

FEB 2 1994

**2nd INTERNATIONAL CONFERENCE ON
ARTIFICIAL INTELLIGENCE AND EDUCATION**

University of Exeter, U.K.

September 2nd - 3rd, 1985

Sponsored by the Society for the Study of Artificial Intelligence and Simulation of Behavior (AISB), Symbolics, Inc., GTE Laboratories, Inc., and Intellect, Ltd.

General Chairman

Masoud Yazdani
Department of Computer Science
University of Exeter
Prince of Wales Road
Exeter, EX4 4PT, U.K.
Telephone: (0392) 77911, Ext. 216

Co-Chairman (USA):

Dr. Robert Lawler
Fundamental Research Laboratory
GTE Laboratories, Inc.
40 Sylvan Road
Waltham, MA 02254, USA
Telephone: (617) 466-2698

Administrative Assistants:

Mrs. Marlene Teague
(Exeter University)
Mrs. Marie McPartland-Conn
(GTE Laboratories)

BAA 7508

educ
LB1028.5
I57
1985

FOREWORD

This collection contains extended abstracts of papers submitted by participants at the 2nd AISB Conference on Artificial Intelligence and Education held September 2-3, 1985 at the University of Exeter, U.K. The first conference in this series was also held at Exeter during Easter 1983.¹ The program of invited lectures, submitted papers, and workshops retain the character of the previous conference. However, this conference has a more extensive research orientation and focus. The conference features the primary aspects of interaction including:

AI Programming Environments for Novices
Intelligent Tutoring Systems
Theoretical Foundations of Learning and Teaching
The Teaching of AI as a Subject
Knowledge Acquisition by People and Machines

With an extended research focus, the last three topics will receive more attention than they did at the previous conference.

The submitted papers have been refereed by the program committee, the members of which will also be giving invited talks or chairing sessions:

Dr. B. du Boulay	Sussex University, UK
Dr. A. Bundy	Edinburgh University, UK
Dr. M. Eisenstadt	Open University, UK
Prof. J. Howe	Edinburgh University, UK
Dr. J. Self	Lancaster University, UK
Dr. D. Sleeman	Stanford University, USA
Dr. R. Young	MRC, Cambridge, UK

Approximately 15 papers from American researchers have been accepted for presentation and more than that number from Europe.

No full proceedings of the conference will be published. However, it is expected that presentations at the conference will be subsequently polished as chapters for an edited volume on the theme of Artificial Intelligence and Education.

¹ The volume **New Horizons in Educational Computing**, M. Yazdani, ed., (John Wiley and Sons, Inc.), summarizes papers presented at that conference and other works of attendees.

TABLE OF CONTENTS

Author	Title
J. Barchan	French Grammar Analyzer – pg. 1
J. Bonar	Architectures for Intelligent Tutoring – pg. 6
B. du Boulay	Computers Teaching Programming – pg. 8
J. Briggs et al.	Toolkits for Naive Users – pg. 12
A. Bundy	Teaching AI Programming to Non-Scientists – pg. 16
A. deCallatay	Spontaneous Rule Acquisition from Experience in Learning Robots – pg. 20
K. Carley	Knowledge Acquisition as a Social Phenomenon – pg. 27
S. Cerri and P. Landini	Misconceptions in Multilingual Translations – pg. 30
C. Colbourne et al.	Micro-PROLOG is the Classroom – pg. 35
A. DiSessa	Computers and Experience in Learning Science – pg. 37
G. Drescher	Object LOGO: Object-Oriented Programming Refined – pg. 38
G. Drescher	The Schema Mechanism – Translating Piaget Into LISP – pg. 41
M. Elsom-Cook	Learner-Based User Modelling – pg. 43
W. Feurzig	Algebra Slaves and Agents in a LOGO-Based Mathematic Curriculum – pg. 45
J.R. Frederiksen and B. White	A System for Teaching a Qualitative Understanding of Electrical Circuit Behavior and Troubleshooting – pg. 47
G. Groen	The Epistemics of Computer Based Microworlds – pg. 49
M. Gross	A Laboratory for Exploring Design Constraints – pg. 51
B. Harvey	The Young Computer Scientist: A Curriculum Plan – pg. 53
C. Hoyes et al.	LOGO Programming Activities – pg. 58
N.E. Johnson and J.M. Ogborn	Eliciting and Representing Knowledge for Learner Models A Study in Maths and Science Education – pg. 60
Y. Kareev and J. Avrahami	Teaching Children AI – pg. 64
R. Lawler	Computer Microworlds and Reading – pg. 67
R. Lawler	Development of Structures Through Interaction – pg. 70

B. Lees	Model Building in Computing Education – pg. 75
H. Lieberman	An Example Based Environment for Beginning Programmers – pg. 78
L. Miller	The Role of Cognitive Theory in Educational Reform – pg. 79
S. Ohlson	Cognitive Diagnosis, Pedagogical Methods, and Instructional Actions in a Rational Number Tutor – pg. 80
M. Owen	Searching for New Microworlds – pg. 82
A. Priest	How People Read Physics Problems – pg. 86
R. Samurcay	Is the AI Approach Adequate to Study How People Gain Complex Knowledge? – pg. 89
P.J. Scott and R.I. Nicholson	Expert Systems and CAL Design – pg. 91
John Self	The Application of Machine Learning to Student Modelling – pg. 93
M. Selfridge	Observational Learning By Computer – pg. 95
D. Sleeman et al.	The Inference and Use of Student User Models – pg. 98
C. Solomon	Can We Build a Mathland? – pg. 99
J. Vazquez-Abad and R. Larose	Enriched Learning Environments on a Microcomputer: Towards Feasibility in the Instructional Use of Expert Systems – pg. 10
H. Wertz	bLISP: An Interactive Programming Environment – pg. 103
S. Wood	Expert Systems in Teacher Education – pg. 105
M.R. Woodroffe	FITS: A Framework for an Intelligent Tutoring System – pg. 107

French Grammar Analyzer

Jonathan Barchan
Department of Computer Science
University of Exeter, U.K.

The French Grammar Analyzer program (Barchan, Woodmansee and Yazdani, 1985) began as an attempt at a rational reconstruction of FROG (Imlah and du Boulay, 1984) and much of its inspiration and ideas are owed to their work: however, as we intend to show, we have made significant improvements to the theory behind the previous work, in addition to incorporating some of the best ideas to be found in FROG.

Examining first the similarities between the French Grammar Analyzer (FGA) and Frog, it is most significant that we have chosen to adopt the same teaching strategy and environment as our predecessors, i.e., the typing in of a sentence in free French on the part of the student which is then parsed by the program with appropriate error messages being reported as and where applicable. The advantages in terms of scope and interest of this form of teaching over traditional "drill and practice" are clear, as are the disadvantages in terms of the absence of a clear idea of who best to put to use the freedom which it offers. Like Imlah and du Boulay, we envisage our program being employed in a somewhat more restricted environment than just the entry of arbitrary French sentences: English sentences could be presented for translation, for example, and/or these sentences could incorporate specific points of grammar which the teacher wished to test.

A further area of similarity between the two programs is in the choice of PROLOG as the implementation language. We too consider that this language, or one similar to it, is necessary in designing what is essentially an attempt to construct an expert system in the teaching of the French language. FGA is basically an exercise in pattern matching along the same lines as FROG, but in the sophistication offered by PROLOG differs radically from earlier "dumb" versions such as Last's (1979) and Farrington's (1985).

In FGA we identify four distinct design features which we consider typify the goals and strengths of the program. These are as follows:

1. It has been designed for future work.
2. Robustness is of prime importance in parsing.
3. At present, only a subset grammar and dictionary for French are provided.

4. It aims at the generation of constructive error messages for the purposes of teaching.

We now turn to the examination of each of these aspects and their implications.

Future Work

In order to ease the task of future extensions, we have gone to great pains to write FGA in a highly modular fashion. The data structures used, parsing techniques and error reporting phase are all intended to be clearly intelligible, distinguishable and modifiable. This sort of "open-ended" approach to programming is held to be of particular importance in the field under consideration: because no finite solution can (yet) be seen in the area of natural language parsing, the author of any system must recognize the need for future development and extension of his ideas by someone else in the future and should design his program accordingly. It is our opinion that, although Imlah and du Boulay recognized and drew attention to this requirement, in practice we have succeeded better in realizing it. Of particular note is the attempt to keep the grammar rules of French, the parsing techniques and the error reporting modules separate and clearly distinct from one another, where our predecessors tended to amalgamate them and render them all in an intrinsic and separable part of the whole.

Robustness

Like FROG, FGA anticipates grammatically incorrect input and the need to successfully parse such to its best degree, rather than simply giving up. As already pointed out, there are still big problems in parsing just correct French since it is not possible to specify a complete grammar for the language: there is, therefore, a need to handle grammatically correct but unanticipated structures as well as those which are definitely wrong. When a seemingly erroneous lexeme, word group or structure is encountered there are, effectively, two different explanations: if we take the case of an individual word, it may appear in the dictionary but be incorrect in its context (misspelling, bad ending, incorrect agreement, wrong position/function in the sentence), or it may be potentially quite correct but not present in the subset dictionary. The same holds for structures and the grammar. We are particularly aware of this latter danger when attempting to cope with unknown words, resorting to intelligent guesses dependent on position and spelling to identify their most probable identity and attributes and also informing the user of the assumptions made. To this end a pre-parse section is employed whose task is to make all possible guesses about the role and characteristic of an unidentifiable lexeme, e.g., the work "portes" would be assumed to be:

- a noun, feminine, plural; or
- an adjective, feminine, plural; or
- a verb, second person singular, present indicative (the subjunctive is not catered for in FGA).

The subsequent parse phase will choose among these alternatives as necessary.

The advantage of this method is that it greatly simplifies the parse section since all words can be treated as if they were "known" (in fact they will be flagged as either found or guessed, but the distinction is not relevant here). This means that the parser itself will not have to generate alternative interpretations but only choose among those already offered to it. On the other hand, there is also a disadvantage in that more work than necessary will have been done in the pre-parse phase if only one possibility exists in the parse phase, e.g.,

tu portes le ballon.

There "portes" can only be a verb, and there seems little point in identifying it also as a potential noun and/or verb. A further problem arises from the fact that the guess about the nature of an unidentified lexeme occurs in a totally context-free environment: it is based on spelling alone and is not related to the sentence structure. This means that, for example, if the parser needs an adjective at a certain stage when an unknown word has been found but the pre-parse phase had not thrown up this possibility from the spelling encountered, then the parse will fail rather than generate this solution. Nonetheless, other problems are just as great, if not greater, when the parsing mechanics are designed the other way round.

Subset Grammar and Dictionary

FGA, like FROG and all other attempted natural language parsers, is based on only a subset of the complete French grammar and dictionary. Limitations of time, speed and space, as well as the (current) impossibility of defining a full natural language grammar dictate the need for subsets. In fact, FGA's subset is more restricted than that of FROG. However, it is felt that the latter may have tended to be somewhat over-ambitious in the extent of its attempted coverage of the language and thereby lost the generality, clarity and modularity considered so essential. It is felt that one of the principle virtues of the former is its recognition of a sort of "equation" for natural language parsers which reads something like:

Subset + Modularity → clear and general principles.

Given the constraints within which the work must be done, we think it better to produce solid work on a small subset alone than to attempt too much with a larger cross-section and end up with grammar rules, parsing mechanics and so on permanently intertwined on an "ad hoc" basis. We believe further that it is reasonable to hope that thorough analysis of a subset implies that in principle the ideas (with necessary modifications) should provide a sound platform for tackling progressively larger sets of the whole. For example, it was mentioned above that the pre-parser makes guesses as to the likely identity of an unrecognized word based on its spelling alone: even though only subset of the rules necessary for such predictions is available to FGA, some ninety percent success rate is achieved. This shows the potential for future development.

Error Reporting

The grammar used by FGA's parser is written in a top-down fashion which is well suited to the PROLOG language. Many people – both computer scientists and linguists – may also well

find that grammars written in this form are easier to write, understand and modify than a bottom-up counterpart. An added bonus is the ease with which expected incorrect structures can be anticipated and built into the grammar with an appropriate error message tag. For example, a fair number of students may produce "je les n'aime pas" instead of the correct "je ne les aime pas": an error message to the effect that the pronoun list should be within the negation rather than outside it is incorporated into FGA. Woodmansee writes (Woodmansee, 1985):

"This method has potential for being useful in a rigorous environment such as drill-and-practice tests where common errors can be anticipated and reported."

It may be too much to ask the teacher to predict every error which might occur in the general case, as in Farrington's work, but the means of relatively simple adaptation are available if it is wished to teach a specific point. The result is a system which is capable of trapping the more mundane mistakes on its own while allowing a useful message to be generated for a particular purpose.

Error messages are flagged during the parse but not reported until that phase has terminated. This means that an unsuccessful parse is still able to communicate those mistakes detected before the failure to the error reporter. Furthermore, although the program does not implement a least-cost error analysis, the door is left open for such a future improvement. For instance, if the phrase "les bateau sont sortis..." were encountered it would probably be better to report only the one mistake that "bateau" should be plural than to report three errors of number agreement for "les", "sont" and "sortis" with the singular noun.

Finally, the power of the "dual direction" of PROLOG is utilized to generate correct forms for incorrect ones through the very same rules which state what is and is not correct. The user can then be informed of what such-and-such an incorrect word "should be", hopefully thus providing a most useful teaching lesson.

In summary, it must be restated that many of the best ideas behind FGA were encouraged by those in FROG and that many of the criticisms levelled against the latter should be viewed in this light. We consider FROG to be an excellent introduction to the use of Artificial Intelligence techniques in the sphere of computer-aided language learning, and FGA to have concentrated on and clarified many of the most progressive features of its forerunner with a view towards future development in this area.

Further Development

Drawing on what we regard as the most powerful aspects of these two programs, work is currently being undertaken to try to produce a language-independent computer-aided language teacher. While FROG and FGA can deal only with French, a language-independent program would, it is hoped, be equally at home with any language, given a grammar, dictionary and set of error messages appropriate to that language. However, initial research will be limited

to Western European languages only: French, Italian, Spanish, German and possibly English. These languages, all more or less closely related to Latin, share certain common characteristics which may facilitate the production of a sufficiently general "shell" capable of parsing, and recognizing mistakes in, sentences in that language. It is not claimed that these similarities would hold were an unrelated language, such as Japanese, to be included in the group. Particular interest centers on the two languages of French and German since they both resemble and differ from each other in many respects.

Many difficulties are expected in the attempt to produce such a language-independent system. To take just one example, consider determiner-noun agreements in French and German. In the former language we have to consider both gender and number agreement: but the situation is greatly complicated in the latter since the case (nominative, accusative, dative, genitive) of the noun group must be taken into account as well, which involves information about the function which it is playing in the sentence as a whole.

Further attention will be paid to one area of weakness shared by FROG and FGA: the difficulty of distinguishing grammar from parser from error reporter. A goal will be the specification of the dictionary, grammar and errors in a clear and logical fashion such that they will be easily intelligible to the language-independent shell and at the same time to the human teacher who should be able to understand, write and modify them. The final objective of the project will be to determine the feasibility, or impracticality, of the production of language-independent language teachers as an alternative to the development of individual computer-based tutors for each and every particular language.

Acknowledgements

The author gratefully acknowledges the support given by the British Science and Engineering Council.

References

- Barchan, J., Woodmansee, B. Yazdani, M., "Computer Assisted Instruction Using a French Grammar Analyzer," Research Report No. 126, Department of Computer Science, University of Exeter (1985).
- Farrington, B., "A Micro-Computer Program for Checking Translation at Sentence Level," University of Aberdeen (1985).
- Imlah, W. and du Boulay, B., "Robust Natural Language Parsing in Computer Assisted Language Instruction," Cognitive Studies Programme, University of Sussex (1985).
- Last, R., "On-line Computer-Assisted Modern Language Learning," *Modern Languages*, Vol. 60 (1979).

Computers Teaching Programming

Benedict du Boulay
Cognitive Studies Programme
University of Sussex

Introduction

It seems entirely sensible that we should delegate some of the burden of teaching students how to program to the computer itself. The machine, in some sense, would then become both the object of the student's study as well as the guide to that study.

It turns out, however, to be remarkably hard to build a system that can effectively mimic any of the activities of an insightful teacher: whether it be commenting on the way a student is developing a program, deciding on an effective order for the introduction of new concepts, spotting a possible misconception underlying some syntactic or logical error, deciding whether a given program is a reasonable solution to an exercise or even accurately pinpointing where the errors are and making suggestions about how they might be corrected.

This is hardly surprising. Good teaching borrows from many of our other human abilities and knowledge concerning, say communication, interpersonal relationships and social skills, not to mention any abilities concerned with the topics of instruction. Building a surrogate teacher is thus a problem comparable in magnitude to that of the general Artificial Intelligence enterprise. In the context of Intelligent Computer Aided Instruction these problems surface in the guise of automatically building and maintaining a model of the student under instruction, of determining and executing an effective teaching strategy and of being able to solve the problems that one sets or answer unanticipated questions from the student.

Despite the difficulties, various systems have been built that address different aspects of the above issues. We can divide these roughly into four classes according to the roles they have attempted to play. First, there are systems which have acted as "tutors" in the sense that they have made decisions about how and in what order programming concepts should be introduced and attempted to monitor the student's progress in mastering the new notions. Typical examples here are Cerri et al. (1984) and Elsom-Cook's (1985) systems and to a lesser extent BIP (Barr et al., 1976). Second, there are "coaches" which have stayed more in the background and played a more neutral role as far as putting an ordering on which concepts are learned. Their job is to observe the way that the student tackles a problem and either comment when something seems to be going wrong, provide help when asked or, as in the case of say Miller's (1978)

or Anderson's (1985) system, constrain the student to set about the problem in a particular way. The distinction between a tutor and a coach is sometimes quite blurred in as much as some programs take on both roles, for example Anderson's system.

The third class of system, the "bug-finders", are those that are brought in after the event, as it were, to try to figure out what is wrong (and there is often something wrong) with the student's program and provide advice about what to do. In some ways these may be considered as coaches except that their advice has specifically to be sought by the student and they wait until the program is complete before attempting any analysis. Examples here are the LAURA system (Adam and Laurent, 1980), Goldstein's (1975) MYCROFT and, more recently, PROUST (Johnson and Soloway, 1983).

The final class is more diffuse and consists of those systems that attempt to make the interactions between the student and the programming environment more effective or more understandable. These encompass such aides as structure editors, debuggers, tracers, modelling aids, etc., which have been designed with novices in mind. The BIP system has several features which make it an interesting member of this class. Strictly speaking, one might also include programming languages whose syntax and semantics have simplified and tidied for novice consumption.

The aim of this paper is to survey the strengths and weaknesses of systems that fall into the first three classes in an attempt to understand how to go beyond current capabilities. Less attention will be paid to the fourth category here, important though it is for teaching novices effectively. One class of systems that will not be considered here are frame-based tutors that simply present textual information about programming and then ask questions on that material with little or no capability to examine students' answers which consist of programs as programs.

The remainder of this abstract makes some preliminary remarks about representative systems in each of the categories that have been mentioned.

Tutors

Cerri et al.'s (1984) system TRILL is a tutor for a subset of LISP. It represents its knowledge of LISP in a semantic network and attempts to drill students in various aspects of elementary LISP by generating and then asking questions concerning the behavior of fragments of LISP program. Its interest lies in the way it tries to deal with incorrect answers. The semantic network encodes the various dependencies between different bits of LISP, such as the fact that understanding CDR depends on being able to remove the first s-expression from an s-expression which in its turn depends on being able to identify an s-expression and so on. By generating and then asking supplementary questions the system can determine whether an incorrect answer is due to a simple slip or due to a misunderstanding at a lower level. As the system traverses the network representing an expert's knowledge of LISP, it builds up a picture of the beginner's strengths and weaknesses that is used to guide what kind of question is generated next.

Coaches

Both Gentner's (1979) system, COACH, to teach FLOW, and Miller's (1978) system, SPADE, to teach LOGO, monitored the student as s/he developed a program. Indeed, the former system attempted to understand the significance of every keystroke, including mis-typings. This involved the hard problem of inferring the student's plan from an incomplete and growing program, though the system could make some assumptions because students were supposed to work from a course book. The SPADE system was at pains to inculcate top-down program development methods in the students but one of its problems was that it did not leave the student any flexibility to set about designing the program in any other way. Although a well-organized program ought to be the student's goal, it is not clear how easily a novice can produce this by top down methods.

Anderson's LISP tutor seems to suffer from a similar difficulty in that it embodies a particular view about how a program should be organized and comments immediately that the student starts to depart from the prescribed path. Questions about how much a coach should interfere are still open with Anderson's view being at odds with the more taciturn and self-controlled coaching provided by some other systems.

Bug Detectors

Goldstein's (1975) system, MYCROFT, could detect and repair errors in LOGO programs designed to produce line drawings. It made the program conform to a set of assertions about the geometric properties of the drawing it was supposed to produce. It was able to relate inconsistencies between a program and the assertions via a theory of program planning and of errors in plans and so repair the program. Using related methods, Lukey (1980) has shown how certain errors can be eliminated from Pascal programs by sub-dividing the program into smaller logical chunks. His system, PUDSY, was able to deal with various logical errors which could be detected without reference to a specification for the program. It was also able to derive assertions about a program and match these against a specification, also represented as assertions. By detecting mis-matches and then changing the program to eliminate them, it was able to remove further logical errors.

PROUST is an impressive recent system (Johnson and Soloway, 1983) that can detect a wide variety of logical errors in student's Pascal programs. It makes use of an extensive (human) analysis in many student's answers to two problems set as part of an initial Pascal course. The system is primed with what the teacher thinks the "goals" of a particular programming exercise are (i.e., what the program is intended to do). The system has available a repertoire of "plans" for achieving each of these goals. It systematically attempts to match the various plans onto the code it is presented with. It tries to understand discrepancies by deforming its plans in accordance with bugs that had earlier been detected in the prior empirical studies. Impressive though the system is, it can deal only with a subset of Pascal, with only a limited range of problems and can easily be fooled by pieces of code that are equivalent but not identical to those expected.

