The Repository, a collection of over 2700 technical papers and documents relating to computer science and engineering, is maintained by the Computer Society as a service to the information processing community. Some of the papers have been refereed; others have not.

If you have a paper of interest to the computer field, you are invited to submit two copies, including abstract and index terms, to the IEEE Computer Society Editor-in-Chief, 5855 Naples Plaza, Suite 301, Long Beach, CA 90803. Be sure to include 1) a 50-100 word abstract and 2) a cover letter giving permission to enter the paper in the Repository. Entry in the Repository does not constitute publication.

Note: submit all material unbound.

Photocopy prices are $1.00 per page. Add $1.00 service charge to all orders under 50 pages. Microfiche copies are available for $2.50 for each manuscript under 50 pages, plus $2.50 for each additional 50 pages or fraction thereof.

Be sure to state the R number, listed before the author's last name, of each paper you order. All Repository items must be prepaid except for companies or institutions with established accounts. A $2.00 invoice charge will be added to all non-prepaid orders. Please make your check or money order payable to the IEEE Computer Society.

R77-145—DasGupta, Sumit, Carlos Hartmann, and Luther Rudolph, "Dual-Mode Sequential Logic for Function Independent Fault-Testing" (40 pp., Syracuse University, Syracuse, New York)

This paper presents a method of using hardware redundancy to ease the problem of fault testing in sequential logic networks. Sequential logic networks are constructed using two kinds of dual-mode logic gates, one of which is specifically required to initialize a feedback loop to some logic value. Initially, it is shown that these networks can be tested for all single stuck-at-faults with six function-independent tests. In both cases, the network must have the proper number of extra inputs.

R77-146—Maryanski, Fred J., "A Survey of Developments in Distributed Data Base Management Systems" (37 pp., Kansas State University, Manhattan, Kansas)

This report describes the principal problem areas in distributed data base management system development. Distributed data base systems share many design problems with both single-machine data base systems and computing networks, as well as introducing several new dilemmas. Recent research in these problem areas is presented to provide a picture of the state of the art of distributed data base development. In addition, the current status of the data base industry with respect to distributed processing is evaluated by reporting the current projects and future plans of selected (anonymous) data base vendors.

R77-147—Aldrich, Michael, "Nets of Model Neurones" (21 pp., University of Western Australia, Nedlands, Western Australia)

The limitations of the threshold logic unit (TLU) or perceptron as a model for perception and adaption have been clearly delineated in the literature; but the TLU is, after all, a model for only a single neurone, and it is of some interest to consider nets of such units. What has been lacking hitherto is a result for nets analogous to the Perceptron Convergence Theorem for a single unit; one needs to know how to train a network. In a previous paper such a theorem was proved for some particular nets. In the present paper this result is extended to cover a large class of nets; it is shown how any pattern can be learned by a sufficiently complex net. Some implications of this result are considered, and some conclusions pertinent to practical learning machines and brain-modeling are drawn.

R77-148—Walke, Bernhard, "Multiprogrammed Memory Management for Random-Sized Programs" (17 pp., AEG-Telefunken, Research Institute, Ulm, Federal Republic of Germany)

A probabilistic model of a computer system with multiprogramming and paging is considered. The applied work load is derived from measurements in scientific computer applications and is characterized by a great variance of computing time. Throughput of a cyclic model is computed approximately, assuming program sizes with negative exponential distribution. After a review of previous results for a memory allocation policy with at least two working sets to be loaded, an adaptive memory allocation policy is introduced which dynamically changes the prescribed number n. Thereby, it is possible to reach the goal of always having enough memory available to load the parachor of each program. Simulation results establish the approximations as being very good. CPU scheduling is chosen to be through-put optimal. The results demonstrate the benefits of allocation policies with an adaptive controlled degree of multiprogramming.

R77-149—Walke, Bernhard, "Improved Bounds and an Approximation for a Dynamic Priority Queue" (27 pp., AEG-Telefunken, Ulm, Federal Republic of Germany)

An infinite source model M/G/1 is considered and analyzed for a dynamic (non-preemptive) priority discipline. Arriving customers at a queue are assigned urgency numbers. The customer with smallest urgency-number-plus-arrival-time is served next. This service discipline has the effect of distinguishing between customers with different priorities while at the same time taking into account the undesirability of having low priority customers wait too long. The upper and lower bounds on equilibrium mean waiting times are distinctly improved, and an approximation is introduced which yields good agreement with simulation results. From a heavy traffic solution, new insights are gained into typical properties of this dynamic priority discipline. Examples are given to...
compare results of this discipline with results of the well-known static priority discipline.


ASPEN is a "toy" language that can be used as a sample source language in the teaching of compiler construction. As such, its design incorporates language constructs that can be handled by fundamental compilation techniques and are used in educational texts and yet are expressive, well-structured, and reasonably secure, in keeping with current trends in language design. As a result, ASPEN is GOTO-less, strongly typed, and provides efficient, orthogonal mechanisms for information hiding and parameterized user-defined types. ASPEN's dynamic storage allocation mechanism and its polymorphic structures and procedures can be implemented without a heap and with no run-time support routines other than those needed for format-free input and output of strings and numbers. These language specifications define the language and offer some examples of its use.


The NOR network transduction procedures can be grouped into three classes according to their characteristics: pruning procedures, general procedures, and error-compensation procedures. These procedures are implemented into ten programs: NETTRA-PG1, -P2, -P3, -G1, -G2, -G3, -G4, -E1, -E2, and -E3. Among these ten programs, NETTRA-PG1, -G1, -G2, -G3, -E1, and -E2 can treat problems with fan-in/fan-out restrictions. In order to consider the number of levels in a network also as a restriction, the transduction procedures based on gate substitution (NETTRA-PG1), gate merging (NETTRA-G3), connectable and disconnectable functions (NETTRA-G1, -G2), and error-compensation (NETTRA-E1, -E2) are modified. This report describes systematic modifications of these transduction procedures for level-restriction and the procedures for designing level-restricted initial networks. The current version of level-restricted transduction program attempts to design near-optimal multiple-output, multiple-level, loop-free NOR-gate networks under given fan-in/fan-out restrictions and level restriction.

R77-152—Gannon, John D., "Language Design to Enhance Programming Reliability" (251 pp., University of Toronto, Toronto, Canada)

The language in which programs are written can have a substantial effect on their reliability. This thesis discusses the design of programming languages to enhance the reliability of programs. It presents several design principles, and then applies them to particular language structures. Since one cannot logically prove the validity of such design principles, empirical evidence is needed to support or discredit them. An experiment was performed to measure the effect of nine specific language design decisions in one context. Analysis of the frequency and persistence of errors shows that several decisions had a significant impact on reliability.

