The First Conference on Artificial Intelligence Applications

ISBN NUMBER 0-8186-0624-X
IEEE CATALOG NUMBER 84CH2107-1
LIBRARY OF CONGRESS NUMBER 84-82214
IEEE COMPUTER SOCIETY ORDER NUMBER 624

The First Conference on Artificial Intelligence Applications

Sponsored by IEEE Computer Society
In Cooperation with American Association for Artificial Intelligence

Sheraton, Denver Tech Center • December 5-7, 1984

CAIA

COMPUTER SOCIETY PRESS

IEEE COMPUTER SOCIETY

1884 1984 THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS INC

Published by IEEE Computer Society Press
1109 Spring Street
Suite 300
Silver Spring, MD 20910

ISBN 0-8186-0624-X (paper)
ISBN 0-818[illegible]-[illegible]624-1 (microfiche)
ISBN 0-8186-8624-3 (casebound)
IEEE Catalog No. 84CH2107-1
Library of Congress No. 84-82214
IEEE Computer Society Order No. 624

Order from: IEEE Computer Society
Post Office Box 80452
Worldway Postal Center
Los Angeles, CA 90080

IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854

1884 1984 THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

General Chairman's Message

I would like to welcome everyone to the First IEEE Computer Society Conference on Artificial Intelligence Applications. We have viewed artificial intelligence in the broadest possible manner: any machine behavior that in some sense mimics intelligent human behavior. The papers at this conference reflect this orientation, covering everything from reasoning and expert systems through natural language and computer vision. We have not adhered to any particular school of thought and we have tried to include all. We emphasize applications, but have not excluded theory.

Watch the bulletin boards for the message indicating when the pattern analysis and machine intelligence technical committee will be meeting. If you are interested in helping organize the next artificial intelligence applications conference or in any other way help, feel free to attend the meeting.

Robert M. Haralick
General Chairman

Program Chairman's Message

It is with a great deal of pride and pleasure that I present this proceedings of the First Conference on Artificial Intelligence Applications. Arranging the program for this conference was a shade more difficult since agreeing on what constitutes an "artificial intelligence application" is not without difference of opinion. Since this is the first conference of its kind sponsored by the IEEE Computer Society, I have interpreted the purpose of this conference rather broadly.

I would like to take this opportunity to thank the general chairman, Professor Robert M. Haralick, for his help and the program committee for all the support it has provided. Also, I would like to convey my sincere thanks to the many reviewers for the excellent job of reviewing at such a short notice.

Again, it is a pleasure to thank the IEEE Computer Society for sponsoring this conference and its staff for their marvelous help in making various arrangements. Also, I would like to convey my sincere thanks to the American Association for Artificial Intelligence for cooperating in this conference and to Ms. Claudia Mazzetti for many helpful suggestions.

J.K. Aggarwal
Program Chairman

Conference Committee

General Chairman
R.M. Haralick

Program Chairman
J.K. Aggarwal

Local Arrangements Chairman
David Morgenthaler

Treasurer
Roger Ehrich

Tutorial Chairperson
Linda G. Shapiro

Program Committee
J.K. Aggarwal
B.G. Buchanan
R.O. Duda
R. Reddy
A. Rosenfeld
E.D. Sacerdoti
M. Tenenbaum
D. Waltz
L.A. Zadeh

List of Reviewers

Harry Barrow
David Barstow
S. Bedrosian
James Bennett
Robert Blum
Daniel Bobrow
Piero Bonissone
Robert S. Boyer
Bruce G. Buchanan
Raymond Carhart
Peter Cheeseman
C.H. Chien
Jan Clayton
John Cowles
Richard O. Duda
Robert Engelmore
Susan Ennis
Mark Fox
Thomas Garvey
Michael Genesereth
M. Georgoeff
Peter Hart
Frederick Hayes-Roth
Ramesh Jain
Jonathan King
Vipin Kumar
Douglas Lenat
Mike Magee
Worthy Martin
E.S. McVey
Ryszard Michalski
N. Nandhakumar
G. Novak
Kamran Parsaye
Raj Reddy
Elaine Rich
Azriel Rosenfeld
E.D. Sacerdoti
Robert Simmons
James Slagle
David Smith
J.J. Talavage
Marty Tenenbaum
Allan Terry
R. Tong
Jeffrey Van Baalen
David Waltz
Rand Waltzman
Donald Waterman
Sholom Weiss
Gio Wiederhold
R. Yager
Lofti A. Zadeh

Table of Contents

ORIGINAL PAGE IS
OF POOR QUALITY

*Not received in time for publication.