References

- Anderson, J.R. and Reiser, B.J., "The LISP Tutor," *BYTE* 10, 4 (1985).
- Adam, A. and Laurent, J., "A System to Debug Student Programs," *Artificial Intelligence* 15, 75 (1980).
- Barr, A., Beard, M., Atkinson, R.C., "The Computer as Tutorial Laboratory: The Stanford BIP Project," *International Journal Man-Machine Studies* 8, 567 (1976).
- Cerri, S.A., Fabrizzi, M. and Marsili, G., "The Rather Intelligent Little Lisper," *AISB Quarterly Newsletter* 50 (1984).
- Elsom-Cook, M.T., "Design Considerations of an Intelligent Tutoring System for Programming Languages," University of Warwick, Department of Psychology (1985).
- Gentner, D.R., "Towards an Intelligent Computer Tutor: Procedures for Instructional Systems Development," Academic Press, New York (1979).
- Goldstein, I.P., "Summary of MYCROFT: A System for Understanding Simple Picture Programs," *Artificial Intelligence* 6, 249 (1975).
- Johnston, W.L. and Soloway, E., "PROUST: Knowledge-Based Program Understanding," Yale University, Department of Computer Science, Technical Report 285 (1983).
- Lukey, F.J., "Understanding and Debugging Programs," *International Journal Man-Machine Studies* 12, 189 (1980).
- Miller, M.L., "A Structured Planning and Debugging Environment for Elementary Programming," *International Journal Man-Machine Studies* 11, 79 (1978).

Toolkits for Naive Users
Jonathan Briggs,* Jacqueline Dean, Jon Nicol
School of Education
University of Exeter

The Exeter History and Computing Project is engaged in bottom-up action research in education, using micro-PROLOG as a medium for improving cognitive skills.

The project is exploring ways of introducing both pupils and teachers to logic programming in an accessible form which involves improving the man-machine interface to remove unnecessary barriers to understanding. The project's work has been tailored to classroom realities, e.g., micros with limited RAM.

One thing which has become clear in the course of this research is that the ideal of "logic across the curriculum" is impossible to achieve in its theoretical purity. The constraints of the school curriculum, the wide range of pupil ability and the lack of sufficient in-service training for teachers ensure this.

The project has evolved a workable compromise in the development of compact educational "toolkit" programs. These programs enable naive users to create successful and quite complex software by providing them with logical, structured building blocks, while introducing them gently to many of the key ideas of logic programming. Initially the toolkits have been developed for the Humanities as this is the project's present area of expertise. The project's toolkits are designed to free teachers from the "stultifying activities which supposedly go under the name of ...software" (Avon History teacher, November, 1984) and enable "...teacher and pupils to create their own educational software as and when seems appropriate, thus freeing them from the commercial program writers who may regard 'education' as simply another marketplace for their wares" (Computer Science Teacher, September, 1984). These teachers were commenting favorably on micro-PROLOG toolkits they had encountered.

Four Exeter Project toolkits (PLACES, UTIL, DETECT, SIMPLAY) are described briefly below, together with THE PLAN, developed by Jonathan Briggs, which has great potential use in education.

PLACES

This program enables teachers and pupils to create their own database of place-names for this local area. It provides a simple, natural English querying system, e.g.,

*Logic Programming Associates, Ltd., London

Describe Crediton

Which places before 1066?

Which places contain ford?

It also allows users to extend a program by entering rules specifying new categories which can be queried.

UTIL

This is a tiny convenience program. It is best used in conjunction with one of the compact specific-purpose front-ends. It allows programmers to:

1. Edit their data. The UTIL editor is the same as the SIMPLE editor.
2. List their program, or part thereof. Sentences in the data-base are numbered for easy recall (for alteration or deletion).
3. Delete any clause of a specified relationship.
4. Kill any relationship.

UTIL also contains an error handler which provides appropriate error messages. It thus supplies many of SIMPLE's facilities, while using a fraction of the memory space taken up by SIMPLE.

DETECT

This program allows naive programmers to write large data-bases for exploration by users. These databases essentially contain information which "detective" users must extract in order to solve a mystery. DETECT provides:

1. Five user query words, viz: (a) tell-about, (b) interview, (c) read, (d) search, and (e) dig.
2. Five relationships (predicates) to cover the five categories of information the programmer might use. These are linked to the five query words. The five matching categories of information are: (a) facts about people or things; (b) what people said; (c) relevant written documents; (d) what a physical search would uncover; and (e) what is contained in an area dug up (this is designed for archaeological excavations).
3. Facilities for attractive display of text.
4. The facility to insert synonyms for any query words, or for any subject in the database.

SIMPLAY (Simulations in Micro-PROLOG for Laymen)

This program provides a blueprint for writing simulations. Each programmer inserts his/her own structure and data.

SIMPLAY supplies:

1. Four categories of information which can be entered, using the command word add: start, link, choice, text. For example, "add choice". As with PLAN, SIMPLAY responds with a template for programmers to copy when entering their data. Text enables programmers to

- insert text of their choice; start allows them to specify a starting point of the simulation; link allows definition of the links between events; choice enables them to define the number and type of choices to be presented to the user, and which results follow these choices.
2. Improved screen display, including the screen being cleared after each user input. This represents an advance for micro-PROLOG, which has been – rightly – criticized for its poor visual display.
 3. Ease of use, of the “Press any key to continue” variety.
 4. An optional back-tracking facility for users of the completed simulation.

THE PLAN

THE PLAN, as its name suggests, enables novices to write their own adventure games. It allows them to create locations, to add exits or “doors” between these locations, to place objects or people in the locations, to specify events, to create actions and to determine the results of these actions.

THE PLAN operates in two separate environments:

1. That of the actual adventure game in which the player moves, i.e., gaming mode.
2. Programming mode, which allows users to create the physical world of their game by describing it to the computer, using the categories: position, exit object, action, event, synonym, with the command word add.

For example,

add object

PLAN responds with a prompt which displays the formula for entering the object, i.e.,

(place object)?

The programmer answers, say,

(room3 pistol)

naming the place and the object as requested.

Users can also insert descriptive text for their locations, exits and objects by using the word describe.

To add actions and events, the programmer uses micro-PROLOG’s internal syntax. THE PLAN does have several built-in actions (move, take, drop, help). However, an adventure using only these action words does not have much flexibility. Therefore, THE PLAN allows the creation of new actions using defined command words and variable. As before, a prompt displaying the correct entry pattern is displayed. The programmer types:

add action

PLAN replies:

((user types) (conditions) (results))?

i.e., first give what the user of the game must type in, then the conditions which must be met in order for the action to take place, then the conditions which must be met in order for the action to take place, then specify the results of such action. The programmer can now enter, say:

```
((killx) (WITH x HAVE y weapon y)
(REMOVE X PRINT ( x lies dead on the floor)))
```

i.e., in order for his command to kill someone, (x) to be carried out, the adventurer must meet these conditions: he must be in the same location as x (WITH X), and he must HAVE in his possession a y, such y having previously been defined as a weapon. If these conditions are met PLAN will produce the result by REMOVE-ing x from the game and PRINTing the final message on the screen, inserting the name of the dead person in place of the x.

Despite its apparent complexity, pupils of all ages master THE PLAN easily. Its well-defined categories of information and the templates it provides for users ensure this.

These toolkits have major implications for educational practice and theory because:

1. They introduce children to logic programming, which is essentially the art of clear description and straight thinking. These are highly desirable skills across the curriculum and for life. These skills are being acquired in a palatable and motivating environment.
2. They allow pupils to control their own learning.
3. They enable pupils to encounter and program computers successfully without having to master complex code.
4. They make learning in specific subject areas more effective, as pupils must master and structure their subject matter in order to turn it into a viable computer program.

Teaching AI Programming to Non-Scientists

Alan Bundy

Department of Artificial Intelligence
University of Edinburgh

Most AI courses at universities are given either a postgraduate level or as a final year option in a computer science honors program. But one of the consequences of having a department of artificial intelligence is the requirement and the opportunity to teach AI to students from a much wider range of abilities and backgrounds. It is good to have this opportunity. AI is going to pervade all our lives via experts systems in the home, workplace, surgery, etc., ICAL in the school and robotics in the factory. It is important to raise public awareness of the scope and limitations of this technology so that people can control it and take advantage of the opportunities it offers them to enrich their lives.

However, teaching AI to students who lack a strong mathematical/science background raise special problems. In order to bridge the gap between theory and practice in AI it is necessary to illustrate abstract algorithms with computer programs. Teaching programming to non-science students is the major one of these problems. The computational techniques we want to illustrate are not trivial, but non-scientists not specializing in AI usually have neither the ability to read or write complex programs nor the time to learn this ability. To bridge the gap we must use a programming language with high-level, AI-oriented primitives in which AI programs with non-trivial behavior can be implemented in a straight-forward and transparent way. That is, we want to be able to build parsers, inference engines, line finders, etc., which are easy for novices to understand.

We have decided to adopt PROLOG as our main teaching language because it contains high-level primitives for: pattern matching, representing rules and facts, and search. It has proved possible to illustrate our lectures with one- or two-page PROLOG programs for: the missionaries and cannibals problem, geometric analogy problems, RTN parsing, English dialog, etc. These then serve as the basis for student practical work and projects. For instance, the program for geometric analogy is based on half of Evan's LISP program [Evans 1964]. This was sufficiently interesting to earn its author a Ph.D. in 1964. It was so large that the two halves had to be separately loaded into the MIT PDP6 and was one of the largest LISP programs of its time. The PROLOG version is under two pages of code, including extensive comments, and there is almost a one to one correspondence between the PROLOG clauses and the abstract algorithm described in the lecture notes.

PROLOG AI programs may have a future beyond university AI courses, e.g., in schools. AI programs can be a lot of fun, e.g., Eliza can be a vehicle for all sorts of amusing dialogue. They can be a lot more interesting than some of the arithmetic examples currently used to teach Basic in schools. They can also be used in a wide variety of areas, c.f. the use of logic databases to illustrate a number of different parts of the school curriculum [Ennals 1982]. Can they also be made simple enough that schoolchildren could learn to write them?

The disadvantage of PROLOG is that it does not provide all the primitives one needs, for instance, our PROLOG Eliza program occupies less than a page of code, most of which is transparent. However, it is necessary to augment the PROLOG unification with a string pattern matcher, and the procedure for this, while short, is relatively opaque.

The Sussex University Cognitive Studies Programme have had a similar problem, but have adopted a different solution. Starting with the Pop programming language, they have added a variety of AI packages, e.g., for inference, frames, parsing, linefinding, etc., together with extensive online help packages and a rich development environment. Most recently they have added additional languages, PROLOG and LISP, to make the Poplog system [Sloman et al., 1983]. Using these packages as primitives, Sussex students have been able to build small, but powerful, AI programs.

The disadvantage of the Poplog solution is that the array of packages becomes bewildering. For instance, the notion of variable in the Pop language and the PROLOG language is subtly different and can lead to confusion, e.g., try to assign $X+1$ to X in PROLOG. It is not just the range of packages but the different underlying virtual machines that cause the confusion.

Programming students must learn to predict the behavior of programs, so that they can debug them and develop confidence in their programming ability. This involves understanding the underlying virtual machine of the programming language and being able to simulate it on a program. Students with a strong mathematical/science background can often induce the virtual machine from the worked examples given by the teacher. Non-science students find this more difficult. For them it is better (a) if the programming language has a simple virtual machine, (b) if this is taught to them explicitly.

For its power, PROLOG has a simple virtual machine, but this is not always taught explicitly. An examination of [Clocksin and Mellish, 1981] and [Kowalski, 1979] revealed six different virtual machines [Bundy, 1984]. In the first edition of [Clocksin and Mellish, 1981] different virtual machines are used in an ad hoc way to explain different aspects of PROLOG, so that a student can fail to see any overall pattern in the behavior of the interpreter. These apparently different virtual machines are either only superficially different accounts of the same machine or explain different parts of it. Furthermore, the terminology used in most PROLOG error messages, program traces, etc., does not correspond to the virtual machines described in the textbooks.

1. The list of PROLOG clauses.
2. The goals and subgoals set up during the running of a program and how these are related.
3. The relationship between the goals and clauses, i.e., how goals are satisfied by matching against clause heads and how the clause bodies are introduced as new subgoals.
4. The substitutions generated by this matching, especially the answer substitution.

If all this information is displayed for a reasonably complex program then the information is overwhelming. It is necessary to be able to zoom and focus. At one time one might want to concentrate on the goal structure, e.g., an AND/OR tree, without displaying the PROLOG clauses or the substitutions. At another time one might want to see just the clauses. At yet another time one might want to see all the information listed above, but only in relation to a particular part of the computation. However, the introduction of the different parts of the virtual machine and the zooming and focussing must make clear that what is displayed in all part of a larger and coherent story [Bundy, 1984].

Some critics might argue that it is not necessary to describe the virtual machine; that PROLOG programs can and should be understood declaratively; and that a procedural account of PROLOG is both superfluous and dangerous. This might be true of some utopian logical programming language, but it is not true of current PROLOGs. It is necessary for PROLOG programmers to understand the virtual machine in order that they can use non-logical features like cut, assert, write, etc., and so that they can order their clauses and their literals to avoid problems and make their programs efficient. Study of PROLOG programmers suggest that they develop their programs procedurally and use the declarative reading as a check, rather than develop them declaratively in the first place.

The high level primitives of PROLOG enable some interesting AI programs to be succinctly implemented. It has great potential as a teaching language for AI, and more widely as a programming language for non-scientists. Care is required to explain its workings with a coherent and simple virtual machine. The teaching materials (e.g., textbooks, lectures, etc.) and the PROLOG implementation (e.g., error messages, tracer, etc.) should be based on the same virtual machine. Ad hoc extensions to PROLOG which make a simple virtual machine impossible should be resisted. However, PROLOG can be improved by the accretion of Poplog-like packages provided they were, in turn, implemented as simple and transparent PROLOG programs which the students can understand and modify as well as use as primitives.

Acknowledgements

I am grateful to Paul Brna and Helen Pain for comments on this abstract. The research was supported by ESRC grant H00230012.

References

- Bundy, A., "What Stories Should We Tell PROLOG Students?" Working Paper 156, Dept. of Artificial Intelligence, Edinburgh (1984).
- Clocksin, W.F. and Mellish, C.S., "Programming in PROLOG," Springer Verlag (1981).
- Ennals, J.R., "Beginning Micro-PROLOG," Ellis Horwood, Chichester (1982).
- Evans, T.G., "A Heuristic Program to Solve Geometric Analogy Problems," *J.S.C.C.*, April 1964.
- Kowalski, R., "Artificial Intelligence Series: Logic for Problem Solving," North Holland (1979).
- Sloman, A., Hardy, S., and Gibson, J. "POPLOG: A Multilanguage Program Development Environment," *Information Technology: Research and Development 2* (1983).

Spontaneous Rule Acquisition from Experience in Learning Robots

Armand M. de Callatay
IBM International Education Center
Ch. de Bruxelles, 135, La Hulpe, B-1310, Belgium

Summary

Human learning cannot be fully understood without understanding animal learning. The following is a study of skill acquisition in mammals. The results are extended to the sensory-motor scheme of Piaget which is a learning model for children and also for animals. Skill acquisition is simulated by a robot operating with AI techniques (knowledge network, learning of rules, and problem resolution). The main findings are: knowledge is represented as events: new events must be understood, i.e., coded in rules, before being stored; events are added to the memory and do not modify previous information; events are retrieved as documents in information retrieval (IR). To learn, a child applies a known sequence of actions to new contexts. The generated events are stored (assimilation). Efficient sequences of motor commands are induced from these events (accommodation). It is concluded that, to optimize learning, the student profile of an intelligent computer assisted instruction (ICAI) package must have a comprehensive history.

Natural Intelligence Processes

Skill acquisition in animals is viewed as a data acquisition system followed by other learning methods [e.g., in Michalski et al. 1984]. To simulate this animal learning, a robot controller has been programmed to generate spontaneous moves and to build a knowledge database from these experiences. The simulated robot has a human-like arm and learns how to reach a target [Callatay 1983]. The programs of the robot's controller are in the hardware of the proposed single instruction multiple data (SIMD) computer. This composite controller system has a central component able to execute most AI programs [Callatay 1984]. The learning method is described when explaining the machine. The features of our brain model are the following:

1. **Symbolic Knowledge:** Knowledge is represented by a network of nodes called physical symbols (Newell 1980). The nodes are the spiny cells of the brain cortex. The nodes are connected by directed links called *pointers* in artificial intelligence (AI). In *connectionist architectures*, the pointers are actual wires with potential connections to other cells. In the brain, a cell has one output wire which has potential contacts with up to 100,000 cells.

The contacts are made through small dendritic appendages called *spines*. The contact closure could be made by spine resorption [Callatay 1985]. A node representing a concept is the attachment point of the relations representing its properties. This architecture is compatible with the laws of thought taught in philosophy. The symbolic representation corresponds to the mental world described by Plato.

The connections store relations that are related to events which are true if based on facts or false if based on contradicted internal events. The basis of logic programming corresponds to the laws of thought described by Aristotle and Leibnitz.

2. **Confidence value:** A value (a weight) is attached to each connection and represents the confidence that such a relation is true. The value is computed by the frequency of successful use.
3. **Brain state:** The state of the brain is the state of the symbols which are activated or *instantiated* at a particular point in time. Each activated cell transmits a *logical impulse* (a fast train of neural impulses). Only one cell per neuron group may be active. The processing is rhythmic [Callatay 1969]. A symbol stays active during one or many periods.
4. **Production rule:** Each cell works as an association device storing a *production rule* [Newell and Simon 1972]. A production rule is written $C1 \& \dots \& Cn \rightarrow A$. When all *conditions* 'C' are activated, the *action* 'A' is executed. More specifically, each group of neurons works as an IR system retrieving the best document (the action) answering the query composed with the keywords (the conditions) carried by the activated relations. IR algorithms use inverted files with von Neumann architectures or *Recording Content Addressable Memories* (RCAM) with suggested parallel architecture [Callatay 1981]. The rules coding the events are the content of the *long term memory* (LTM). The activated cell represents the *short term memory* (STM). Thus the content of STM is already built in LTM. The associations between cells simultaneously active in STM may be added to LTM. Temporal strings of symbols can be parsed (replaced by associations and structures) because the past states are simultaneously active in LTM. The brain has 10 billions of neurons to store one trillion of relations. It learns by adding 300 new relations per second on average.
5. **Database machine:** The associative function may be simulated on a microprocessor able to answer a query. This processor contains also a permanent memory concerning a knowledge domain. A system containing such connected processors is called a *database machine*.
6. **Query processors:** Groups of cells are linked by cables and form a *Directed Network Processor System* (DNPS) [Callatay 1984]. Processor systems with a Single Instruction (IR query in RCAM) and parallel processing of Multiple Data are called SIMD. Such networks can process most AI programs: produce a response with a *finite state machine*, parse a string of temporal data (sentences) to instantiate the corresponding semantic tree

in the knowledge network, answer queries like an *expert system* based on a production rule algorithm, or solve programs like a *Prolog interpreter* with breadth-first search [Callatay 1984]. The programs are made with component cabling, and the knowledge stored in the RCAM is learned by experience.

7. **Learning robot:** The brain is simulated by a robot and not by an isolated computer. The robot learns the laws of the external world by storing rules describing its experiences. The external world includes the robot and its brain.
8. **Interface:** The analog/symbolic interface (the transcendental relation in philosophy) is provided by the following input/output mechanisms: the sensory data from *feature extractors* are periodically sampled and categorized, i.e., transformed in symbols. The output symbols are set-points of *servo-mechanisms* controlling continuous muscle contractions. The processing is done in long cycles (130ms).
9. **State to state functions:** Each state is represented by a symbol and each event is represented by a production rule. For example: when a spontaneous action 'A', initiated by a muscle contraction during a period of fixed duration, occurs with the arm in position 'S', a resulting position 'I' is reached after a fixed time period. This event is recorded in memory as: *the present arm position 'S' and the intended arm position 'I' produce a motor command 'A' to reach the target in that context*. The rule *Sensation and Intention produce Action* is abbreviated as $S \& I \rightarrow A$. The external world must be almost repetitive to be understandable. The rules are generated by random trials. A simulation has shown that a two-dimensional human-like robot arm can learn all movements in one day with a precision of one centimeter. The processor, which has no ALU, stores and processes these rules in 30,000 cells (cerebrum cortex function). The simulated apparatus includes an additional controller stopping the arm after it has been thrown by an initial muscular contraction. This correction controller is located in the cerebellum.
10. **Intention set-up:** During training, the intentions are initially taken randomly, then some sequences of intentions are selected, then commands from higher processors provoke local intentions to achieve an intended complex result. A high intention 'I' and a high context 'S' produce a high command 'A' that sets the low-level intentions. The low-level execution generates results whose association should produce a high result corresponding to 'I'. To avoid combinatory explosion, equivalent results are put in one category (chunk). Some simulations in specific cases have shown that well-designed algorithms can ensure a useful categorization [Callatay 1983].
11. **Hierarchical control:** Like in a corporation, messages circulate in the bidirectional hierarchy of the brain. Commands coming from top offices are analyzed in each department to find the related local instructions. Eventually external actions are executed at the base. Then result reports are summarized at each level. The top result corresponds to the intention if all intermediate decisions were based on correct rules. These rules may have

been learned in different contexts. These operations correspond to a problem resolution in PROLOG [Callatay 1984]. Logic theory shows that the resolution method gives *true* results, i.e., generates possible events, because the inferences are based on axioms which are *true* experimental facts. The process of top-down and bottom-up message transmission is repeated up to four times per second in the brain.

12. **Intuition and verification:** The knowledge recorded from the above learning is made of the relations experienced during an episode. The events must be understood, i.e., coded as rules, in order to be added in storage [Schank 1985]. *Knowledge is not made of simple rules, but of complex structures which are globally true*, as stated by Quine. The brain knowledge network is a group of independent episodes from which general rules can be discovered by *induction*, but these induced rules may become false after a new —. Each planned intention is associated with the present situation and compared with previous similar episodes. When this parallel processing shows that similar actions are suggested by all evoked episodes and that no threatening episode is evoked, this rule is intuitively induced. Some cases, like disjunctions or incomplete knowledge, are ignored by the above induction method. Therefore the intuitive search may be erroneous. Scientists verify their intuitions by a dialectical process where every step of the deduction is analyzed.
13. **Sensory-motor scheme of Piaget:** The natural tendency of children is to repeat a frequently rewarding sequence of actions which are thus applied to many situations. Then the particularities of these new events are recorded (*assimilation*). When specific events are recognized later on, a better reaction may be taken (*accommodation*). This process seems to correspond to the sensory-motor scheme of Piaget (1947).
14. **Additive database:** Learning is done by recording rules (events) which are added to the network, but which do not modify any knowledge. The irreversibility of the contact closure is required to ensure logical consistency and parallelism [Callatay 1981]. Thus a *historical database* of facts is created. Redundant information may be stored and processed in parallel in this *additive database*. Only the memory part used by the symbolic processing is added. The choice of intentions is adaptively learned. The intentions that had enhanced pleasure or reduced pain are more likely to be repeated.
15. **Memory of episodes:** Events which are repeated are not stored again, if the possible addition of redundant memories is not called a new storage. At each cycle lasting from one second to a few minutes, a top cell is activated and linked with the active events. This mechanism forms a *memory of episode*, and is required to recognize that an event has been repeated.
16. **Recognition:** The most similar episode is recognized at once by the IR organization. Melodies and sentences are transformed into instantiated structures before being recognized. Only the discrepancies relative to a known event are recorded. A top control *enables* (first level of activation) the areas where the previous episode was learned, so that a minimum of relations storing the event can be added there to the knowledge network.

