II. HEURISTIC SEARCH
- A. Heuristic Search Overview
- B. Search Spaces
  - 0. Overview
  - 1. State-space representation
  - 2. state-space search
  - 3. Problem-reduction representation
  - 4. AND-OR trees and graphs
- C. "Blind" Search Strategies
  - 1. Overview
  - 2. Breadth-first searching
  - 3. Depth-first searching
  - 4. Bi-directional searching
  - 5. Minimaxing
  - 6. Alpha-Beta searching
- D. Using Heuristics to Improve the Search
  - 1. Overview
  - 2. Best-first searching
  - 3. Hill climbing
  - 4. Means-ends analysis
  - 5. Hierarchical search, planning in abstract spaces
  - 6. Branch and bound searching
  - 7. Band-width searching
- E. Programs employing (based on) heuristic search
  - 1. Overview
  - 2. Historically important problem solvers
    - a) GPS
    - b) Strips
    - c) Gelernter's Geom. Program

III. AI Languages
- A. Early list-processing languages
- B. Language/system features
  - 0. Overview of current LP languages
  - 1. Control structures
  - 2. Data Structures (lists, associations,
  - 3. Pattern Matching in AI languages
  - 4. Deductive mechanisms
- C. Current languages/systems
  - 1. LISP, the basic idea
  - 2. INTERLISP
  - 3. QLISP (mention QA4)
  - 4. SAIL/LEAP
  - 5. PLANNER
  - 6. CONNIVER
  - 7. SLIP
  - 8. POP-2
  - 9. SNOBOL
  - 10. QA3/PROLOGUE

IV. Representation of Knowledge
- A. Overviews
  1. Survey of representation techniques
  2. Issues and problems in representation theory
- B. Representation Schemes
  1. Predicate calculus
  2. Semantic nets -- Quillian, Hendrix, LNR
  3. Production rules
  4. MERLIN
  5. Procedures (SHRDLU, actors, demons)
  6. Frames
  7. Componential analysis
  8. Scripts
  9. KRL
  10. Multiple Knowledge sources - Blackboard
  11. Query languages
  12. FOL

V. SPEECH UNDERSTANDING SYSTEMS
- A. Overview (include a mention of ac. proc.)
- B. Integration of Multiple Sources of Knowledge
- C. The ARPA speech systems
  1. HEARSAY I
  2. HEARSAY II
  3. SPEECHLIS
  4. SDC-SRI System (VDMS)
  5. DRAGON

VI. Natural Language
- A. Overview - History & Issues
- B. Representation of Meaning
- C. Grammars and Parsing
  1. Review of formal grammars
  2. Extended grammars
     a. Transformational grammars
     b. Systemic grammars
     c. Case Grammars
  3. Parsing techniques
     a. Overview of parsing techniques
     b. Augmented transition nets, Woods
     c. CHARTS - GSP
- D. Text Generating systems
- E. Machine Translation
  1. Overview & history
  2. Wilks' machine translation work
- F. Famous Natural Language systems
  1. Early NL systems (SAD-SAM through ELIZA)
  2. PARRY
  3. MARGIE
  4. LUNAR
  5. SHRDLU, Winograd

VII. Applications-oriented AI research (overview)
- A. Chemistry
  - 1. Mass spectrometry - DENDRAL
  - 2. Organic Synthesis - overview
- B. Medicine
  - 1. MYCIN
  - 2. Others
- C. Psychology and Psychiatry
  - 1. Protocol Analysis (Waterman and Newell)
- D. Math systems
  - 1. REDUCE
  - 2. MACSYMA (mention SAINT)
- E. Business and Management Science Applications
  - 1. Assembly line/ power distrib.
- F. Miscellaneous
  - 1. LUNAR
  - 2. Education
  - 3. SCHOLAR
  - 4. SOPHIE
  - 5. SRI computer-based consultation
  - 6. RAND--RITA production rule system
  - 7. Randevous - Query languages

VIII. AUTOMATIC PROGRAMMING
- A. Overview
- B. Program Specification Techniques
- C. Program Synthesis techniques
  - 0. Overview
  - 1. Traces
  - 2. Examples
  - 3. Problem solving applications to AP
    - a. Sussman's Hacker
    - b. Program Synthesis by Theorem Proving
  - 4. Codification of Programming Knowledge
  - 5. Integrated AP Systems
- D. Program optimization techniques
- E. Programmer's aids
- F. Program verification