Session IA: Natural Language Interface

Cochairmen

R. Simmons
University of Texas at Austin
R.L. Kashyap
Purdue University

FROM MENUS TO INTENTIONS IN MAN-MACHINE DIALOGUE

Rob't F. Simmons

University of Texas, Austin, 1984

Abstract

Operating systems are designed to achieve goals not to recognize user intentions. But the use of *Help* systems and Menu-selection make them more useful and friendly. Natural Language interfaces must go farther by guessing what the user wants and what is meant but not specified. Natural language programming systems -- still in infancy -- promise explicit capability for a user to define his/her intentions explicitly. Text Knowledge systems -- at the research frontier -- bewilder us with the complexity of intentions expressed and implied. Current techniques for recognizing intentions and computing appropriate representations and responses are discussed in this paper.[1]

Of Mouse and Menu

A modern computing system includes hundreds of operators and programs each of which has its own control language. It is a technological marvel for accomplishing my information processing intentions -- providing only that I figure out how to use its capabilities to achieve my goals. The description and documentation of such a system entails several fat volumes of difficult text, so a user may frequently settle for using the system in limited ways to avoid the extensive effort required to learn it thoroughly. The complexity of such systems easily distinguishes between the casual, specialized user who knows an editor, a programming language, a text formatter, etc. from the "wizard-hacker" who can use the system to accomplish vastly more varied operations. The wizard-hacker knows much of the operating system and numerous control languages for its multitudinous operations; he/she is an indispensable consultant and teacher in any computer laboratory.

For the best equipped modern laboratories, specialized work stations have become available -- LISP machines, SUNs, etc. -- which are large, fast microcomputers supported by megabytes of internal memory, hundreds of megabytes of local disk storage, and somewhere in the background a gigantic file server, providing trillions of megs of longterm memory, and network communications with the world. More importantly, they include bit-map displays on the order of 1000 X 1000 pixels and a "mouse" whose motion translates into movements of a pointer on the display. One or more buttons on top of the mouse are used to select a portion of the screen (actually of the array that the screen displays) in order to focus on specific data. A most common use of the mouse is to select a choice from a menu; the pointer is guided to one of the choices that are displayed on the screen, a button is pressed on the mouse, and the system passes control to the procedure selected from the menu. In its turn, this procedure may present additional menus or actually accomplish some task. A menu is usually a box drawn on the screen containing cells with the names of the procedures that can be selected, although it may take other display forms such as a boxed border of the screen in which choices are shown.

Apple Corporation's MacIntosh is correctly advertised as a desk assistant for people who have no desire to program. With a one-button mouse and hierarchies of menus, the bit-map display presents to the user a large set of options for drawing, text preparation, editing, type-setting, display, and printing of data. One or two hour's experience with such a system is sufficient to enable most users to graduate from their typewriter to a high-technology information control device that not only replaces the printing functions of the typewriter, but adds capabilities for drawing diagrams and artwork, for storing files, for graphically displaying and charting data, and for endless creative activities. As long as the user is content with the capabilities provided by the menus, he/she is almost fully protected from asking impossible questions -- only meaningful sequences of menu choices are provided -- and the system is equally protected from user errors.

In general, the operating system, whether mouse and menu-controlled or controlled by formal language commands, provides a set of operations that a user can combine to accomplish a set of information processing goals. These goals, with careful design and a bit of luck, will represent what the system can understand of the user's *intentions*. From the users' viewpoint, it will be seen that these operations are primitives at the bottom of the human goal system. In menu-controlled systems the possible processing goals are completely pre-specified and the user need only select a permissible sequence to accomplish his/her goal if it is provided. In formal-language controlled systems, the user may be offered much more freedom in accomplishing goals within the capability of the system, but at the cost of learning several formal command languages at the operating system and production program levels, and subject to the ever-present potential of making errors. In a sense, the menu-controlled system anticipates a limited set of users' intentions and provides capabilities to accomplish them, while the formal language system can provide a vast space of possible intentions that the user can realize -- if he/she learns the several operating and production system languages that are required.