17. **Habituation:** The threshold of instantiated cells is permanently increased, except if the event is interesting (rewarding or threatening). Perception of boring sensations or decision to repeat recognized boring events are discarded because the threshold of the cells representing these symbols is high. This habituation mechanism described by Pavlov provokes an innate tendency to search for novelty, prevents the search of unsuccessful hypothesis in problem resolution, and does not destroy logical consistency of knowledge.
18. **Attention focus:** The symbols used in the rules are selected features of the overall context. These symbols are in the attention focus. This focus is normally well selected by innate mechanisms for sensory-motor mechanisms. Selecting a correct focus for more intellectual operations is done by problem solving.
19. **Symbol selection:** The rule $S \& I \rightarrow A$ may be useless in the following situations:
 - (a) 'S' may refer to a situation description that does not influence the result of the experiment.
 - (b) 'I' is irrelevant when it concerns an impossible result in the present context.
 - (c) 'A' may once have succeeded by chance, although it does not change the result.

Adaptive learning, independent of the symbolic processing, changes the weights used for selection of the physical symbols 'I', 'S', and 'A' that are part of efficient rules. The weights are learned by trial and assessment [Callatay 1985a].
20. **Effort/arousal choice:** Some behaviors need a persistent effort to be achieved, but when an interesting event occurs, this may interrupt the present behavior and change the focus and the intention. This choice seems to depend on the chemical balance of dopamine and acetylcholine and is learned by assessing each event [Pribam and McGuinness 1975].
21. **Piaget's stages:** According to our model, each brain unit has the same processing functions, but some components are the meta-level of others because their inputs/outputs are connected to these processors and not to the sensors/actuators. A unit in a database machine cannot abstract the information it contains. Abstraction is made by other hardware elements located in a meta-level. The top levels of a child's brain hierarchy are not mature, so he is unable to reason with abstract data. The addition of each meta-level improves the capability to abstract problems and to make complex representations. The maturation of each level may correspond to the levels and stages of development described by Piaget (1975).
22. **Alternate hierarchy:** There is more than one hierarchy. A direct hierarchy controls the coordinate execution of concurrent movements. Meta-rules or recommendations in expert systems are rules that choose which type of execution rule should apply. This control is done in a meta-level of the program. In the database machine, a hardware meta-level can change the intentions decided by the direct hierarchy. This method may be used to alter

the response of the sensory-motor direct hierarchy to take account of visual or auditive perceptions [Callatay 1985a].

23. **Common sense:** Most mistakes contrary to common sense are produced by undue abstraction. That man has better common sense than the computer because he does not reason spontaneously with abstract rules [Callatay 1985a].

Learning in Education

From the above learning theory, one can retrieve the efficient education techniques already used by teachers. These methods should also be included in ICAI packages:

1. **Showing examples of gradual complexity.** Thus children can immediately translate them into production rules in order to memorize these events. This assimilation rule has been described in detail by Herbart.
2. **Teaching efficient strategies** for experimentation; correcting wrong ones; presenting problems that may be badly resolved (the error shows that the strategy must be amended). This Socratic method was used by Plato.
3. **Giving problems** to check how important properties are understood. Concepts provide methods to internally create a use (meaning) from clues; therefore badly understood concepts are equivalent to erroneous strategies; these exercises permit learning of the categories represented by words and expressions.
4. **Preventing habituation** by avoiding repetition of events that the student already knows.

These methods require that teachers or ICAI packages remember the Y-examples, concepts, and strategies that the student has already learned. A *student profile* seems required for intelligent tutoring. It should include historical records for examples presented, concept properties already understood, and strategy variations correctly performed.

Acknowledgements: The author thanks G. Darquenne, M. King, M. Sintzoff, C. Trullemans, C. Veraart, and M. Watzenboeck. The opinions developed here are those of the author and not necessarily those of IBM.

References

- Callatay, A. de, "Brain Model with Periodic Processing," *Curr. Mod. Biol.* 2, 307 (1969).
- Callatay, A. de, "The Additive Database," *IBM IEC Technical Report*, IEC-0014. La Hulpe, Belgium (May 1981).
- Callatay, A. de, "the 'Result as Goal' Learning Method: Computer Simulations of Self-learning Robots," *IBM IEC*, La Hulpe, Belgium (Jan. 1983).
- Callatay, A. de, "A Logic Program Interpreter in a Distributed Network of Processors," *IBM IEC Research Report*, IEC-0042. La Hulpe, Belgium (March 1984).

Callatay, A. de, "A Neural Architecture for Symbolic Processing," *COGNITIVA Proceedings*, CESTA, Paris, 605 (1985).

Callatay, A. de, *Artificial and Natural Intelligence: Processor Systems Compared with the Human Brain*. to be published (1985a).

Michalski, R.S., Carbonell, J.G., and Mitchell, T.M., *Machine Learning: An Artificial Intelligence Approach*, Springer Verlag, Berlin, 25 (1983).

Newell, A., "Physical Symbols Systems." *Cognitive Science* 4, 135 (1980).

Newell, A., and Simon, H.A., *Human Problem Solving*, Prentice Hall, Englewood Cliffs, New Jersey (1972).

Piaget, J., *La Psychologie de l'Intelligence*, A. Colin, Paris (1947).

Piaget, J., *L'Equilibration des Structures Cognitives: Probleme Centrale du Developpement*, Etudes d'Epistemologie Genetique, XXIII. Presses Universitaires de France, Paris (1975).

Pribam, K.H., and McGuiness, D., "Arousal, Activation, and Effort in the Control of Attention," *Psychological Review* 82, No. 2, 116 (1975).

Schank, R., "Looking at Learning." In *Progress in Artificial Intelligence*, L. Steels and J.A. Campbell (eds.), Ellis Horwood, London (1985).

Knowledge Acquisition as a Social Phenomena

Kathleen Carley
Department of Social Science
Carnegie-Mellon University

The acquisition of knowledge, the development of the individual's knowledge bases, does not occur in a vacuum. The social context, both social structure (the perceived regularities in the social network) and social knowledge appear to be related to the individual's acquisition of knowledge. In this paper, the relationship between the social world and individual knowledge acquisition is explored. A general theory is postulated and then formulated as a model. The basic theory is predicated on the assumption that social interaction is the driving force behind knowledge acquisition.

The individual's cognitive structure, specifically his knowledge bases, are being continuously constructed by the individual as he moves through the series of tasks that constitute daily experience. Some of the information that he acquires will be new to the society, garnered through his own personal experience and not previously known to another individual in the society. Such discoveries are relatively rare. Most information is already known by someone in the society and the individual learns this information by interaction with someone who already possesses this knowledge. Further, what knowledge the individual acquires is a function of the knowledge he currently possesses. Hence, what the individual learns is intimately constrained by whom he communicates with and by his position in the social network. In the short run, cognitive development, the acquisition of new knowledge by the individual, becomes primarily a by-product of human interaction. This same interaction recursively defines the social structure.

Social knowledge is generally thought of as *that information which everyone knows*. I suggest that social knowledge does not exist independent of the individual in the society; rather, its existence requires tacit consensus over perceived regularities. The development of social knowledge, social learning, involves the evolution or development of this tacit consensus. Social learning occurs as individuals' cognitive structures co-evolve. Social knowledge is known by 'everyone' in the society simply because it is precisely that information that exists somewhere in 'everyones' set of knowledge bases. Thus, social knowledge, as shared cognitive structure, is also a by-product of human interaction. Social knowledge is a dynamic entity which is continuously constructed as members enter and leave society. The social validity of a particular piece of information increases as more individuals add it to one of their respective knowledge bases.

To increase the precision of the foregoing model, a knowledge representation scheme is proposed. In essence, the individual's cognitive structure is viewed as being composed of, at the articulable level, a set of interconnected knowledge bases. Each knowledge base is composed of a set of pieces of information. Based on the foregoing model of the world, and the proposed knowledge representation scheme, a precise, albeit simple, model of knowledge acquisition at the individual level as a function of social interaction is proposed. The inter-relationship between individual knowledge and social knowledge is also modelled. In the proposed model, social knowledge is modelled as the intersection of the individuals' knowledge bases. Social knowledge becomes that set of pieces of information which everyone shares.

Based on the proposed models, a set of hypothesis about the nature of individual and social knowledge are proposed and then tested. The data used is non-experimental and is drawn from a larger study of the relationship between social structure and cognitive structure. The subjects were a group of 45 undergraduates at MIT, all members of the same living group — Third East — who were in the process of selecting a new graduate resident, or tutor, for their living group. The data base includes interaction data, free-form interview data about tutors, voting behavior, and so on. The interaction data was analyzed using network tools to extract the underlying social structure (e.g., the block modelling program CONCOR used). The students were interviewed several times throughout the tutor selection process as to their perception of the concept tutor, as well as on other facets of the selection task. These interviews were coded as knowledge bases, one base per person per time period. A set of procedures that allow the researcher to relate social structure to cognitive structure were used to analyze this data. The procedures, collectively referred to as *Frame Technology*, are an attempt at broaching the gap between network tools (e.g., block modelling and cluster analysis) used to look at social groups and the formalisms in artificial intelligence for representing cognitive structure (e.g., frames and transition networks). Although the tools are somewhat novel, the focus here is not on the tools, but on results observed when one looks at the relationship between the individuals' interaction patterns and the changes that occur in the individuals' knowledge bases over time. In particular, I am looking at the way in which the students' perception of the concept tutor changes throughout the period of time during which the students are selecting a new tutor for their living group as a function of their social position.

A selection of results pertaining to social learning will be presented. These include, but are not limited to the following:

1. Social structure affects knowledge acquisition; e.g., individuals in tightly knit social groups develop similar cognitive structures but not necessarily similar evaluations of those structures.
2. Social knowledge evolves slowly and relative to shared experience.
3. Individual knowledge acquisition is limited; e.g., individuals and groups talk past each other because of the lack of critical paths in their respective knowledge bases.

A final point, although these findings may not be completely unique, the procedures used permit a quantified and systematic study of such issues. Thus, we can begin to empirically gauge how tight the group must be to inhibit the members from learning outside knowledge, how fast social knowledge evolves, and how much information is needed for there not to be miscommunication, and so on.

Misconceptions in Multilingual Translations

Stefano A. Cerri* and Paola Landini
Artificial Intelligence Laboratory
Department of Informatics
University of Pisa
Corso, Italia, 40
I-56100 PISA

Introduction

The analysis, diagnosis and remedial misconceptions in translating "difficult" words¹ in context into or from a foreign language have been reported in previous papers [Cerri and Breuker, 1980; Cerri and Merger, 1983].

The research described here underlies the design and implementation of an Intelligent Tutoring System – called ALICE² – which behaves in a substantially improved way with respect to its predecessor ELISA.

The improvements can be ascribed to the following achievements:

1. The identification of a limited and powerful set of fundamental criteria for the representation into a single, frame-based, multilingual conceptual kernel of correspondences among words and the contexts which they may be inserted in. This representation is used by the system in order to know the conceptual meaning of words as denoted by the contexts in which they can appear, and consequently to infer a partial classification of translations.

While the representation of ELISA's linguistic competence in a semantic network did not account for many linguistic aspects relevant for a correct interpretation of real student's behavior in the task of translation – such as: interference, priorities, etc.; cfr. the suggestions in [Cerri and Merger, 1983] – ALICE's frames embody a complete description of linguistic features allowing a complete diagnosis of students' errors.

Moreover, ALICE's representational framework has been designed explicitly for an easy extension to other languages. In order to add a new language ALICE's framework requires one to relate simply the new word's meanings to the existing unique conceptual kernel.

*On leave at the Artificial Intelligence Laboratory, Vrije Universiteit, Pleinlaan 2, 1050 Brussels

¹"words" may denote composite linguistic units, such as the conjunction *by the time*.

²ALICE is an acronym for "A Little Intelligent Conjunction Expert".

The representation of knowledge in ALICE is described in another paper [Cerri and Landini, 1985].

2. The diagnostic strategies have been substantially improved according to student models embodying a representation of misconceptions which leads to an adequate remedial by a dialogue focussing on the student's problems. The principles, methods and techniques underlying these diagnostic dialogues are briefly described in this paper.

The ALICE system is implemented on a DEC VAX 780, in FRANZ LISP and MRS [Genesereth et al., 1984].

From a technological viewpoint, ALICE's interest relies on the following considerations:

- a. A complete representation of knowledge regarding the discriminative use of Italian-French conjunctions is available [Merger and Cerri, 1983].
- b. A complete representation of knowledge concerning the Italian-French-English temporal conjunctions has been produced [Landini, 1985].
- c. It is generative in the sense that it provides a ICAI program for each ordered pair of languages chosen within the available ones.
- d. While ELISA's use was severely limited by its costs and performance, ALICE is viable for a concrete educational setting.

Finally, ALICE is the outcome of a rather unique effort in the construction of ITS in foreign language learning providing diagnostic dialogues. Other reported experiences apply AI techniques mainly to the generation and parsing of natural language texts, without dealing with student models [Bates et al., 1981; Weischedel et al., 1978].

Mistake, Suggested Cause and Misconception

When the student makes a mistake, there is a hypothesis about its possible cause. The suggested cause of a mistake does not coincide with the explicit representation of the student's conviction, which is usually referred to in the literature as misconception.

For instance, if the student uses *as soon as* instead of *as long as*, we identify the cause in the similarity between the two conjunctions. Such a mistake and the suggested cause select a diagnostic dialogue of a specified type with the aim of acquiring a description of the student's behavior regarding the use of each of the conjunctions under consideration in any usable conceptual meaning (in general: regarding also conjunctions and concepts which may eventually be confused with the ones identified by the original mistake).

Once the diagnostic interaction has allowed to gain sufficient information to interpret the student's behavior as a plausible misconception, ALICE explains to the student its conclusions by means of a remedial message. Since the full history of the dialogue is available to the system, some deductions may relate to interactions preceding the one which caused the current mistake.

For instance, the interpretation of the diagnostic dialogue for the referred mistake may result in:

Inverted use of the conjunctions (with respect to their meaning).

Absorption (i.e., using just one of them).

Total confusion (i.e., indiscriminate use of both of them).

For instance, the student may discriminate between the two concepts by using both conjunctions in one meaning and only one of them in the other meaning (**partial confusion**).

In case of unplausible patterns, the diagnostic dialogue continues under the postulate that the underlying misconception to be inferred includes the one discovered but is a plausible one (e.g., partial confusion as an incomplete manifestation of total confusion).

If the search results in a failure, the remedial message will not give an interpretation of the student model, but will summarize it and give a hint about the cause identified at the beginning of the diagnostic.

Translation as a Composite Function

Let us call **source language** and **target language** respectively the languages from which and into which we translate. These may correspond either to the mother and foreign language or vice versa.

The task of translating a conjunction in a context can be split into two sequential processes.

The first process consists in **identifying the conceptual meaning** to be attributed to the source language conjunction within the specific context. The second process consists in **choosing a conjunction of the target language** suitable for denoting the conceptual meaning identified.

A translation mistake may arise from an error in the application of the function modelling the first process, or the second one or the composition of both.

For instance, consider the following English sentences:

- a. Tell me why you are so happy. (interrogation)
- b. I'm late because I missed the bus. (reason)

In Italian, the conceptual meaning of the two conjunctions are represented by the single *perchè*. Therefore, if the Italian student makes a mistake in translating *perchè*, the cause is the impossibility of discrimination of the two conceptual meanings on the basis of Italian conjunctions.

Let us present now two examples of misconceptions involving the first and the second process, respectively.

Assume that the student translates *perchè* in an interrogative context by *because*. The cause of the mistake suggests to inquire if the student knows the word *why* by asking him/her to translate other interrogative clauses, among which direct interrogative clauses.³

If the student translates *perchè* in indirect clauses only with *because* and in direct interrogative clauses only with *why*, we deduce that since he knows the word *why* he/she did not recognize the indirect interrogative and interpreted it as a clause of reason. This is an example of incorrect identification of the conceptual meaning, i.e., an error regarding the first process.

If instead he/she translates *perchè* always with *because* in both interrogative conceptual meanings, we deduce that an absorption of *why* by *because* occurred. As we are reasonably sure that the student recognizes direct interrogatives, we interpret his/her behavior as due to the association of the conjunction *because* to all interrogative clauses. This is an example of misconception regarding the second process.

Simple and Complex Errors

It will not be possible to enter here in a detailed description of the errors and misconceptions identified. The interested reader may refer to [Landini, 1985]. However, we will shortly describe which principles the classification is based on and will give some examples.

Depending on the cause of errors, we split them into two groups: simple errors and complex errors.

An error is called simple if its cause is sufficient to identify a misconception, so that it is not necessary to enter the diagnostic phase in order to remediate the error.

For instance, let us suppose that an Italian student translates the temporal conjunction *ogni volta che* by *every time that*. Since the Italian conjunction is composed of more words (cause), the student translates each one separately and then constructs an expression which he/she believes to be a possible translation (misconception). Notice that the same type of error may be made by an English student when translating *every time* by *ogni volta* instead of *ogni volta che*.

An error is called complex if the cause can support more than one misconception, so that it is necessary to enter the diagnostic phase in order to find out which one interprets the student's behavior. The two examples of errors shown in the previous sections are complex errors.

Let's give another example which is related to the linguistic notion of interference. The Italian conjunction *come* is not always translated by the French conjunction *comme* and vice versa.

³Notice that in direct interrogative clauses, *why* and *perchè* are modifiers of the principal clause, not conjunctions. ELISA's representation of conceptual meanings would not have discriminated between direct and indirect interrogative clauses, because it considered only conjunctions and also because both concepts are denoted by *perchè* and *why*. ALICE's representation included this discrimination as it is relevant for explaining many errors typical of Italian students of English and therefore for guiding an adequate diagnostic dialogue to a sensible remedial.

Since the two conjunctions sound similar,⁴ mistakes can arise in both directions. Depending on whether the student consistently thinks *come* and *comme* to correspond to each other, we may infer different misconceptions.

Conclusion

In this paper we presented a brief description of fundamental issues onto which the design and development of the Intelligent Foreign Language Tutor ALICE has been based. While ALICE is operational, complete and efficient to be used as it is in a real educational setting, we are aware that it might grow in its diagnostic competence as a result of concrete experimentation.

Aside from the specific interest of our analysis of misconceptions occurring in the task of translation of "difficult" words in context, we believe that the methods and techniques can be extended to other analogous tasks. For instance, we are currently thinking of the diagnosis of mistakes in understanding and translating the meaning of clauses as a function of the assignment of focus.

References

- Bates, M., Beinashowitz, J., Ingria, R., and Wilson, K, "Generative Tutorial Systems," *Proc. ADCIS Conf.*, Atlanta, GA (1981).
- Cerri, S.A. and Breuker, J., "A Rather Intelligent Language Teacher," *Artificial Intelligence, Proc. AISB Conf.*, Amsterdam (1980).
- Cerri, S.A. and Landini, P., "Representation of Knowledge Relevant to Multilingual Translation" (in preparation).
- Cerri, S.A. and Merger, M.F., "Learning Translation Skills with a Knowledge-Based Tutor: French-Italian Conjunctions in Context," *Proc. 1st Eur. Conf. Assoc. Comput. Linguistics*, Walker (ed.), SRI International, Menlo Park, CA, (1983).
- Genesereth, M.R., Greiner, R., Grinberg, M.R. and Smith, D.E., "The MRS Dictionary," HPP-80-24, Stanford Heur. Programming Project (1984).
- Landini, P., "Modelli concettuali e rappresentazione del dialogo diagnostico nella traduzione multilingue" (Conceptual Models and Representation of the Diagnostic Dialogue in Multilingual Translation), Thesis, Department of Informatics, University of Pisa (1985).
- Weischedel, R.M.; Voge, W.M. and James, M., "An Artificial Intelligence Approach to Language Instruction," *Artificial Intelligence 10*, (1978).

⁴Notice that interference is different from similarity as in the first case the conjunctions involved belong to two different languages, whereas in the second one they both belong to the target language.

Micro-PROLOG in the Classroom

Christopher Colbourn, Paul Light and David Smith
Department of Psychology
University of Southampton

This paper will describe the objectives, procedures and some preliminary results of an ESRC-funded project *Peer Interaction and Logic Programming* running from August 1984 to July 1985.

In broad terms, the aim of the project is to characterize the learning processes involved, and some of the conceptual difficulties which arise, when the first year comprehensive school children are introduced to micro-PROLOG in a school setting. It has been claimed that micro-PROLOG is both readily learnable and provides a potentially powerful teaching tool for use in conventional school curricula. This would seem to apply particularly well to the humanities, where the representation and organization of declarative knowledge often lies at the heart of the educational enterprise. Our own use of micro-PROLOG has involved utilizing its elementary database handling facilities in the context of a humanities foundation-year curriculum. The existing curriculum already emphasizes evidence and inference.

A specific focus of the research is an evaluation of the role of interaction between children in working groups at the computer keyboard. Previous studies at Southampton and elsewhere have shown that collaborative problem solving can, under certain circumstances, be an effective stimulus to individual learning. This has been demonstrated both with non-computer- and with computer-based tasks, but typically both task and setting have been highly contrived, and lacking in ecological validity in relation to classroom settings. In particular, children working individually have lacked any access to other children. The present study allows an assessment of peer interaction as a facilitator of learning in a more realistic context.

Sixty-four children were involved at the outset, though there has been some attrition. The children have weekly sessions in groups of eight. For some of these groups the children work individually, eight Sinclair Spectrum microcomputer systems being available. For other groups only four such systems are available and the children work in pairs. For others again, only two systems are present so that they are working in groups of four. Collaboration is explicitly encouraged in the cases where children are working in twos or fours, but even the children working individually with the microcomputer have free access to one another.

Allocation of children to conditions and to particular working groups is arbitrary, except that all groups are same sex. Time sampled videotape records of ongoing work have been taken, as well as videotapes of children working together on certain preset problems. Programs produced by individuals or groups have been saved. In addition, pencil and paper measures of individual children's grasp of micro-PROLOG have been taken. Attitude measures have also been taken from the children involved in the project and from matched comparison groups not taking part. An analysis of much or all of this material should be available by the time of the conference.

Computers and Experience in Learning Science

Andrea DiSessa

Massachusetts Institute of Technology

The relation between computers and human experience is a complex one. In fact, my purpose is not at all to reach a conclusion on this subject, but merely to open a few perspectives on it so that the complexity is clear. Some of these perspectives begin from a point of view which, at the first level, is often used to criticize the potential of computers in education. But I hope to show that judgements that classify computational experience as, for example, necessarily artificial or symbolic in comparison to "real world" experiences are premature at best, often based on preconceptions which disappear on closer inspection.

