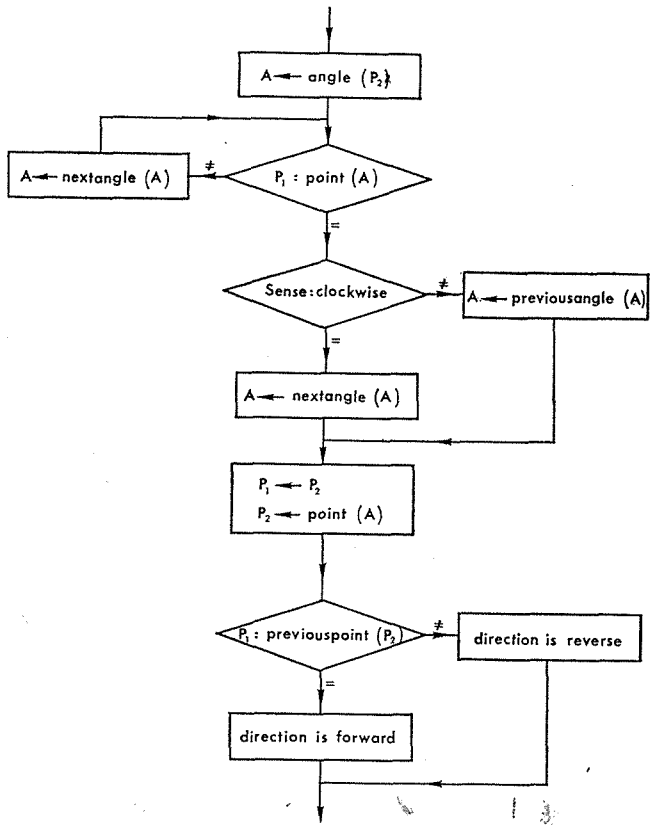


Figure 14. Choosing the vertex beyond a junction.

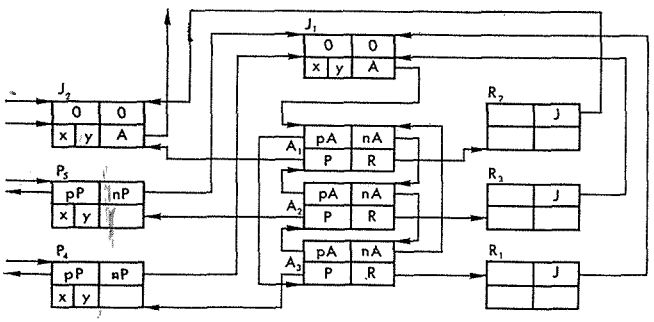


Figure 13. Data structure at a junction.

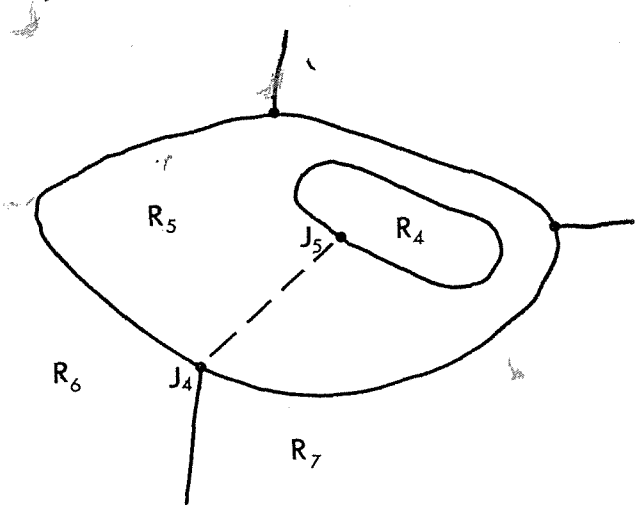


Figure 15. A virtual edge.

of this region (i.e., the outer boundaries of the mapped area) are linked to the (0, 0) corner of the frame by virtual edges, see Fig. 19.

Entry to the structure is by the address of any of its elements. In many applications it will be convenient to associate regions into classes using attribute rings, i.e., rings of elements pointing to regions having some common attribute (c.f. map colour). However, an external table of region names and addresses will usually be needed to allow direct access to particular regions, and to cater for multi-valued attributes. The junction at (0, 0) is a convenient entry to the outer boundary of the map.

In developing this data structure the aim was to exhibit the pictorial information contained in a map of region boundaries. This has been achieved by building a structure having relations between its elements analogous to the pictorial relations between

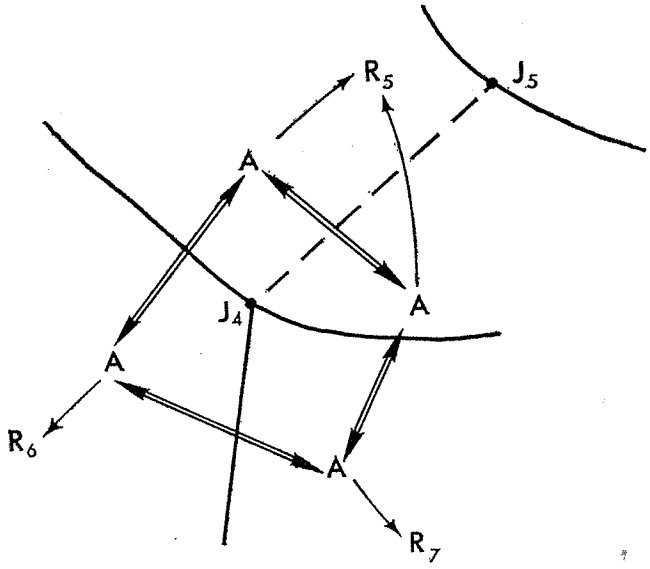


Figure 16. The angle ring at a virtual edge.

map elements: points belong to the boundaries of two regions; junctions belong to the boundaries of three or more regions. Then, the ambiguities introduced by this approach have been resolved by the mechanism of an angle ring at each junction, displaying the cyclic ordering of both regions and adjacent vertices at the junction, and interrelating regions with their adjacent boundary vertices.

5. Implementation

The data structure described does not constitute an efficient method of storage, since each boundary vertex requires one element in the structure. There are several possible approaches to improving the structure in this respect. We can ensure that boundary curves are approximated by polygons with as few sides as possible, within the accuracy required. Further improvements can be made by recording point positions relative to previous points, commencing with the absolute position of each junction, and pack-

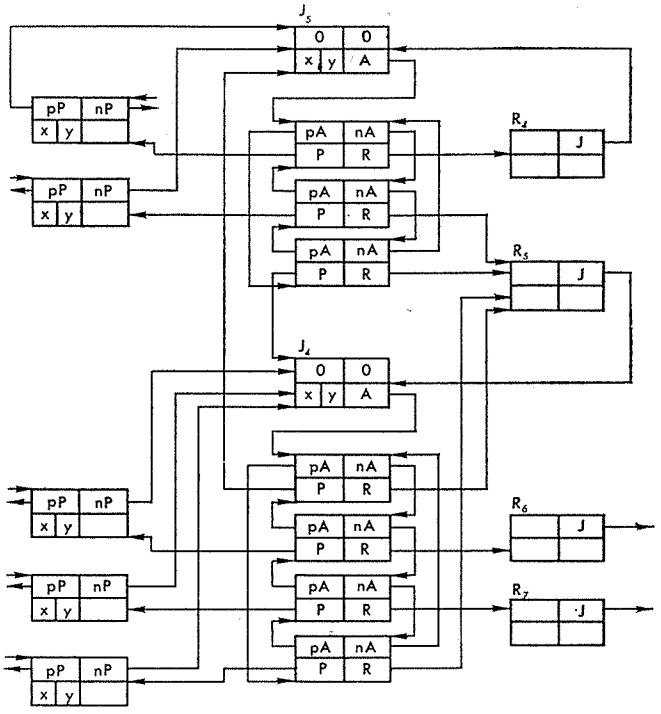


Figure 17. Data structure at a virtual edge.

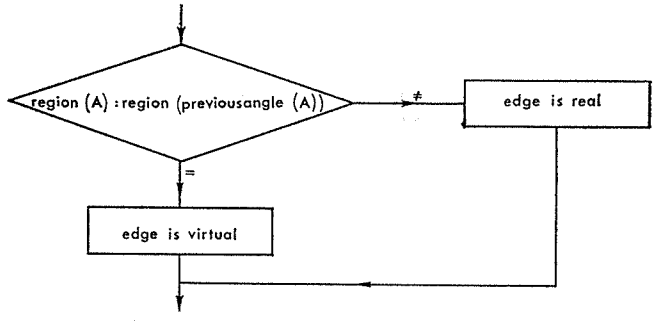


Figure 18. Distinguishing a virtual edge.