In the menu systems, intentionality is explicit and limited; in the formal language system it is largely left to the user (or to analysts who might consider all the possible, legal combinations of operations in order to describe the system). But human intentions are much more complicated, hierarchically ordered systems of abstract and concrete goals. A MacIntosh user, for example, might have a desire to win fame and fortune by publishing a paper communicating a simple description of some hitherto complex process. Perhaps

[1]Supported by NSF Grant IST-8403028

that high-level goal might have brought him to the purchase of his new computer and to the learning of skills in its use. To write the paper this user may have read widely, invented a technique, and walked long miles while he organized his thoughts for communication. When finally, he selects MacWrite and begins to prepare his paper, a new set of intentions or goals control his behavior. He may wish to illustrate his paper with diagrams, and embellish it with special type fonts; he may wish to include some humor, he may consult dictionaries or thesauri to select vocabulary, and insure correct spelling.

In any case, the operations provided by the computer correspond to only the most primitive and concrete of human goals; so intentions, as provided by the operating system, are only the basic building blocks in achieving vastly more complex intentions by its human users. At the operating system level, computers reliably apply the programs a user calls; user intentions must be translated to program capabilities. The system helps with information and documentation files, but only rarely recognizes the users' goals.

Natural Language Interfaces -- NLI

Natural spoken and written languages play a very large role in human communications and it was completely predictable from the inception of computers that efforts would be made to use natural language to communicate with them. The limited practical problem of querying databases using English became a most attractive goal, and one that resulted in the technically satisfying accomplishment of suitable grammars for translating subsets of English to formal query languages. But immediately following the technical success of working systems, tests with user communities showed that much more was required. One of the first problems to emerge was that many users first had to learn to type and to use a computer at the operating system level. More seriously, the typical user expected that a system that "understood" English, could understand what the user meant in asking a question and, in general, expected the system to be intelligent in a human way -- since it used a human language. In addition the user usually had no way to know the limits of the system's comprehension of natural language and thus no way of restricting the vocabulary and structure of his queries to those limits. The consequence appears to be that current users of NLI find some elementary and short English forms with which the system is successful and ignore the more complex capabilities which are more prone to failure.

A recent experiment [6] evaluating the use of an NLI English interface to a database compared the success of English queries with those of queries in a simple formal query language, SQL. For this experiment a set of problems was chosen for which answers could be derived from the DB whether asked in English or in SQL. Users were required under various experimental conditions to formulate a series of questions either in SQL or English to obtain the answers that would solve the problem. The findings were entirely surprising. The users succeeded in discovering answers to only about 46% of the questions when using SQL and for about 22% using English. Analysis revealed that in SQL, when a question failed there was enough feedback to enable (sometimes) successful revision, but in English the lack of feedback as to what went wrong left the user only the recourse of random paraphrase.[2] Such experiments as this demonstrate that the human factors problems associated with NLI are crucial; technical linguistic success is only a beginning

Descriptive Statistics for Study

Subjects	20	
Problems	39	
Tasks		
NLS	42	
SQL	45	
Total		87
Queries		
NLS	656	
SQL	425	
Total		1081

SQL - Formal Query Language
NLS - English Interface

Results

TASKS	NLS	SQL
Essentially correct	17 1	44 2
Partially solved	34 2	23 3
Not Solved	48 7	32 5
QUERIES		
Essentially correct	22 3	45 6
Correctable	75 5	57 0

Conclusions:

SQL CURRENTLY SUPERIOR TO NLS
NLS BRIEFER THAN SQL
NLS LACKS FEEDBACK TO USER

Subjects performed more poorly in both languages than laboratory studies predicted.

Figure 1:
EVALUATION OF NATURAL LANGUAGE FOR DATA RETRIEVAL

M. Jarke, *et. al.*[6]

Tennant [10] first studied human intentions with regard to natural language query systems and noted how poorly the NLI fitted expectations -- or how unrealistic user expectations were with respect to the capabilities of NLI. He and Thompson [11] adopted menu-control as a possible solution for the problem. In their approach, the user selects an option to query or input data; at that point a menu shows how a command can begin. Selecting an option causes a new menu to appear showing choices for possible continuations. This method insures that the user remains within the English subset and that his queries remain within semantic and pragmatic bounds of the system. Use of a mouse or voice selection largely eliminates typing problems. All in all, the menu-driven NLI results in error-free use, accompanied by satisfactory feedback showing the user the resulting translation to a simple formal language. The authors note, however, that the method will become extremely cumbersome if the NL subset becomes very large. A reader may also notice that the style of the English they use is already reminiscent of the flavor of database formal languages.