R77-153—Turnbull, Christopher J. M., "Deterministic Left to Right Parsing" (254 pp., University of Toronto, Toronto, Canada)

A parser model is presented whose structure is a generalization of that of the well known LR(k) parsers. In particular, various classes of parsers that would be both practical and efficient to use in a compiler are examined. Associated with these classes of parsers is a hierarchy of type-O grammars, each grammatical class being defined in terms of the form and structure of derivations. Parsers based on these deterministic regular parsable (DRP) grammars will detect any errors as soon as possible during a left to right scan of their input. A practical parser generator, applicable to some DRP grammars, is developed. Closure and decidability results of this new hierarchy are examined. Of particular interest is the fact that there is no parser generator creating a parser that will detect errors as soon as possible in a parse for all DRP grammars; such an algorithm exists for LR(k) grammars. Much of the research related to LR(k) parsing is applicable to the new parsers discussed in this thesis.

R77-154—Tannenbaum, Andrew S., "A General-Purpose Macroprocessor as a Poor Man's Compiler-Compiler" (16 pp., Der Vrije Universiteit, Amsterdam, The Netherlands)

A method for quickly producing compilers for high-level languages is described. The technique consists of feeding a description of the language to be translated to a general purpose macroprocessor. Used in this way, the macroprocessor functions as a compiler-compiler, providing automatic parsing, lexical scanning, symbol table operations, and handling of syntactic errors. A complete syntactic and semantic description of WHILE statement (except for boolean expression processing) is given in only seven lines as an example. A system programming language implemented by this method is discussed in order to illustrate the main ideas. The compiler produced for this language is compared to other compilers produced by conventional methods.

R77-155—Atwell, Kenneth R., Frank D. Anzelmo, and Raymond A. D’Angelo, "Facility for Integrated Data Organization" (26 pp., Ohio State University, Columbus, Ohio)

In developing its chemical information system, Chemical Abstracts Service (CAS) has built computer files which require hundreds of millions of bytes of storage. To cope with the size and variability of these files and to support a major portion of these files on direct-access media, CAS has developed a data management system and has implemented it on the IBM System/370. This data management system, which is called Facility for Integrated Data Organization (FIDO), accommodates files on disk or tape with a varying number of hierarchically related records under one key and varying length keys within a file. FIDO provides access control to data files by primary keys through primary indexes or by inverted keys through secondary indexes. FIDO assists in maintaining file integrity by limiting file access to authorized processes and by creating an audit trail of direct-access file transactions. FIDO provides the application programmer with a powerful set of commands to access records and data elements. Implementation of FIDO has permitted CAS to operate with greater generality, flexibility, and security.

R77-156—DeRemer, Frank and Hans Kron, "Programming-in-the-Large vs. Programming-in-the-Small" (8 pp., University of California, Santa Cruz, California)

The activity of writing large programs is distinguished from that of writing small ones. Large programs are defined as systems consisting of many small programs (modules) possibly written by different people. Languages for programming-in-the-small, i.e., languages not unlike the common programming languages of today, are needed for writing modules. Also needed is a "module interconnection language" for knitting those modules together into an integrated whole and for providing an overview that formally records the intent of the programmer(s) and that can be checked for consistency by a compiler. This paper explores the software reliability aspects of such an interconnection language. Emphasis is placed on facilities for information hiding and for defining layers of virtual machines.

R77-157—Howden, William E., "Reliability of the Path Analysis Testing Strategy" (28 pp., University of California, San Diego, La Jolla, California)

A set of test data T for a program P is reliable if it reveals that P contains an error whenever P is correct. A set of tests T is reliable and produces the correct output for each element of T, then P is a correct program. Test data generation strategies are procedures for generating sets of test data. A testing strategy is reliable for a program P if P produces a reliable set of test data for P. It is proved that an effective testing strategy which is reliable for all programs cannot be constructed. A description of the path analysis testing strategy is presented. In the path analysis testing strategy, it is assumed that certain errors are not related. A method for analyzing the reliability of path testing is introduced. The method is used to characterize certain classes of programs and program errors for which the path analysis strategy is reliable. Examples of published incorrect programs are included.

R77-158—Downes, William E., "Symbolic Testing and the DISSECT Symbolic Evaluation System" (43 pp., University of California, San Diego, La Jolla, California)

Symbolic testing and a symbolic evaluation system called DISSECT are described.
The principle features of DISSECT are outlined. The results of two classes of experiments in the use of symbolic evaluation are summarized. Several classes of program errors are defined and the reliability of symbolic testing in finding bugs is related to the classes of errors. The relationship of symbolic evaluation systems like DISSECT to classes of program errors and to other kinds of program testing and program analysis tools is also discussed. Desirable improvements in DISSECT, whose importance was revealed by the experiments, are mentioned.

R77-159—Stoutemyer, David R., "Computing with Impure Numbers: Automatic Consistency Checking and Units Conversion Using Computer Algebra" (11 pp., University of Hawaii, Honolulu, Hawaii)

This note shows how computer-algebra systems may be used to include symbolic physical units in computer calculations, with automatic detection of dimensionality-inhomogeneous formulas and automatic conversion of inconsistent units in a dimensionally-homogeneous formula. Inhomogeneity errors are a prevalent type that is undetected in traditional programming languages, and the user is relieved of the tedious, error-prone units-convosrion process.

R77-160—Hamlet, R. G., "Testing Programs with the Aid of a Compiler" (54 pp., University of Maryland, College Park, Maryland)

If finite input/output specifications are added to the syntax of programs, these specifications can be verified at compile time. Programs which carry adequate tests with them in this way would be resistant to maintenance errors. If the specifications are independent of program details, they are easy to give and unlikely to contain errors in common with the program. Furthermore, certain finite specifications are in that they can be used to check the control and expression structure of a program as well as any tests can. A testing system based on a compiler is described, in which compiled code is utilized under interactive control, but "semantic" errors are reported in the style of conventional syntax errors. The implementation is entirely in the high-level language on which the system is based, using some novel ideas for improving documentation without sacrificing efficiency.

R77-161—Kuhl, Jon G. and Sudhakar M. Reddy, "A Multicode Single-Transition-Time State Assignment for Asynchronous Sequential Machines" (29 pp., University of Iowa, Iowa City, Iowa)

A multicode single-transition-state assignment for normal mode asynchronous sequential machines is given. The proposed state assignment requires 2 \( \log_2{n} \) state variables for flow tables with \( n \) states.