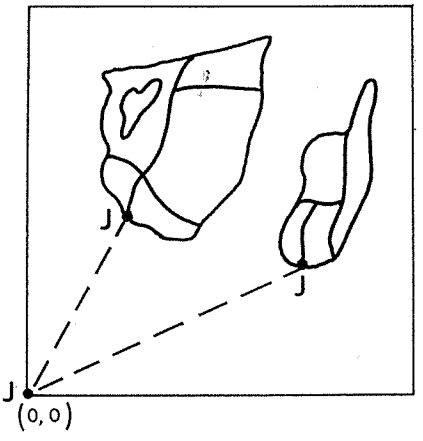


Figure 19. Linking the map to the co-ordinate frame.

ing several adjacent points into the one "point" element of the structure. Another solution is to remove the point strings completely and store them externally, but this is likely to result in a serious loss in processing efficiency where positional data are required.

There appears to be no processor entirely suitable as a vehicle for this representation. List processors are inherently unsuitable as has been concluded earlier. It would certainly be possible to use the ring processor CORAL (Roberts, 1964), but only at the price on considerable structural elaboration and less efficient storage. Implementation at assembly level is planned; the elements of the structure have been kept a uniform size in order to simplify the handling of free-space. It should be stressed that our concern is to develop a tool for research on land resources information. If our aim were research into storage structures it might well be desirable to use a generalised processing language.

To prove the adopted structure, a simple region map has been hand coded into its machine representation, and region boundaries successfully plotted using a CDC 3600 assembly language version of the procedure given in the Appendix. A compact and efficient program was easily achieved: the program occupied 64 words (excluding service routines) and traversed region boundaries at average rates of 20 microseconds per point, and 80 microseconds per 3-fold junction (excluding the time needed to call and execute the plotting routine). No programs have yet been written to recover pictorial information from the data structure: it is expected that these also will be simple and efficient.

6. Conclusion

The representation developed neither aims at nor achieves great efficiency in storing map information. It does, however, achieve its object of allowing easy and efficient access to both the pictorial and positional information in a map of region boundaries. Future work will establish how useful a tool it is for the manipulation of this data.

7. Acknowledgments

I am indebted to Dr. M. B. Clowes for discussions, suggestions and encouragement during the preparation of this paper. I would also like to thank Mr. T. Pearcey, Dr. T. G. Chapman and Mr. R. M. Scott for helpful criticism.

References

1. CHRISTIAN, C. S., and STEWART, G. A. (1953): "General Report on Survey of Katherine-Darwin Region, 1946", CSIRO Aust. Land Res. Ser. No. 1.
2. LOOMIS, R. G. (1965): "Boundary Networks", *Comm. Assoc. Comp. Mach.*, Vol. 8, p. 44.
3. PFALTZ, J. L., SNIVELY, J. W., and ROSENFELD, A. (1967): "Local and Global Picture Processing by Computer." Presented at Symposium on Automatic Photointerpretation, Washington, D.C., May 1967.
4. ROBERTS, L. G. (1964): "Graphical Communication and Control Languages", Proc. Second Congress on the Information System Sciences, p. 211.
5. ROSS, D. T. (1961): "A Generalised Technique for Symbol Manipulation and Numerical Calculation", *Comm. Assoc. Comp. Mach.*, Vol. 4, p. 147.
6. SUTHERLAND, I. E. (1963): "Sketchpad: A Man-Machine Graphical Communication System", Tech. Rep. No. 296, Lincoln Laboratory, MIT, Cambridge, Mass.
7. TOMLINSON, R. F. (1967): "An Introduction to the Geo-Information System of the Canadian Land Inventory", ARDA, Canada Dept. Forestry and Rural Development, Ottawa.

Appendix

```
procedure figure (r, c); value r, c; integer r, c;
comment: figure traverses the boundary of region,
address r, in a clockwise (c = 1) or anti-clockwise
(c = 0) sense, and draws the boundary on a plotter.
The procedure plot (x, y, i) moves the plotter pen from
its current position to position (x, y) with pen up
(i = 1) or down (i = 0). The function z(w), where z
is one of the integer procedures named below, has
the value of the z field of the element at address w
(see Fig. 12);
begin
    integer p1, p2, a, v, f, ip1 ip2; integer procedure
    nextpoint, previouspoint, nextangle, previousangle,
    point, region, angle, junction, x, y; procedure plot;
    comment: given region address r and sense c, find
    an initial junction, address p2, and previous point,
    address p1, on the region boundary;
    p2 := junction(r); a := angle(p2);
t:  if r ≠ region (a) then begin a := nextangle(a);
                                 go to t end;
    if c = 0 then a := nextangle(a);
    p1 := point(a);
    comment: save initial points and initiate plot;
    ip1 := p1; ip2 := p2;
    plot(x(p2), y(p2), l);
    comment: given junction, address p2, and previous
    point, address p1, step p2, p1 through the junction
    (the old p2 becomes the new p1);
m:  a := angle(p2);
n:  if p1 ≠ point(a) then begin a := nextangle(a);
                                 go to n end;
    if c = 1 then a := nextangle(a)
             else a := previousangle(a);
    p1 := p2; p2 := point(a);
    comment: set f = 1 if direction along new point
    list is forward (i.e., nP field indicates next element),
    f = 0 if reverse;
    if previouspoint(p2) = p1 then f := 1 else f := 0;
    comment: set v = 1 if new edge is virtual, v = 0
    if real;
    if region(a) = region(previousangle(a))
        then v := 1 else v := 0;
    comment: plot the new edge;
s:  plot(x(p2), y(p2), v); v := 0;
    comment: test if p2 is a junction. If not, step p2,
    p1 along the point list;
    if nextpoint(p2) ≠ 0 then
        begin p1 := p2;
            if f = 1 then p2 := nextpoint (p2)
                     else p2 := previouspoint(p2);
            go to s end;
    comment: test for completion of boundary traverse;
    if ip1 ≠ p1 v ip2 ≠ p2 then go to m end
```

A Note On Nonlinear Regression Analysis

By Janusz Kowalik*

SUMMARY.—A general analysis of a nonlinear regression problem, utilizing a number of Gauss methods has been performed. These methods have been demonstrated to be unreliable, in that they do not always lead to a solution and the reasons for their failure have been given. It has been suggested that the variable metric method by Davidon, Fletcher and Powell is more satisfactory technique, since its use always gives convergence. There are provided numerical examples which illustrate the relative merits of the Gauss and the variable metric methods.

The general problem involves the fitting of K sets of values

$$y_k, (x_{1k}, x_{2k}, \ldots x_{Rk}) = x_k;\ k = 1, 2, \ldots K \quad (1)$$

to the function $f(x_k, v)$, which is nonlinear in the variables $(v_1, v_2, \ldots v_N) = v$, so as to obtain the values of v, which minimize the sum of squares

$$F(v) = \sum_{k=1}^{K} (f(x_k, v) - y_k)^2 = \sum_{k=1}^{K} d^2(x_k, v) \quad (2)$$

The following notation will be used:
A is (K, N) matrix with components

$$a_{k,\,n} = \frac{\partial f(x_k, v^i)}{\partial v_n} \quad (3)$$

B is a (N, N) matrix

$$B = \sum_{k=1}^{K} d(x_k, v^i) B_k \quad (4)$$

where

$$(B_k)_{j,\,m} = \frac{\partial^2 f(x_k, v^i)}{\partial v_j \partial v_m} \quad (5)$$

and d is a column vector with the elements $d(x_1, v^i)$, $d(x_2, v^i), \ldots d(x_k, v^i)$.

In the neighbourhood of v^i, the nonlinear function $F(v)$ can be approximated by the following quadratic form $F(v^i + \delta) = F(v^i) + 2\delta^T A^T d + \delta^T (A^T A + B)\delta$ (6) whose maximum is attained when δ satisfies the system of equations

$$A^T d + (A^T A + B)\delta = 0 \quad (7)$$

If it be assumed, that the point v^i is near the minimum and that the elements of $d(x_k, v^i)$ are small, or if the components of B_k are small, then matrix B can be ignored. Thus there is obtained a Gauss system of normal equations

$$(A^T A)\delta = -A^T d \quad (8)$$

which can be considered as being accurate only if $f(x_k, v)$ is linear in the variables v.