Perspective 1: Will computers dominate our children's experience in the future?

Perspective 2: Is experience direct experience with the world a pipeline to reality? Aren't all computer experiences artificial, replacements for the real thing?

Perspective 3: How can you discover anything new from a sequential, deterministic program? Can computers help our students to re-experience their world in profound new ways?

Perspective 4: Can computers allow us to "experientialize" symbols and formalisms in ways we haven't been able to with other technology?

Perspective 5: Is it really plausible to propose to develop a deep understanding of science based on experience with computational microworlds?

Object LOGO: Object-Oriented Programming Refined

Gary L. Drescher
Artificial Intelligence Laboratory
Massachusetts Institute of Technology

In object-oriented programming, one constructs entities, called objects, each with its own *state* and *behavior*. A new object is constructed either from scratch, or as a *specialization* of another object; a specialized object *shares* the attributes of the object it's based on, but can be given additional (or alternative) attributes of its own.

Object LOGO is an extension of LOGO, designed to facilitate object-oriented programming without drastically changing the appearance of the language. In Object LOGO, one communicates with an object simply by telling it to execute some ordinary LOGO expression – for example:

```
TELL :TURTLE4 [FD 100 RT 90]
```

An object may have its own local versions of certain functions and certain variables, so that the same expression may be interpreted somewhat differently by different objects.

Building an object, and giving it its own functions and variables, is straightforward. For example:

```
MAKE "TALKER SOMETHING
TELL :TALKER [HAVE "NAME "GERTRUDE]
TELL :TALKER [HOWTO SPEAK :MSG]
PRINT (SENTENCE :NAME "SAYS :MSG)]
```

The function `SOMETHING` creates and outputs a fresh object (which we save here as the value of `TALKER`). The function `HAVE` causes the current object (the one we're currently `TELL`ing to do something; in this case, `TALKER`) to establish its own version of some variable (in this case, `NAME`), which is assigned a value (in this case, `GERTRUDE`). Similarly, `HOWTO` is used to define a function (here, the function `SAY`, with argument `:MSG`) locally to the current object. Now we can say:

```
TELL :TALKER [SAY "HI]
```

and LOGO prints GERTRUDE SAYS HI.

The function KINDOF creates a new object as a specialization of another. For instance, we might make a object just like :TALKER, except that the new object "shouts" when it is told to SAY something.

```
MAKE "SHOUTER KINDOF :TALKER
TELL :SHOUTER [HOWTO SAY :MSG
  TYPE "HEY!
  USUAL.SAY :MSG]
TELL :SHOUTER [SAY [HELLO THERE]]
LOGO prints: HEY! GERTRUDE SAYS HELLO THERE
```

Initially, :SHOUTER shares :TALKER's version of the variable NAME and the function SAY. But :SHOUTER is then given its own definition of SAY. :SHOUTER's version of SAY begins by typing HEY! . Then, by using Object LOGO's "USUAL" facility, :SHOUTER's SAY function invokes the "usual" version of SAY, meaning the version that :SHOUTER would be sharing with :TALKER, had not :SHOUTER's own definition of SAY superceded :TALKER's.

The traditional distinction between *class* and *instance* is not built into Object LOGO, but is easily constructed when needed. Suppose, for example, we wish to have a number of different talkers and shouters, each with its own name. We can say:

```
MAKE "TK1 KINDOF :TALKER TELL :TK1 [HAVE "NAME "CHRIS]
MAKE "TK2 KINDOF :TALKER TELL :TK2 [HAVE "NAME "KIM]
MAKE "SH1 KINDOF :SHOUTER TELL :SH1 [HAVE "NAME "FRED]
```

Here, :TALKER and :SHOUTER are regarded as each specifying a class of objects; :TK1, :TK2, and :SH1 are instances of those classes. The instance objects are specializations (made by KINDOF) of their respective class objects. An instance inherits its "behavior" from a class (by sharing the class's function definitions, e.g., SAY); but the instance has its own individual "state" (by having its own version of some variable, e.g., NAME).

To summarize, Object LOGO's distinctive features are that:

1. An object is invoked by telling it to run an ordinary LOGO expression.
2. An object can be given its own variables and functions, interactively and incrementally.

3. The distinction between class and instance is not required, but is easy to establ. For beginners, this means that the (rather subtle) distinction can be *ignored* (as is was above until the last example). And for advanced users, a uniform paradigm of specialization, realized by the primitive KINDOF, is used to make both instances and specialized classes.

The Schema Mechanism – Translating Piaget Into LISP

Gary L. Drescher

This paper presents a constructivist model of human development during infancy. According to constructivism, the elements of mental representation – even such basic elements as the concept of physical object – are constructed afresh by each individual, rather than being innately supplied. A (partially-specified, yet-unimplemented) mechanism, the Schema Mechanism, is proposed here; this mechanism is intended to achieve a series of cognitive constructions characteristic of infants' sensorimotor-stage development. I rely primarily on the Piagetian account of this development; the Schema Mechanism is an attempt to *explain* the developmental progression that Piaget describes.

Included here is a sketch of the proposed Schema Mechanism, and highlights of a hypothetical scenario of the mechanism's operation. The Schema Mechanism starts with a set of sensory and motor primitives as its sole units of representation. As with the Piagetian neonate, this leads to a "solipsist" conception: the world consists of sensory impressions transformed by motor actions. My scenario suggests how the mechanism might progress from there to conceiving of objects in space – representing an object independently of how it is currently perceived, or even *whether* it is currently perceived. The details of this progression parallel the Piagetian development of object conception from the first through fifth sensorimotor stage.

The Schema Mechanism focuses, on Piaget's concept of the competition and evolution of cognitive *schemas*. The schema is modelled here as an assertion that a given *action* has a particular *result*, provided that certain *context* conditions are met. Context and result states are represented in terms of binary (true-false) elements called *items*. Schemas are used both as assertions about the world, and as elements of plans to achieve goals.

A mechanism of *attribution* causes a schema's assertions to be extended or revised according to the observed effects of the schema's action. A schema discovers additional results of its activity; and if these results are unreliable, further context conditions are identified to enhance reliability. New schemas are formed to express these discoveries. Typical associationist models assume that only a small number of possibly-relevant events occur at once. But attribution is designed to deal with perhaps thousands of recognized events (or aspects of events) occurring together, and with the importance of identifying *conjunctions* of events as context conditions or as results. A technique is proposed to sort through the 2^n possible conjunctions of n elements with only n^2 worth of exhaustive tabulation.

It is not enough to acquire connections among existing terms of representation. A constructivist systems's greatest challenge is to extend its own conceptual vocabulary, to abstract beyond the initially-supplied primitives. In addition to building new schemas in terms of extant actions and items, the Schema Mechanism constructs new actions and items.

Each newly-achievable result defines a new *composite action* – the action of achieving that result. A composite action is implemented by a coordination of schemas that chain to a particular result; these schemas are used to achieve that result when the composite action is initiated. As with primitive actions, the attribution machinery looks for the results of composite actions, forming new schemas to describe the discovered results.

Most importantly, the Schema Mechanism constructs *synthetic items* to postulate the existence of additional components of reality when now state-elements prove inadequate to explain observed events. A synthetic item is at first an unanalyzed "dummy" explanation of an unusual occurrence; the item represents whatever condition is responsible for that occurrence. A synthetic item gains content by the discovery, and inference, of circumstances under which the condition represented by the item can be deemed to obtain. Synthetic items, like primitive ones, come to be incorporated (via the attribution mechanism) in the contents and results of new schemas.

Learner-Based User Modelling

Mark Elsom-Cook
Human Cognitive Research Laboratory
Open University
Milton Keynes, U.K.

There have been several attempts in the tutoring systems literature to move from an expert-based paradigm to one which is more closely linked with the state and capacities of the pupil [Goldstein 1982]. Such a transition requires a tutoring system to embody some model of the learning process. At present, there are no comprehensive and well-developed theories of human learning which are suitable to this task.

The purpose of this paper is to establish the usefulness of less than complete models of the learning process. In particular, it describes the user-modelling aspects of a system for tutoring programming languages (IMPART), and introduces the concept of "bounded user-modelling".

IMPART is a guided discovery learning environment for programming languages. The pupil interacts with a language interpreter via a syntax-directed editor. This part of the system is described [Elsom-Cook 1982]. Unguided learning in such an environment was found to be unsatisfactory. Consequently, a tutoring system was designed which is intended to sit "on the side" of such an environment and provide guidance for the pupil. This provides a framework for the nature of such guidance.

The user-modelling aspects of the tutoring system combine a well-known machine learning methodology [Michalski 1983] with a semantic representation for programming languages. Together these components form a unit which receives the same information which a pupil would receive when interacting with a language interpreter. The user-modelling component applies the learning model to this data, and for each "event" which occurs in the environment, produces a pair of "event descriptions" which are intended to delimit the range of hypotheses which the pupil could have formed based upon that event. This information is passed to the remaining part of the tutoring system which attempts to locate the pupils' beliefs in this spectrum of hypotheses, and to guide the pupil toward the most general correct hypothesis. These event descriptions place bounds on the amount which the pupil could have learned from the event. The teacher can act to narrow the distance between these bounds. Because there is no absolute representation of the knowledge of the pupil, this method will be termed "bounded user-modelling".

Since the learning model can also make use of "event descriptions" taken from earlier events, it can compound incorrect hypotheses in the same way that a pupil could. In effect, this permits it to build a "buggy" language description corresponding to the language which the pupil thinks she is using. It is important to note that, although this is an errorful variant of the correct model of the language, it has not been derived by assuming that the pupil is a "deficient expert". This enables the system to overcome many of the criticisms which can be made of expert-based (and mal-rule based) approaches.

The present learning algorithm uses five rules of selective generalization and one rule (based on pattern-matching) of constructive generalization. As a separate unit, the learning mechanism has been shown to "learn" a reasonable subset of LISP by observation. Its performance deteriorates when confronted with events requiring disjunctive descriptions. This part of the IMPART system is implemented in PROLOG.

References

- Elsom-Cook, M.T., "MATILDA: A Menu and Template Interface to LISP," M.Sc. Thesis, Warwick University (1982).
- Goldstein, I.P., "The Genetic Graph," in "Intelligent Tutoring Systems," D. Sleeman and J.S. Brown (eds.), Academic Press (1982).
- Michalski, R.S., "A Theory and Methodology of Inductive Learning," in "Machine Learning," R.S. Michalski, T.M. Mitchell, J.G. Carbonell (eds.), Tioga Press (1983).

Algebra Slaves and Agents in a LOGO-Based Mathematic Curriculum

Wallace Feurzeig
BBN Laboratories
Cambridge, MA

For many students the ideas and methods of algebra appear obscure and mysterious, their sense and purpose unclear, and their applicability to anything genuinely real or interesting very remote. Most students fail to acquire an understanding of the key concepts. Even when they understand the meaning of variables, expressions, and equations, they often lack the strategic knowledge required to motivate and direct the global planning and detailed execution of an attack on a problem. Their conceptual and strategic difficulties are compounded by their lack of skill in arithmetic and symbolic manipulation. Complex sequences of operations, like subtracting an expression from both sides of an equation or expanding a product of two terms, are very difficult for beginning students. Their buggy performance in carrying out the detailed manipulative work greatly confounds and frustrates their acquisition and assimilation of the most important and central ideas and strategies, despite their inherent simplicity.

We plan to confront these difficulties and show how they can be overcome. We are developing a LOGO-based algebra course with two major components:

1. Programming projects designed to teach the notions and art of algebra as exemplified in contexts that are meaningful and interesting to students,
2. A new class of expert instructional tools designed to serve students as slaves, demonstrators and mentors in performing algebraic manipulations.

These experts tools are a set of symbolic manipulation programs employing pattern recognition methods for performing the standard manipulations of high school algebra. Each tool is a specialist designed to perform a very specific function such as parsing, simplifying, evaluating, or canonicalizing an expression; collecting, grouping, or transposing terms; dividing both sides of an equation by an expression. Embedded in these programs are the facilities required to do all the computational operations required for solving algebra problems in the high school course. These tools are qualitatively different from the symbol manipulation programs currently implemented on microcomputers, such as muMath, TK!Solver, and VisiCalc. These are powerful performance systems but they are not powerful as instructional aids. They do not provide capabilities for demonstrating and explaining their performance as they work through a problem. And they have no facilities for monitoring and critiquing a student's work in attempting to solve

a problem. We require more intelligent and articulate instructional tools that are accessible to young students.

Our tools will have three main modes of use: execution mode, which simply performs the operation requested by the user (just as current tools do); demonstration mode, which performs the requested operation incrementally, explaining its strategy and its mode of performance of the component steps in straightforward terms along the way; and practice mode, in which the student tries to solve a problem with the assistance of the tools, which perform the requested manipulations, monitor the student's performance, diagnose errors and difficulties, and describe major features of the student's strategies relative to those of the expert. By providing these powerful aids, we can effectively separate out the difficulties in performing the formal and manipulative aspects of algebra work from those encountered in learning the central conceptual and strategic content. If we can successfully address the latter issues and help students acquire a strategic understanding of algebra, the tools will become useful and uniquely valuable to them for solving problems. Otherwise, however, students will not know how to use the tools.

Our plan for developing this understanding of algebra depends centrally on the use of LOGO programming ideas and activities. The LOGO work spans the entire course, covering most traditional algebra topics and units, e.g., Variables, Expressions, and Relations; Equations and Inequalities; Polynomials; Functions; Story Problems; and Estimation. The key algebra ideas and methods are introduced and developed in the context of LOGO programming. The LOGO sequences used in these developments often depart significantly in approach and content from the corresponding topics in the traditional course. The sequences and tools will be described in the talk.

A System for Teaching a Qualitative Understanding of Electrical Circuit Behavior and Troubleshooting

John R. Frederiksen and Barbara Y. White

BBN Laboratories

Cambridge, MA

The goal of this project was to develop an intelligent tutoring system for teaching qualitative reasoning about electrical circuits in general, and troubleshooting of automotive electrical systems in particular. The system is designed to provide a simulation environment in which students can solve circuit problems. It also serves a tutorial function and can be called upon to solve problems and demonstrate to students the reasoning involved. The instructional objective is for students to acquire, through their exposure to a sequence of carefully selected circuit problems, a sequence of increasingly sophisticated models of circuit behavior.

To this end, we have formulated a series of cognitive models that correspond to different levels of understanding of electrical circuit concepts. Each model is upwardly compatible with higher level models, but is restricted in the set of circuit problems for which it is adequate. While these mental models vary in the complexity of the circuits about which they can reason, they all represent "runnable" qualitative simulations and enable the student (or the system) to predict the behavior of circuits of a given level of complexity.

Circuit problems given to students include (1) making predictions about circuit behavior, and (2) troubleshooting or isolating faults within circuits. Corresponding to each of these two types of problems are two tutoring systems: (1) a qualitative, causal simulation of electrical circuits that illustrates principles for reasoning about circuits; and (2) an "expert" troubleshooter that demonstrates a strategy for isolating faults within circuits and that incorporates the same type of reasoning as that involved in predicting circuit behavior.

These tutoring systems are based upon a qualitative approach to troubleshooting taken by an expert mechanic whom we have studied. His methods are based upon the fundamental idea of a circuit: For a device to operate, there must be an electrical potential across the device. When such an electrical potential exists, a current will flow through the device (provided it is conductive), causing it in some cases to change its state. Knowledge of the device, in the form of a device model, allows one to infer what state the device will enter and under what conditions.

When troubleshooting, the expert's goal is to choose a test point that divides the circuit into two parts and then to infer which portion of the circuit, that in parallel or that in series with a test light, is faulty. When the fault has been isolated to within a portion of the circuit, the expert then moves to a new test point within the faulty part of the circuit and recursively applies the troubleshooting logic. The approach of our expert has the advantage of being generally applicable to a large class of series-parallel circuits. However, it presupposes a knowledge of electrical circuits and requires an ability to reason qualitatively about circuits.

Given the need to teach electrical principles and their implications as a prerequisite to teaching troubleshooting, we created an instructional environment that is capable of demonstrating and providing practice in applying circuit principles. The basis of this system is a progression of increasingly sophisticated, qualitative causal models that utilize knowledge of the structure of the circuit, the functioning of the devices within the circuit, and the basic electrical principles to simulate circuit behavior. These simulations needed to be robust in permitting faults to be introduced or circuits to be modified without requiring a new model for each perturbation in the circuit. By utilizing context free functional models for devices, along with topological search processes for evaluating electrical potential across devices, we are able to construct circuit models that accurately simulate the behavior of a large class of circuits in both faulted and unfaulted states.

For any given circuit in the teaching sequence, the student is asked to predict the effects of faults that the system has introduced, and to locate unknown faults. At any time, the student can call upon a model of circuit behavior to explain the functioning of the circuit, or upon a troubleshooting model to describe how it would isolate a particular fault in the circuit. Each tutorial program utilizes a model that articulates reasoning at a level of explanation that is appropriate for the particular stage in instruction. We are currently conducting research to determine whether the set of activities that we have developed, combined with the explanatory power of the progression of circuit and troubleshooting models, enables students to acquire a qualitative understanding of circuit behavior and troubleshooting logic.

Acknowledgements

The project includes, in addition to the authors, Wallace Feurzeig and Frank Ritter. This work is supported by the Office of Naval Research and the Army Research Institute, under ONR contract N00014-82-C-0580 with the Personnel and Training Research Program.

The Epistemics of Computer Based Microworlds

Guy J. Groen

Department of Educational Psychology
McGill University

In recent years, the notion of a microworld has appeared frequently both in the artificial intelligence literature and in discussions of educational applications of computers. However, its usefulness has been hampered by the fact that it has never been precisely defined. The purpose of this paper is to suggest a more precise definition and consider some of its implications.

In artificial intelligence, a microworld is usually viewed as referring to a domain of objects with "nice" representational properties. In the field of education, it seems to mean something more. It is a computer based environment that a student can explore and possibly extend. It also provides the basis of theory of what might be learned in such an environment. Specifically, Papert (1980) has suggested an amalgam of Piagetian theory with artificial intelligence in which microworlds are "incubators for powerful ideas". There is currently widespread controversy regarding what is, or is not, a powerful idea. Since there is less controversy regarding instances of microworlds, it is possible that a more precise definition may also clarify the notion of a powerful idea.

The basis of the definition is a formalization of Piaget's notion of structure (Groen 1978). Basically, a structure is a set of states and transformations between states. An important property is that the transformations should be modular. Each transformation, with all its sub-transformations, can be viewed as an object, and an appropriate linking topology can be defined between these objects.

A microworld is a structure with certain additional properties. The two most important are:

1. A transformation can be undone to go back to the previous state.
2. There should exist mappings (homomorphisms) to other structures that are representations of concrete actions in the "real world".

Such a definition is sufficient to distinguish between microworlds and non-microworlds. Within LOGO, turtle graphics is a microworld. The states are the possible transformations of the turtle. The commands are the transformations. The procedures are objects. A topology can be defined either in terms of the superprocedure-subprocedure hierarchy or in terms of

communication between procedures. Every turtle movement can be undone. Every procedure can be mapped onto an appropriate object in a variety of concrete worlds such as that of drawings with pencil and paper. On the other hand, arbitrary sets of numbers, words or lists are not microworlds. Neither are programming languages.

One advantage of a definition along these lines is that it provides a means of using the techniques of modern cognitive psychology, such as protocol analysis and propositional analysis, to analyze what is learned in a microworld. The notion of a microworld presented here corresponds closely to recent definitions of mental models in terms of object topologies. Recent research by the author, in collaboration with V. Patel (e.g., Patel and Groen, in press) indicates that it is possible to analyze empirical protocols of subjects in complex verbal problem solving tasks, using propositional analysis techniques, to yield hierarchical rule systems with object-like properties. Such rule systems are fragmentary in the case of novices but connected in the case of experts.

References

Groen, G.J., "The Theoretical Idea of Piaget and Educational Practice," in P. Suppes (ed.), "Impact of Research on Education: Some Case Studies," Washington, DC, National Academy of Education (1978).

Papert, S., "Mindstorms," Basic Books, NY (1980).

Patel, V. and Groen, G.J., "Knowledge Based Solution Strategies in Medical Reasoning," *Cognitive Science*, in press.

A Laboratory for Exploring Design Constraints

Mark Gross

Design Theory and Methods Group
Department of Architecture
Massachusetts Institute of Technology

This paper describes a model of design expertise based on constraint exploration and presents a computer-based learning laboratory for design. We are concerned in particular with the design of physical layouts, for example floorplans. Constraints are the representation for design knowledge; the laboratory provides tools for the management, manipulation, solution, and resolution of systems of constraints. The operations in the laboratory are described and simple examples for the design of floorplan layouts are given. Some details of implementing a constraint exploring design laboratory using an object oriented version of LISP/LOGO are presented. The remainder of this abstract summarizes our model of design, the relation of the computer-based laboratory to the model, and our motivations for undertaking the work.

Design differs from problem-solving domains such as diagnosis in several ways. The most important difference is that design problems have no single best solutions; there are many competing objectives by which to keep score. That is, design does not optimize globally because there are too many objectives. These design objectives and constraints shift and change constantly throughout the design process. The design constraints as initially understood and described by the designer may be inconsistent, incomplete, and redundant; sorting and sifting the constraints into coherent sets is part of the design process. In any design domain there are natural laws, particular requirements of the problems at hand, and conventions of style and common practice. All these constrain the space of alternatives, and every design decision the designer makes imposes additional constraints on the design to bound the search for good alternatives.

We see the computer as an aid in the search for good alternatives. But computational models of expertise that have proved successful in problem-solving domains seem inappropriate for design. Rule based expert systems, for example, are programmed to find a single best solution. And procedure-based models require a preconception of the problem to be solved, but in design the problem description evolves simultaneously with the design alternatives under consideration. Our laboratory supports the design process by calculating consequences of new constraints and variable settings, by maintaining a history of the design process, supporting dependency directed retraction and providing justifications for inferences made by the machine, by automatically

partitioning large systems of constraints into workably-sized chunks, and by permitting multiple views of a design.

Learning to design is made more difficult by the absence of a formal design theory or method. We are interested in what designers know and how they use this knowledge in designing. Much of design knowledge can be described as constraints on design attributes; therefore our model and laboratory deal with organizing and manipulating design knowledge represented as constraints. We believe that our model affords a better and more explicit understanding of design expertise and competence and can advance design education as well as practice. The computer provides a laboratory for testing our model of design expertise as well as a tool for amplifying our design efforts by making systematic exploration of complicated designs affordable.

The Young Computer Scientist: A Curriculum Plan

Brian Harvey

Art, Media and Technology Center
Massachusetts Institute of Technology

For the elementary school student who is first learning how to program a computer, there is a great deal of intellectual meat in the experience. The very idea of an algorithm – a *formal* description of the steps in solving a problem – is new and important. The early literature from MIT LOGO Lab describes many other “powerful ideas” like modularity and debugging.