This thesis describes the design and implementations of a PASCAL-based programing system on the PDP-11 minicomputer. The system consists of a compiler for the language PASCAL-11, written in itself, together with a set of MACRO-11 routines which transform a PDP-11 into a virtual PASCAL machine. Extensive diagnostic and utility facilities are provided to allow easy debugging and optimization of PASCAL-11 programs. A number of extensions have been made to the base language PASCAL, including augmentation of the file handling abilities of the language in an operating system independent manner. This dialect of PASCAL allows easy separation of a program into overlays; the run-time overlay management system is completely automatic and transparent to the user. The system is coded in a sufficiently modular manner to allow it to be moved to either a different processor or a different operating system. Consideration is given to the possibilities of coding much of the run-time support package in PASCAL-11, and to writing operating systems components in this language.


This paper presents the error-compensation procedures which can find near optimal NOR (NAND) networks under fan-in and/or fan-out restrictions by using the concept of a compatible set of permissible functions with errors (CSPE). Two computer programs, NETTRA-E1-FIFO and NETTRA-E2-FIFO, are implemented to realize the error-compensation procedures under fan-in and/or fan-out restrictions. Both programs have capabilities in transforming a large non-optimal NOR network under fan-in and/or fan-out restrictions into a near-optimal one by reducing the number of gates and connections. The results of the computer experiments are also included in this paper.

R77-164—Elshoff, James L, "The Influence of Structured Programming on PL/I Program Profiles" (19 pp., General Motors Corporation, Warren, Michigan)

Two sets of commercial PL/I programs are studied. The sets represent programing practice before and after the introduction of structured programming techniques. The use of structured programming is found to make a measurable difference on the quality of programs. A few minor changes in the use of the PL/I programming language are noted. Significant modifications to the control structure of the programs and improvements in the qualitative aspects of the programs are discussed. Although the programs are much improved, further alterations can make the programs still better. The author concludes that the time and training required to introduce structured programming techniques to programmers will begin paying dividends in the form of better programs within six months.

Order by R-number. Use the Repository order form on p. 128

R77-165—Stone, Harold S., "Multiprocessor Scheduling with the Aid of Network Flow Algorithms" (44 pp., University of Massachusetts, Amherst, Massachusetts)

In a distributed computing system a modular program must have its modules assigned among the processors so as to avoid excessive interprocessor communication while taking advantage of specific efficiencies of some processors in executing some program modules. This paper shows that this program module assignment problem can be solved efficiently by making use of the well-known Ford-Fulkerson algorithm for finding minimum flows in commodity networks as modified by Edmonds and Karp and Dinic. A solution to the two-processor problem is given, and, extension to the problem of \( n \) processors is considered with partial results given without a complete efficient solution.

R77-166—Hamilton, M. and S. Zeldin, "Higher Order Software—A Methodology for Writing Software" (109 pp., C.S. Draper Laboratory, Cambridge, Massachusetts)

The key to software reliability is to design, develop and manage software with a formalized methodology which can be used by computer scientists and applications engineers to describe and specify reliable software with a high degree of reliability. The methodology of higher order software (HOS), specifically aimed toward large-scale multiprogrammed/multiprocessor systems, is dedicated to systems reliability. With six axioms as the basis, a specific set of software development processes is defined as if it were one complete and consistent computable system. Some of the derived theorems provide for reconfiguration of real-time multiprogrammed processes, communication between functions, and error correction and recovery of data and data integrity. The authors envision a scheme in which the definition of a given system can be described with an HOS specification language which, by its very nature, enforces the axioms with the use of each construct. A system defined in HOS can be analyzed automatically for axiomatic consistency by the design analyzer, without program execution, and by the structural executive analyzer on a real-time basis. The result is that a software system can be developed efficiently with reliable interfaces. This is significant since interface testing in a large system accounts for approximately 75% of the verification effort.

R77-167—Williams, Robin and Gary M. Giddings, "A Picture-Building System" (13 pp., IBM Research Laboratory, San Jose, California)

The picture-building system (PBS) deals with the problem of creating and manipulating data structures for applications using computer graphics. PBS has a data definition facility that enables one to specify structures needed for a given application and a manipulation facility for loading and editing the data and displaying the
image stored in the structures. The structures are defined in a relational data base and allow graphical attributes to be specified in the same way as non-graphical attributes. There is a deliberate attempt to separate data from programs which is in keeping with other data base developments. It is also possible to mix text and vector data with raster scan data which the authors have found to be very useful in some graphical applications. Some examples and a brief description of the color display system are given.

R77-168—Elgot, Calvin C., "Structured Programming With and Without Go To Statements" (45 pp., IBM Thomas J. Watson Research Center, Yorktown Heights, New York)

While "Dijkstra flowchart schemes (built out of assignment statement schemes by means of composition, IF—THEN and WHILEDO) are simple and perspicuous, they lack the descriptive power of flowchart schemes (provided additional "variables" are not permitted). On the other hand the analogous multi-exit CACI schemes introduced in this paper, which are virtually as simple and perspicuous as Dijkstra schemes, describe exactly the same computations. Processes as flowchart schemes (without the aid of additional variables). One theorem makes contact with "readable flowgraphs," an active area in its own right.

R77-169—Pottie, Marshall R., "Analysis of Some Simple Algorithms for Smoothing Noisy Numeric Characters" (149 pp., University of Maryland, College Park, Maryland)

This thesis provides a comparison of existing smoothing algorithms which have been used for image enhancement operations on alphanumeric characters. Modifications to the original algorithms were experimented with and new algorithms developed. All algorithms and modifications were tested on numeric characters using the standard Highleyman data set distributed by the IEEE Computer Society. The results show that a new algorithm designed around an iterative inter-element connectivity approach gives consistent enhanced results which are superior to the results of the other methods. An extensive set of appendices is included to display the results of the experimentation.

R77-170—Lala, J. H. and Albert L. Hopkins, Jr., "Performance Evaluation of a Real-time Multiplier" (40 pp., C. S. Draper Laboratory, Cambridge, Massachusetts)

A three-processor bus-centered multiprocessor is analyzed from the performance viewpoint in a real-time environment. The real-time environment is simulated by a set of periodic jobs. An executive program is developed to dispatch jobs using two different scheduling strategies. Software techniques are used to monitor the system performance. The measured performance criteria include throughput efficiency and job starting delay. The probability distribution function of the delay and the maximum percentile delay functions are for the first time analyzed in detail. A theoretical model, called the 3-Erlang Markov process model, of the multiprocessor has also been developed. Some of the important experimental results are compared to the theoretical results.


This paper aims to assist the person who needs to solve stiff ordinary differential equations. First, the problem area and the basic difficulty are identified by responding to several fundamental questions: Why is it worthwhile to distinguish a special class of problems termed “stiff”? What are stiff problems? Where do they arise? How can we recognize them? Second, the characteristics shared by methods for the numerical solution of stiff problems are described. These characteristics have important implications as to the convenience and efficiency of solution of even routine problems. Understanding them is indispensable to the assembling of codes for the very efficient solution of special problems or for solving exceptionally large problems at all. Third, the authors discuss what is meant by “solving” a differential equation numerically and what might be reasonably expected in the case of stiff problems.