IX. THEOREM PROVING
- A. Overview
- B. Resolution Theorem Proving
  - 1. Basic resolution method
  - 2. Syntactic ordering strategies
  - 3. Semantic & syntactic refinement
- C. Non-resolution theorem proving
  - 0. Overview
  - 1. Natural deduction
  - 2. Boyer-Moore
  - 3. LCF
- D. Uses of theorem proving
  - 1. Use in question answering

2. Use in problem solving
3. Theorem Proving languages
4. Man-machine theorem proving

E. Predicate Calculus
F. Proof checkers

X. Human Information Processing -- Psychology
A. Perception
B. Memory and Learning
1. Basic structures and processes in IPP
2. Memory Models
a. semantic net memory models
b. HAM (Anderson & Bower)
c. EPAM
d. Productions (HPS)
e. Conceptual Dependency
C. Psycholinguistics
D. Human Problem Solving
0. Overview
1. PBG's
2. Human chess problem solving
E. Behavioral Modeling
1. Belief Systems
2. Conversational Postulates (Grice, TW)
3. PARRY

XI. VISION
A. Overview
B. Polyhedral or Blocks World Vision
1. Overview
2. Guzman
3. Falk
4. Waltz
C. Scene Analysis
1. Overview
2. Template Matching
3. Edge Detection
4. Homogeneous Coordinates
5. Line Description
6. Noise Removal
7. Shape Description
8. Region Growing (Yakamovsky, Olander)
9. Contour Following
10. Spatial Filtering
11. Front End Particulars
12. Syntactic Methods
13. Descriptive Methods
D. Robot and Industrial Vision Systems
1. Overview and State of the Art
2. Hardware
E. Pattern Recognition
1. Overview
2. Statistical Methods and Applications

3. Descriptive Methods and Applications
F. Miscellaneous
1. Multisensory Images
2. Perceptrons

XII. ROBOTICS
A. Overview
B. Robot Planning and Problem Solving
C. Arms
D. Present Day Industrial Robots
E. Robotics Programming Languages

XIII. Learning and Inductive Inference
A. Overview
B. Samuel Checker program
C. Winston -- concept formation
D. Pattern extrapolation problems--Simon,
E. Overview of Induction
F. AQVAL (Michalski at U.Ill)
G. Parameter adjustment of linear functions
H. Rote learning
I. D.A. Waterman's machine learning of heuristics
J. Learning by debugging
K. Learning by parameter Adaptation
L. Signature & move phase tables

XIV. Reasoning and Planning
A. Reasoning by analogy
1. Overview
2. ZORBA
B. planning
1. NOAH
2. ABSTRIPS

Appendix III

SUMMARY OF MAINSAIL LANGUAGE FEATURES

MAINSAIL LANGUAGE FEATURES

Clark R. Wilcox
Stanford University

Portable ALGOL-like language with dynamic memory support

MAINSAIL is an ALGOL-like language with dynamic memory support for strings, arrays, records, modules and files. The driving force behind its design is that it provide for the development of portable software. At the same time, low-level features allow the programmer to deal with the underlying representation of data aggregates. These low-level features have made it possible for most of the runtime system to be written as MAINSAIL modules.

Intended applications

MAINSAIL is not oriented toward any particular application. The flexible use of memory makes it suitable for tasks with memory requirements which are difficult to predict prior to execution, as is often the case with knowledge representation. The string capabilities facilitate word processing applications such as compilers, text editors and document preparation, and "friendly" interactive programming. These same facilities require runtime support, so that a MAINSAIL program is not a stand-alone body of code, and thus may not be appropriate for some primitive system utilities.

Portability

A primary goal is that compatible implementations be provided on a variety of computer systems. Programs which are written for portability should be able to execute on any of the implementations with the same effect. Such programs must adhere to reasonable constraints with regard to data and memory ranges, as described in the language manual. Programs which violate these constraints are not considered portable, and thus may behave differently on different implementations. This design for portability raises a number of questions with regard to how well MAINSAIL will fit any particular machine. It is too early to provide a conclusive answer to such concerns, though it appears that many machines will efficiently support MAINSAIL implementations.