In order to solve the system (8) it is necessary that v^i is not a stationary point, i.e. $A^T d \neq 0$. The solution gives

$$v^{i+1} = v^i + \delta \quad (10)$$

a new approximation to the minimum of $F(v)$. This iterative procedure is equivalent to the steepest descent method

$$v^{i+1} = v^i - \tfrac{1}{2}(A^T A)^{-1} \operatorname{grad} F(v^i) \quad (11)$$

where the metric is defined by the positive definite matrix $A^T A$. Unfortunately, the matrix $A^T A$ is often ill-conditioned so that the solution of system (8) can give a very large step δ. This can result in the value of $F(v^i + \delta)$ being much greater than $F(v^i)$. Theoretically, it should always be possible to find such $\lambda^i > 0$ that

$$F(v^i + \lambda^i \delta) < F(v^i) \quad (12)$$

but in practice vector $(A^T A)^{-1}$ grad $F(v^i)$ is often almost tangential to the level surface through v^i, and in such a case no improvement of $F(v)$ can be made.

To improve the Gauss method a number of modifications have been suggested. These cause the problem to become well-conditioned and restrict the values of δ to a neighbourhood of v^i, where the above considerations are valid. One of these modifications [1], [2], [8] introduces a new metric matrix

$$M = A^T A + \mu I \quad (13)$$

where μ is a positive scalar and I the unit matrix. It follows that as μ grows, the solution $\delta(\mu)$ of the equation

$$M\delta = -A^T d \quad (14)$$

tends to zero and thus it is possible to find a value of μ for which $v^i + \delta$ lies in an arbitrary neighbourhood of v^i. However, as $\mu \to \infty$ then δ becomes proportional to the gradient vector at v^i. In this case searching for a minimum of $F(v^i + \lambda\delta)$ is equivalent to the method of steepest descent, which is known to give poor convergence.

It has also been suggested that the Gauss method can be stabilised by the introduction of damping factors $0 < \beta^i < 1$, which gives the following iterative procedure,

$$v^{i+1} = v^i + \beta^i \delta \quad (15)$$

where δ is obtained from (8), and $\beta^i \to 1$ when v^i approaches the solution. There are, however, at least two reasons that the minimization along the direction δ,

$$\min_{\lambda^i} F(v^i + \lambda^i \delta) \quad (16)$$

is preferable.

1. The value of $\lambda^i_{\min}$ gives a precise location of a minimum along δ and can be found without costly computations.
2. The optimal step $\lambda^i_{\min}$ is varying from step to step and if an initial guess of β^0 is unfortunate, then the

* *Computer Centre, Australian National University, Canberra, A.C.T.* *Manuscript received May, 1967.*

Gauss procedure with the damping can have a slow convergence, or even a divergence.

For these reasons, although better than the full steps Gauss method, the damping technique is not reliable as a general procedure.

Other modifications [1], [2], [3], [4] of the Gauss method have made use of the fact that problem (8) can be transformed to an equivalent one with a diagonal or triangular coefficient matrix, which can be easily manipulated.

An approach which differs markedly from those mentioned above is the direct use of the conjugate gradient techniques [5], [6], [7]. In our experience Davidon's method has proved to be the most powerful of all gradient techniques*.

This method defines v^{i+1} from v^i according to the relationship

$$v^{i+1} = v^i - \lambda^i H^i \text{ grad } F(v^i) \tag{17}$$

where H^i is a positive definite matrix, and λ^i, is a scalar, which minimizes $F(v^{i+1}) = F(v^i, \lambda^i)$. Davidon's procedure starts with an arbitrary, positive definite H^o, for example the unit matrix, and generates variable metric matrices H^i by simple matrix manipulations. As the minimum is approached, the matrices H^i converge to the inverse of hessian which is defined by

$$h_{m,n} = \frac{\partial^2 F(v)}{\partial v_m \partial v_n} \tag{18}$$

This helps to accelerate convergence in the neighbourhood of the minimum and also provides a ready approximation to the least-squares variance-covariance matrix.

* *Broyden [10] regards Davidon's method as a version of the Newton method.*

Davidon's method is stable in the sense that

$$F(v^{i+1}) < F(v^i) \tag{19}$$

and in addition possesses quadratic convergence, i.e. the minimum of an n-dimensional quadratic is located in n iterations.

The numerical experiments have shown that the method works efficiently even if starting points are only poor approximates to the minimum.

Such experiments were carried out by computing the best fit of data points to the function

$$f(x, v) = \frac{v_1(x^2 + v_2 x)}{x^2 + v_3 x + v_4} \tag{20}$$

which is the general form of an equation that can account for the results obtained for reactions catalyzed by certain allosteric enzymes. The calculations were performed on an I.B.M. 360/50 computer.

The variable metric method has been compared with the Gauss method and its modification in the sense that after solving the set of equations (8) the minimum of $F(v^i + \lambda^i \delta)$ along the vector δ, has been sought.

Two typical results are shown in Table I. These were obtained by computing eleven function values for x in the range of $0{\cdot}0625 - 4$ (f_T, Table II) and using reasonably good starting points.

The latter values were $v_I = (0{\cdot}25, 0{\cdot}39, 0{\cdot}415, 0{\cdot}39)$ and $v_{II} = (0{\cdot}33, 0{\cdot}45, 0{\cdot}516, 0{\cdot}3)$ which may be compared with theoretical values of $v_T = (0{\cdot}2, 0{\cdot}3, 0{\cdot}3, 0{\cdot}18)$. It may be noted that, if the data are ideal and the starting points are close to the solution, then both methods work well, i.e. the final values of v are close to the theoretical values. Further, it is apparent that similar computation times are requested for the two methods. The number of iterations in the modified Gauss method is considerably smaller, but this advantage is offset by more time-consuming procedure of calculating the descent directions, δ. The following standard errors of

TABLE I

Method	Starting Point	Solution	Starting and Final Values of F(v)	No. of Iterations	Execution Time (Sec.)
DAVIDON	I	.199941 .303143 .301524 .181475	4.4×10^{-3} 8.8×10^{-9}	13	4.5
	II	.199912 .303523 .301648 .181530	4.2×10^{-2} 5.7×10^{-9}	18	4.8
MODIFIED GAUSS	I	.199912 .303522 .301648 .181530	4.4×10^{-3} 5.7×10^{-9}	5	4.1
	II	.199912 .303523 .301648 .181530	4.2×10^{-2} 5.7×10^{-9}	4	4.0

TABLE II

X	4	2	1	.5	.25	.167	.125	.1	.0833	.0714	.0625
f_T	.1979	.1925	.1757	.1379	.0866	.0605	.0456	.0364	.0301	.0257	.0224
f_D	.1957	.1947	.1735	.16	.0844	.0627	.0456	.0342	.0323	.0235	.0246
f_S	.1944	.1928	.1824	.1489	.0927	.0625	.0456	.0354	.0287	.0241	.0207

$v_{opt} = (0{\cdot}199912, 0{\cdot}303523, 0{\cdot}301648, 0{\cdot}181530)$ were obtained; (a) $7{\cdot}01 \times 10^{-5}$, $1{\cdot}56 \times 10^{-3}$, $7{\cdot}35 \times 10^{-4}$, $9{\cdot}24 \times 10^{-4}$ and (b) $4{\cdot}92 \times 10^{-5}$, $3{\cdot}03 \times 10^{-3}$, $5{\cdot}14 \times 10^{-4}$, $6{\cdot}07 \times 10^{-4}$. The former were calculated using the variance–covariance matrix $(A^TA)^{-1}$ and the latter using the final value of the inverted hessian H^i.

When the data to be fitted are inaccurate due to experimental limitations, the modified Gauss method very often does not converge to the solution. The matrix $(A^TA)^{-1}$ is a poor approximation of $(A^TA + B)^{-1}$, shows serious ill-conditioning and thus is an unfortunate metric matrix. This would suggest that the Gauss method is not likely to work, when the experimental function values y_k are considerably in error and consequently the differences $d(x_k, v^i)$ are not small enough to eliminate matrix B from the equations (7). Unfortunately, a precision of the observed data is usually limited by a particular technique of measurement.

Davidon's method works well with the same efficiency, when function values y_k vary from ideal values. It has been found that observed data, which could not be analysed by Gauss method, could be analysed by Davidon's procedure. The results indicate that some points were considerably in error with respect to the standard error of the overall fit. When these were eliminated, both the Gauss and Davidon methods of analysis yielded identical, satisfactory results. The ability of Davidon's method to work in case of poorly determined parameters has great advantage that it permits this behaviour to be detected.