R77-172—Banerjee, Upal, "Data Dependence in Ordinary Programs" (45 pp., Report No. UIUCDCS-R-76-837, University of Illinois, Urbana, Illinois)

Certain conditions have been obtained under which there is no dependence between two assignment statements inside an IF-free loop in a language like Fortran. An algorithmic technique to solve the problem is described. The array subscripts are assumed to be general polynomials in the index variables.


This document describes the ACSES quiz system as implemented on the PLATO CAI system. The quiz system allows an instructor of a computer science course access to a pool of PLATO quizzes for use with PLATO instructional computer science lessons that will be utilized by the instructor’s students. The use of a quiz with an instructional lesson provides both a study aid for each student and a means of assessing the effectiveness and thoroughness of a computer science lesson. This latter provision is accomplished through the collection of quiz question data which can serve as an analyzable instructional lesson (or quiz) weakness. This report describes the quiz system from both an instructor’s and a student’s standpoint, discusses how the system is monitored and maintained, and gives an example of the potential for evaluation. Additionally, it contains a self-contained paper, “Guidelines for Developing Computer Science Quizzes,” for use by authors of quizzes to be added to the system pool.


Burst techniques are considered as possible alternatives to conventional ones in the realm of digital transmission of analog signals. Three demonstration systems using various burst techniques are discussed in detail.


CLEOPATRA is a general-purpose language with features suitable for systems programming. A compiler for the language CLEOPATRA has been implemented in two passes. This report describes the analysis pass which produces an intermediate text suitable for the code generation pass. The analysis pass was written in PL/I for the IBM 360/370. Due to the facilities of the language, the analysis pass requires innovative data structures and algorithms—these are reported herein.


Although there are many open theoretical questions for stiff problems, non-stiff order, variable step methods, it seems unlikely that there will be any major computational advances as far as integration methods for the general nonlinear problem. The major drawback of many of the methods that have been proposed in the last few years is that they involve too many matrix operations, or operations on larger matrices than in some of the more straightforward techniques such as the backward differentiation formulas. However, advances in techniques for handling large, sparse matrices may affect the choice of integration method for very large problems. This paper examines possible improvements that might occur through the use of very inaccurate Jacobians. Such Jacobians might be chosen so that the LU decomposition is particularly simple. Some methods are shown to remain stable in spite of very large errors in the Jacobian. These results are very encouraging but suggest that further speed increases in very large problems are possible.

R77-177—Sanjai, S. and P. V. Sankar, "A Note on Control Memory Minimization in Microprogrammed Digital Computers" (8 pp., Tata Institute of Fundamental Research, Bombay, India)

The problem of minimizing the bit dimension of control memories in microprogrammed computers is considered. The authors define a new concept of spanning compatible classes (SCC) as those compatible classes which span the entire set of ROM words. Based on the SCC, they demonstrate that it is possible (under cer-
taint conditions) to come up with a better solution (lower cost) than is reported in the literature.

R77-178—Lenfant, Jacques, “Parallel Transfers of Data: A Structure Favoring Frequently Used Permutations” (39 pp., I.R.I.S.A. Laboratoire d’Informatique, Renens, France)

Benes binary network can realize any one-to-one mapping of its 2n inputs onto its 2n outputs. Several authors have proposed algorithms which compute control patterns for this network from any bijective assignment. However these algorithms are both time-consuming and space-consuming. In order to meet the time constraints arising from the use of Benes network as the alignment network of a parallel computer, another approach must be chosen. This paper considers typical functions and shows that the set of needed permutations of data is very small, as compared to the whole symmetric group. The author gathers frequently used bijections into five families. For each family he presents an algorithm that can control the two state-switches on the fly, as the vector of data passes through the network. Finally he describes one possible scheme to implement an instruction “Trigger a Frequently Used Bication.”


This relational model for data base organization clearly defined basic algebraic concepts. As a consequence, questions of a semantic nature arise. A data model is discussed that takes into account basic semantic properties and permits a data base to be partitioned into independent object types and associations among them. Independent object types are described by characteristics which are meaningful modification units. This data model permits more precise analysis of the ideas behind the relational theory, in particular the meaning of functional dependency and normalization. Insertion-deletion rules for relations in third normal form are developed, and proposals are made to avoid some existing drawbacks.


The development of an optical disk recorder with an on-line storage capacity of 1012 bits is discussed. Recording is done by laser machining of micro-sized pits in a thin tellurium film allowing direct-read-after-writing (DRAW) of the information. The recording format maps each bit of information to a recorded pit. The tellurium is deposited on a clear plastic disk configured to be self-protecting and easily handled. A major program objective was to demonstrate the feasibility of an inexpensive recorder and disk.

R77-181—Capowski, Joseph J., “The Neuroscience Display Processor” (50 pp., University of North Carolina, Chapel Hill, North Carolina)

From five years of experience, the computer graphics needs of the UNC neuroscience computer user’s community have been defined. These needs are: refreshed line drawings of 2D graphs and waveforms and 3D structures, hardware matrix multiplication, simple line rejection, orthographic and perspective projection, output to CRT or plotter, refresh by host, and ease of programming. No commercially available system meets these needs at a reasonable cost. A neuroscience display processor (NDP) was built to fulfill these needs. It is a 82700 graphics system designed as a peripheral device for a general purpose minicomputer. Three features characterize the NDP: a 12-bit product is produced by a combinatorial multiplier in 190 nsec; a complete line is rejected if either of its endpoints is out of view; and CRT deflection ramps between line endpoints are produced by analog RC circuitry. A software package to allow easy programming of the NDP has been written. The machine is in production use; response from users has been very good.

R77-182—Blum, E. K., “The Semantics of Programming Languages: Part I” (23 pp., TRW, Redondo Beach, California)

A theoretical basis for the semantical concept of programming languages is presented. The concepts of computational structure and symbolic computational structure are introduced as a basis for a precise definition of a programming system and the semantic operator of such a system. A precise definition of formalizability of the semantics of a programming system is given. Part II of this report, R77-183, is also available.

R77-183—Blum, E. K., “The Semantics of Programming Languages: Part II” (29 pp., TRW, Redondo Beach, California)

The definition of formalizability of one programming system in another, given in “The Semantics of Programming Languages: Part I,” is used as the basis for the definition of a meta-system, called SEMANOL, for programming the semantics of such languages as Jovial, Algol, Fortran, etc. The syntax and semantics of SEMANOL are described. “The Semantics of Programming Languages: Part I” is available as R77-182.