Modularity

In addition to the more obvious effects the machine-independent design has on data types and operations, it also necessitates a model of runtime interactions which can be supported on a broad range of computers. In particular MAINSAIL must be able to execute in a limited address space, which means that programs must be broken into pieces (modules) which need be in memory only when executing. The inability to characterize linkage and overlay systems in a

machine-independent manner has forced MAINSAIL to take over these functions, and thus assume duties often considered part of the operating system.

A MAINSAIL program consists of an open-ended collection of modules, i.e., the programmer need not specify what modules make up a program. The modules may originate from many files at execution, as contrasted to the common approach of having a single "save file" or "load module" which may contain an overlay structure.

The modules are compiled separately and assembled into a form which does not require linkage prior to execution. MAINSAIL resolves all inter-module references at runtime. Modules are automatically brought into memory as needed. If there is insufficient room in memory for an incoming module, MAINSAIL automatically swaps out one or more resident modules to make room. This swapping could involve i/o to an external device or memory mapping. Modules are position-independent, i.e., they do not contain references to fixed memory locations. Thus they may be moved about during execution, and need not be swapped into the same memory locations from which they were swapped out. This generalization of the traditional overlay structure will make possible the implementation of sizeable programs in a limited address space, while at the same time utilizing the minimum possible memory on larger systems.

Range of data types

In order to allow efficient operation on machines with a small word size, yet access to large values when necessary, MAINSAIL offers both short and "long" data types: integer, long integer, real, long real, bits and long bits. In practice the long forms are used much less frequently than the short forms, and thus can be simulated if necessary with no major degradation in efficiency. These data ranges have been chosen to fit the range of machines for which MAINSAIL is intended.

Strings

A MAINSAIL string is a variable length sequence of characters. The programmer does not need to specify a maximum length for a string as is common in many languages. Instead, MAINSAIL keeps track of the current number of characters in a string and automatically handles storage allocation. Most existing general-purpose languages have omitted a full implementation of strings, apparently under the assumption that they could not be efficiently implemented, and were dispensable. However, the hardware design trend is toward microprogrammed instruction sets which support string operations, in view of the increasing acceptance of computers for word-processing.

Classes, records and pointers

MAINSAIL employs a general notion of "class" as a collection of data and procedures fields. Classes serve two purposes: they specify the interfaces through which modules communicate with one another; and they are used as templates for the creation of and access to records. A record is a dynamically allocated memory area which contains data corresponding to the fields of the class to which it belongs. The fields of a record are accessed by means of a pointer to the record, combined with the name of the field. The pointer must have been associated with the record's class when it was declared.

The notion of "prefix class" was introduced to provide for a hierarchy of classes. A class which is declared with a prefix class is conceptually made a member of the prefix class, and inherits the fields of the prefix class as its initial fields. For example, the concept "doubly-linked list" may be represented as a class with two pointer fields, say "left" and "right". Any other class will automatically inherit these two fields if it is defined as a doubly-linked-list class.

The language contains rules which govern the use of pointers according to the relationships between classes and prefix classes. MAINSAIL provides for secure use of pointers in the majority of cases, but allows insecure operations if desired.

Arrays

MAINSAIL's implementation of arrays is quite flexible in that it allows the programmer full control over the creation and disposal of arrays. This is to be contrasted with classical ALGOL, where array allocation is tied to block structure. An array is actually a pointer to a record, and thus is allowed many of the same constructs provided for pointers, such as assignment, equality comparison, and parameter passing. An array may be a field of a class, so that any number of records may be allocated which contain array fields. This capability is particularly useful in image processing, where flexible array allocation can significantly simplify program logic.

Procedures

Procedures play a major role in MAINSAIL. Procedures may be typed for use in expressions. There are three simple parameter passing mechanisms: USES passes the value; PRODUCES passes a value back to the caller; and MODIFIES passes and returns a value. Optional arguments, repeatable arguments, and generic procedures provide useful syntactic constructs. Any procedure may be invoked recursively. Other procedure characteristics are COMPILETIME (if all arguments are constants, the procedure is evaluated during compilation), INLINE (produces "in-line" code), and CODED (supports assembly language coding).

Embedded assembly language

A number of facilities support the use of assembly language within a MAINSAIL program: CODED procedures, the Code statement, and the various forms of encoding variable offsets. Of course assembly language cannot appear within a machine-independent program, but nevertheless there are many instances when the target machine is known. The MAINSAIL interface to each operating system makes extensive use of the assembly language facilities.