We can notice here that a menu containing natural language queries is a decision tree; beginning with say, "How many", branches exist for all categories in the database. For

---

[2]It is important to mention that the version of the NL system used in this study was not fully debugged, and that the computer facility used was sporadically available. Generally such studies show that NL questioning is about 70% satisfactory.

each such category, continuations for further qualifications and constraints are provided as shown below.

```
HOW MANY -- EMPLOYEES ----with-- QTY --CHILDREN
         -- MANAGERS          |-- SALARY of--QTY
         -- SALESMEN          |-- SENIORITY of
                                  --QTY -- years
         -- PARTS             |-- etc
         -- ASSEMBLIES
         -- etc
```

The first menu offers EMPLOYEES, MANAGERS, etc. When EMPLOYEES is chosen, the *with* choices appear as shown. If PARTS had been the choice, different continuations would have been presented.

Many years ago a noted researcher was laughed out of IBM research for supporting a machine translation system based on an exhaustive list of phrase equivalences for two natural languages. It was obvious that the language used in ordinary texts was too large to yield to such an approach. For NLI applied to database and operating system control, this exhaustive approach is, for the moment at least, more successful because the command languages into which the NL is translated are still relatively small and grammars are used to form economical descriptions of the NL subset.

Another human factors problem concerns the nature of the NLI answers to queries. Obvious to most researchers, the answers should be in a readable NL form, and so they have been in most systems. Not so obvious until studied by Kaplan [7] and Allen [1], is the requirement that the system estimate the users' intentions in asking the question and so respond appropriately. Allen used a train station example like the following:

Query: Is there a train to Zurich?
Response: Track 5 at 2100.

The clerk might have responded directly to the question with the answer, "Yes", but then the user would almost certainly have asked about time and location. The clerk, anticipating the questioner's intention to take or meet the train, returned a cooperative response providing the additional information. If there were no train to Zurich, the clerk might have responded,

"No, but an express bus leaves the bus station at noon daily."

In this event the clerk again is cooperating by anticipating the questioner's probable intention. Computational methods for dealing with intentions at this level are only beginning to be understood.

Kaplan emphasizes the importance of corrective responses in a database environment. In asking a question to a student database, the questioner may have certain misapprehensions. His example dialogue is paraphrased below:

Q: Which students failed CS135 in Spring 1984?
A: None.
Q: Did anyone get a D in CS135 in Spring 1984?
A: No.
Q: How many students passed CS135 in Spring 1984?
A: None.
Q: Was CS135 offered in Spring 1984?
A: No.

Aha! How much better if the system recognizes our misapprehension and immediately informs us that the course was not offered. Kaplan's computational solution is simple and attractive. Suppose the data structure has the following pattern:

(1:STUDENT 2:GRADE 3:COURSE-NAME)

The following failures are possible:

1: No such student
2: No such grade
3: No such course
1-2: No such grade for this student
1-3: No such course for this student
2-3: No such grade in this course

Reporting one of these reasons for failing a question serves to correct user presuppositions.

Programming in Natural Language, NLP

In the early history of computing, *automatic programming* meant the development of assembly languages that allowed the programmer to use English mnemonics for binary operation codes, and octal or decimal numbers to refer to computer-memory locations. Soon the meaning changed to refer to programming in higher level languages such as Fortran or Lisp. But that reference also changed to the present (confused) notion that automatic programming includes any method for specifying to a computer the requirements of a program in such a manner that it could be automatically compiled, somehow excluding the ordinary methods of specifying the structures and operations directly in a compilable programming language. The current notion seems to be to state what a program should do and allow a very smart compiler to determine the "how". Specification languages of choice include input output arrays, first order logic, and occasionally English and other natural languages.

Several experimental systems have been reported that allow a user to write programs in a strictly restricted English subset. Two recent examples include van Baalan's dissertation [12] in which recursive Lisp functions were compiled from English descriptions, and Biermann and Ballard's [3] experiments comparing the efficacy of programming array-manipulation problems in an English programming-subset and in PLC. The latter study reported the challenging finding that sophomore level programming students were able to program these operations faster and with fewer mistakes in English than in PLC. I take this report as a curiosity much in need of additional confirmation and of further study of conditions under which the findings hold.