TABLE III

variables / number of steps	v_1	v_2	v_3	v_4	F(v)
START	.25	.39	.415	.39	5.31×10^{-3}
5	.1835	.5466	.2729	.2786	3.95×10^{-4}
15	.1916	.2132	.1148	.1499	3.12×10^{-4}
30	.1928	.1913	.1230	.1360	3.07×10^{-4}

The following numerical example illustrates a failure of the Gauss method, when the experimental values of y_k are in error. The exact values of $f(x, v_T) = f_T$ have been disturbed by $\pm 10\%$ of the lowest f_T, so that $f_D = f_T \pm 0{\cdot}0022$, except $f_D(0{\cdot}5) = f_T(0{\cdot}5) + 0{\cdot}0221$ and $f_D(0{\cdot}125 = f_T(0{\cdot}125)$ (see Table II). For the starting point $v_{III} = (0{\cdot}25, 0{\cdot}39, 0{\cdot}415, 0{\cdot}39)$ the Gauss method failed to converge, while Davidon's method successfully converged to $v_S = (0{\cdot}1928, 0{\cdot}1913, 0{\cdot}1230, 0{\cdot}1360)$, (Table III). Values of $f(x, v_S) = f_S$ are shown in Table III.

The numerical example, which illustrates a similar case, but when real experimental data were considered, is given in the reference [9].

REFERENCES

1. MORRISON, David D.: Methods for Nonlinear Least Square Problems and Convergence Proofs, JPL Seminar Proceedings, Space Technology Laboratories, Inc., Los Angeles, California, 1960.
2. FEDER, D. P.: Lens Design Viewed as an Optimization Process, Recent Advances in Optimization Techniques, ed. by A. Lavi and Th. P. Vogl; John Wiley & Sons, 1966.
3. PEGIS, R. J., GREY, D. S., VOGL, T. P., and RIGLER, A. K.: The Generalized Orthonormal Optimization Program and its Applications, Recent Advances in Optimization Techniques, ed. by A. Lavi and Th. P. Vogl; John Wiley and Sons, 1966.
4. JENKINS, M.: Computation Centre, Stanford University, work in progress.
5. DAVIDON, W. C.: Variable Metric Method for Minimization, A.E.C. Research and Development Report, ANL-5990, 1959.
6. FLETCHER, R., and POWELL, M. J. D.: A Rapidly Descent Method for Minimization, Comp. Journ., 6, 1963.
7. FLETCHER, R., and REEVES, C. M.: Function Minimization by Conjugate Gradients, Comp. Journ., 7, 1964.
8. BERZTISS, A. T., and THORNEYCROFT, R. T.: Some Experiments in Nonlinear Estimation, Proceedings of the Australian Computer Conference, Canberra, 16-28 May, 1966.
9. KOWALIK, J., and MORRISON, J. F.: Analysis of Kinetic Data for Allosteric Enzyme Reactions as a Nonlinear Regression Problem (to be published).
10. BROYDEN, C. G.: A Class of Methods for Solving Nonlinear Simultaneous Equations, Mathematics of Computation, 19, 1965.

IFIP—Its Structure And Its Aims

By Dr. A. P. Speiser*

President, International Federation for Information Processing

The International Federation for Information Processing (IFIP) was born out of the computer revolution. This revolution is far more than the introduction of the electronic computer into the engineering and mathematical world, as was originally anticipated. It consists of the penetration of digital machines into almost every activity, be it human or mechanical, where formally encoded information (so-called data) has to be handled or processed in any way. Besides the more obvious fields such as accounting and scheduling, the computer is taking over more and more subjects previously considered to be the exclusive domain of the human mind. Computers are now programmed to translate languages, to compose music, and the term "artificial intelligence" aptly describes a discipline of research which attempts to find out how far machines can ultimately penetrate into the area of human mental activity.

ORIGINS

In 1959, ten years after the inauguration in the United Kingdom of the world's first stored program computer, UNESCO sponsored the first International Conference on Information Processing. This took place in Paris. In the planning meetings during the year preceding the conference, national representatives from nine countries met to organise the scientific program and to develop an international exhibition of information processing equipment. At the conference, almost 2,000 delegates from 37 countries attended the scientific presentations.

Even before the conference, it was apparent that future international meetings and other activities were essential to the development of information sciences throughout the world and that the need existed for a suitable international body. Appropriate statutes were drafted, and IFIP came into existence in January, 1960, with 13 national societies as members.

WHAT IS IFIP?

The International Federation for Information Processing is a multi-national federation of professional-technical societies, or groups of societies, concerned with information processing. Its membership currently comprises 26 of those countries most active in the information sciences. The aims of IFIP are:

- To sponsor international conferences and symposia on information processing, including mathematical, engineering and business aspects.
- To establish international committees to undertake special tasks falling within the spheres of action of its national member societies.
- To advance the interests of member societies through international co-operation in the field of information processing.

In achieving these aims, IFIP fulfills the need for better world-wide communication and increased understanding among scientists of all nations of the role which information processing can play in accelerating technical and scientific progress. Emphasis is clearly on the term "international". In the General Assembly the rule "one country—one vote" is established, thereby giving each national group, whether large or small, equal rights. Officers are elected by the General Assembly from among its members.

IFIP CONGRESSES

Every three years IFIP holds a world-wide congress following the pattern of the Paris conference. Congress 62 in Munich and Congress 65 in New York (over 5,000 persons attended the latter) were successful in attracting the world's leading specialists in all fields of information processing. IFIP Congress 68 is to be held in Edinburgh and a program is being prepared which will include the latest scientific and technical developments. The proceedings of IFIP Congresses are published in book form. The published proceedings are of lasting value and have become prime reference tools for researchers in all countries.

TECHNICAL COMMITTEES

In a continuing program devoted to promoting the world-wide development of the information sciences, IFIP has established a number of technical committees and working groups whose influence is now strongly felt at both international and national levels.

* *Dr. A. P. Speiser is Director of Research with A. G. Brown Boveri & Company, Switzerland.*

IFIP TC-1 Terminology

Every new field develops a jargon of its own for quick and easy communication. In the earlier days of information processing each computer-development group made its own contributions to the jargon, and each language group had its own jargon in this field. The first Technical Committee established by IFIP was formed to establish a terminology of computers and data-processing devices, equipment, media and systems, to reduce misunderstandings and to promote accuracy in the exchange of information.

To avoid duplication of effort, the IFIP Technical Committee 1 on Terminology was affiliated with a similar committee within the International Computation Centre in Rome, forming "IFIP/ICC Technical Committee 1, Terminology". Its work has been directed toward developing a multilingual vocabulary of information processing concepts and terms. Current plans call for the publication of a number of monolingual volumes, each consisting of two parts. The first part contains the definitions of over 1,500 information processing concepts for which there is an international consensus. These concepts are defined in the language of each volume. Appended to each concept is a term that connotes the concept. Entries of the concepts are arranged so that related objects and ideas are grouped together for easy reference. The arrangement of the concepts is the same in all volumes. Each entry bears a unique key consisting of a letter and a numeral, which is also identical in all volumes.

The second part of each monolingual volume consists of an alphabetical listing of terms in the language of the volume. Listed with each term is the unique key of the concept to which the term applies.

The English volume was published in 1966 and has found wide acceptance. Translations into Danish, Dutch, French, Italian, Russian, Spanish and other languages are under way.

IFIP TC-2 Programming Languages

While computers have their own, or machine, languages, there are also many special languages by which man communicates with the great variety of computers, and orders their operations. With the dynamic growth of information processing, a great proliferation of languages created another Tower of Babel. Men could not communicate with machines without learning a machine's particular language. Computers had very little communication compatibility with each other, even computers produced by the same manufacturer. While manufacturers, users' groups and government organisations struggled with the problem of devising common languages oriented to users' needs, it was evident that the field was too new to accept tight standardisation; and that the language framework must be flexible to permit future refinement and expansion.

IFIP Technical Committee TC-2 was established to promote the development specifications, and refinement of common programming languages with provisions for future revision, expansion and improvement.

IFIP TC-3 Education

Until IFIP formed its Technical Committee 3 on education, there was no world-wide central clearing house for educational material about the information sciences. A typical project of TC-3 is its joint action with the International Computation Centre in Rome, in organising a Seminar in Automatic Data Processing during 1965/66. This Seminar lasted about six months; its purpose was the education of teachers from the developing countries. Further seminars of this kind are in the planning stage. These seminars are typical examples of the benefits IFIP can provide for those countries in which the computer sciences are less well developed.

Plans are also being made for training scientists from other disciplines in the use of information processing to advance their own individual technologies. Part of this plan calls for promoting a working liaison with international organisations having allied interests so that the groups may co-operate in projects that will further mutual aims.