Compiletime support

Most present-day compilers were designed to work in a sequential access mode, and suffer from the resulting limitations. The MAINSAIL compiler was designed with the understanding that the source files would be on random-access devices, so that it need not progress through the file in a strictly linear fashion. Any number of nested input files are allowed, in fact the same file may be scanned several times during compilation (contrast this with a compiler designed for input from punched card decks).

Compilation involves interaction with the user in that the programmer can put messages in a source file which are displayed during compilation. The user can affect the course of the compilation by specifying the names of files to be compiled as requested by directives within the file being compiled, and by defining values which govern the scanning of the source text. The compiler has the ability to quickly search through a file for the text to be compiled as specified either by earlier source text, or interactively by the user. This allows a single file to be made a repository of fragments of source text needed during many different compilations, and quickly searched during a particular compilation.

Conditional compilation allows an arbitrarily complicated expression (ultimately made up of constant operands) to be evaluated by the compiler to determine whether a particular segment of the source file is to be ignored. In general, the compiler will evaluate all expressions involving only constant operands (of type boolean, (long) integer, (long) bits, and string) and compiletime procedures. These facilities are quite important when building a large parameterized system.

A save and restore facility allows the current state of the symbol table to be saved. It may be restored during a later compilation to avoid recompiling unchanged text. This is particularly useful for the development of a collection of modules all of which utilize one or more common "header" files.

A comprehensive macro facility provides for the definition of constants, arbitrary text, and arbitrary text with parameters. Many commonly used constants are predefined, especially as needed by the system procedures to simplify passing of bits parameters consisting of predefined "flags".

File system

A simple yet powerful file system has been designed which, like all features of MAINSAIL, is guaranteed on every implementation. When a file is opened for use, the program specifies whether it contains text or data (binary), and whether access is sequential or random. A fundamental assumption is the ability to communicate with a controlling terminal, called the tty ("teletype"). For example, error message are output to tty, and a response is expected.

Appendix IV

MICROPROGRAMMED MAINSAIL PLANS

Plans for a Microprogrammed Implementation of MAINSAIL

Clark R. Wilcox
Stanford University

In this appendix we shall discuss our plans for a microprogrammed implementation of MAINSAIL. The goal of this research is to determine the feasibility of distributing a cost-effective integrated hardware-software programming environment. A computer which operates under the control of a microprogrammable control store offers a new approach to efficient program execution which we summarize below. We feel this approach could offer the means of developing reasonably-priced computing resources with the capability of executing programs which are too demanding for present mini-computers. It appears that such machines may be widely available within a few years. We propose to purchase the necessary hardware to enable us to develop a microprogrammed MAINSAIL implementation.

The emulation approach to high-level language implementation

Traditional implementations of high-level language involve translation to the fixed machine languages of the target machines. Such machine languages have not been designed for the efficient representation of high-level languages, with the result that an excessive number of overhead instructions are required to map the high-level language into its directly-executable machine code "surrogate". With the advent of microprogrammable computers with writable control stores, a different approach appears to have great promise for the efficient execution of high-level languages.

A micro-coded computer executes the instructions in main memory under control of the micro program. Thus the machine code may be viewed as data which is interpreted, or emulated, by the micro program, rather than as direct signals to the hardware. The micro program is written in a more primitive machine code called micro code, which (usually) directly controls the hardware. Most micro-coded computers have been designed for the emulation of a particular machine code, and thus the micro-code is simply a means of reducing the complexity of the hardware while perhaps providing a "higher-level" machine code. The micro-code is placed into a high-speed memory (relative to main memory), so that many micro instructions can be executed in the time it takes to fetch a single instruction from main memory.

The same technique of interpreting a particular machine code with a micro program can be broadened to the ability to interpret an arbitrary machine code. Such a micro computer is called a "soft" machine, or "universal host", since it is not oriented toward any particular machine code. Instead, the language implementor chooses a suitable machine-code representation. A compiler is constructed which translates into this representation, and a micro program is

written which interprets the representation. This approach is known as a "directly executable language", or DEL, since the high-level language has been translated into a form tailor-made for it. The unnecessary overhead instructions are eliminated, with a resulting decrease in program representation and increase in execution speed. There is evidence [3,4,6] that this approach can provide substantial dividends.