This paper describes the approach used in developing a functional simulator of a large data processing system (a ballistic missile defense system that operates in a real-time, multiprocessor mode). It describes and discusses four essential features of the functional modeling concept: (1) capability to readily accept changes in the functional structure of the system by modifying a set of functional operators in a logic control chain that prescribes both the logical sequence of subsystems tasking and data processing requirement to the simulator to make it emulate the operation of the real system; (2) ability to make parameter changes on simulated routines that dictate process and I/O work specifications and routine priority levels to the simulator for process structure modifications; (3) capability to simulate the system at varying levels of functional detail so that some subsystems can be simulated at different functional levels using the same architecture; (4) in-depth visibility of these subsystems’ performance is gained; and (5) flexibility in the organization of the model, permitting three-way expansion of the scope of the simulation by considering more sophisticated threats to increase data processing loads and software requirements, by augmenting the number of hardware units to compensate for the increase or change in data traffic, and by modifying the process to reflect the change in data handling requirements.

R77-185—Lanzano, Bernadine C., “Program Automated Documentation Methods” (27 pp., TRW, Redondo Beach, California)

Several methods for automatically generating and maintaining documentation for TRW’s computers systems are being used, and other procedures are under examination. This paper presents: (1) a short synopsis of the mission analysis and trajectory simulation (MATS) program to provide an understanding of the size and complexity of one simulation for which documentation is mandatory; (2) a description of a program which assists in automating the documentation of subroutines; (3) an expose of two flowcharting programs; (4) some notes on useful program internal cross-reference information; (5) an implementation of a text editing program available in a time-shared computer system environment; (6) a preview of a proposed system which would aid in program development and documentation which utilizes a graphics display console; and (7) a recommendation for software standardization.


Development and application of a functional simulation technique for evaluation of alternate, on-line, data processing configurations is described. Centralized and decentralized systems were simulated; under various file structures, to determine relative performance characteristics. The LCC simulation technique was used. It involves a set of functional operators which analysts apply in constructing models of elements within the system. The models represented (1) data processing and communications hardware, (2) operating system and applications programs, (3) file structure and data base, (4) on-line system users, and (5) the external environment. The functional operators, mechanized in Fortran IV, plus a set of Fortran IV program packages, SALSIM, are used to generate the simulation program.


An algebraic system for representing data structures is developed using con-
cepts from boolean algebra and from Blum’s mathematical theory of the semantics of programming languages. A data structure is defined as a finite family $D$ of data sets and a finite family $R$ of partial relations with each relation in $R$ defined as a family of all possible relations that are represented as matrices having 1’s and 0’s as elements. Conventional data structure constructs, such as fields, records, and files, are represented as formulas in the data structure algebra. The data structure as well as the logical structure and the relation between the two can be represented in the algebra.

**R77-188—Gibson, C. G. and L. R. Railing, “Verification Guidelines”** (57 pp., TRW, Redondo Beach, California)

This report contains specific guidelines and procedures for development, checkout, and verification of Fortran computer programs. The emphasis is on what should be accomplished during each phase of the software development process in order to make verification easier and more effective. The fundamental approach is one of keeping verification considerations uppermost in mind throughout the software development process. The scope of the document includes the verification process for non-real-time Fortran computer programs. Although many of the guidelines presented can be applied to real-time and flight software, no attempt has been made to explicitly address verification problems inherent in this type of software. Software development procedures are specified only as they impact the verification process.

**R77-189—Wolverton, R. W., “The Cost of Developing Large-Scale Software”** (21 pp., TRW, Redondo Beach, California)

The work of software cost forecasting falls into two parts. First structural forecasts are made; then the absolute, dollar-volume forecasts are calculated. Structural forecasts describe the technology and functioning of a software project, but not its cost. These forecasts are based on the project’s life cycle from the structural forecasts. Judgement, technical knowledge, and economic research should combine in making the structural forecasts. This paper presents a methodology based on a 25 x 7 structural forecast matrix which has been used by TRW with good results over the past few years. It presents a cost-estimating algorithm and describes five traditional methods of software cost forecasting: top-down estimating, similarities and differences estimating, ratio estimating, standards estimating, and bottom-up estimating. All forecasting methods suffer from the need for a valid cost data base for many estimating situations based on some methodology and historical results. Topics for further work and study are suggested.


Extensive study and many applications have shown that a formal approach to software quality assurance system can provide three capabilities that could not be as easily attained with manual testing and checkout procedures. These capabilities are (1) quantification of test effectiveness, (2) optimization of the testing process, and (3) increased reliability. Three case studies are presented to describe several examples of the application of automated software quality assurance tools. A data base comparator, a test effectiveness measurement tool, and a test result excess cost are described in sufficient detail to illustrate their utility in support of a complete software testing process. Summary remarks include another case history in which increased effectiveness was achieved with a decrease in the cost of testing and a reduction in the elapsed time to complete the testing job. Potential cost benefits from the use of automated software quality assurance tools are summarized. The prediction of extended future development and application of such tools is made.


The development and systematic application of an integrated set of automated tools is suggested here as an important approach toward attaining cost effectiveness throughout the software production cycle. A typical software development process is discussed in terms of major phases of design, implementation, verification, maintenance, and disposal. For each development phase, problems which can result in excessive difficulty or rework and subsequent cost are identified, and some tools which might effectively reduce or eliminate the problems are described. Finally, a combination of automation, verification, maintenance/modification, and documentation tools are described in detail. The current status of each tool (i.e., conceptual, in-design, developmental, or operational), its primary phases of application, general capabilities, and a brief illustration of output are provided.

**R77-192—Brown, J. R., “Practical Applications of Automated Software Tools”** (6 pp., TRW, Redondo Beach, California)

One of the most serious difficulties confronting software certification is a general failure to determine and declare what we expect certification to achieve and “certified” to mean. It is suggested that certification is dependent upon the rigor or thoroughness with which software is tested, and some recently developed measures of measuring thoroughness are described. The measures (including percentage of executable source statements exercised, branching tests, and logic paths exercised) are computed for a typical computer program. Support tools which automatically provide measurements of testing thoroughness are discussed. Several examples are presented to illustrate use of tools to modify or add to existing test cases to cause previously undetected program errors to be exercised and thus increase the various computed measures of testing thoroughness.