INTERNATIONAL CO-OPERATION

While the work of IFIP in formation-processing sciences inevitably impinges on the activities of other international organisations where information processing is having an influence, this is viewed as a rich opportunity for co-operation and co-ordination of activities in order that common goals may be achieved more effectively and more quickly. For example, two conferences on "Application of Digital Computers for Process Control" jointly organised by IFIP and the International Federation for Automatic Control (IFAC) were held in Stockholm in 1964 and in France in 1967. A number of further conferences, jointly organised with other societies, have taken place or are being planned. As a matter of policy, the proceedings of all these conferences are published and made available to the scientific world.

SPECIAL INTEREST GROUPS

The formation of Special Interest Groups, the constitutional basis for which was created in 1966, marks an important step in IFIP's evolution. Such Groups have as their members national special interest groups, with the restriction that there shall be only one such Group in each country. Special Interest Groups have a greater degree of independence and autonomy in the formulation and implementation of their activities than Technical Committees. They propose their own program which is submitted to the IFIP General Assembly for approval.

The first IFIP group in this category is the IFIP Administrative Data Processing Group. Its purpose is to promote and co-ordinate research, education and exchange of experience in the field of information processing as applied to organisational, economic and administrative problems in public and business administration. This constitutes a welcome opportunity for IFIP to extend its activities into the area of the administrative (as against the predominantly scientific) sphere. The need for such a broadening has been felt in several of our member countries. A part of the Group's activities will extend into the field of Education; this portion will be closely co-ordinated with the IFIP Technical Committee on Education.

IFIP'S FINANCIAL STRUCTURE

The Federation's only regular financial resource is the membership contributions paid by the 26 member societies. Further income results from royalties from publications. Congresses and conferences are budgeted on a break-even basis, assuming a reasonable attendance figure. In some cases attendance has been higher than expected, which resulted in a surplus.

This is indeed a narrow basis for operation and one might rightly ask how the Federation has been able with such modest resources to gain the position which it occupies and to conduct activities as diverse as those described. The answer is both simple and impressive: Over the past eight years countless men and women have devoted time and effort far beyond the call of duty and have spent weeks or even months to serve on committees and groups on behalf of IFIP. IFIP has been fortunate to enlist their services, as well as the support of their employers. To them IFIP owes its success and we are confident that this all-important support will continue.

The tasks computers do transcend national boundaries and the response to the work IFIP has already accomplished demonstrates the value of an international forum where common problems can be discussed. IFIP will continue this work for the benefit of all countries where information processing is growing and developing.

The Fourth Australian Computer Conference

The fourth computer conference under the auspices of the Australian National Committee on Computation and Automatic Control (ANCCAC) will be held in Adelaide from August 12th to 16th, 1969. The organisation and running of the conference has been delegated to the Australian Computer Society which has entrusted the matter to its South Australian branch.

As in the case of the 1966 conference held in Canberra two committees have been established, an organising committee to deal with policy matters and an executive committee concerned with the detailed work arising from the decisions of the former committee.

Membership of the committees are as follows:

Organising Committee:

Sir Arthur Rymill, Chairman.
Sir Henry Basten, Deputy Chairman.
Sir Ian McLennan, Broken Hill Proprietary Ltd.
Sir James Vernon, Colonial Sugar Refinery.
Sir Frederick White, C.S.I.R.O.
Sir James Holden, General Motors-Holden's Pty. Ltd.
Sir Edward Hayward, John Martin & Company Limited.
Professor G. M. Badger, Vice-Chancellor, University of Adelaide.
Professor P. H. Karmel, Vice-Chancellor, Flinders University.
Dr. M. W. Woods, Weapons Research Establishment.
Mr. K. M. Archer, Commonwealth Statistician.
Mr. A. B. Barker, Chairman, Municipal Tramways Trust.
Mr. D. H. Brown, Chrysler Australia Limited.
Mr. L. J. Cohn, Chairman, ANCCAC.
Mr. M. L. Dennis, S.A. Public Service Commissioner.
Mr. D. G. McFarling, Advertiser Newspapers Limited.
Mr. T. Pearcey, President, Australian Computer Society.
Mr. B. J. Giles, Chairman, Executive Committee.
Mr. R. O'Brien, Secretary.

Executive Committee:

Mr. B. J. Giles, Chairman.
Professor J. A. Ovenstone, Deputy Chairman (Scientific).
Mr. P. R. Giblett, Deputy Chairman (Commercial).
Mr. P. R. Giblett, Chairman, Publicity Sub-Committee.
Dr. G. W. Hill, Chairman, Programme Committee.
Mr. P. C. Cornish, Chairman, Exhibits.
Mr. M. J. Shertock, Chairman, Conference Evaluation.
Mr. W. E. Skinner, Chairman, Accommodation & Facilities.
Mr. P. R. Benyon, Chairman, Special Activities.
To be Appointed: Chairman, Finance.
Mr. R. O'Brien, Secretary.

The International Federation For Information Processing (IFIP)

The fourth IFIP Congress is to be held in Edinburgh, United Kingdom, from 5th to 10th August, 1968, under the Patronage of Her Majesty the Queen. The Congress organising committee has called for papers in the categories of mathematics, software, hardware applications and education. In addition to the presentation of invited papers and panel discussions, a large part of the program will be reserved for submitted papers of 20 minutes duration. During the week of the Congress an exhibition will be held in the Waverley Market, in the centre of Edinburgh, highlighting current computer technology and future trends. This will include a comprehensive display of computing equipment, services, techniques and applications.

The Chairman of the Program Committee is F. Genuys, IBM France, Boite Postale 82-01 75-Paris, France, and press enquiries may be directed to John Fowler & Partners Ltd., Grand Buildings, Trafalgar Square London WC2. Other enquiries and applications for registration may be directed to IFIP Congress Office, The British Computer Society, 23 Dorset Square, London, N.W.1, England.

Australian Representatives in the various areas are as follows:—

TA1: Computing methods and analysis, computing methods and algebra, combinatorial and discrete mathematics, theory of machines, theory of algorithms—Professor J. A. Ovenstone, Computer Centre, University of Adelaide, Adelaide, S.A. 5001.

TA2: Operating systems, programming languages, compilers, etc., parallel programming, data structures—Professor J. M. Bennett, Basser Computing Department, University of Sydney, Newtown, N.S.W. 2042.

TA3: Analogue and hybrid computers, computing systems, real-time systems, components and circuitry, graphical display and input, data transmission—Professor M. W. Allen, Department of Electrical Engineering, University of New South Wales, Kensington, N.S.W. 2033.

TA4: Applications to the physical and life sciences, applications to engineering, applications to linguistics, artificial intelligence, applications to library sciences — Dr. F. Hirst, Computing Department, University of Melbourne, Parkville, Vic. 3052.

TA5: Applications to Management and Business, applications to social sciences, applications to the arts and humanities, applications to education—Dr. D. Fenna, ADP Branch, P.M.G.'s Dept., 31 Flinders Lane, Melbourne. 3000.

In addition, Dr. C. Bellamy, Director of the Computer Centre, Monash University, Clayton, Vic., is the Australian corresponding member of Technical Committee, TC-3, on Education.

Obituary

The death of Professor Sir Thomas MacFarland Cherry, F.R.S., occurred on 21st November, 1966.

Born in 1898, Professor Cherry was educated at Scotch College, Melbourne; University of Melbourne and the University of Cambridge, England. He became Professor of Mathematics in the University of Melbourne at the age of 31 and remained at Melbourne until his retirement at the end of 1963.

His contributions to the field of mathematics were outstanding and in recognition of his work he was elected a Fellow of the Royal Society in 1954, had the degree of Doctor of Science conferred by the University of Cambridge and received honorary degrees from the Australian National University and the University of Western Australia. A foundation Fellow of the Australian Academy of Science, he was a member of its first Council and became its President in 1961. He was knighted in 1965.

In 1955, Professor Cherry was instrumental in arranging the transfer of the computer CSIRAC from the C.S.I.R.O. to the University of Melbourne, and he served on the CSIRAC administration committee until his retirement. He was an intensely practical mathematician, and as such was deeply interested in numerical analysis. At an age when most men are looking forward to retirement, he became an expert computer programmer in the days when the only computer languages were binary machine languages and the only computations scientific. Professor Cherry was an excellent lecturer and his lectures were always stimulating and thought-provoking.

When the Victorian Computer Society was inaugurated on 15th February, 1961, Professor Cherry was elected Foundation President, which office he held for two years. In the early days of V.C.S., he exerted a great deal of effort in fostering the infant Society and, in recognition of his endeavours, he was created the first Honorary Member of the newly formed Australian Computer Society in 1966.