A MAINSAIL Directly Executable Language (DEL)

We propose to design a MAINSAIL DEL and implement it on a microprogrammable computer. The goal is to evaluate the economic and technical advantages of exporting a combined hardware-software environment for program development and distribution. In particular, we want to orient MAINSAIL's design and implementation toward such an emulation approach and compare the resulting "MAINSAIL machine" with conventional implementations.

We are interested in determining whether a "soft" machine of this sort can be provided cheaply enough to serve as a basis for the distribution of software which presently requires expensive hardware facilities. Hardware which can be specifically tailored for high-level language execution may provide the quickest route to the economically viable distribution of programs which exceed the limits of present general-purpose mini-computers.

This work will complement the on-going implementations of MAINSAIL on conventional hardware. Thus we will be in a unique position to compare the two approaches. We expect the MAINSAIL DEL to outperform other MAINSAIL implementations in much the same way that DELtran (a DEL for FORTRAN II) outperforms FORTRAN II [3]. Initial measurements show that the DELtran representation is less than one fifth the size of the code generated by the FORTRAN-H optimizing compiler, and executes about five times faster.

MAINSAIL is perhaps better suited to the emulation approach than FORTRAN because of the locality of reference provided by procedures, records and modules. A preliminary DEL has already been designed for MAINSAIL, but further work is necessary before we can predict (or demonstrate) size and execution comparisons with standard implementations. There is much work to be done in determining the efficient representation of ALGOL-like languages for the purpose of emulation, and providing data from actual implementations.

A MAINSAIL DEL could provide facilities which are impossible to provide in an efficient manner on conventional machines. These facilities relate to the monitoring of the program during execution. Since the emulator is simply a program written in micro code, it can be made to perform any kind of execution-time checks with no need to alter the DEL. By contrast, the MAINSAIL compiler must generate different code depending on the amount of checking to be performed.

The emulator can also provide execution profiles and comprehensive debugging facilities such as instruction traps and single stepping. We expect to provide several emulators which are oriented toward particular types of execution, e.g. a "fast" emulator which maximizes execution speed, a "careful" emulator which provides comprehensive runtime checks, a "performance monitoring" emulator which gathers information concerning program execution, and a "debugging" emulator which allows interactive debugging.

Another advantage of the emulation approach is the simplifications in the compiler. Since the compiler will translate MAINSAIL to its own DEL, the code generators become almost trivial. MAINSAIL operations which require many instructions on existing machines can be compactly represented with the DEL. The compiler need not worry about register optimization since there will be no registers in the DEL representation. Since the MAINSAIL DEL is a close representation of the source code, there is no reason to "drop into assembly language" since any "sensible" program which could be written in the DEL could more easily be written in MAINSAIL.

Hardware support

To support this development, we propose the purchase of a dynamically micro-programmable machine with such supporting hardware as is necessary. This machine should be a universal host in the sense that it is not already oriented towards a particular machine code. Its software support is of little consequence since we will design our own operating system and high-level language support.

We are interested in implementing sophisticated programs, and thus require a large address space (say 24 bits) and 32-bit arithmetic. We need sufficient control store, say 16K words, to support a debugging emulator and selected parts of the operating system. The micro store must be able to quickly transfer words to and from main memory, in particular we want to be able to quickly switch emulators. There must be facilities for interface to a variety of peripherals, and to other computers.

There are some machines now available along these general lines (e.g. [1]), with the introduction of more imminent. Indeed, manufacturers are beginning to include user-microprogrammable features with new models of their traditional hardware, e.g. Digital Equipment Corporation's PDP-11/60 and Data General's Eclipse.

One such machine, EMMY, has been developed by the Stanford Emulation Laboratory, under the direction of Professor Michael Flynn of the Department of Electrical Engineering [2,5]. EMMY is a universal host machine which closely fits our needs. It is an unbiased yet efficient host for a wide range of target machine architectures. EMMY is scheduled to go into production in late 1977 by ICL of England (the emulation laboratory has been involved in the development of a prototype). We feel that this machine would suit our needs, but further evaluation is necessary.

We expect most of development of the MAINSAIL DEL to be independent of any particular micro program representation. In particular, we are not at this time proposing to carry out any hardware design to orient the host processor towards MAINSAIL, though this approach would be reasonable if a large number of processors were to be distributed solely to support MAINSAIL execution.