For the Computer Science major at a university, there is also much intellectual meat on the menu. There is the detailed study of known methods for particular kinds of problems, like formal grammars for compiler construction or queues and semaphors in operating systems. There is the study of computing as a mathematical systems, in automata theory or recursive function theory.

For the teenager, though, the situation is different. The kinds of expertise that are easily pursued tend to be intellectually barren. There is the detailed study of PEEKs and POKEs for some particular brand of microcomputer. There are techniques for breaking into someone else's computer. And there is the struggle to cram a few more instructions into a limited memory. Any of these pursuits can be entertaining, but none is really worthwhile.

Barriers to Worthwhile Computing for Adolescents

There are several reasons for the general lack of worthwhile computing experiences for adolescents. One reason is the weak hardware usually available. LOGO for microcomputers is a fairly recent phenomenon; the ubiquitous BASIC is a strong deterrent to any sort of structured thinking about programming issues. Also, the isolated home computer with small, removable disk storage doesn't build a programming community, like the ones that grow inevitably on any large, timeshared machine.

Another problem is the shortage of truly educational literature for this age group. Sometimes a *Byte* article addresses an intellectually significant issue, but the usual magazine fare tends more toward commercial boosterism, or toward sample programs presented without discussion and written in the worst possible BASIC style. There seems to be hundred of programming books, but all addressed to the complete beginner whose greatest need is for reassurance.

The schools have not, in general, filled the gap. One reason is that there are very few teachers in high schools with any real training in Computer Science. The economic reasons for this shortage are too well-known to need repeating here. A more fundamental reason, though, is that most secondary schools think about educational computing from a perspective other than that of Computer Science.

One such perspective is that of the computer as tool, to be used in teaching the traditional curriculum. Simulation programs, drill programs, word processing for the English class, and computer-based laboratory instruments are all part of this approach. I think that some pieces of the tool perspective are more valuable than others; I like word processing; I don't like drill. But my point is not to argue with the approach, except to the extent that it drives out the different approach that I want to suggest.

Another common perspective is that of "computer literacy". This catchy slogan seems to mean different things to different people. Sometimes it means job training, either to prepare students for computer programming jobs, or less convincingly, to prepare them for any job, because all jobs in the future will "involve" computers. Sometimes, proponents of the "literacy" approach concede that universal programming lessons as job training is nonsense, but suggest that there is some other lesson in their courses that everyone needs: critical thinking skills, the knowledge of the informed citizen, or whatever. I have argued elsewhere that there is no substance to this perspective in any of its forms. The danger of the "literacy" approach is that, by trying to fill a universal need, it uses up all available resources, and ensures a watered-down curriculum.

Our high schools now generally attempt to teach a serious, intellectually sound physics course. They teach it to students who are interested in science. It is not a "physics literacy" course that attempts to appeal to every student, although schools may, in fact, encourage every student to study physics. But anyone who proposes a similar approach to computer programming is often called an elitist.

A Computer Science Curriculum

Not every adolescent needs to study computer programming. But for those who are interested, that study can offer two benefits: an applied, practical approach to mathematical ideas, and the same disciplined apprenticeship that other young people may get from art, drama, or journalism. For both of these purposes, I think the study of programming should be based on the ideas of Computer Science.

What I've just said isn't new. It's always been part of the LOGO tradition that we're fundamentally teaching mathematics. So, for example, the mathematical idea of *composition of functions* is at the core of LISP, the language from which LOGO was derived. I think that we should make that idea explicit in teaching LOGO programming to high school students. That means paying more attention to LOGO's *evaluation* mechanism than is usually done.

I see three stages in a teenager's study of computer programming. In the first stage, he or she must learn the rules of the game; the syntax and semantics of a programming language. At

this stage, formal teaching in groups can be very valuable. In the second stage, the newly-hatched programmer wants and needs a lot of practical experience. At best, it is the individual learner who provides the initiative and the direction; the teacher looks over shoulders and offers occasional advice about style and technique. After some months or even years at the second stage, some students will start complaining of boredom. They've gone as far as they can, with their limited understanding of Computer Science ideas, in stretching their programming skills. They're then ready for the third stage, a return to more formal instruction, this time to get acquainted with some of the more powerful formal structures they'll meet in college.

I've been working on a series of books to guide young computer scientists through these three stages. There are three books in the series, one for each stage. I'm trying to make the books usable either in a school setting or by an independent learner at home.

Stage One: Introduction

At first glance, it must seem that most kids of high school age don't need an introduction to programming. By now, everyone learns LOGO in the third grade, and nearly everyone learns BASIC, alas, somewhere along the line.

One the other hand, the approach taken to LOGO in the elementary school is geared to quick results for unsophisticated students. The emphasis is on commands rather than operations: lots of FORWARD and RIGHT, not much POS and HEADING - let alone FIRST and BUTFIRST. As a result, the central issue of composition of functions is often handwaved away, along with the idea of evaluation of inputs. And of course our students' exposure to BASIC does more harm than good; it teaches them many subtle bad ideas that we must help them unlearn, like verb-object syntax and universal use of global variables.

The first topics are procedures and evaluation. These go together, because evaluation in LOGO is primarily a matter of invoking procedures. It's important to emphasize that user-defined procedures have the same status in LOGO as primitive procedures. Secondary issues that must be discussed to allow even the simplest LOGO instructions are quoting and data types (words and lists). But it's possible to get in a lot of programming practice before introducing variables, so I do.

After variables come predicates (and IF) and recursion. Unlike the usual pedagogic approach for younger students, my preferred technique is to begin with an example of recursion that is *not* tail recursive. This makes the topic seem more mysterious at first, but it also makes it impossible for BASIC experts to handwave recursion into a funny kind of looping.

Up to this point, I deal only with the list processing part of LOGO, not with graphics. I'm bending over backwards to avoid anything that looks cute, like a turtle, to try to overcome LOGO's reputation as a baby language. But after the chapter on recursion, there's a detour through turtle graphics, pointing out some of the sophisticated ideas in that part of LOGO. (Locality is an example, in contrast to the global approach of Cartesian graphics.)

A chapter on writing recursive operations completes the "how-to" part of the book. The remaining chapters deal with planning, debugging, iteration and mapping tools, and some advanced LOGO features like CATCH and THROW.

Stage Two: Style and Technique

The second stage, after our hero learns how to program and before he or she gets bored with it, is likely to last several years. (I suppose it can last someone's entire life.) A learner at this stage is least amenable to the usual formal class structure, and instead needs minimal supervision in a computer lab with a lot of peer communication.

The role of a teacher, or a curriculum developer, at this stage is twofold. One part is to provide examples of programming projects, for those students who get stuck thinking up their own. The other is to look over the apprentice programmer's shoulder pointing out stylistic flaws (although not necessarily at every opportunity) and suggesting new techniques that might make some problems easier.

I hope to meet both of these needs, as much as any book can meet needs like these, with a collection of worked-out programming projects, including commentary on the style and techniques used in each. The projects are not necessarily written in "perfect" textbook style; instead, the commentary points out deviations from that style, suggests alternatives, and also suggests why I wrote the program the way I did.

The techniques used in the projects include indirect assignment (a computed first input to MAKE), keyword inputs, and recursion through the mutual invocation of two procedures. I've made no attempt to order the projects, by difficulty or by any other scheme. My guess is that nobody will read the book straight through anyway; instead, the idea is to flip through the book until you find something that seems interesting, study that project, and then go develop one of your own.

Stage Three: Computer Science

The best students will eventually get bored with writing more and more computer programs. The cure for boredom isn't just to find a new thrill, but rather to find a deeper one. We've really been doing computer science all along, emphasizing ideas like locality and modularity from the beginning. But now it seems appropriate to let the mathematical nature of the activity become more apparent, thinking more formally than in the earlier, more practical stages.

My hope for this stage is to make the same ideas that are taught to Computer Science majors in graduate school accessible to teenagers. For each topic, I try to find a "hands-on" approach. For example, in automata theory, there is a theorem saying that finite state machines are equivalent to regular expressions. (A finite state machine is a *procedural* notation for a problem; it says, "first do this, then do that." A regular expression is a *declarative* notation for the same class of problem; it says, "here is the result I want." The deep equivalence between the two notations is

a smaller-scale version of the deep equivalence between a procedural language like LOGO and a declarative language like PROLOG.) In any standard text on automata theory, you'll find a very elegant formal proof of this theorem. Instead, I present a LOGO program that actually translates a regular expression into a finite state machine. Readers can actually play with the program, perhaps by using it as a black box to experiments with the translation of various regular expressions, or perhaps by taking apart the program itself and seeing how it carries out the translation.

Other topics to be covered in this third book include data structures, combinatorics, analysis of algorithms, and artificial intelligence.

Future Directions

For the current generation of common educational computers, LOGO is clearly the language of choice for secondary school computer science. LOGO is more powerful and cleaner than Pascal; it directs the learner's attention to interesting issues, rather than to syntactic details and implementation concerns.

Looking to the future, though, there are several things LOGO needs to be truly adequate to this purpose. First, LOGO *compilers* are needed; interpreted LOGO is simply too slow for large projects. Second, some of the extra memory in the newer computers should be spent on new primitives for iteration, mapping, bit manipulation, and arrays. Third, LOGO should provide a rich operating system interface, like LISP on a LISP Machine.

What I'm suggesting here is to move in a direction different from the one usually proposed: iconic programming, microworlds, "no programming language at all." I'm not saying that the other direction is wrong. But I hope that "no threshold" is not pursued at the cost of forgetting "no ceiling."

LOGO Programming Activities

Celia Hoyles, Rosamund Sutherland, and Joan Evans

Institute of Education

London University

The LOGO Math Project (Hoyles, Sutherland, and Evans 1985) which commenced in September 1983 is a three year longitudinal study with children who were 11/12 years old at the beginning of the period of research. We have chosen to work in two first year mixed-ability classrooms which have adopted a pupil-centered approach to the learning of LOGO and the research is carried out during "normal" mathematics lessons. In one classroom the pupils have had the use of two RML 380z computers and in the other classroom the pupils had the use of four Commodore 64 computers. The pupils work in pairs and are given the freedom initially to devise their own goals and decide how they might achieve them. As researchers, we have acted as participant observers in the classroom, conducting detailed case studies of four pairs of pupils from each school. Recording of the pupils' LOGO work and spoken language (the video recorder is connected between the computer and the monitor), together with pupil and teacher interview data, have provided an extremely powerful database and allowed us to analyze in depth the progression in learning. When and why we need to intervene during the pupils' LOGO activity is being analyzed and the consequences of these interventions in terms of the pupils' motivation and learning recorded. We are also analyzing the nature and extent of the collaborative work between pupil pairs and how this is related to the goals chosen and to the mathematical processes and the problem strategies adopted. This presentation will focus on the types of programming activity identified in our research. These are:

Working at a Syntactical Level

This activity consists of the use of primitives, procedures (or sequences of these) without any understanding of how or why the "act" as they do. For example, random typing of commands, "copying" from other pupils or from a handbook, or merely adapting a pattern which by chance produced a pleasing result. Our observations have led us to believe that pupils who work at a syntactical level are not provoked to think about the processes involved in their work and this tends to be detrimental to their learning.

"Making Sense Of"

This is exploratory activity where pupils are trying out a new idea or procedure and reflecting on what is happening. Sometimes such activity is completely non goal-directed, sometimes it takes place within goal-directed activity, and sometimes a goal emerges from the activity. We have found that "making sense of" activity is important if pupils are to develop an understanding of the processes involved in LOGO programming and would suggest that pupils be encouraged to "take time out" from working towards predefined goals to explore how a new process works.

Goal Directed

This is activity aimed at achieving a specific product (even if this is actually modified during the programming activity). From the research data two separate dimensions along which goals can be classified have been identified:

Pictorial	Abstract
Loosely defined	Well defined

It is hypothesized that the "position" of the goal with respect to these two dimensions will affect pupil interaction and behavior. In addition local and global activity within goal-directed activity have been distinguished. Local activity tends to focus on the graphics or text output. Global activity tends to focus on a mental plan.

This presentation will include examples of the above programming activities, drawn from data collected during 1983/5 as illustration of the activities themselves and of the conclusions drawn from our case study work.

References

Hoyles, C., Sutherland, R., and Evans, J., "A Preliminary Investigation of the Pupil-Centered Approach to the Learning of LOGO in the Secondary School Mathematics Classroom (1983-1984)," LOGO Maths Project, Institute of Education, 20 Bedford Way, London WC1H OAL, (1985).

**Eliciting and Representing Knowledge for Learner Models:
A Study in Maths and Science Education**

Dr. N.E. Johnson
Dept. of Electrical Engineering & Electronics
Brunel University
Uxbridge, Middx., U.K.

Prof. J.M. Ogborn
Science Department
Institute of Education
University of London
20 Bedford Way
London, WC1, U.K.

Within knowledge engineering there is a presumption that expert knowledge is stable, structured, and describable. In education, such assumptions cannot unquestionably be made about the knowledge of children, even though it is of course well established in educational research that children can possess complex and coherent forms of knowledge which, to an adult investigator, appear as exotic as a hitherto unrecorded language does to a field linguist. Our concern is to explore these issues, and in doing so to develop further existing tools for knowledge description. In this paper we first describe a specific study in mathematics and science education and then indicate further directions of research.

In exploring the nature of eliciting and formalizing the knowledge of child arithmetic "experts" [Johnson 1983], no machine implementation of the knowledge exists, nor was it planned, for the major thrust of the project was to concentrate on the nature of enquiry and the problems of representing non-standard knowledge. Here an early commitment to a logico-mathematical scheme of representation can easily lead to an incomplete, or even quite inappropriate, account; mindful of that danger, we did not commit ourselves to any of the representation schemes regularly used in AI. Nevertheless, the specific study was akin to AI in its commitment to providing both detailed and explicit accounts of the knowledge. We thus accepted the current aim if not the available methods.

We began by regarding our child informants as experts in their own knowledge with us, as investigators, as naive. It was not just romantic psychology to regard the children as experts; it was in keeping with a long tradition of anthropological social science research. We also felt

that if we took our own naivete seriously, then we should make few initial assumptions about the form or nature of the children's knowledge. Two things seemed essential:

1. We must make the knowledge a public entity, something the child could admit as his own, otherwise our own prejudices/views would mask the children's "real" views.
2. We needed a representation scheme which was sufficiently flexible to capture aspects of the knowledge, but was not a purely adult conception of the domain. Hence the natural choices of set theory, logics, or error classifications, which might be appropriate in parts later, would not be suitable initially.

As a theoretical basis for the first goal we chose Pask's Conversation Theory [Ogborn and Johnson 1983, Pask 1976]. We developed a semi-structured interview which gave the child more than the customary control over topics of discussion and had a built-in device (teachback) to check the adult participant's version of the child's knowledge. Rather than present (and probably intimidate) the child with an arithmetic problem to solve we asked the child to teach the adult "what you know about sums". We operated at Pask's two levels of Level 0 (descriptions and explanations of procedures and facts) and Level 1 (why things work - explanations of the Level 0 explanations). The child would teach the adult a procedure for doing, say, an arithmetic operation at Level 0. The adult would then teach-it-back until the child was sure the adult was doing the same thing (note not necessarily *thinking* the same thing). Then the adult would elicit Level 1 knowledge in the form of things known about the procedure (perhaps the child's learning history or why the operation could be trusted). The adult would teach again until the child approved the adult's version. Only then could the adult lay claim to *understanding* the child's knowledge.

Thus over a period of six half-hour interviews with each of eight children there emerged a detailed picture of the children's knowledge in a publicly agreed form, embedded in interview transcript and nodes and arrows summaries.

Some time later we began the task in 2. of representing the knowledge embedded in two boxes of transcripts and summaries in some more rigorous scheme. We chose two different formalisms, in order to compare their potentialities: Rissland Michener's adaptation of Minsky's Frames [Michener 1978, Minsky 1974] and Systemic Networks. The latter are a formalism developed in Systemic Linguistics to represent grammar in functional terms [Bliss, Monk, and Ogborn 1983]. Systemic Networks found a novel application in this study and had the following appealing features:

1. The content-free formalism suggests ways of structuring the author's ideas of how to interpret/model qualitative data (here ideas in ordinary English conversation). It is a scheme which aids the author in coming to know his data or "forming data-driven concepts" - an essential stage in this or any other knowledge finding activity.
2. The formalism provides only three or four links and the author provides all nodes, and names of nodes. The nodes can therefore be made as expressive as the author chooses and the structure is purpose-built to reflect chosen characteristics of the domain.

3. It is possible to build a full bodied network on a single sheet of paper and the author may then explain or communicate the child's knowledge through the network(s).

Overall, networks are an aid to the analyst, a highly expressive but compact representation scheme with transparent structure making for communicability. We feel these are highly desirable features in an "in house" representation scheme within a team building an AI system.

Contrary to the spirit of AI we did not then proceed to justify the cognitive reality of our "models" by realizing them in a suite of programs, but searched elsewhere for evidence that we had elicited real knowledge. We found three types of evidence:

1. Anecdotal evidence in the form of episodes of real expert behavior in the children.
2. A prediction of the eight children's performance on a standard maths test.
3. A further piece of field work in the form of a simple recognition task. We asked 113 other children to categorize (agree, don't agree, don't know) 60 statements embodying fragments of the elicited knowledge.

Subsequent quantitative analysis of 2. showed that the investigator had a crude prediction success rate of 70% ($= 64$ for 1 df. and $= 0.47$) with interesting local variations. In 3. most children tended to agree with the statements made by the 8 interviewed children (86% classed as decisive and agreeing). Thus we are fairly confident of the success of our elicitation technique.

Our present work proceeds on a variety of fronts. Applications to science teaching continue in the following:

1. The application of Hayes' (1982) theory of naive physics to the learning of mechanics [Ogborn 1985].
2. Application of network representation to the design of structured knowledge-eliciting questionnaires (Koulaidis, unpublished).

It is also continuing beyond academic education studies in two projects at Brunel University:

1. An SERC-funded project, "Knowledge Transfer Methodologies in the Context of Electronic Systems Design" (SERC No. GR/C/6883).
2. A joint project with Alvey and Telecomputing called "Business Applications for Expert Systems."

More generally, and closer to current work in IKBS, we see the conversational approach and network building as having some potential in ITS in specific domains. Fruitful areas would be identifying the general cognitive set of a student's knowledge and delimiting the set of cognitively real errors which subsequent empirical investigation would validate. It is also an approach worth considering when training researchers in cognitive modelling and knowledge elicitation techniques [Johnson and Johnson 1985].

References

- Bliss, J., Monk, M., and Ogborn, J.M., "Qualitative Data Analysis for Educational Research: A Guide to the Use of Systemic Networks," Croom Helm (1983).
- Hayes, P.J., "The Naive Physics Manifesto," in "Expert Systems in the Microelectronic Age," D. Michie (ed.), Edinburgh University Press (1979).
- Johnson, L. and Johnson, N.E., "Using Teachback Methodology for Interviewing," in "Knowledge Elicitation for Expert Systems: A Practical Handbook," A. Kydd (ed.), Plenum Press (in press).
- Johnson, N.E., "Elicitation and Representation of Children's Arithmetic Knowledge," Ph.D. Thesis, Chelsea College, University of London, (1983).
- Minsky, M., "A Framework for Representing Knowledge," AI Memo 306, MIT.
- (Michener) Rissland, E.R., "Understanding Understanding Mathematics," *Cognitive Science* 2, No. 4, 361 (1978).
- Ogborn, J.M., "Understanding Student's Understandings: An Example From Dynamics," *European Journal of Science Education* 7, No. 2 (1985).
- Ogborn, J.M. and Johnson, L., "Conversation Theory," *Kybernetes* 13, 7 (1984).
- Pask, G., "Conversation Theory: Applications in Education and Epistemology," Elsevier (1976).

Teaching Children AI

Yaakov Kareev and Judith Avrahami

The Goldie Rotman Center for Cognitive Science in Education

School of Education

The Hebrew University of Jerusalem

The educational system constantly faces the challenge of preparing today's children for life in the world of tomorrow. This world is certain to differ from that of today in the expected prevalence of intelligent machines.

Drawing on our experience in developing and teaching a computer literacy course for junior highschool students, we present an outline of an introductory course in artificial intelligence, designed to meet the challenge by making children aware of the capabilities of intelligent machines and of the way they work.

We aim the course at children of the age 14-15 years – an age at which most children still attend the public educational system and at the same time have already acquired the formal thinking necessary to benefit from the course. In this paper we present our notions concerning the contents of the course and the methods for teaching it.

Since the intelligent systems of the future are bound to differ from those of today, we consider it unnecessary to provide an exhaustive description of present-day systems. The course should focus on the basic ideas underlying such systems, and use state-of-the-art examples for illustration and increased motivation only.

Since human intelligence and its manifestations are the major source of basic ideas employed in artificial intelligence, we start by increasing our students' awareness of human mental activity.

Two additional steps are required to explain how such intelligent activity may be realized on an artifact:

1. The components of the relevant cognitive activity have to be expressed in information processing terms (e.g., input and output, storage and retrieval of information, searches, branching, comparisons and evaluations). Intelligent behavior is then explained as a process consisting of such components. Such an explanation can rely on meta-cognitive insights and introspection.

2. A demonstration of how the process formulated in the former stage could be translated into a working computer model. Naturally, the students should be somewhat familiar with a computer language and its capabilities.

Experience indicates that the course should not be taught through lecturing alone. To motivate our students and provide them with recent experiences as raw materials for the kind of analysis described above, we have them interact with actual computer programs bearing on the topics in question.

A typical lesson starts with a preliminary discussion of the mental operations involved in the activity to come, in order to prime the students to their own cognitive processes. Next comes the actual experience within the chosen AI program. In the discussion which follows, students are first encouraged to review the formerly discussed mental operations in view of their recent experience, and then to consider how these operations may be ordered and combined to form a well-defined, finite process, producing the observed behavior. The lesson ends with some hints as to how such a process could be translated into a computer program.

For an activity to be included in the course, it has to represent some AI area and to fulfill three criteria:

1. It should easily give itself to meta-cognitive and/or introspective analysis.
2. A working program representing the area should be available.
3. The activity should enable a clear and easy demonstration of at least one of the basic principles underlying the workings of AI programs.

Game playing can be used as an illustration of our approach. It obviously fulfills the first two criteria, and the mental activities involved in it can illustrate a number of basic principles:

1. Playing by the rules of the game, employing a strategy – usage of previously stored information.
2. Identification of meaningful board positions – top-down search driven by already stored information pattern recognition.
3. Appropriate response upon recognition of meaningful board positions – rules expressed as condition-action pairs.
4. Evaluation of alternative moves in the absence of forcing patterns – forward search, pruning of search trees, simultaneous consideration of different aspects to reach comparable values.
5. Learning from experience – new data added to already stored knowledge as a result of backtracking meaningful situations, the necessity of a higher-level control structure to decide when to start a “learning routine”.