His interests were not all academic; he retained a life-long interest in the Boy Scout movement, in which he was a Rover leader, and he was one of the founders of the Melbourne University Mountaineering Club. Rock-climbers know "Cherry's flake", a steep rock pitch named after its first conqueror.

Our sympathy is extended to Lady Cherry and her daughter.

—R.P.H.

The Standards Association Of Australia

The Standards Association of Australia (SAA) has promoted active participation by Australian interests in the International Standards Organisation (ISO) and the International Electrotechnical Commission (IEC) Sub-Committees TC97 and TC53 on Computers and Information Processing. The SAA is a member body of both ISO and IEC, and has established a Committee MS/20 to advise on the various topics in the computer and information processing field covered by ISO and IEC.

The first meeting of MS/20 took place in Canberra on 13th September, 1966, and set up a number of Sub-Committees to review and abstract for the SAA Monthly Information Sheet the work of ISO/TC97 and IEC/TC53 Sub-Committees. The current standards activities are as follows:

SC1 — Vocabulary: To provide a multilingual glossary for information processing systems and related subjects covered in the general scope of ISO/TC97 and, where appropriate, abbreviations and letter symbols.

SC2—Character sets and Coding: The standardisation of character sets, character meanings, the grouping of character sets into information, coded representation and the identification of it for the interchange of information between data processing systems and associated equipment; also to report on the problems related to analog devices.

SC3—Character Recognition: The standardisation of input and output character forms for the interchange of information between data processing equipments and associated equipments utilising only humanly-legible printed character sets, i.e., character recognition. This sub-committee has two working groups, on Optical Character Recognition and on Magnetic Ink Character Recognition.

SC4—Input-Output: The standardisation of those physical characteristics of input-output media which are required for the interchange of digital and/or coded information among information processing systems and systems of associated equipment. There are four Working Groups: Magnetic Tape, Punched Cards, Punched Tape, Input-Output Equipment.

SC5—Programming Languages: The standardisation and specification of common programming languages of broad utility, with provision for revision, expansion and strengthening, and for the definition and approval of test problems. There is a Working Group on Programming Languages for Numerical Control of Machine Tools.

SC6—Digital Data Transmission: To determine and define the system parameters governing the operational action and reaction between communication systems and digital data generating and receiving systems.

SC7—Problem Definition and Analysis: To establish appropriate standards on definition and analysis of information processing problems in order to define the means, the format, the context and other techniques which will provide a representation of these problems.

SC8—Numerical Control of Machine Tools: Any standardisation problem concerning numerical control of machines.

Working Group K: Data Elements and their Coded Representation: The standardisation of data elements and their coded representation to facilitate information interchange and data processing.

Advisory Committee on Patents: To serve as the ISO/TC97 liaison with the ISO/IEC Committee on Patents, and to provide a focal point for questions relating to patents in ISO/TC97.

As they become available we will endeavour to publish titles of ISO documents received.

The scope of these Sub-Committees and the current status of development of standards will be found in ISO/TC97 (Secretariat-101)156 of April, 1967. Copies of published ISO and IEC documents on computers and information processing may be obtained from:—

Mr. C. McGuinness,
Standards Association of Australia,
Temple Court,
422 Collins Street,
MELBOURNE, VIC. 3000.

A charge is made for the copying and is related to the size of the document. Mr. McGuinness is also the Engineer Secretary of MS/20 and will channel comments on ISO/IEC documents to the appropriate MS/20 sub-committee. The following documents were recently made available.

ISO/TC97(Sec.-65)103. Draft ISO Proposal: Implementation of 6 and 7 bit coded character set on punched tape.

ISO/TC97/SC2(Sec.-70)226. Summary of voting on Second ISO Draft Proposal 97/2N216: Implementation of the 7 bit coded character set on 12 row punched cards.

ISO/TC97/SC2(Sec.-73)238. First ISO Draft Proposal: Magnetic tape labels and file structure for information exchange.

ISO/TC97/SC4/WG1(USA-10)87. U.S. Comments on German CRC proposal. (Deals with cyclic error codes for magnetic tape—Ed.)

ISO/TC97/SC2(ECMA-17)223. Draft proposal for coding of character sets for magnetical and optical character recognition.

ISO/TC97/SC2(Paris '67-14)265. Draft resolutions taken during the meeting of sub-committee ISO/TC97/SC2 Character sets and Coding (Paris 13th to 16th March, 1967).

ISO/TC97/SC2(Germany-9)236. German considerations on basic rules of an 8 bit code.

ISO/TC97/SC4(Secretariat-35)65. Fourth draft ISO proposal: Specifications for 7 track 200 RPI magnetic tape. (45c)

ISO/TC97/SC4(Secretariat-36)66. Third draft ISO proposal: Specifications for 9 track 200 RPI magnetic tape. (45c)

ISO/TC97/SC4(Secretariat-37)67. Second draft ISO proposal: Specifications for unrecorded magnetic tape (200 & 800 RPI, NRZI). (60c)

ISO/TC97/SC1(Sec.-19)48. Second working document on General Terms.

ISO/TC97/SC4/WG1(Secretariat-34)85. Second draft proposal: Specifications for 9 track 800 RPI magnetic tape.

ISO/TC97/SC4/WG1(USA-12)89. Proposed USA standard: Recorded magnetic tape for information interchange (800 CPI, NRZI). ($1.15)

ISO/TC97/SC4/WG1(USA-11)88. USA working paper on Skew and comments on document ISO/97/4/1 N 68. (55c)

ISO/TC97/SC4/WG1(USA-10)87. US comment on German CRC proposal. (35c)

ISO/TC97/SC4/WG3(USA-8)65. Proposed American standard for take-up reels for one inch perforated tape for information interchange.

ISO/TC97/SC4/WG3(Secret.-21)71. General requirements for data interchange on punched paper tape.

ISO/TC97/SC4/WG2(Sec.-18)51. Second ISO draft proposal: Specifications for unpunched paper cards.

ISO/TC97/SC4/WG2(Sec.-19)52. Second ISO draft proposal: Dimensions and location of rectangular punched holes in 80 column punched paper cards. ($1.30)

ISO/TC97/SC4/WG2(Sec.-17)50. Draft report for the second meeting of the working group ISO/TC97/SC4/WG2 punched cards.

ISO/TC97/SC5(USA-25)176. Proposed U.S.A. Standard Cobol. ($26.00)

Branch Notes

It is intended to set aside in the Journal, space for Branch notes that may be of interest or topical at the time of publication. Such notes could appropriately include past or future Branch programs, notices or reports of seminars, brief reports of special interest group activities and other items on personalities or Branch endeavours that may have a general news value. Branches will be given appropriate notice of the dates at which issues are scheduled to appear; generally, it will be necessary to have material for publication in the hands of the Editor about two months ahead of the date of issue. Not all Branches are represented in this first issue and it is hoped that a better coverage will be presented in future issues.

CANBERRA

The following lectures have been given at General Meetings of the Canberra Branch during 1967.

February, 1967—The Organization of Programs, Sentences and Pictures—Dr. M. B. Clowes, C.S.I.R.O.

March, 1967—The Problem of Speech Recognition by Computers—Mr. T. Pearcey, C.S.I.R.O.

April, 1967—The Auditor and ADP—Mr. D. G. Lloyd, Commonwealth Audit Office.

May, 1967—Computers in the Population Census—Mr. H. F. Brophy, Bureau of Census and Statistics.

June, 1967—Modern Developments in Computers in the U.S.S.R.—Professor A. D. Smirnov, Academy of Sciences, U.S.S.R.

July, 1967—Careers and Training in the EDP Field—Mr. B. Z. DeFerranti, I.C.T., Australia.

August, 1967—RAAF Supply Management—What Is Wanted from Data Processing and How We Are Going About Getting It—Group-Captain A. T. McHutchison, RAAF EDP Centre.

August, 1967—ADP File Systems, with Special Reference to Storage, Retrieval and Linkage—Mr. S. Nordbotten, Central Bureau of Statistics, Norway.

September, 1967—Picture Processing by a Computer—Professor A. Rosenfeld, University of Maryland, U.S.A.

October, 1967—Simscript—A Tool for Decision-makers—Mr. J. S. Armstrong, Department of Forestry, ANU, and Mr. L. P. Tognetti, Research School of Biological Sciences, ANU.

November, 1967—System Concepts and Software Design of CARBINE—Mr. J. Marquet, Control Data, Australia.