References

1. Burroughs Corp., "B-1700 Systems Reference Manual," Burroughs Corp., Detroit, Michigan, 1972.

2. Flynn, M. J., Hoevel, L. W., and Neuhauser, C. J., "The Stanford Emulation Laboratory," Digital Systems Lab., Technical Report No. 118, Stanford University, June 1976.

3. Hoevel, L. W. and Flynn, M. J., "The Structure of Directly Executed Languages: A New Theory of Interpretive System Support," Digital Systems Laboratory, Technical Report No. 130, Stanford University, March 1977.

4. Hoevel, L. W., "DELtran Principles of Operation," Digital Systems Laboratory, Technical Note No. 108, Stanford University, March 1977.

5. Neuhauser, C. J., "An Emulation Oriented, Dynamic Microprogrammable Processor," Digital Systems Lab., Technical Note No. 65, Stanford University, October 1975.

6. Wilner, W., "Burroughs B-1700 Memory Utilization," AFIPS Proceedings, Vol. 41-I, FJCC, 1972, pp. 579-586.

Appendix V

AIM MANAGEMENT COMMITTEE MEMBERSHIP

The following are the membership lists of the various SUMEX-AIM management committees at the present time:

AIM EXECUTIVE COMMITTEE:

LEDERBERG, Joshua, Ph.D. (Chairman)
Department of Genetics, S331
Stanford University Medical Center
Stanford, California 94305
(415) 497-5801

AMAREL, Saul, Ph.D.
Department of Computer Science
Rutgers University
New Brunswick, New Jersey 08903
(201) 932-3546

BAKER, William R., Jr., Ph.D. (Executive Secretary)
Biotechnology Resources Program
National Institutes of Health
Building 31, Room 5B43
9000 Rockville Pike
Bethesda, Maryland 20014
(301) 496-5411

LINDBERG, Donald, M.D. (Adv Grp Member)
605 Lewis Hall
University of Missouri
Columbia, Missouri 65201
(314) 882-6966

MYERS, Jack D., M.D.
School of Medicine
Scaife Hall, 1291
University of Pittsburgh
Pittsburgh, Pennsylvania 15261
(412) 624-2649

AIM ADVISORY GROUP:

LINDBERG, Donald, M.D. (Chairman)
605 Lewis Hall
University of Missouri
Columbia, Missouri 65201
(314) 882-6966

AMAREL, Saul, Ph.D.
Department of Computer Science
Rutgers University
New Brunswick, New Jersey 08903
(201) 932-3546

BAKER, William R., Jr., Ph.D. (Executive Secretary)
Biotechnology Resources Program
National Institutes of Health
Building 31, Room 5B43
9000 Rockville Pike
Bethesda, Maryland 20014
(301) 496-5411

BOBROW, Daniel G., Ph.D. [Term expiring]
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304
(415) 494-4438

FEIGENBAUM, Edward, Ph.D.
Department of Computer Science
Polya Hall, Room 213
Stanford University
Stanford, California 94305
(415) 497-4079

FELDMAN, Jerome, Ph.D. [Term expiring]
Department of Computer Science
University of Rochester
Rochester, New York
(716) 275-5671

LEDERBERG, Joshua, Ph.D. (Ex-officio)
Principal Investigator - SUMEX
Department of Genetics, S331
Stanford University Medical Center
Stanford, California 94305
(415) 497-5801

MILLER, George, Ph.D. [Term expiring]
The Rockefeller University
1230 York Avenue
New York, New York 10021
(212) 360-1801

MOHLER, William C., M.D.
Associate Director
Division of Computer Research and Technology
National Institutes of Health
Building 12A, Room 3033
9000 Rockville Pike
Bethesda, Maryland 20014
(301) 496-1168

MYERS, Jack D., M.D.
School of Medicine
Scaife Hall, 1291
University of Pittsburgh
Pittsburgh, Pennsylvania 15261
(412) 624-2649

REDDY, D.R., Ph.D. [Term expiring]
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania
(412) 621-2600, Ext. 149

SAFIR, Aran, M.D.
Department of Ophthalmology
Mount Sinai School of Medicine
City University of New York
Fifth Avenue and 100th Street
New York, New York 10029
(212) 369-4721

STANFORD COMMUNITY ADVISORY COMMITTEE:

LEDERBERG, Dr. Joshua (Chairman)
Principal Investigator - SUMEX
Department of Genetics, S331
Stanford University Medical Center
Stanford, California 94305
(415) 497-5801