**R77-193—Lipow, M., “Estimation of Software Package Residual Errors”** (13 pp., TRW, Redondo Beach, California)

A method for estimating the number of errors remaining in a software package is proposed and analyzed. It is based upon a scheme, proposed by H. D. Mills of IBM, in which a set of two errors is “seeded” into the software. A specified number of tests is conducted, each test capable of finding one of the indigenous or unknown errors, or one of the seeded errors with the same probability, or of finding no error. The method esti- mately the two parameters: $n_i$, the unknown number of residual errors, and $q$, the unknown probability of detecting either a seeded or an indigenous error. The method requires that the seeding scheme works, of course, is because observed detection rate of seeded errors with a known number of seeded errors remaining yields information on the number of residual, indigenous errors, since the latter errors are detected at the same rate, by assumption. In addition to finding maximum likelihood estimators for $n_i$ and $q$, the conditional distribution of the maximum likelihood estimator for $n_i$, given the total number of errors detected, is given analytically and also computed in two errors cases. The distribution calculation also yields the mean and the root mean square deviation of the estimator for $n_i$, and consequently an indication of its bias and mean square error true values of $n_i$, number of seeded errors, and observed total numbers of errors.


This paper describes an automated method for designing an optimal set of test cases which exercise all branches in the source code of the user’s software modules. The method has been implemented for Fortran source code and is operational on NASA/JSC’s Univac 1108 and 11/11 system. The paper defines ideal criteria for verification, demonstrates the impracticality of attempting to exercise every path, and arrives at a practical, measurable objective: to find the minimum set of paths which exhausts exercise all logical branches in the module. Limitations of the present system are described. New techniques being investigated to rectify these limitations are discussed.

**R77-195—Nelson, E. C., “The Dynamics of Data Structures”** (13 pp., TRW, Redondo Beach, California)

The data structure algebra previously developed is extended to provide a capa-
bility to represent the dynamic processes in which changes are made to data structures. The capability is applied to describing the changes to a GIM data structure by statements involving the verbs ADD, CHANGE, DELETE, and LIST.

R77-196—Nelson, E. C., “A Statistical Basis for Software Reliability Assessment” (11 pp., TRW, Redondo Beach, California)

A mathematical definition of the reliability of a computer program is developed from the mathematical definitions of a program and program execution given in Blum’s mathematical theory of the semantics of programming languages. The reliability so defined is measurable, and it is related to the structural properties of computer programs using concepts borrowed from the PACE system of automated software test tools.

R77-197—Blum, E. K., “SEMANOL: A Formal System for the Semantics of Programming Languages” (37 pp., TRW, Redondo Beach, California)

The theory of semantics described in earlier reports is extended and modified. A new version of the SEMANOL system (called SEMANOL 73) is described, with reference to the extended theory. Some related documents are “The Semantics of Programming Languages: Part I” (R77-182) and “The Semantics of Programming Languages: Part II” (R77-183).

R77-198—Lipow, M., “Application of Algebraic Methods to Computer Program Analysis” (24 pp., TRW, Redondo Beach, California)

An extension of Dilworth’s theorem on partially ordered sets is shown to be useful in determining the minimum number of test cases needed to exercise all segments (or branches) of a computer program at least once each. Using the known adjacencies of segments of code, various programs can be developed graph models are utilized to determine the number of logic paths in the program. Some suggestions for future studies using various matrices associated with directed graphs are given. It is believed that some of the methods discussed can be of assistance during the design phase, before the programs are completely coded, in producing more efficient and error-free programs.

R77-199—Boehm, Barry W., “Some Steps Toward Formal and Automated Aids to Software Requirements Analysis and Design” (12 pp., TRW, Redondo Beach, California)

This paper describes a formalism for requirements analysis and design which includes a properties-oriented formalism for requirements traceability, consistency, and completeness checking, the Requirements/Properties Matrix, and a process-oriented formalism similar to others in being based on working with blackbox descriptions of modules. An initial manual version of the technique has worked successfully in two small-to-medium software projects. It is now being extended to handle large software projects in a semiautomated fashion.

R77-200—Kaspar, H., and D. S. Miller, “Job and Jobstream Modeling in the TRW Timeshared System Simulation” (28 pp., TRW, Redondo Beach, California)

The method of modeling user jobs and jobstream utilized in the simulation of the TRW Timeshared System (TRW/TSS) is presented in a brief description of the TRW/TSS. Simulation objectives are given. An overview of the simulator is presented. The actual TRW/TSS user jobstream is given in detail. This is followed by two methods of modeling, namely, a statistical and a job-trace modeling approach which forms the central discussion of this paper. Sections on calibration and validation of these jobstream models and a presentation of results and output to date conclude the article.

R77-201—Wolerton, R. W., “Paradoxes in Management: Software Standards and Procedures” (26 pp., TRW, Redondo Beach, California)

The purpose of this paper is to show what happens within industry when the government keeps adding software development and documentation standards to the set open to the contractor. The result is often contrary to the expectation—hence paradoxical. This paper briefly examines government rules and regulations which influence the contractor’s technical approach, his software development procedures, and his product control. It discusses the process of reducing standards, attempts at standardization of software development procedures, promising solutions to current problems in documentation standards, and promising solutions to current problems in programming standards.


This paper summarizes the results of an ACM/NBC Workshop on Computer Performance Evaluation. Computer performance evaluation (CPE) was selected as the subject of an ACM/NBC Workshop because of the significant leverage that CPE activities can have on computer usage. This paper describes a number of conclusions abstracted from the discussions as well as some technical results and conclusions formally adopted by the participants. While several of these conclusions indicate that improvements are needed in performance analysis tools, another suggests that improved application of CPE could be achieved by better documentation of analysis approaches. More integration of data collection and modeling are considered necessary for the performance analysis field to develop its full potential. Participants noted that the common emphasis on data collection or modeling, to the exclusion of considering objectives, often seriously degrades the value of performance analyses; the only savings that really count from a performance analysis are the ones that appear on the bottom line of the balance sheet.

R77-203—Boehm, Barry W. and B. G. Dexter, Jr., “Some Software Considerations for Developing Countries” (20 pp., TRW, Redondo Beach, California)

Most computer system acquisitions in developing countries are made primarily on a hardware price-performance basis. However, the critical importance in developing countries is usually neither cash nor hardware, but expert software personnel. Policies in developing countries for computer system acquisition should, therefore, focus on training, documentation, maintainability, and expert software support. This paper provides guidelines for such emphasis during evaluation, acquisition, and development of computing systems and software packages. It also points out other mechanisms such as automated software development tools for conserving scarce software personnel.