Other activities, such as querying of expert systems, using intelligent computer aided instruction, adding knowledge to an existing system, or interaction with a robot, can all be used in a similar vein to introduce other basic principles of AI and to augment some of those already presented.

For the course to achieve its goals, students should also come to realize, by means of references appearing throughout the course materials, in which directions AI systems are expected to develop:

The variety of AI programs is expected to be larger, pertaining to new and hitherto untapped areas of human endeavor.

AI programs will consist of more primitive components, thus gaining more depth.

AI systems will contain more sophisticated control structures to direct and coordinate the variety of programs forming the complex systems of the future.

**Computer Microworlds and Reading
An Analysis for Their Systematic Application**

Robert W. Lawler and Gretchen P. Lawler
GTE Laboratories, Inc.
Waltham, MA

This discussion is a worked example of how learnability analysis can provide guidance for the design of computer-based educational materials. Learning can be seen as a consequence of problem solving in particular cases. It occurs when one achieves a solution which is able to be used later. "Anchoring with variation", first described by Tversky and Kahneman in 1974, is a common and important process because it provides a general framework through which one can discuss the basic situation wherein one tries to deal with something imperfectly understood in terms of what is already well known. Such a process is common for many people in various domains of problem solving.

Our purpose in the following discussion is to explore some possible implications of this process for reading education as a worked example of how educational technology presents us an opportunity for reconceptualizing instruction.

Our vision of the role of problem solving in learning to read relates that role to problem solving in mental calculation. By definition, pre-readers have limited ability to construct oral words from letter strings, although they may still recognize certain common words like "stop" and "exit". Further, they are able to recognize some individual strings of letter symbols as words through a very strong use of context dependency. We ask what could be anchors of variation for the decoding of alphabetic words. In general, these anchors are accidental and depend upon experience as do those for mental calculation. People with unusual experience may have surprisingly idiosyncratic anchors of variation. Computer-based microworlds introduce the possibility of creating materials through which a designed, non-accidental set of systematic anchors could be introduced to pre-readers. If it were possible to create a more nearly complete collection of anchors for the interpretation of monosyllabic words, we could change significantly the process of learning to read. We could make reading much more accessible to many more people in a relatively effective and congenial fashion.

As we turn to the systematic design of a set of reading anchors, our attempt will focus on the learning of whole words. If simple processes of monosyllabic modification can become richly productive in the child's everyday experience and if the child has a sufficiently rich repertoire

of well known words to modify, the invention of this procedure and its good development by every child should be expected. **Mastery of the phonological code will derive through variations from recognizing a very well chosen vocabulary of whole word anchors.**

Computer-based microworlds can be made which most children will find quite congenial. The essential reason is that the flexibility of computer systems makes it possible to design objects and procedures within these microworlds which are tailored to the specific experience of a young child. Using arbitrary symbol strings (which may be words) as the tools of control, a child is able to make objects appear and manipulate them in a virtual world that can be shaped to her own purposes. There is no reason why experience with a multitude of such microworlds cannot be based upon a vocabulary which will present as anchors for variation the most productive words of the language.

I propose introducing children to an increased number of whole words, distributed over a broad range of language sounds, by offering them experience with multiple, word-oriented microworlds at a very early age. Breaking words into an initial phonetic cluster and a syllabic residue would be worked out naturally using the vocabulary presented and established through experiences with these multiple microworlds. This would be followed by either the spontaneous or the directed reconstitution of other monosyllables through anchoring with variation. The key difference is the introduction of a multitude of monosyllables through computer-based microworlds.

The size of syllabaries in languages such as Indian Cherokee (85 signs) and African Vi (200 signs) indicates an order of magnitude for syllable-signs people can easily recognize and distinguish as significant components of written language. If we ask children to recognize 200 words, it's probably not too much. Recognizing a increased number of whole words, 200-500, at the age of four or five should not be uncommon. The systematic approach we propose will cover the phonetic range of English in such a fashion that it will both make the structure of the language more comprehensible and make it accessible to the flexibility of computer implementation. Choosing to deal with monosyllabic words in itself offers children instruction in what are significant units in the written representation of the language.

English has the potential for more than 60 thousand monosyllables. Exclusion of terminal consonant clusters brings the number of possible monosyllables down to a more manageable 15 thousand, known as the reduced space. This space is about 20 percent full; approximately 3000 such monosyllables actually exist in English. We've chosen to represent these monosyllables as an initial phonemic cluster plus residue. Within the 3000 monosyllables of the reduced space the number of residues is approximately 700. 113 of these appear in 10 or more monosyllables; 191 are in five or more. If children could recognize these two hundred residues, they would have a syllabic basis from which they could interpret 73 percent of the 3000 monosyllables in the reduced space. In this specific sense, these would be the optimally productive residues for pre-readers to recognize.

Many words do end with consonant clusters. We have attempted to answer the question "How many?" and to explore the organization possible to impose upon them through a residue-oriented analysis. We found out which consonants cluster together and generated the maximal list of existing English monosyllables. From this we deleted archaic, arcane, dialectic, vulgar, and insulting words. Sixty-six terminal clusters were found to form 4280 monosyllables in addition to those of the reduced space for a grand total of approximately 7000 words. There are nearly 1200 residues in that collection of 4280 monosyllables. The summary of the analysis is that among the nearly 1900 different residue spellings of English monosyllables, only 260 appear in more than ten words, and about 550 in more than five. The first number is clearly within the range of words knowable through that sort of associational memorization called upon within syllabic written languages. If children can learn as many as 500 or 600 different correspondences between sounds and spelling patterns, their knowledge of such residues, coupled with the ability to modify interpretations of letter strings by anchoring with variation, will cover a major portion of the phonetic-orthographic correspondences of the English language. We believe this extensive, concrete foundation of word and sound knowledge will permit children to read well enough that the instruction they will need will be primarily a refining and perfection of such knowledge.

The primary design conclusion is that if we create computer-based microworlds which use words with these level 10 and level 5 residues as the names for their entities and their actions, we will be providing a set of systematically generated monosyllabic anchors which promises to be highly effective for children's interpretation of many words they will encounter in reading English.

The potential revolutionary impact of such a prereading curriculum is worth exploring. A second possible significant outcome is that such experiences could alter the balance of the learner's perception of the orderliness of language. Experience with words that are archetypical in terms of the phonological correspondence may change the child's view of language to one of a rule-governed code with a lot of exceptions. We could make the orderliness more salient than the disorder. We could, in effect, change the perception of "figure" and "ground" in the child's early experience of the written language.

References

Kahneman, D. and Tversky, A. "Judgment Under Uncertainty: Heuristics and Biases." *Science* 185, 1124 (1974). Reprinted in "Judgment Under Uncertainty: Heuristics and Biases," D. Kahneman, P. Slovic and A. Tversky (eds.), Cambridge University Press (1982).

The Development of Structures Through Interaction

Robert W. Lawler
GTE Laboratories, Inc.
Waltham, MA

Vygotsky asserts that the internalization of socially rooted and historically developed activities is the feature distinguishing between human and subhuman psychology.¹ Piaget claims that the development of knowledge can only be understood through appreciating the history of structural changes deriving from experience.² Lewin argues that a central task for the cognitive sciences is going beyond the search for statistical norms of behavior to the detailed consideration of particular cases of individuals interacting with their milieu.³ I have studied the learning of one child in breadth and in detail through a period of six months and more.⁴ The general objective of THIS effort is to use detail of THAT study to address some of the cited themes in a more computational fashion than was attempted in Lawler '85. Some progress has been made to date, reported in "Learning Concrete Strategies Through Interaction" and "The Internalization of External Processes".⁵

The Structuralist Assumption

The structuralist assumption is in explaining saltations of performance. An outstanding example comes from recent work in modelling the evolution of vertebrate flight. Calpe, Balda, and Willis argue⁶ that insectivorous dinosaurs' use of forelimbs to stabilize their movement while running and jumping to snap at bugs can explain, in terms of positive feedback in a physical model, the development of those forelimbs into wings capable of the power stroke of flight. The theory measures good adaptedness with the criterion of "foraging volume", the space within

¹See especially Chapter 4 of "Mind in Society."

²This position, which permeates Piaget's work, is encapsulated in his paradoxical slogan: There can be no development without structures, nor any structures without development. See Piaget 1976 for an introduction to his ideas.

³See "The Conflict between Aristotelian and Galilean Modes of Thought in Contemporary Psychology" in Lewin, '35.

⁴See "Computer Experience and Cognitive Development," Lawler, '85.

⁵These essays are to be subsumed in a more ambitious work now underway with the working title "The Development of Structures through Interaction".

⁶As reported in "How Did Vertebrates Take to the Air?" by Roger Lewin in the Research News section, *Science*, July 1, 1983.

which the creature might capture bugs in a given time. A fast jumper that can stand on two feet will be better able to capture bugs from a larger space than will a slowpoke with four feet solidly on the ground.

Three different physical changes led to increases in foraging volume, bipedalism, increased running speed, and controlled jumping. Because of the physics of the world, limbs extended to control the trajectory during a high-speed jump experience lift. Since this lift increases with increased speed and since it also increases stability of the running creature, a positive feedback condition develops where more speed produces more lift and this in turn permits more speed. The creature takes off. Flattening of the forelimbs would increase lift, as would the effective extension of the proto-wing surface by the transformation of scales into longer feathers. Once in the air, those very limb movements which facilitate trajectory control, when coordinated, serve as the power stroke of flight. This story exemplifies the crossover in application of a structure with adaptive value for one function to some new function which provides a different adaptive advantage.⁷

The Search for Anterior Structures

Since structures exist in evolved physical things and the idea of co-adaptive evolution of structures is an effective form of explanation, it is not bizarre to explore the extent to which information structures may be argued to exist in the human mind and to inquire about how they might prove useful in mechanical minds. For those who may hold suspect such analyses as cognitive studies permit, I note that unanticipated structures have been found even in physiological experiments. Satinoff argues⁸ that the smooth temperature control performance of mammals is based upon the multiple integration of disparate temperature control systems characteristic of earlier developed life forms. I have argued that mature skills can arise from such small but significant changes in the organization of pre-existing, fragmentary bodies of knowledge⁹ which represent the things of everyday experience and operations on them. Piaget's "conservation" experiments are strong evidence that there exist organizations of knowledge about the world quite different from the collections of ideas held by most adults.¹⁰ Such studies raise to salience the issue of what precisely the structural precursors of mature performance might be. If only we could specify the character and function of antecedent structures, we could explain large scale behavior changes as saltations emergent from minimal internal organizational changes. Even if one does not believe in the essential reality of large-scale information structures in the human mind, the precise articulation of problems of learning (both natural and artificial) from a structuralist position gives us a language for discussing issues which may be beyond the reach or more atomistic descriptions.

⁷This is co-adaptive evolution of structures; the character of structures which names their re-applicability is "functional lability".

⁸In "Neural Organization and Evolution of Thermal Regulation in Mammals", *Science* 201, pp. 15-22, 7 July 1978.

⁹Chapter 2 of Lawler 1985, sets out such an argument.

¹⁰See, for example, "The Child's Conception of Number", Piaget 1952.

From Neophyte to Master

From a function-oriented structuralist view of human learning, a central problem is explaining the development of a person from neophyte to master. The specific case on which I am now working can characterize the general problem. A child has been observed beginning to play Tictactoe strategically¹¹ by imitating a three move plan for establishing a fork another child performed. The characteristics of her knowledge at that time were egocentricity and particularity. EGOCENTRICITY: she did not attend to the moves of her opponent unless they directly interfered with her single plan. PARTICULARITY: when her sole plan was blocked, she was unable to develop any alternative. How, from such a beginning, is it possible for a mind to develop powerful notions, such as the offensive use of forcing in achieving forks, and methods of mastery, such as a scheme of games classification which permits immediate prediction of victory for over half the games possible to play once the first two moves are specified? In this specific case, I believe the collection of processes for a computational model of such a path of development must involve at least these:

1. **Data structures which embody the minimum encoding of knowledge needed to perform actions learned from imitating others to achieve a recognizable goal.**
2. **Interaction-driven reflexive construction, which includes the concrete learning of:**
 - a. Specific constraints upon egocentric plans.
 - b. New goals recognized when surprises occur.
 - c. New plans constructed as variations of prior plans to achieve newly recognized goals.
3. **Redescription of goal elements of concrete data structures based upon inversion of bug descriptions: the essential use of such processes is the introduction of new terms into the descriptions of goals and plans.**
4. **Representing the role of "society" in the internalization of external processes: the constriction of the context of action, coupled with interior goals to continue activity, leads to functional re-application of structures developed for some other purpose to meet the challenge of the context-constriction; the importance of this process is in permitting the development of "virtual experience"; this amounts to the introduction of a Vygotsky-like view into modelling of mind.**
5. **Reflexive abstraction, a functional analysis of genesis, as exemplified in Bourbaki's description of the generality of axiomatic systems:**
 - a. "A mathematician who tries to carry out a proof thinks of a well-defined mathematical object, which he is studying just at this moment. If he now believes that he has found a proof, he notices then, as he carefully examines all the sequences of inference, that only very few of the special properties in the object at issue have really played any significant role in the proof. It is consequently possible to carry out the same proof also for other objects possessing only those properties which had to be used. Here lies the simple idea of the axiomatic method: instead of explaining which objects should

¹¹The study on which this description is based appears as Chapter 4 of Lawler 1985.

be examined, one has to specify only the properties of the objects which are to be used. These properties are placed as axioms at the start. It is no longer necessary to explain what the objects that should be studied really are...." N. Bourbaki;¹²

One use of such a process would be to explain the recognition that symmetry applies with power in the domain of Tictactoe.

6. **Reflexive classification, a feature oriented organization of things:** Piaget contrasts the former, reflexive abstraction, with classificatory or Aristotelian abstraction,¹² demeaning the latter somewhat by referring to it as "simple"; to the extent that classification is an automatic process, perhaps an outgrowth of such mechanisms of memory organizations as Schank¹³ and Kolodner¹⁴ have proposed, it must be distinguished from a consciously articulated application of such classification processes on a more abstract level. For example, a pattern-oriented classification of Tictactoe games into types based on the first two moves permits an easy to use scheme by which masterful play can be achieved without forward play in the tree of possible games. The challenge is to imagine how a consciously articulated process of such classificatory abstraction could arise from an "automatic" side effect of memory organization; Piaget's answer would be, of course, by reflexive abstraction. The attempt to build a model with such a set of characteristics is underway.

References

- Calpe, Balda, and Willis. Work reported in "How Did Vertebrates Take to the Air?" by Roger Lewin, *Science*, July 1, 1983.
- Fang, J., "Towards a Philosophy of Modern Mathematics," Paideia Series in Modern Mathematics, Vol.1, Hauppauge, NY (1970).
- Kolodner, J., "Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model," Lawrence Erlbaum Associates, Hillsdale, NJ (1984).
- Lawler, R., "Computer Experience and Cognitive Development," Ellis Horwood, Ltd., Chichester, England, and John Wiley Inc., NY (1985).
- Lawler, R. and Selfridge, O., "Learning Concrete Strategies through Interaction," *Proceedings of the Cognitive Science Society Annual Conference*, (1985).
- Lawler, R., "The Internalization of External Processes," unpublished draft.
- Lewin, K., "A Dynamic Theory of Personality," translated by Adams and Zener.
- Piaget, J., "The Psychology of Intelligence," Littlefield, Adams and Co., Totowa, NJ (1976).

¹²Cited in Fang, 1970, p.69.

¹²In "Biology and Knowledge", Piaget 1971, p. 320.

¹³"Dynamic Memory," Schank 1982.

¹⁴"Retrieval and Organizational Strategies in Conceptual Memory," Kolodner 1980.

Piaget, J., "Biology and Knowledge," University of Chicago Press, Chicago, IL (1971).

Piaget, J., "The Child's Conception of Number," W. W. Norton and Co., NY (1952).

Satinoff, "Neural Organization and the Evolution of Thermal Regulation in Mammals", *Science* 201, pp. 15-22, 7 July 1978.

Schank, R., "Dynamic Memory," Cambridge University Press (1982).

Vygotsky, L.S., "Mind in Society," M. Cole, V. John-Steiner, S. Scribner, and E. Souberman (eds.), Harvard University Press, Cambridge, MA (1978).

Model Building in Computing Education

B. Lees

Department of Computing Science
Paisley College of Technology
Paisley, PA1 2BE, Scotland

A particularly appealing feature of LOGO is the role that it can play as a modelling language [Howe 1984]. LOGO provides pupils with a friendly and powerful modelling environment with which to build models which may represent, for example, mathematical structures or physical systems. Its turtle graphics facilities may be used to advantage to give rapid graphical feedback to show clearly to pupils whether their procedures work; and if not, to give some indication of perhaps where the errors may lie. By this process, models may be built up incrementally through an iterative sequence of: model building, testing, debugging and extension. LOGO provides a good balance of powerful features and simple syntax in a friendly environment; its effectiveness as a powerful learning tool has been amply demonstrated in areas such as mathematics [Howe 1982] and physics [DiSessa 1980].

But why not adapt this model building approach to other subject areas – for example, computer science education? It is perhaps remarkable that relatively little software has been developed which is specifically designed to assist the learning of concepts in computer science itself. Consider, for instance, computer architecture. The aims in teaching a course in this area might be to develop an understanding of the principles of operation of computer processors, and the inter-relationships between hardware and software elements; also to investigate and compare the characteristics and operation of different machine architectures. Here is a possible area of study in which a model building approach may usefully be applied.

This paper will describe an interactive modelling system which has been designed to give practical support to an undergraduate course in computer architecture. An outline of the system facilities is given, after which some possible modes of use are suggested.

Computer Architecture Modelling System

One possible approach, which might assist in the realization of the aims noted above, is through the provision of an interactive computer architecture modelling system. Such a facility would permit students to construct software models of computer systems by the selection and inter-connection of components chosen from a set of basic building blocks (these building blocks have been designed to reflect the functional characteristics of corresponding hardware components).

An interactive modelling system, designed to meet the above requirements, has been implemented in Pascal to support the teaching of undergraduate courses in computer architecture. A fundamental decision in the design of the modelling system was: whether to extend an existing language, by the provision of a set of procedures to represent the building blocks; or whether to create a special purpose modelling language, for interpretation by a program written in an implementation language? It was decided to adopt this latter approach [Lees 1984].

With the modelling system it is possible to create and investigate working models of any conventional architecture machine. It can, for example, be used to model complete microprocessors, or to focus attention on one particular aspect of a processor. The system comprises three compatible model building modules.

Hardware Modelling

Firstly, a set of functional software building blocks is provided which represent the different kinds of logical elements in a computer processor. The building blocks are chosen so as to correspond to items such as registers, buses, adders and memory units. These building blocks may be regarded as templates, each of which represents the functions of a corresponding hardware component. Models are built by creating instances of the required building blocks and by specifying certain parameters for each block; for example, its identifier, wordlength and input/output connections. By this means it is possible to create a software model of any desired piece of hardware from a kit of parts, in a way that is analogous to the construction of, say, a Lego model. The actual process of model building proceeds either by using a tutorial dialogue (for beginners) or by means of a concise model definition language. The system checks the model for inconsistencies before setting up the required interconnections between components.

Microprogramming

Once constructed, such a "static" model may be brought to life by writing short microprograms which specify sequences of operations to control the data transfer and processing operations of the functional building blocks. This employs the second component of the system, in the form of a simple microprogram definition language. Three types of microinstruction are provided: to control the operations of model components; to enable branching within the microprograms; and to halt the execution of a microprogram. This system facility may be used, for example, to define and test machine language operations, and thereby enable machine language programs to be "executed" on the modelled hardware through interpretation by an appropriate set of microprograms (such machine language programs being stored in the modelled memory units).

Assembler Modelling

A third module of the system allows the modelling of assemblers, in order that the models may also be programmed at the assembly language level. This module enables the user to match the assembly language instructions, instruction formats and directives to the corresponding machine language binary codes, formats and assembler operations. Although it is intended that the

assembler modeller be used by students mainly to model relatively simple assembly language instruction formats, its scope is such as to allow the more sophisticated pdf-11 instruction formats also to be modelled.

Finally, to facilitate model building in stages, through an iterative sequence of model definition, testing, debugging and extension, an integral set of editing and file storage facilities is also incorporated in the system.

Discussion

The system offers sufficient flexibility to allow students to build their own models or, alternatively, to modify or extend models which have been pre-defined by the teacher. By this means, models of either existing computer systems or novel machine architectures may be set up and studied.

The system may be used by students to assist their understanding of computer systems concepts through practical model building activities. Students, may, for example, construct simple models of real or hypothetical computer systems which help to focus their attention on the fundamental principles of machine operations. Alternatively, models of the hardware of complete processors may be created. Once built, such models may then be programmed to model a subset of a machine language instruction set. An important aim in the application of the system is that it may help to foster an appreciation by students of the inter-relationships between the hardware and software elements of computer systems.

Acknowledgement

The author is very grateful for the continued guidance and helpful suggestions of Professor Jim Howe and Dr. Hugh Noble in this project.

References

- DiSessa, A.A., "Computation as a Physical and Intellectual Environment for Learning Physics," *Computer and Education*, 4, 67 (1980).
- Howe, J.A.M. et al., "Teaching Mathematics Through Programming in the Classroom," *Computers and Education*, 6(1), 85 (1982).
- Howe, J.A.M., "Model Building, Mathematics and LOGO," in M. Yazdani (ed.), "New Horizons in Educational Computing," Ellis Horwood (1984).
- Lees, B., "An Interactive Modelling System to Assist the Teaching of Computer Architecture," *Computers and Education*, 8(4) 419 (1984).

An Example Based Environment for Beginning Programmers

Henry Lieberman

Artificial Intelligence Laboratory
Massachusetts Institute of Technology

Making a good programming environment for beginning programmers is an enterprise which can exploit the strong connections between machine learning and human learning. Applying what we know about teaching and learning to improve the programming environment can result in a system which allows beginners to more readily acquire programming skills.

Surprisingly, a universally accepted principle of good teaching and good learning has not been taken seriously enough in designing programming environments – *learning by example*. A good teacher presents examples of how to solve problems, and points out what is important about the examples. The student generalizes from the examples to learn principles and techniques. This paper describes a programming environment called *Tinker*, in which a beginning programmer presents examples to the machine, distinguishing accidental and essential aspects of the examples. The programmer demonstrates how to handle the specific examples, and the machine formulates a procedure for handling the general case. Because people are much better at thinking about concrete examples than they are at thinking about abstractions, and because examples provide immediate feedback, *Tinker* is a more congenial environment for a beginner than conventional programming systems.