COHEN, Stanley N., M.D.
Department of Clinical Pharmacology, S169
Stanford University Medical Center
Stanford, California 94305
(415) 497-5315

DJERASSI, Dr. Carl
Department of Chemistry, Stauffer I-106
Stanford University
Stanford, California 94305
(415) 497-2783

FEIGENBAUM, Dr. Edward
Serra House
Department of Computer Science
Stanford University
Stanford, California 94305
(415) 497-4878

LEVINTHAL, Dr. Elliott C.
Department of Genetics, S047
Stanford University Medical Center
Stanford, California 94305
(415) 497-5813

Appendix VI

USER INFORMATION - GENERAL BROCHURE

Revised May 1976

Appendix VII

GUIDELINES FOR PROSPECTIVE USERS

SUMEX-AIM RESOURCE
INFORMATION FOR POTENTIAL USERS

National users may gain access to the facility resources through an advisory panel for a national program in Artificial Intelligence in Medicine (AIM). The AIM Advisory Group consists of members-at-large of the AI and medical communities, facility users and the Principal Investigator of SUMEX as an ex-officio member. A representative of the National Institutes of Health-Biotechnology Resources Program (NIH-BRP) serves as Executive Secretary.

Under its enabling 5-year grant, the SUMEX-AIM computing resource is allocated to qualified users without fee. This, of course, entails a careful review of the merits and priorities of proposed applications. At the direction of the Advisory Group, expenses related to communications and transportation to allow specific users to visit the facility also may be covered.

USER QUALIFICATIONS

The SUMEX-AIM facility is a community effort, not merely a machine service. Applications for membership are judged on the basis of the following criteria:

1) The scientific interest and merit of the proposed research and its relevance to the health research missions of the NIH.

2) The congruence of research needs and goals to the AI functions of SUMEX-AIM as opposed to other computing alternatives.

3) The user's prospective contributions and role in the community, with respect to computer science, e.g., developing and sharing new systems or applications programs, sharing use of special hardware, etc.

4) The user's potential for substantive scientific cooperation with the community, e.g., to share expert knowledge in relevant scientific specialties.

5) The quantitative demands for specific elements of the SUMEX-AIM resource, taking account of both mean and ceiling requirements.

In many respects, this requires a different kind of information for judgment of proposals than that required for routine grant applications seeking monetary funding support. Information furnished by users also is indispensible to the SUMEX staff in conducting their planning, reporting and operational functions.

The following questionnaire encompasses the main issues concerning the Advisory Group. However, this should neither obstruct clear and imaginative presentation nor restrict format of the application. The potential user should prepare a statement in his own words using previously published material or other documents where applicable. In this respect, the questionnaire may be most useful as a checklist and reference for finding in other documentation the most cogent replies to the questions raised.

For users mounting complex and especially non-standard systems, the decision to affiliate with SUMEX may entail a heavy investment that would be at risk if the arrangement were suddenly terminated. The Advisory Group endeavors to follow a responsible and sensitive policy along these lines--one reason for cautious deliberation; and even in the harshest contingencies, it will make every effort to facilitate graceful entry and departure of qualified users. Conversely, it must have credible information about thoughtful plans for long-term requirements including eventual alternatives to SUMEX-AIM. SUMEX-AIM is a research resource, not an operational vehicle for health care. Many programs are expected to be investigated, developed and demonstrated on SUMEX-AIM with spinoffs for practical implementation on other systems. In some cases, the size, scope and probable validation of clinical trials would preclude their being undertaken on SUMEX-AIM as now constituted. Please be as explicit as possible in your plans for such outcomes.

Applicants, therefore, should submit:

1) One to two-page outline of the proposal.

2) Response to questionnaire, cross-referenced to supporting documents where applicable.

3) Supporting documents.

4) List of submitted materials, cross-referenced.

We would welcome a draft (2 copies) of your submission for informal comment if you so desire. However, for formal consideration by the SUMEX-AIM Advisory Group, please submit 13 copies of the material requested above in final form.

Elliott Levinthal, Ph.D.
AIM User Liaison
SUMEX-AIM Computer Project
c/o Department of Genetics, S047
Stanford University Medical Center
Stanford, California 94305
Telephone: (415) 497-5813

May, 1976

SUMEX-AIM RESOURCE

QUESTIONNAIRE FOR POTENTIAL USERS

Please provide either a brief reply to the following or cite supporting documents.