R77-204—Boehm, Barry W., R. K. McClean, and D. B. Urfirslans, “Computer Experience with Automated Aids to the Design of Large-Scale Reliable Software” (11 pp., TRW, Redondo Beach, California)

This paper summarizes some recent experience in analyzing and eliminating errors from the early design of large software projects. It begins by pointing out some of the significant differences in software error incidence between large and small software projects. The most striking contrast, illustrated by project data, is the large correspondence of design errors over coding errors on largescale projects. The paper next presents a taxonomy of software error causes and some analyses of the design error data. Based on this analysis of observational data, a hypothesis was derived regarding the potential cost-effectiveness of an automated aid to detecting inconsistencies between assertions about the nature of inputs and outputs of the various elements (functions, modules, data bases, etc.) of the software design. This hypothesis was tested by developing a prototype version of such an aid, the Design Assertion Consistency Checker (DACC), using TRW’s Generalized Information Management and Control System (GIM) and using it on a large-scale software project with 186 elements and 967 assertions about their inputs and outputs. Of the 121,000 possible mismatches between input and output assertions, DACC found 518, at a cost in computer time of 430. Most of the mismatches resulted from shortfalls in the initial version of DACC or the initial data preparation. In general, the data confirmed the hypothesis about the general utility of a DACC capability for large software projects.


The paper presents an evaluation of a novel methodology for evaluation of testing in support of operational reliability assessment and prediction. The methodology features an incremental evaluation of the representativeness of a set of development and validation test cases together with
definition of additional test cases to enhance those qualities. An experimental application of the techniques to a small program is provided as an illustration of the proposed use of the methodology for operational software reliability estimation. The relationship between structured exercise testing thoroughness and operational usage representativeness is discussed; the specification of a quantified reliability requirement and an explicit, required representativeness measure (or confidence) is identified as integral to effective application of the proposed reliability testing methodology; efforts to extend, formalize, and generalize the methodology are described; and expected benefits, as well as potential problems and limitations, are identified.


Diagnostic emulation is a means of reducing high costs in the development and verification of highly reliable software. This improved technique, which is based on microprogram emulation technology, is significantly more cost effective than conventional interpretive computer simulation for performing software checkout and testing. A computer description language is shown to be an essential capability for developing diagnostic emulators, and experimental results demonstrate the feasibility and reduced cost of the concept. Further TRW research based on diagnostic emulation has been initiated to develop the Flexible Analysis, Simulation, and Test (FAST) facility.

R77-207—Thayer, T. A., “Understanding Software Through Analysis of Empirical Data” (20 pp., TRW, Redondo Beach, California)

This paper discusses the collection and analysis of data available during development, testing, and operational use of software systems as a means of determining software quality in quantifiable terms. The approach to data collection and analysis taken by TRW in a study of three systems is described, including some study results and identification of necessary improvements in the collection and analysis processes. This paper treats both the long-range and the near-term payoffs of such studies in an attempt to answer the question “Why collect data at all?”

R77-208—Nelson, E. C., “Software Reliability” (21 pp., TRW, Redondo Beach, California)

Although the reliability of software has been a major problem in most computer applications and in computer-based hardware systems, great strides have been made in understanding the problem and in developing solutions to it. This progress, surveyed in this paper, has involved collection and analysis of data on actual software problems, development of a mathematical theory of software reliability, software management practices that apply a well-structured discipline to the development and testing of computer programs, software development tools and techniques that automate part of the process and define and enforce good programming practices, and software test tools and techniques that automate portions of the testing process and increase the thoroughness and cost-effectiveness of testing. The result is a method for testing developed with the new tools, techniques, and practices, a dramatic reduction in the number of problems found in operational use. The progress is due to the increasing interest, attention, and effort applied to the problem.

R77-209—Williams, R. D., “Managing the Development of Reliable Software” (8 pp., TRW, Redondo Beach, California)

This paper discusses the serious need for improved management methods and software development technology to support the production of truly reliable software and presents experience gained on TRW’s Site Defense Program in addressing that need. The basic theme is simply this: the program attempted to achieve a better understanding of how to specify, design, build, and validate reliable software; translated that understanding into an integrated and exceptionally disciplined software development methodology; and used, evaluated, and validated the methodology enough to recognize and report substantial early success and continuing progress toward that goal.

R77-210—Bell, Thomas E., “Computer Performance Management Through Control Limits” (12 pp., TRW, Redondo Beach, California)

Industrial control limits, routinely applied elsewhere for improving management, should also be employed in computer centers. A control limit is the chosen value of a variable which, when exceeded, implies that control action should be taken to return the variable to the controlled area. Some control limits for computer installations (e.g., response time or up-time) are directed toward the installations’ objectives while others (e.g., CPU utilization or total memory in use) indicate how the process is operating. Both types of control limits are required for successful computer center management because their use tends to expose bad management; of course, they also expose good management and provide a technique to aid good managers do their jobs better.

R77-211—Bell, Thomas E. and David C. Bixler, “A Flow-Oriented Requirements Statement Language” (19 pp., TRW, Redondo Beach, California)

TRW, through the Software Requirements Engineering Program, is developing a systematic approach to requirements engineering as part of the Ballistic Missile Defense Advanced Technology Center (BMDATC) software development research program to improve techniques for developing new computer software. A key part of the approach is the Requirements Statement Language (RSL), a formal language for stating requirements. Use of RSL enforces a discipline on the requirements engineer and additionally provides a specification that is machine-processable. Instead of the traditional specification of individual module requirements, RSL approaches the problem from an integrated, flow-oriented viewpoint. The requirements are stated as sequences of processes which must be executed to provide response to a stimulus. Other portions of RSL deal with the specification of the processing step (as those included in high-level specifications or models in procedural code) of the processes as well as indications of the inputs and outputs necessary for each one. The information given in the RSL specification of requirements is collected in a centralized data base. Tools operated on the data base analyze the specification for consistency and completeness, and generate a simulation of the specified system so that dynamic interactions may be studied. As a result, the developer can begin to see the consequences of a set of requirements before they are even complete, and the software system’s ultimate user can have the requirements validated before beginning full-scale development.

R77-212—Bell, Thomas E. and Thomas A. Thayer, “Software Requirements: Are They Really a Problem?” (24 pp., TRW, Redondo Beach, California)

Although requirements arise naturally from an obvious need, or do they come about only through diligent effort—and even then contain problems? Data on two very different types of software requirements were analyzed to determine what kinds of problems occur and whether these problems are important. The results are dramatic: software requirements are important, and their problems are surprisingly similar across projects. New software engineering techniques are clearly needed to improve both the development and statement of requirements.

R77-213—Bell, Thomas E., David C. Bixler, and Margaret E. Dyer, “An Extendable Approach to Computer-Aided Software Requirements Engineering” (29 pp., TRW, Redondo Beach, California)

The development of system requirements has been recognized as one of the major problems in the process of developing computer systems. Software requirements are analyzed to determine what kinds of problems develop, and the RSL requirements engineering system is used to maintain and analyze such requirements. This system includes the Requirements Specification Language (RSL), a flow-oriented language for the expression of software requirements, and the Requirements Engineering and Validation System (REVS), a software package which includes a translator from RSL, a data base for maintaining the description of system requirements, and a collection of tools to analyze the information in the data base. The system emphasizes a balance between the use of the creativity of human thought processes and the rigor and thoroughness of computer analysis. To maintain this balance, two key design points—extensibility—were followed throughout the system. Both the language and the software are easily extended, but adequate locks are placed on extensions, and limitations are imposed on use, so that discipline is augmented rather than decreased.