The Role of Cognitive Theory in Educational Reform

John Laurence Miller
York University and
The Clarke Institute of Psychiatry

My goal in this paper is to present a contrast between two intellectual universes, those of Piaget on one hand and Minsky and Papert on the other, as exemplars of two distinct points of view from which to approach child psychology, the theoretical and humanistic perspectives. Within Piaget's world, the child is seen as a living organism, perhaps more complex, but subject to the same biological laws as other organisms. The fundamental principles of cognitive growth, assimilation and accommodation, differentiation and coordination, and developmental stage, are biological. Furthermore, Piaget's basic research question, to identify precursors of human adult knowledge structures, is isomorphic with the problem in developmental and evolutionary biology to identify precursors of human adult organic structures. In contrast, Minsky and Papert see the child as upwardly rather than downwardly continuous, meaning like an adult instead of like an organism. The effect is to reduce the difference between adult and child, leading us to perceive children as more like ourselves and therefore more as our equals.

I will pursue this contrast in the context of the current interest in educational reform, by comparing a Piagetian approach to education, the concept of an educational "micro-world", developed by Lawler, with the LOGO language and the theory of Turtle Geometry, developed by Papert and his colleagues. I believe on theoretical grounds that the Piagetian approach ought to be more successful in developing originality and technical excellence and that the humanistic approach ought to be more successful in teaching intellectual breadth and independence of thought. Our final conclusion will therefore be that further research is needed to clarify the effect of educational methodology on the development of intellectual qualities such as these.

Cognitive Diagnosis, Pedagogical Methods, and Instructional Actions in Teaching Understanding of Rational Numbers

Stellan Ohlsson
University of Pittsburgh

It is argued that the major promise of computer tutors resides in their potential for moment-by-moment adaptation to the cognitive needs of the student. This view puts emphasis on cognitive diagnosis and on flexibility of pedagogic decision making, rather than on task analysis or on design of user interfaces. Furthermore, this view implies that a computer tutor should be capable of a wide range of instructional actions. A review of *cognitive diagnosis* shows that (a) Experimental Psychology has largely ignored the problem, (b) expert classroom teachers do not, in fact, base their instruction on diagnosis of their students, and (c) that current computerized techniques for diagnosis are either too simple minded, as in so-called overlayed models, or too ambitious, as in the DPF system by Ohlsson and Langley. The implications of these facts for computer tutoring are discussed. A review of *pedagogical methods* reveals several different images or metaphors for teaching: teaching as problem solving, teaching as supervision, teaching as an expert skill, teaching as communication of knowledge, etc. The implications of these diverging models of teaching are discussed. A review of *instructional actions* reveals that classroom teachers employ a very wide range of different types of instructional devices, implying that current computer tutors are too narrow in their action capabilities. A summary of taxonomy of instructional actions is presented.

The above considerations are applied to the design of a computer tutor for teaching children to understand rational numbers ("fractions"). The arithmetic of rational numbers is characterized by simple algorithms in combination with difficult and complex conceptual content. It is therefore an appropriate domain in which to attempt tutoring of mathematical understanding. The basic pedagogical method explored in this tutor is to teach the mathematical theory by providing a concrete interpretation of it, i.e., a set of physical objects which behave like the mathematical entities when manipulated according to certain rules. In addition, we hypothesize that manipulation of multiple interpretations of a mathematical theory will facilitate learning of that theory. This pedagogical method is implemented in a computer tutor which features intermediate level diagnosis and a wide range of instructional actions. The main design decisions with respect to the tutor's control structure are (a) to combine a goal-tree with a linear priority-ordering over the leaves of the tree, (b) to divide the rules of the pedagogic strategy into goal-specific rules, which apply only when the relevant goal is active, and general rules,

which can apply at any time, and (c) to let rules vote for action alternatives, rather than control behavior directly; the rule interpreter resolves conflicts between action alternatives by counting positive and negative votes. We hypothesize that these devices will allow us to combine local flexibility with global direction of the tutoring effort.

Searching For New Microworlds

Martin Owen

Department of Mathematics and Computing
Roehampton Institute of Higher Education

"In my vision, the child programs the computer and, in doing so, both acquires a sense of mastery over a piece of the most modern and powerful technology and establishes contact with some of the deepest ideas from science, from mathematics, and from the art of intellectual model building."

— Seymour Papert, *Mindstorms*

Probably the seminal and most widely read book on the subject of computers and learning is Seymour Papert's *Mindstorms*. In this book Papert makes strong claims for the virtue of the use of programming to explore ideas. In the book he describes constrained computer environments called microworlds which embody sets of ideas. These microworlds are best described as a set of programming constructs which come from a programming language and constraints in the form of programs composed for a particular discipline. Using the programming constructs the child explores the constraints.

In *Mindstorms*, Papert describes three computer microworlds in some detail. The principal microworld is the world of the TURTLE which is a microworld used to introduce the programming constructs of LOGO, i.e., a microworld about microworlds. More recent classroom uses of LOGO have often focussed on this aspect of LOGO, to the exclusion of other uses, and have used it to teach some concepts of geometry. The other two computer microworlds described refer to the comparison of Newton's and Aristotle's mechanics in which turtles take on dynamic properties and respond to forces with the property of velocity. The other microworld relates to the structure of sentences. My work as part of the MEP Capital Region/IBM "PC as a Whole School Resource" Project is to develop exemplars of other microworlds.

Papert suggests that this microworld approach has two beneficial methods of operation. Firstly the computer provides the means of externalizing intuitive expectations about phenomena. When these intuitions are made available as programs they become more accessible for testing and reflection. Secondly the computational ideas can be taken up as materials for the process of remodelling. Exploration, and the gaining of understanding in science, is not an exercise in finding correct answers but resolving conflict between observed phenomena and our interpretations of them. This often means taking a mental journey from intuitive understandings of

a situation to the formalization of our knowledge of the situation. The interactive capabilities of the personal computer and the formalisms of a computer program combine to provide students with a marvellous tool for that exploration. There are some interesting analogies from other spheres of education. In art, a greater understanding of technique in representing nature comes from a better understanding of nature; also a better understanding of nature is gained by developing techniques to represent nature. Further, one of the benefits of an essay in education is to make the student externalize his understanding of the topic as a means of coming to understand.

There is also a resonance here with ideas expressed about the way people learn. Learning is viewed as a process of building increasingly sophisticated models from our universe of experience, externalizing them, and attempting to reconcile them with the experiences of our senses and of our fellow human beings.

There is implicit in the LOGO philosophy the notion of an "object to think with," and the project has attempted, wherever possible, to offer "objects to think with" within the microworlds.

Thus far the project has explored the following areas:

Turtles in Space: a microworld in which the turtle can take properties of mass, velocity, and charge as can another object, the screen, which can act as a force field which can also have the properties of epicentres. This microworld offers numerous opportunities in the study of physics [see Owen 1985a].

Toucan Music: another possibility for an animal in the cybernetic zoo. The creature plays a keyboard on either screen or (yet to be developed) in reality. Various timbres can be provided to the creature analogous to SETPC. This is not performance music but a tool for exploring ideas in music such as tonality, harmony, tempo, rhythm, and transposition. The microworld also has some virtue in being body syntonic with playing the glockenspiel [see Owen and Harrison 1985b].

Ferret's Burrows: hunting and sorting and storing information. An attempt at a curriculum for developing a computer information processing curriculum that is analogous to the programming curriculum of the turtle. The microworld encompasses set manipulation and develops to logic programming [see Owen 1985c].

Turtles Down on the Farm: a case study in modelling in the humanities providing tool kits for displaying information and linking in with the information processing procedures [see Owen 1985d].

Turtles Through the Looking Glass: composition and language with LOGO. An integrative microworld which looks on the ways that the microworlds used and developed by students can be further used to tell stories. As well as some direct explorations in language processing in LOGO using sentence generators and ELIZA-like dialogue systems the position of the turtle on the screen can be predicated (like positions on a chess board)

for the turtle to go on an adventurous journey, the plot of which is both interactive and the summation of many students' work [see Owen 1985e].

The "case study" models developed in this project can reflect two methods of operation. In the first instance, they are a case study of what might be achieved by students who have learnt LOGO earlier in their education. They would exercise and refine their knowledge by developing simulations themselves. Executing the programs verifies their understandings. Alternatively the models themselves offer curriculum materials for the exploration of the ideas in the particular domain. These models are transparent to the user, available for manipulation and extension under LOGO program control using either a command mode (possibly using trace facilities which explain the program/model during its execution) or within LOGO programs which extend the application of the model, e.g., in the ATOM model (Turtles in Space) Rutherford's alpha particle scattering experiment may be simulated as a computer program using the student's own procedures and the procedures provided. The two methods are not exclusive, and it is entirely possible that students could be given more or less of the constraints of the microworld or be asked (or hopefully decide for themselves) to program in the constraints of the model themselves.

Teaching in this way presents problems however. How does it fit into a timetable and exam syllabus? Without wishing to dodge the issue, one could cite similar responses in the past to the introduction of syllabi which demanded more practical work in science teaching. It was felt that it is better to know less detailed fact and to possess a better understanding of the concepts and processes of science. Educational innovation always pays some cost in the loss of what has gone before. If one examines "Nuffield" O level examinations in physics there is less requirement for students to recall the formal experiments to prove fact X or to explore quite so much detail in the study of heat, sound, or optics. Recently more conceptual approaches are evident, with an emphasis on concepts like waves or energy. Thus there has been a tendency in a direction which could support a microworld utilizing methodology.

Managing the work requires a teaching style which can cope with less coordinated activity than may be found in a traditional classroom. Students will have independent, and perhaps false, perceptions. The teacher needs to be sensitive to the students' need to explore and yet be aware of the need to sustain motivation and avoid major pitfalls. Teachers who have been developing the skills associated with project may well have something to offer here, although the literature and support are very sparse.

The problem of examination and assessment is also less insurmountable than may be imagined. The sciences at A level and at C.S.E. often include an element of assessment based on project work which replaces the more traditional set practical. I do not wish to suggest that the recorded use of a computer model should replace tests of practical skills; rather, the very fact that practical skills are assessed serves as an exemplar that learning and understanding in science is already assessed by means other than formal written examinations.

I have no evaluation reports to offer. As more powerful computers enter the schools I hope that ideas such as these will be tried. As an educational innovation the use of microworlds as methods of exploring complex ideas is open to the evaluation of their users. They do not present a neutral option for introduction in the curriculum, they require a degree of commitment. They involve a commitment to the teaching of programming skills (or the use of previously acquired skills) and the will to let students explore.

References

Owen (1985a-e). "Occasional Papers of the Department of Mathematics and Computing," RIHE (some forthcoming).

How People Read Physics Problems

Tony Priest
Oxford Polytechnic, U.K.

Experimental Data

Studies of physics problem solving have shown that:

1. Problem solvers alternate between reading and solving activities. They do not read all and then solve all [1].
2. Experts can often guess the relevant solution principle before they read all the questions [2,3].
3. Some people re-interpret questions that refer to impossible situations so that they describe possible ones [4]. This can take place before any equations are formed.
4. Both experts and novices draw sketches of problem situations [5].
5. Problem solvers often re-read parts of the question [1].

Explanation

The paper reviews current theories dealing with this data by Scanlon, Hawkridge and O'Shea [6], Chi, Feltovich and Glaser [2], and also McDermott and Larkin [5].

Three hypotheses are proposed in order to explain the data:

1. The Hidden Curriculum Assumption of Non-Redundancy.
2. Pragmatic Interpretation Checking by Sketch Construction.
3. Adjacency Checking by Sketch Analysis.

The hidden curriculum assumption of non-redundancy is the assumption that all the quantities mentioned in the problem are necessary for its solution. This allows problemsolvers to select plausible physical principles to use as they read the question [7]. If a number of quantities are mentioned, and only one principle relates quantities of this type, an experienced problemsolver will hypothesize that this principle will be made use of in the solution. This explains how experts are able to guess the solution principle to a problem before having read in the whole question. If we assume that problemsolvers note down plausible principles as they are recognized, we also have an explanation for the alternation of reading and problemsolving activities.

The motivation for constructing a list of plausible solution principles is that in most cases it will greatly reduce the amount of search needed before a problemsolver finds an appropriate principle to use.

A sketch will be taken in this paper as being an arbitrary metric instantiation of a problem description, either external or mental. It performs a dual function in the problemsolving process.

Firstly, it enables the natural language interpretation to be checked for consistency.

If the problem description, as understood by the parser, is contradictory or physically impossible, it will be impossible to draw a sketch of it. This can be used as a check on the output of the parser, and failure of the sketching mechanism can be used to indicate the necessity for an alternate parse of the question. Such a process is sufficient to explain Paige and Simon's [4] finding that some problemsolvers rejected a correct parse of a problem describing an impossible situation. Allwood [8] claims that experts undertake more deliberate checks on their actions than novices – if the sketch is thus utilized for checking, it would explain why experts draw sketches as much as novices do.

Secondly, the existence of a sketch enables a problemsolver to answer questions of the form "What physical quantities are related to object X?", by looking only at the sketch representation of X.

This permits local identification of such things as forces acting on a body. In a natural language representation, or a coded form thereof, the descriptions of different forces acting on the same body may appear at widely different points, and thus cannot be identified without seeing the whole problem representation.

Larkin [9] states that the commonest error of novice problemsolvers is to omit a relevant force acting on a body, and therefore any mechanism which reduces the likelihood of such an error is clearly useful.

Conclusion

The process of reading and representing a physics problem is a complex one, involving other processes than natural language understanding.

Problemsolvers make use of arbitrary sketches and meta-knowledge about the nature of textbook problems to check problem consistency, reduce search, and facilitate the identification of relevant quantities when using a physical principle.

References

1. Scanlon, E., Hawkrige, C. and Evertz, R., "Modelling Physics Problem-Solving: Production Rule Models for Two Problems," Open University CAL Group Technical Report No. 39.

2. Chi, M., Feltovich, P. and Glaser, R., "Categorization and Representation of Physics Problems by Experts and Novices," *Cognitive Science* 5, p. 121-152.
3. Hinsley, D., Hayes, J. and Simon, H., "From Words to Equations - Meaning and Representation in Algebra Word Problems," in "Cognitive Processes in Comprehension," edited by Just and Carpenter.
4. Paige, G. and Simon, H., "Cognitive Processes in Solving Algebra Word Problems," in "Problem Solving," edited by B. Kleinmutz.
5. McDermott, J. and Larkin, J., "Re-Representing Textbook Physics Problems," *Proceedings of the 2nd National Conference of the Canadian Society for Computational Studies of Intelligence*.
6. Scanlon, E., Hawkrigde, C. and O'Shea, T., "Modelling Physics Problem Solving: Scripts and Production Rules," Open University CAL Group Technical Report No. 36.
7. Priest, A., "Inference Strategies in Physics Problem Solving," *Proceedings of the 1985 Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour (AISB)*.
8. Allwood, C.M., "Error Detection Processes in Statistical Problem Solving," *Cognitive Science* 8, pg. 413-437.
9. Larkin, J., "The Cognition of Learning Physics," Applied Cognitive Psychology Paper No. 1, Department of Cognitive Psychology, Carnegie Mellon University.

Is the AI Approach Sufficient to Study How People Gain Complex Knowledge?

Renan Samurcay
Centre d'Etudes des Processus
Cognitifs et du Langage

The main question for us is to study the learning, and therefore the transmission of, complex scientific knowledge such as mathematics, computer science, physics, etc.... What kind of elements does the AI approach give us to understand this very complex process?

In this presentation we will try to show why it is impossible to answer this question if we don't integrate a cognitive, epistemological and didactical approach to teaching and learning into our theoretical framework. We will illustrate our purpose on a particular domain, the learning of programming by novice students.

The kind of knowledge we want to study has two main characteristics:

1. It is institutionally taught and most often cannot be learned spontaneously;
2. It is cognitively constructed by the pupils, and this construction is a very long process.

In this context the theoretical concepts and methodological issues which we use to identify, analyze and describe the learning process of pupils become very important and determine the ultimate success or failure of our teaching efforts. For instance, consider the acquisition of the concept of variables, which is essential in learning programming. The cognitive and epistemological analysis of this concept necessitates that the variable be defined as an *operational invariant* which should be constructed by pupils during the long process of learning and in different teaching situations. The notions of *representation*, *analysis of situations*, and *didactical contract* are then essential to analyze behaviors observed in problem-solving.

The conceptual framework proposed by AI approaches (general mechanisms of learning, learning by analogy, the independence of the knowledge representation from the specific content of domains...), in its actual state, does not allow comprehension of the complexity of the teaching situations in which three partners interact: the pupil, the teacher and the knowledge. Moreover, this interaction is specific to this particular knowledge.

The analysis of some teaching situations for learning programming will allow us to illustrate the theoretical framework described above. In particular, we will focus our attention on the

acquisition of the concept of variables by pupils belonging to different age groups. We will try to answer questions such as:

1. What are the different activities that can be related to such-or-such aspects of this concept?
2. What did the pupils learn? How can it be observed and verified? What are the criteria of the acquisition?
3. Was the class of problems chosen the best for teaching objectives? How can they be modified for the new experiences?

Expert Systems and CAL Design

P.J. Scott and R.I. Nicholson

Department of Psychology

University of Sheffield

One major problem for CAL is the provision of good quality software in the range of applications required for any routine classroom use. Wasserman and Gutz (1982) concluded "Software design and development is the weakest link in the system development process. It has become extraordinarily and prohibitively expensive and remains unpredictable in terms of economically and dependably producing programs that run reliably, correctly, and efficiently". Recent approaches to CAL production (e.g., NDPCAL, CIC, Five Ways, ITMA Project) vary considerably in detail, but concur on the need for a team effort, requiring the input of teachers, programmers and "CAL designers" who are responsible for ascertaining the teachers' requirements and communicating them to the programmers in the form of a "program specification". As Watson (1983) remarks "...the successful development of any unit depends heavily on the CAL design skills of certain key people" [p. 175].

Clearly, we must streamline the program creation process. Current theoretical approaches to CAL development attack the problem of automating the teaching component. The insight of Expert Systems research, and the key to its commercial success, is that the expert modelled by the system should be a scarce and expensive resource. Though teachers may well exhibit a high degree of teaching skill, they are in plentiful and underpaid supply. No, the scarce resource in the CAL production team is the "CAL designer", and the research reported here demonstrates the outstanding benefits conferred by automating the design process.

Two decisions had to be taken early on: what type of CAL design to model first, and how to represent the knowledge entailed. Most CAL is still of an "instructional" form involving a series of questions to which there is one and only one correct answer, and this is undoubtedly the easiest form of CAL to model. Analysis of existing instructional CAL programs indicated that most program requirements could be represented within a general "script" (Schank, 1975) made up of "frames", each of which might include further frames, primitives, or variables. For instance, the frame "present one question" might involve the primitive "clear screen", the frame "set up window for question" (whose effect would depend upon whether the variable "question type" was set to "text" or "graphics"), and the frame "cue answer". The power of the script representation is that it is completely specified once all the variable "slots" have been filled, and that it is possible to specify default values for all the variables. Consequently,

the user need only give a few details to arrive at a preliminary specification. Our CAL design program is based on a script written in PROLOG and runs on a Bleasdale micro running Poplog under UNIX. It interacts with the user to derive a program specification in the form of an instantiation of all the variables in the script.

This program specification is language independent, and could be used as the basis of program code in almost any language on any micro. However, in order to demonstrate its sufficiency, we decided to implement the script in BBC Basic. We wrote a utility procedure corresponding exactly to each primitive and thus were able to represent each frame in terms of procedure calls, passing the variables as arguments. The PROLOG designer program selects the relevant frames on the basis of the specification, then assembles the necessary utilities from a BBC Basic library. The assembled code is then directed to a terminal working in BBC mode and the program is run automatically (normally using dummy questions selected from a question library). Following this initial trial, the user may then adjust the script and specify the questions required. The end result of this is that a teacher can take full control over program development, and the specify-encode-refine process that would have taken several months may now be achieved in an hour or so.

References

- Wasserman, A.I. and Gutz, S., "The Future of Programming," "Instructional Software," Walker and Hess, Wadsworth Publishing (1984).
- Watson, D.M., "A Model for the Production of CAL Material," *Computers and Education* 7, pp. 167-176 (1983).

The Application of Machine Learning to Student Modelling

John Self

Department of Computer Science
University of Lancaster

This paper considers the prospects for applying machine learning techniques to the problem of maintaining the dynamic student models needed for intelligent tutoring.

To be concrete, we will imagine a learner endeavoring to understand a climate classification scheme through exploring a database containing details of some 1500 weather stations all classified into one of sixteen types (Ar, Am, etc.). To clarify the potential role of machine learning ideas, we will assume that we aim to build a tutorial system to guide the learner's exploration but that this system does not itself know the classification rules (if they exist).

This is clearly a concept learning task, for which many existing machine learning programs might be thought appropriate. We will consider the focussing algorithm since this provides the most general framework.

First of all, there are several limitations of focussing which cause difficulty even with this somewhat idealized task, namely: the discovery of the lattice on which focussing is based, the handling of disjunction, the role of probabilistic learning, and the learning of formulas (as in the BACON series). More serious is the problem of the learner's background knowledge. Machine concept learners tend to assume that background knowledge has no role: this assumption is clearly false for human learners. They bring to bear both general knowledge (e.g., given an array of figures, the maximum and minimum might be particularly important) and specific knowledge (e.g., they may already have a rough idea of a "desert" and may seek to identify this with one of the given types). Indeed, without this background knowledge no human learner would be motivated to explore the database at all!

To apply focussing to our problem we must try to take account of what is known of human concept learning. Most machine concept learning work is of a rigorous, abstract nature more akin to the psychologists' "hypothesis theory" discarded ten years ago. Their more recent "prototype theory" has had little impact on machine concept learning (unless one counts Lenat's work, the unrigorous nature of which causes problems of its own). Some of the difficulties with machine concept learning are an artifact resulting from over-formality, and there may be advantages (certainly for our purposes) in relaxing this formality.

We also consider the possibilities for collaborative concept learning (as indeed was proposed by Lenat, but not pursued). Even if we stick to the limited but relatively well-understood focussing algorithm, various techniques are possible. First, we must escape from the "instance presentation" paradigm: learners should be provided with query languages to enable them to ask pointed questions, for example, to clarify boundary conditions – "What is the lowest monthly rainfall for any climate of Type Ar?" A tutor with an adequate student model and theory of learning may be able to monitor the potential appropriateness of such questions, or in fact to suggest alternative questions. This would require some theory of Socratic teaching along the lines begun by Collins. Maybe, if machine concept learners are effective at learning concepts there is scope for explaining their own learning strategies to human learners, and so for developing meta-level skills, which is usually the real educational purpose.

Finally, we will assess the state of machine learning approaches to procedural learning and the possibility of intelligent tutoring for procedural tasks.