A) MEDICAL AND COMPUTER SCIENCE GOALS

1) Describe the proposed research to be undertaken on the SUMEX-AIM resource.

2) How is this research presently supported? Please identify application and award statements in which the contingency of SUMEX-AIM availability is indicated. What is the current status of any application for grant support of related research by any federal agency? Please note if you have received notification of any disapproval or approval, pending funding, within the past three years. Budgetary information should be furnished where it concerns operating costs and personnel for computing support. Please furnish any contextual information concerning previous evaluation of your research plans by other scientific review groups.

3) What is the relevance of your research to the AI approach of SUMEX-AIM as opposed to other computing alternatives?

B) COLLABORATIVE COMMUNITY BUILDING

1) Will the programs designed in your research efforts have some possible general application to problems analogous to that research?

2) What application programs already publically available can you use in your research? Are these available on SUMEX-AIM or elsewhere?

3) What opportunities or difficulties do you anticipate with regard to making available your programs to other collaborators within a reasonable interval of publication of your work?

4) Are you interested in discussing with the SUMEX staff possible ways in which other artificial-intelligence research capabilities might interrelate with your work?

5) If approved as a user, would you advise us regarding collaborative opportunities similar to yours with other investigators in your field?

C) HARDWARE AND SOFTWARE REQUIREMENTS

1) What computer facilities are you now using in connection with your research or do you have available at your institution? In what respect do these not meet your research requirements?

2) What languages do you either use or wish to use? Will your research require the addition of major system programs or languages to the system? Will you maintain them? If you are committed to systems not now maintained at SUMEX, what effort would be required for conversion to and maintenance on the PDP-10 - TENEX system? What are the merits of the alternative plan of converting your application programs to one of the already available standards? Would the latter facilitate the objectives of Part B), Collaborative Community Building?

3) Can you estimate your requirements for CPU utilization and disk space? What time of day will your CPU utilization occur? Would it be convenient or possible for you to use the system during off-peak periods? Please indicate (as best you can) the basis for these estimates and the consequences of various levels of restriction or relaxation of access to different resources. SUMEX-AIM's tangible resources can be measured in terms of:

   a) CPU cycles.

   b) Connect time and communications.

   c) User terminals (In special cases these may be supported by SUMEX-AIM.).

   d) Disk space.

   e) Off-line media-printer outputs, tapes (At most, limited quantities to be mailed.).

   Can you estimate your requirements? With respect to a) and b), there are loading problems during the daily cycle.--Can you indicate the relative utility of prime-time (0900-1600 PST) vs. off-peak access?

4) What are your communication plans (TYMNET, ARPANET, other)? How will your communication and terminal costs be met? See following note concerning network connections to SUMEX-AIM.

5) If this is a development project, please indicate your long-term plans for software implementation in an applied context keeping in mind the research mission of SUMEX-AIM.

Our procedures are still evolving, and we welcome your suggestions about this framework for exchanging information. Needless to say, each question should be qualified a) "insofar as relevant to your proposal", and b) "to the extent of available information".

Please do not force a reply to a question that seems inappropriate. We prefer that you label it as such so that it can be dealt with properly in future dialogue.

Above all, we are eager to work with potential users in any way that would help minimize bureaucratic burdens and still permit a responsible regard for our accountability both to the NIH and the public. Please do not hesitate to address the substance of these requirements in the format most applicable to you.

NETWORK CONNECTIONS TO SUMEX-AIM

TYMNET

Attached is a list of available TYMNET nodes and associated telephone numbers. The cost to users of using TYMNET is the telephone charge from user location to the nearest TYMNET node. This is available only for communication to SUMEX-AIM and not for other facilities that may be connected to TYMNET. In some cases, there are "foreign exchanges" set up by users. These may offer less expensive communication. Details of these possibilities can best be learned by calling the nearest TYMNET node. The telephone company can provide information on comparative costs of leased lines, toll charges, etc. The initial capital investment for TYMNET installation as well as login and hourly charges is provided by SUMEX-AIM. Standard usage charges on TYMNET are approximately $3/connect-hour.

ARPANET

SUMEX-AIM is connected to the ARPANET. Our name is SUMEX-AIM; our nickname is AIM. We support the new TELNET protocol. Our network address is decimal 56, octal 70. This provides convenient access for ARPANET Hosts and Associates and those who have accounts with ARPANET.