June 1977

119
R77-214—Allford, Mack W., "A Requirements Engineering Methodology for Real-Time Processing Requirements" (28 pp., TRW, Redondo Beach, California)  
This paper describes a methodology for the generation of software requirements for large real-time unmanned weapons systems. It describes what needs to be done by the intermediate products, and how to use automated aids to improve the quality of the product. An example is provided to illustrate the methodology steps and their products and the benefits. The results of some experimental applications are summarized.

R77-215—Boehm, Barry W., "Software Engineering" (41 pp., TRW, Redondo Beach, California)  
This paper provides a definition of the term “software engineering” and a survey of the current state of the art and library in the field. The survey covers the technology available in the various phases of the software life-cycle—requirements engineering, design, coding, test, and maintenance—and in the overall software management and integrated technology-management approaches. It is oriented primarily toward discussing the domain of applicability of techniques (where and when they work) rather than how they work in detail. To cover the latter, an extensive set of 104 references is provided.

This is the final technical report for the Software Reliability Study, performed by TRW for the Rome Air Development Center. It presents results of a study of data, principally error data, collected from four software development projects. These data were analyzed to determine what might be learned about various types of errors in the software: the effectiveness of the development and test strategies in preventing and detecting errors, respectively; and the reliability of the software itself. This report also provides guidelines for data collection and analysis on other projects: data that are generally available, how project data were collected in this study, and some observed realities concerning the data collection and analysis processes. Finally, the most recent work of TRW's Mathematical Theory of Software Reliability (MTSR), the Nelson model, is presented. This is complemented by a survey of software reliability models currently available in the software community.

R77-217—Doring, Richard, L. R. Whitlock, and W. J. Hansen, "Details of an Experimental Videotape Evaluation of an Interactive Exam System" (85 pp., Report No. UIUCDCS-R-77-782, University of Illinois, Urbana, Illinois)  
This report provides details of an experiment conducted at the University of Illinois. An analysis of the data is contained in a companion paper (see R77-188, May 1977 Computer) about the student performance on an interactive examination.

An inferential computer consultant is being designed and implemented at the University of Illinois by a research group headed by Professor R. S. Michalski. The computer consultant is intended to extend the capabilities of current information systems by including deductive capabilities and introducing inductive capabilities. Induction is performed using variable-valued logic techniques on sets of facts called event sets. These event sets are most naturally stored using relational tables as proposed by Codd. In order to allow for the creation and manipulation of these relational tables, a data base sub-language has been developed. The description of this sub-language is the object of this thesis.

R77-219—Guttag, John V., "The Specification and Application to Programming of Abstract Data Types" (159 pp., University of Toronto, Toronto, Canada)  
Abstract data types can play a significant role in the development of software that is reliable, efficient, and flexible. Unfortunately, approaches available within programming languages fail to provide a mechanism for disentangling the abstract meaning of a data structure from a particular representation of it. Many extra-programming-language techniques, e.g., operational semantics, also exhibit this failing, plus a lack of formalism that tends to lead to specifications that are inconsistent or ambiguous. This thesis presents an algebraic specification technique that overcomes these problems. It consists of two parts: a syntactic specification provides the names, ranges, and domains of the operations associated with the type; a set of relations defines the meaning of the operations by stating their relationships to another. The principal difficulty lies in ascertaining whether or not a specification is sufficient to fully define the type. For this reason, a mechanical technique for verifying the sufficient-completeness of certain classes of algebraic specifications has been devised. Theorematic properties and application to top-down program design and verification are also discussed.

R77-220—Sethi, A. S. and Narsingh Deo, "Interference in Multiprocessor Systems with Varying Memory Access Probabilities" (52 pp., Indian Institute of Technology, Kanpur, India)  
Past studies of memory interference in multiprocessor systems have generally assumed that the references of each processor are uniformly distributed among the memory modules. This paper develops a model with local referencing, which reflects more closely the behavior of real-life programs. This model is analyzed using Markov chain techniques, and expressions are derived for the multiprocessor performance. New expressions are also obtained for the performance in the traditional uniform reference model and are compared with other expressions available in the literature. Results of a composition study are given to show the accuracy of the expressions for both models.

R77-221—Han, Yih-Wu and Larry L. Kinney, "Petri Net Reduction and Verification" (52 pp., Honeywell, Minneapolis, Minnesota)  
Petri nets and other labeled graphs have been widely used to model real systems and events. In order to make the application practical, a complex Petri net is usually reduced to a simpler net which possesses the same properties. Other cases use a hierarchical representation which suppresses unnecessary information and allows later expansion if needed. To verify the expansion, some systematic methods are still needed to verify that the original Petri net and the expanded net possess the same properties of interest. This paper presents a method for reducing live and bounded Petri nets which possess the properties of consistent behavior. An intuitive interpretation of these properties is suggested and illustrated with examples. Techniques for determining the consistency, invariance, transition firing ratios, and threshold number of a Petri net are also suggested.

R77-222—Airapetian, A. N. and J. F. McDonald, "Improved Test Set Generation Algorithm for Combinational Circuit Control" (18 pp., Polytechnic Institute of Yerevan, Armenia U.S.S.R.)  
This paper considers the generation of tests for detection of logic failures using an extension of Roth's D-algorithm. It is assumed that the circuit is composed of ANDs, ORs, NANDs, NORs, and NOTs and that only a single arbitrary failure can occur in the boolean tree and boolean graph type circuits studied. It is shown that the elaborated algorithms require less calculation than the D-algorithm.

The problem considered in this paper is test generation for detection of logical failures using an extension of the Roth D-algorithm. Both stuck-at failures and arbitrary failures are considered. New D-cubes are proposed for the control of the given gate. It is shown that the D-intersection proposed by Roth and a new type of intersection called the A-intersection will ensure the sensitive path in the circuit for any case of the given gate. The proposed algorithm, called the A-algorithm, based on these control D-cubes of a gate can generate tests for any stuck-at failure and for any arbitrary failure in the boolean tree and boolean graph type circuits. A comparison is made of the A-algorithm and the D-algorithm and it is shown that the A-algorithm requires less calculation to derive tests. In particular, for any arbitrary gate in the boolean tree type circuit the A-algorithm demands only half the calculation to generate all tests.