December, 1967—Comments on Some Legal Aspects of Data Processing by Computer—Mr. K. S. Pope, I.C.T., Australia.

The talk given by Mr. Pope in December is the one which won him selection as the A.C.S. Lecturer for 1967.

Special Interest Groups are active in the fields of Operations Research, Iverson Notation and Numerical Analysis. These groups would welcome exchanges with groups working in these areas elsewhere. They may be contacted c/- P.O. Box 258, Kingston, A.C.T. 2604, Australia.

NEW SOUTH WALES

The final activity of the N.S.W. Branch for 1967 is the annual Christmas party to be held on 12th December in Sydney at a location still to be fixed at time of writing. The 1968 activities of the Branch will start with a meeting on 23rd January when Mr. L. G. Gerrand of P.A. Management Consultants, Pty. Ltd., will speak on Work Study in the Computer Room. On 13th February, Mr. J. E. Marr, General Manager of Australian Data Processing Pty. Ltd., will give his address as retiring Chairman of the Branch. The Annual General Meeting of the Branch will be held on 12th March, 1968. These three meetings will be held in the A.M.P. Theatrette, Circular Quay, at 6.30 p.m., with coffee and biscuits available from 6.15 p.m. Meetings are planned to be held on the second Tuesday of each month during 1968 in

the A.M.P. Theatrette and the same arrangements will apply. Visitors are always welcome and details will be available from the Hon. Secretary, Box R157, Royal Exchange P.O., Sydney, 2000.

QUEENSLAND

The lecture program arranged for the Queensland Branch for 1967/68 is:—

April, 1967—Some Desirable Properties of a Programming Language.

May, 1967—Modern Developments in Computers in the U.S.S.R.

June, 1967 — Symposium: Selection of Computer Personnel.

July, 1967—Management Information Systems.

July, 1967—Pitfalls Confronting Management in the Use of EDP.

August, 1967—Service Bureau Work: The Bureau Point of View.

September, 1967—Simulating the Plant Designer.

October, 1967—Applications of Computers to Banking.

November, 1967—Symposium: Stock Control.

February, 1968—Auditing of EDP Installations.

March, 1968—Annual General Meeting.

April, 1968—Computing in the Insurance Industry.

In February of this year the Branch, in conjunction with the Queensland Regional Group of the Royal Institute of Public Administration, held a one-day Conference at the University of Queensland on the topic "Computers and Administration". Both local and interstate speakers gave addresses and the conference was extremely successful with an attendance of over 200 delegates. It is hoped to publish the papers delivered at the Conference.

The Branch is endeavouring to form Workshop Study Groups to provide members with the opportunity to deepen their knowledge in relevant fields. A group has been formed on O.R. Techniques and another is currently being formed on Standards.

VICTORIA

During 1967, the Victorian Branch launched a comprehensive lecture program known as Computing For Profit—An Exercise in Good Management. This program consisted of four series:

Series A : Systems and Information Science
Series B : Programming and Operations
Series C : Advanced Applications
Series D : Special Meetings

Series A and B were designed to cater for those interested in commercial data processing and Series C was designed to cover technical and scientific applications. Series D was set aside for special speakers of international reputation to talk on either commercial or scientific applications.

A total of about 25 meetings has been arranged for the year covering all series. The following sample of talks given will illustrate the range and variety of topics included in the lecture program.

May, 1967—Getting Across to Management—Mr. W. C. Cusworth, Ford Motor Co. of Australia (Series A).

May, 1967—Analogue Computers as an Aid to Process Control System Design—Mr. J. K. Ellis, I.C.I.A.N.Z. (Series C).

June, 1967—Tape V. Disk—Myth or Reality—Mr. R. D. McKindlay, I.B.M. Australia (Series B).

June, 1967—QANTAM Project—Airline Real-time System —Mr. E. S. Burley, assisted by Mr. G. Tanner, QANTAS (Series D).

July, 1967—Case Study—Inventory Control at Balm Paints—Mr. P. S. Newman, Balm Paints Ltd. (Series A).

July, 1967—Operating and Monitor Systems—Mr. R. P. Harris, University of Melbourne (Series B).

July, 1967—Numerical Control of Machine Tools—Mr. A. G. Jones, Department of Supply (Series C).

August, 1967—The Use of Computers in Organizations— Mr. D. Dodds, Unilever (Aust.) Pty. Ltd. (Series A).

August, 1967—Problems and Hurdles in Setting Up a Computer System—Mr. C. V. Learmonth, A.C.I. Ltd. (Series B).

August, 1967—Picture Grammars—Dr. M. B. Clowes, C.S.I.R.O. (Series C).

September, 1967—Legal Aspects of Data Processing by Computer—Mr. K. S. Pope, I.C.T., Australia (Series D).

The time and effort of organising and conducting this program has been well repaid by the almost doubling of Branch membership since the beginning of the year. The Branch proposes to continue much the same type of program for 1968.

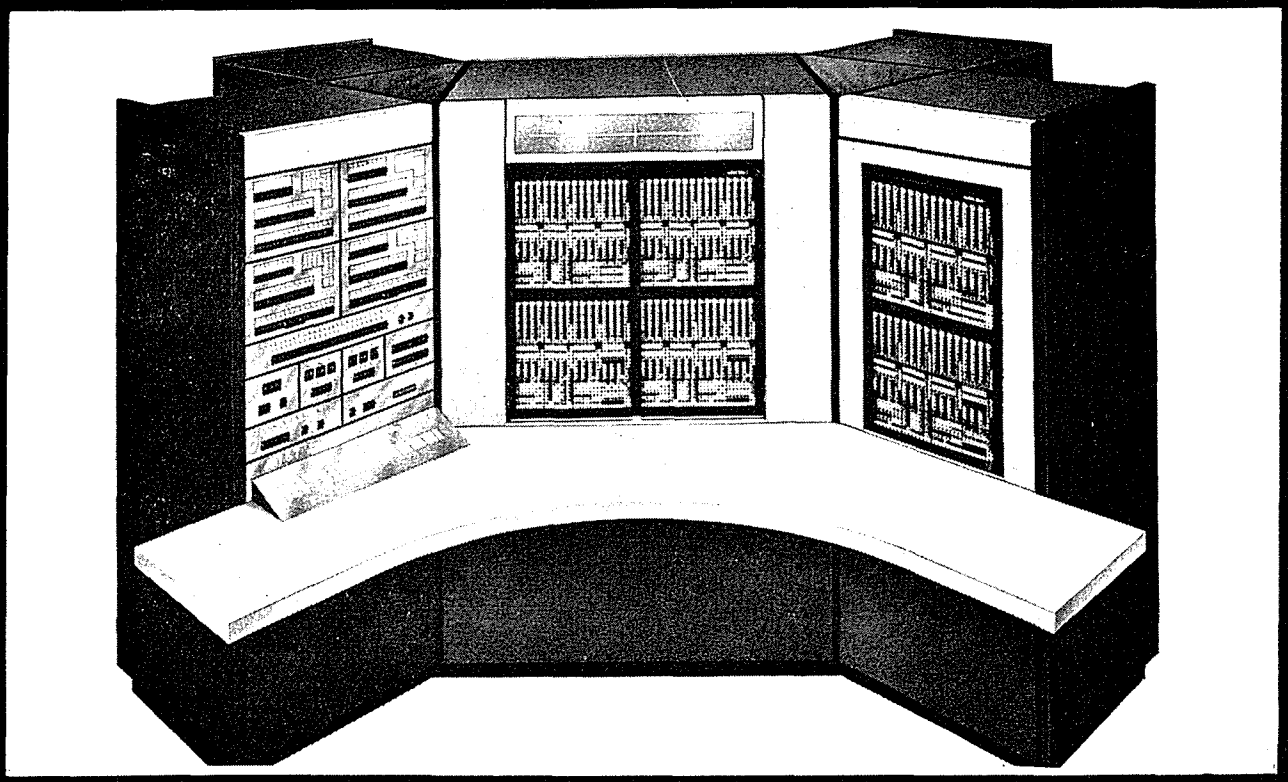
The Applied Dynamics Four is totally new; in hybrid organisational concept, performance characteristics, speed, reliability, and flexibility.

☐ A totally new approach to man-machine communications simplifies programming by users more expert in their own fields than in computer technology.

☐ A totally new approach to components, wiring and mounting techniques really gets to grips with the problem of field expansion—*and* the practicalities of available space.