As in any programming language, defining a system of programs in English allows the programmer to organize and specify the intentions which are to be achieved by the program. In the preceding section we saw that those intentions might include predictions of the intentions of the user, if for example, the program is to simulate an information clerk at a train station. But the use of an English subset as the programming vehicle suggests that the English compiler may be heavily burdened with the necessity for guessing what the programmer may mean by what he/she did not say directly; English uses pronouns, noun phrases, and elliptical references as ordinary parts of its structure and we would consider an English programming-subset sadly lacking if it did not recognize these. For example in a programming experiment by Yeong Ho Yu [13], translating from an English subset to procedural logic as a programming language, objects in the robot blocks world are defined as follows:

There is a table and a robotarm.
[TABLE ISA TABLE]
[ROBOTARM ISA ROBOTARM]

A white block is on the table and a blue block is on the white block.
[B1 ISA BLOCK][WHITE B1][ON B1 TABLE]
[B2 ISA BLOCK][BLUE B2][ON B2 B1]
There is a red block on the blue block.
[B3 ISA BLOCK][RED B3][ON B3 B2]
The block is a pyramid.
[B3 ISA PYRAMID]

In these sentences, the compiler must accomplish the easy task of recognizing that "the table" co-refers to the object "a table" refers to, and similarly recognize the repeated references to "blue block" and "white block". In the last sentence the interpreter must decide that "the block" refers to the red one. In this case, since "the block" is a pyramid which cannot support any block, the conclusion is certain. But if the last statement were "the block is large" we could not determine whether the red or blue blocks were intended without further information.

In another example, Yu states the English rule,

It is always the case that the table is larger than anything.

The phrase "It is always the case that" is recognized as one way of introducing a rule with no antecedents and "anything" is recognized as a free variable. The translation to a procedural logic rule is:

[LARGER TABLE X] <

a universally true assertion over all possible values for X. (Note that "<" represents a backward pointing logical implication sign, and that variables to its left are universally quantified.)

Another English rule states,

If a block is larger than another,
the other is smaller than it.

Here, "a block" is interpreted as a free variable, say X; "another" is a free variable, Y; "the other" is coreferential with "another" so it is replaced by Y; and "it" is coreferential with "a block" and so replaced by X; resulting in the translation to a procedural logic rule as follows:

[SMALLER Y X] < [LARGER X Y]

A more general system could attend to the type, "block" and compile<

[SMALLER Y X]
< [BLOCK X][BLOCK Y][LARGER X Y]

What is notable here, is that procedural logic uses variables in a manner not dissimilar to some uses of English pronouns and noun phrases, thus facilitating the translation from English.

Describing robot commands to the blocks world stretches our use of English, requiring some programming jargon. Consider the following definitions:

A block is clear if it is a supporter unless something is on it.

[CLEAR X]
< [SUPPORTER X] [UNLESS(ON W X)]

{Note here, that any variable occurring exclusively to the right of the backward-implication sign is existentially quantified.}

A block is a supporter unless it is a pyramid.

[SUPPORTER X]
< [UNLESS (ISA X PYRAMID)]

A block is clear if something is on it and that something is put on the table.

[CLEAR X] < [ON W X][PUTON W TABLE]

A block is picked up if the block is clear, and if the robotarm is clear, and if it is deleted that the block is on something, and if it is asserted that the block is on the robotarm.

[PICKUP X] < [CLEAR X][CLEAR ROBOTARM]
[DELETE [ON X Y]]
[ASSERT[ON X ROBOTARM]]

A block is put on X if the block is picked up and if X is clear and if it is deleted that the block is on the robotarm and it is asserted that the block is on X.

[PUTON Y X] < [PICKUP Y] [CLEAR X]
[DELETE (ON Y ROBOTARM)]
[ASSERT (ON Y X)]

The concepts "clear", "supporter", and "robotarm" are jargon defined only with respect to the blocks world. The explicit use of multiple "ands" and of the concepts "unless", "deleted", and "added" are required in the subset to simplify the parsing and translation process. The passive form of sentences is chosen to make the descriptions a reasonably acceptable form of natural English.

Yu's experimental NLP system for the blocks world accepts descriptive statements, rules, questions, and commands; in addition it accepts special consistency rules to maintain database integrity. His grammar translates these four types of English statements into procedural logic and runs the resulting programs, providing the user with an English-subset capability for defining, questioning, and commanding a simulated robot in the blocks world.

The grammar for translating these statements into programs is domain independent in the sense that rules concerned with "if-then","unless", "delete", "assert", etc. are equally applicable to describing programs in any domain, not just the blocks world. Similarly, rules for transforming descriptive sentences apply regardless of their content. In contrast, the content of the English descriptions is domain dependent; in describing a tree search algorithm, for example, we might state,

A tree (X.Y) is a root X and branches Y.
A branch (U.V) is a tree U and branches V.
NIL is a branch.
A goal X is solved if X is the root of a tree.
A goal X is solved if X is the root of a branch of a tree.