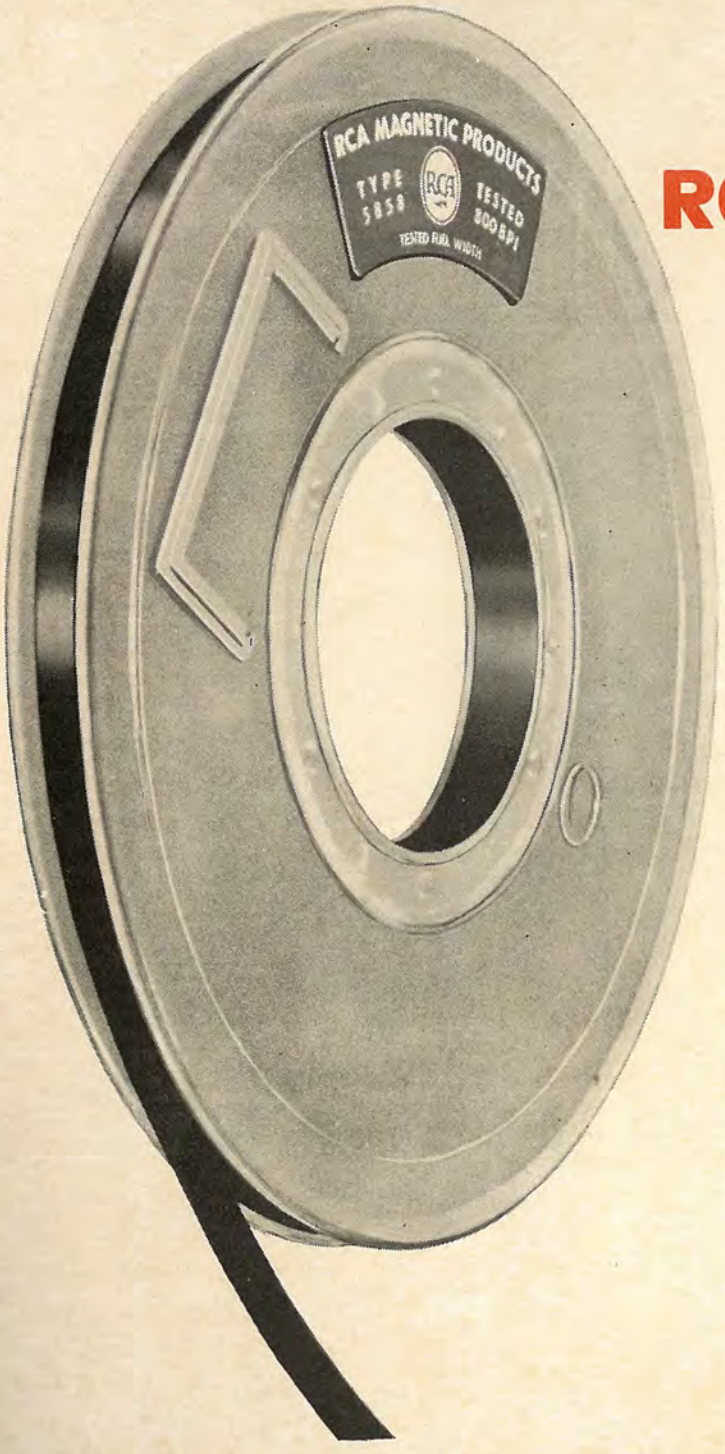
☐ The A D-4 Hybrid is ready and able to cope with your existing problem load alone, or linked to a general purpose digital computer. As a genuinely minimum initial console that can be expanded smoothly and economically, it makes obsolescence obsolete.

'This new hybrid makes computer obsolescence obsolete'



38.5784

Scientific & Industrial Equipment PHILIPS



RCA COMPUTER TAPE IS DEFECT-FREE ERROR-FREE

We have the toughest tests in the industry to make sure it stays that way. At our Test Centre, at Indianapolis, every reel of tape we make is tested from end to end, and some the full width as well.

Our tapes are good. After all, who could make tape better than the people who make the computers? As we do.

We are also in continuous research, developing tomorrow's tapes.

But what's most important, is that these tapes we make in the U.S. are now generally available from our Australian offices through our new Magnetic Products Department; which has been established for your benefit and service.

The following tapes are now available:

RCA Heavy-duty, Industry-compatible Tapes

RCA 5258 LONGLIFE. Engineered and manufactured for maximum performance in computer installations utilising 800 BPI Industry-compatible 7 channel tape drives.

RCA 5458 LONGLIFE. For maximum performance in installations utilising 556 or 200 BPI Industry-compatible 7 channel tape drives.

RCA 5858 LONGLIFE This tape is the culmination of years of research. Full width tested, it is designed to give error-free performance on RCA, IBM and other Industry-compatible tape drives that utilise 800 BPI 7 or 9 channel configurations.

RCA LONGLIFE TAPE features a unique oxide formulation which wears for tens of thousands of error-free passes, while maintaining minimum head wear, with no oxide build-up.
Full specifications and details are available from all RCA State offices.

The most trusted name in magnetic products

RCA OF AUSTRALIA PTY. LTD.

An Associate Company of the Radio Corporation of America

SYDNEY: 219 Elizabeth St. 2000. 61 8541. **MELBOURNE:** 2 Stephenson St., Richmond 3121. 42 4586. **ADELAIDE:** 99 Currie St. 5000. 51 7870.
BRISBANE: 173 Ann St. 4000. 2 7884. **PERTH:** 280 Stirling St. 6000. 28 5057.

THE AUSTRALIAN COMPUTER JOURNAL

Volume 1, Number 1 • November, 1967

The Australian Computer Journal is the official publication of the Australian Computer Society.

Office-Bearers for 1967:

President: T. PEARCEY

Vice-President: P. M. MURTON

Secretary/Treasurer:
R. W. RUTLEDGE
Colonial Sugar Refining Company Ltd., 1 O'Connell St., Sydney 2000, N.S.W., Australia.
Tel.: 2-0515.

Editor: T. PEARCEY,
C.S.I.R.O., P.O. Box 109,
Canberra City, A.C.T. 2600.
Tel.: 4-0455, Ext. 502.

Editorial Committee:

J. M. BENNETT, E. S. BURLEY,

E. H. GIBBS

Published by:

Australian Trade Publications Pty. Ltd., 28 Chippen St., Chippendale 2008, N.S.W. 69-6880.
Vic.: 386 Flinders Lane, Melbourne 3000. 61-3806.
England: 4a Bloomsbury Square, London, WC1. Holborn 3779.
U.S.A.: 1560 Broadway, New York, 36. LTI 3755.
Japan: 34 3-chome, Hatagaya, Shibuya-ku, Tokyo. 463-4665.

Printed by:

Publicity Press Ltd., 29-31 Meagher St., Chippendale, N.S.W. 2008

Contents :

7 *Comments On Some Legal Aspects Of Data Processing by Computer*
By K. S. POPE

15 *Mixed-Data Classificatory Programs I. Agglomerative Systems*
By Dr. G. N. LANCE and W. T. WILLIAMS

21 *A Basis For A Theory Of Programming Languages*
By J. G. SANDERSON

28 *On Intelligence, Intelligent Automata And Robots*
By D. L. OVERHEU

37 *Computer Science Education In Australian Universities*
By P. D. JONES

44 *A Computer Representation Of Plane Region Boundaries*
By B. G. COOK

51 *A Note On Nonlinear Regression Analysis*
By J. KOWALIK

54 *IFIP—Its Structure And Its Aims*
By Dr. A. P. SPEISER

Special features :

2. Notes On Submission Of Papers • 4. Book Review • 6. Editorial • 36. Book Reviews • 57. International Federation For Information Processing • 57. Obituary • 58. Standards Association of Australia • 59. Australian Computer Society Branch Notes

REPRINTS: Fifty copies of reprints will be provided to authors. Additional reprints can be obtained, for which the scale of charges may be obtained from the publisher.

PRICE: The price of copies of the Australian Computer Journal to members of the Society and additional copies to members is $A1.00 per copy.

MEMBERSHIP: Membership of the Society is via a Branch. Branches are autonomous, and may charge different membership fees. Information can be obtained from the following Branch Secretaries:
Canberra: Mr. E. H. Gibbs, P.O. Box 258, Kingston, A.C.T. 2604. **N.S.W.:** Mr. J. E. Marr, Box B 157R, Royal Exchange P.O. 2001. **QLD.:** Mr. L. Olsen, P.O. Box 245, Brisbane 4001. **S.A.:** Mr. A. E. Norman, Institute of Technology, Adelaide 5000. **VIC.:** Mr. P. R. Masters, P.O. Box 104, South Melbourne 3205. **W.A.:** Mr. D. C. Carpenter, P.O. Box K/835, Perth 6001.