Observational Learning by Computer

Mallory Selfridge

Dept. of Electrical Engineering and Computer Science
University of Connecticut

Donald Dickerson

Dept. of Psychology
University of Connecticut

What is Observational Learning?

Observational learning is learning that occurs by observing the behavior of others. A great deal of children's learning is of this type (Bandura, 1977). For example, a child observing a parent's activity and its results would later be able to perform the same action or activity. This implies that the child has discriminated an observed activity from irrelevant input, remembered the observed activity and is able to translate his memories into behavior. This paper considers Bandura's (1977) account of observational learning, and presents an alternative approach. This approach is implemented in a computer program that learns a new activity through observation.

Bandura's Theory of Observational Learning

Bandura (1977) describes a theory of observational learning based on empirical research on children. Their theory proposed that observational learning could be explained by a four-part process. First, the child attends and perceives the appropriate activity. Second, the child retains a memory of his perception of the activity. Third, the child can use the retained memory to generate a behavioral imitation of the activity. Fourth, the child observes the consequences of the attended activity, and uses this to index the retained memory.

Bandura proposes that the child employs imagery to represent the observed activity. An imaginal representation of an observed activity consists of a sequence of images of that activity. Bandura proposes that an image sequence is produced by processes of attention and perceptions, and stored in memory for later use in imitation. This view implies that imitation occurs when the child has a goal-image that matches the final image in a stored sequence. To imitate, the child retrieves the stored sequence and somehow transforms the image into behavior.

Several aspects of Bandura's theory are open to question. First, Bandura does not describe how children know what aspects of the observed activity should be attended to and what aspects are irrelevant. Second, he proposes that an observation is remembered as a sequence of images and

"abstract representations," but does not specify the details either. Third, he fails to address the problem of producing behavior from image sequences or abstract representation.

Observational Learning as Plan Recognition

The approach to observational learning presented here is based on plan recognition techniques (Wilensky, 1983). It addresses the problem of attention and thus can learn from observations containing irrelevant information. It includes a detailed symbolic representation for observed and remembered activity, and a detailed proposal for mapping from observation into representation. Finally, it incorporates a specific mechanism for producing behavior from a remembered representation.

Our approach is based around the notion of goal-driven behavior. When a child has a goal to achieve, he searches a knowledge base of goal-plan information, and retrieves a rule. This rule specifies a plan to achieve that goal; plans consist of sequences of either subgoals or direct motor action. This process of reducing higher level goals to sequences of sub-goals and motor actions continues until a sequence consisting purely of motor actions has been obtained, at which point the child can perform the motor actions to achieve the original goal.

Observational learning occurs when the child observes an activity and tries to "explain" it in terms of higher level goals. We propose that an activity is represented as a sequence of "snapshots" of the activity, and that this sequence represents the observation at its lowest level. To explain the activity, a child searches his knowledge of goal-plan information to see if a sub-sequence of snapshots matches the plan to achieve some goal. If so, the child has explained the snapshots as being intended to achieve the goal, and replaces the snapshots in the representation by the goal. Since the goals so inferred can be subgoals of higher goals, the process of examining elements of the observation and inferring their underlying goals continues until no further inferences are possible. At this point the child has explained the observation in the most abstract terms possible, as a sequence of high level goals.

Learning occurs when a child does not know a rule to achieve a goal. If he observes a demonstration of an activity resulting in that goal, he builds a new planning rule whose goal is his original goal, and whose plan is the sequence of high level (sub) goals which explain the observation. This rule is now available for planning. If the child later has that goal again, he will be able to retrieve a rule to achieve it, and will thus imitate the original behavior. Note that this process is hierarchical, in that learned rules can be used to help explain later observations, and ultimately behavior of considerable complexity can be imitated. Irrelevant facts within a given snapshot are ignored as a side effect of plan recognition. Since plan recognition involves inferring a certain goal whenever the plan components of a planning rule match snapshot components, irrelevant snapshot components are simply eliminated.

Observational Learning by Computer

The approach described above has been implemented in a computer program which learns by observation in the domain of small-scale industrial assembly (Selfridge and Levas, 1983; Levas

and Selfridge, 1974). This domain is relevant to the domain of child learning because small scale assembly has much in common with the motor activities of children. This computer program is written in FranzLISP on a VAX 11/780, and accesses two small two five-degree-of-freedom manipulators via an Apple II+ microcomputer. It learns to assemble small electric motors by observing assembly demonstrations by a user. It can later correctly assemble the motor, in effect imitating the original demonstration.

The program observes the user demonstrating an assembly by manually controlling the manipulators to carry out the assembly. The system "observes" the demonstration by monitoring the manipulator configurations and geometric descriptions of workspace objects to build a description of the demonstration such as would be returned by a high-performance vision system. It infers the plan underlying the demonstration by applying plan-recognition techniques to the description of the demonstration, in effect planning in "reverse". It stores the inferred plan as a plan-goal rule, and subsequently uses it to perform the assembly plans in the presence of multiple snapshot components. Its attention is focussed on the relevant areas when the plan recognition process compares the current observation against the plan part of a planning rule and ignores snapshot components which are not found in the plan part.

Conclusions

The success of this program suggests that the underlying theory is a plausible account of observational learning. In particular, it provides a detailed model of a possible mechanism underlying observational learning which may be open to empirical validation. It also addresses issues not covered by Bandura. First, it proposes that the plan recognition process provides attention focussing as a side effect. Second, it proposes a particular, specific representation for the membered observation as goal-indexed planning rules. Third, it suggests that remembered observations are translated into behavior by a planning process which uses the learned planning rules.

References

- Bandura, A., "Social Learning Theory," Prentice-Hall, Englewood Cliffs, NJ (1977).
- Selfridge, M. and Levas, A., "Teaching Robots by Example," *Proceedings of the 13th International Symposium on Industrial Robotics*, Chicago, IL (1983).
- Levas, A. and Selfridge, M., "A User-Friendly High-Level Robot Teaching System," *Proceedings of the IEEE International Conference on Robotics*, Atlanta, GA (1984).
- Wilensky, R., "Planning and Understanding," Addison-Wesley, Reading, MA (1983).

The Inference and Use of Student (User) Models

Derek Sleeman, Haym Hirsh and InYung Kim
School of Education and Knowledge Systems Lab
Stanford University

For a decade or more the importance of having accurate student models to guide Intelligent Tutoring Systems (ITSs) has been stressed. I will give an overview of the several types of models which have been inferred and will talk in some detail about PIXIE which uses process-orientated models and a system which infers overlay models. Currently, all these techniques effectively determine whether the current user's behavior falls within a previously defined model-space. The focus of some current work is to see whether these techniques can be extended to be more data-sensitive. That is, whether highly "original" behavior can be modelled. (Some examples of highly "creative" behavior will also be given.)

Finally, issues which arise in the use of models to control (remedial) dialogues will be discussed.

Can We Build a Mathland

Cynthia Solomon
80 Ellery Street
Cambridge, MA 02138

In exploring possible relationships between people and computers I decided to look at the computer in two ways: as an interactive textbook in control of the user and as an expressive medium under control of the user. Within these two frameworks, there are many interpretations of how computers and people interact. I recently studied four contrasting applications of computers to elementary mathematics education. In coming in touch with why Pat Suppes chose drill-and-practice as a teaching strategy I gained insight into his theories of learning and his view of what mathematics is. A very different interpretation of the computer as a textbook was offered by Bob Davis's use of the Plato system, in which he adopted a Socratic style of teaching to present Madison Project materials. In looking at the computer as an expressive medium, I wanted to understand more about people who used BASIC and so probed some of Tom Dwyer's work. He wanted to give people an immediate opportunity to take advantage of the computer's potential power and so opted to use easily available materials. He used BASIC and whatever devices were accessible and easily attachable to his computer systems. Finally, I took a more structured look at Papert's thinking.

Although Suppes, Davis and Dwyer propose three different computer-based learning environments, they seem to share a common view of the content of school mathematics: that it consists of arithmetic, algebra and geometry, and that it is teachable. Papert has a different view of the content of present-day mathematics. He sees most of present-day school mathematics as denatured and alienating and outside of a child's concerns. He envisions the computer as a "Mathland" where the computer becomes an instrument for children to talk in mathematics about their everyday life experiences and where children learn mathematics as naturally as they learn to speak.

Papert defines mathematics in its broadest sense. Mathematics is a source of powerful ideas resulting in rigorous, logical thinking with applications to all activities – to the whole person and to the whole environment. Mathematics is not a collection of facts or techniques, although facts and techniques are by-products. Mathematics provides ideas which help people to think about their lives, to organize knowledge, and develop socially, emotionally and intellectually. This "holistic" view of mathematics is supported by the computer's presence.

Papert sees learning as a constructive process. He believes that one of Piaget's most important contributions is not that there are stages of development, but that people possess different theories about the world. Children's theories contrast sharply with adult theories. Another important and related idea from Piaget is that even babies have theories and these theories are modified as the child grows. How are these theories transformed? What experiences and knowledge lead children to change their theories? Why do they learn some things without formal instruction and not learn other things despite formal instruction? These and other such questions motivate Papert's research.

Papert believes children learn best when they are encouraged to draw upon their own intuition and put to use what they already know in developing new ideas. He sees the computer as providing a context in which this kind of learning can happen. For this to happen the computer's assets have to be molded to the child's need and a new kind of mathematics has to be invented which coincides with natural developmental learning processes. Papert has been actively engaged in pursuing this goal.

His vision of computers emerges as many images each linked to one another through an underlying philosophy of creating conditions for people and computer to form relationships with one another which enhance people's sense of themselves and their own self worth, and thus, putting them in touch with their own abilities to learn and develop.

The kind of mathematics Papert has in mind for elementary school children is thus constructive and intuitive rather than rule-driven and formal. This mathematics builds on unique characteristics of computer technology and science. One example is turtle geometry. Papert's geometry differs from Euclid's as radically as most of the modern geometries do and thus differs from present school practice. It is a modern mathematics which could not exist prior to computers; it is a computational geometry. But it differs in other ways from Euclidean geometry. It is an investigation into object construction; using descriptive methods it draws on personal experiences and personal interactions with the objects. Its descriptive power is dynamic and process oriented. It draws on informal methods and heuristics based on personal knowledge. Mathematical ideas, instead of being encountered formally are first introduced as entities like people with their own peculiarities. This approach to mathematics follows within a tradition of intuitionism and constructionism.

Although I believe that Papert's Mathland is possible, I also believe we are far from achieving it or preparing teachers and parents to let children thrive in such a land. I would like to suggest other areas of research and other computational developments that need to take place.

I will discuss ideas from LOGO and show how some of these ideas formed the basis for research that goes beyond LOGO environments currently available on school microcomputers.

**Enriched Learning Environments on a Microcomputer:
Towards Feasibility in the Instructional Use of Expert Systems**

Jesus Vazquez-Abad
Department of Education
Concordia University

Reai Larose
Department of Educational Technology
University of Montreal

In previous papers [1,2] we have discussed the concept of an Operational Learning System (OLS) as a technology capable of providing a learner (or Natural Learning System) with the structuring information that will permit it to function in a new, more complex situation. Briefly, an "instructional situation" occurs whenever a NLS faces a situation demanding a change in its internal environment which requires the involvement of systems or elements thereof, external to the NLS. In an instructional situation, an unstable system is created for as long as the NLS requires to change its internal environment (i.e., for as long as it takes the NLS "to learn"); this ephemeral system was called Amplified Learning System, and OLSs represent a technology aimed at rendering operational such ALSs.

Much of the previous work in OLSs has dealt with rule-learning situations where an algorithmic approach has been followed for both the development of the OLS and in the main instructional strategies [3]. In the present paper we present and discuss another development. A computer-based lesson on circular movement was developed and implemented on an IBM-PC. In the main part of the lesson, students are presented with a simulation where a car has to run on a racing track. This track is composed of straight and circular segments; the car is assumed to be dimensionless, and its mass and the friction (assumed to summarize the interaction between tires and track surface) are given as data (actually, randomly generated). Knowing the radius of the circular segment facing the car, the user is asked to input a speed that the program will compare with the maximum speed allowed for the given mass, friction and radius. At the end of the simulation, a score derived from both the occurrence of accidents (whenever the speed inputted was greater than the maximum one) and the percentage of the optimal-race-run (loss of points due to inputting speed smaller than the maximum ones) is displayed. The object of this part of the lesson is to create an environment (the simulation) in which the student can work towards the discovery of the relationship between force, mass, radius and speed in a

circular movement. The theoretical appropriateness of simulations for discovery learning has been discussed elsewhere [4]. However, our learning environment is enriched by the presence of an advisory facility within the program (even if rudimentary). This "help" device is available while the use is running the simulation; it provides information on concepts and relations relevant to the "instructional objective" without explicit mention of this. As well, the advisory facility shows some kind of "intelligence"; it was built as a Paskian entailment mesh [5,6] of 25 "statements" about 15 concepts related to circular movement. In this sense, our enriched learning environment, albeit rudimentary, fulfills the educational role that Coombs and Alty ask from an expert system [7].

In this paper, drawing on our previous work on OLS, we will present the rationale for enhanced learning environments. Regarding the specific example presented, the discussion will focus on the technical aspects of this particular implementation. Some results of experiments, with this program will be presented and discussed.

References

1. Vazquez-Abad, J. and Larose, R., "Computers, Adaptive Systems and Operational Learning Systems," *Comput. Educ.* 8, No. 1, pp. 27-29 (1984).
2. Larose, R. and Vazquez-Abad, J., "Systems d'apprentissage operationnels: une technologie pour le developpement d'un enseignement adaptatif," presented to the 10th International Congress on Cybernetics, Namur, Belgium, 22-27 August 1983.
3. Larose, R. and Racine, S., "Un algorithme pour la conception de systemes d'apprentissage operationnels," unpublished manuscript, Universite de Montreal, Faculte des sciences de l'education, Groupe de Recherche sur les Processus Cognitifs de l'Etudiant Universitaire (1982).
4. Winer, L.R. and Vazquez-Abad, J., "Towards a Theoretical Framework for Educational Simulations," *Simulation/Games for Learning* 11, No. 3, pp. 114-119 (1981).
5. Pask, G., "Conversation, Cognition and Learning," Elsevier, Amsterdam (1976).
6. Mitchell, P.D., "Representation of Knowledge in CAL courseware," *Comput. Educ.* 6, pp. 61-66 (1982).
7. Coombs, M. and Alty, J., "Expert Systems: An Alternative Paradigm," *Intl. J. Man-Machine Studies* 20, pp. 21-43 (1984).

bVLISP: An Interactive Programming Environment

Harald Wertz
Dept. Informatique
Universite Paris VIII - Vincennes

Constructing programs is rarely a well defined, orderly progress proceeding from abstract specification to design and finally to implementation. It is a tedious, time consuming and complex activity, based often on some method of trial and error or successive refinements, while constructing, testing, and rejecting or modifying different algorithms, heuristics and representations used so far.

Often even, the initial (formal or informal) specification of parts of the developing program will be modified as a result of new insights gained during the process of interactive design and implementation.

In actual programming environments no on-line possibility exists to keep track of the developmental decisions, to access previous or future versions of the system and to switch back and forth between different viewpoints of it. In addition, on-line possibilities to help to evaluate the *range of influence* of intended modifications do only rarely exist.

The construction of complex programs not only creates problems during the development, maintenance, extension or transportation phase, it also creates crucial problems for the *comprehensive* interactive reading or annotation of it.

What is needed is a programming environment permitting the programmer to keep track of all his/her ideas, intentions and decisions underlying every part of his/her program, as well as a possibility to access them at every moment of his/her interaction with it. The programmer needs to have a possibility to create tentative versions of parts of the program, without destroying or otherwise fundamentally changing the structure of the already existing program. S/he should be able – after an incorrect decision – to go back to previous code (or specifications), while keeping the actual version as an annotation of it, perhaps even with comments stating the actual reason for abandoning this line of development. A possibility should be offered to work simultaneously on different independent versions of parts of the program, leaving open the *guided merging* of parts of those different tentative versions into a new and more elaborated one.

In addition, during the development of the program, the programmer needs a possibility to execute *unfinished* programs interactively – without being confronted with already anticipated “fatal error” messages.

bVLISP, the programming environment we have constructed, addresses all those issues: it is able to keep track – through an annotation facility – of all the modifications; it enables – through a multiple evaluation function – the simultaneous use of multiple versions of a program; it accepts temporarily limited solutions, and it permits the execution of unfinished programs and useful annotations of every part of the developing program. This system, built on top of a screen oriented structural editor, offers possibilities to attach descriptors, such as input/output assertions for example, to every portion of the program and to maintain – simultaneously – different versions of the program being written, including tentative hypothetical versions. It comprises a mechanism to maintain minimal consistency between modified parts of code, the non-modified parts of code and attached descriptors, as well as an evaluation module functioning in different modes: *normal evaluation*, *symbolic evaluation* and *checking evaluation*.

The standard programming aids, such as indexors, pretty printers, trace and documentation packages, undo- and history facilities are generalized to handle the descriptors and unfinished programs as well.

These possibilities are obtained by means of three major changes to our LISP interpreter:

1. A new representation of programs as a tree of *all* its versions until now constructed, along with possible attachments to every point in the tree.
2. A sophisticated, structure oriented editor through which all interaction with the LISP system takes place and which works on this tree representation of programs.
3. A special evaluator, able to work in different, programmable modes and contexts.

With this system, the programmer can, at every moment, interactively control the evolution of his/her software architecture.

Expert Systems in Teacher Education

Sharon Wood
Cognitive Studies Programme
University of Sussex
Falmer, Brighton, U.K.

This project involves the application of expert systems technology and AI techniques to teacher education. Expert systems are interactive systems which can question a user about their problem or task, and the circumstances under which the problem occurs. The system embodies the knowledge or "expertise" of someone who is proficient within their field; this expertise, together with the basic information provided by the user, enables the system to deduce the nature of their problem and to specify solutions or advice. Typically, such systems are also capable of explaining their reasoning; why a question was asked or a solution preferred.

This work is taking place through the building of an advisory system – the Trainee Teacher Support System (TTSS) – based on the relationship which exists between trainee teachers and their teacher-tutors. This relationship is characterized as one in which trainee teachers seek advice on problems they experience in the classroom; and in which teacher-tutors comment specifically on observed or described events, with the additional aim of increasing the trainee's understanding of what was taking place.

Education differs from traditional applications for expert systems technology in being a social domain. We have no theories regarding the underlying causality of such domains, as we might have for more traditional science area. Part of this work, therefore, embodies the developments of a "process" model of the classroom domain which describes to some extent some of the contingencies perceived by experienced teachers to exist within classroom situations. This model includes, for instance, the effects of novelty and variety on interest, say, or the effects of motivation on attention and control in the classroom.

The process model represents the "deep" knowledge of an experienced teacher – that which underlies the interpretation of trainees' difficulties and decisions about appropriate advice. Within the TTSS, this process model would essentially fulfill the same role: enabling the system to identify the probable protagonists of a situation the user describes; rather than responding to the superficial characteristics of the description itself. Thus, for instance, dealing with misbehavior which seems to owe as much to the work beyond the ability of the class, as to any deviance in the children themselves, might be better addressed by appropriate re-structuring

and pacing of the work, than any direct attempt to regain control. By adopting a mechanism for identifying such contingencies, one has a basis on which to advise accordingly.

We notice that the terms of advice given by many existing expert systems is essentially prescriptive – with a little explanation thrown in to make it more palatable! A teacher may bring a lesson about successfully in an infinite variety of ways, however; and as far as budding teachers are concerned, one might not wish to stamp out advice in any particular mold which might dampen the development of the individual teaching style. Consequently, the role of the TTSS is envisaged as a delivery vehicle for appropriate, relevant and useful advice on classroom events, which might enable the trainee teacher to make her own, informed decisions regarding future lessons.

The less directive mode of interaction being developed for advising teachers might be of particular interest to those involved in developing systems for educational purposes in general. This would allow one to take on board the advantages of expert systems technology; say, in developing interactive systems; in the formalization of heuristic and strategic knowledge used in teaching; etc. This might be especially true in situations where one is unable to specify categorical solutions to problems, considering the lack of predictive capacity offered by much social science theory, which includes that for describing events which take place in the classroom.

FITS: Framework for an Intelligent Tutoring System

M. R. Woodroffe
Cognitive Studies Centre
Department of Computer Science
University of Essex

This paper describes FITS, an intelligent tutoring system (ITS) shell, and the first implementation of it. The shell is a set of domain-independent procedures which are augmented by domain-dependent knowledge. The domain-dependent knowledge is represented by a hierarchy of plan networks, a number of problems for the student to solve, and a set of heuristic rules to assist in program control.

The tutoring strategy employed by FITS is a form of guided discovery learning, which presents the student with a complex problem, an environment to solve it in, and occasional tutorial guidance.

Introduction

The aim of the FITS project is to implement a domain-independent shell that observes the student solving problems, understands his behavior, and generates predictions as to his probable subsequent behavior. These predictions can then be used to provide helpful guidance when errors occur.

The concept of a domain-independent framework for an ITS was developed by McCalla, Peachey, and Ward (1982), who wrote an ITS shell which represented the domain knowledge in nested AND/OR prerequisite graphs. This approach made the construction of an ITS relatively easy. However, it necessitated a tutoring strategy which was less flexible than that intended for FITS.

Guided discovery learning is used in the WEST program [Burton and Brown 1979], a program to teach the game "How the West was Won". In this game, there are a number of skills to be learned. At each move WEST recognizes the skills the player has used, and compares them with those he should have used. This provides the knowledge for the tutoring element of the program. However, because of the domain there is no representation of the ordering of skills or the way they combine to form higher-level skills. In the domain that FITS is to be used in these are essential concepts. The use of a plan hierarchy enables these to be represented. It

also adds power to the understanding element of the program because it gives the ability to recognize the student's plan and make predictions about his probable intentions.

Similar hierarchies have been used in other interactive programs, such as the Advisor [Genereth 1977]. The FITS hierarchy is unusual because it includes a representation for enquiry actions. Enquiry actions are actions performed by the student to derive information about the environment in which he is working.

This paper will describe the plan hierarchy in some detail, also outlining the program operation and stating some of the future project objectives. All of the examples will be taken from UNIX, the initial test domain.

The Complete Plan Network

Planning generally occurs at a number of levels of abstraction. In UNIX creating a directory file could be considered to be a low-level action, and rearranging the file structure a high-level one. This results in a hierarchy of potential actions, with low-level actions combining to form higher-level actions. The situation is further complicated by actions sometimes being achievable by several combinations of lower-level actions. The resulting structure is a hierarchy of possible plan networks, which is represented explicitly and referred to as the complete plan network (CPN). This explicit representation facilitates the plan recognition process.

The CPN is constructed from actions which contain prerequisites, variable constraints, an expansion, which expresses the action in terms of lower-level actions, and efforts. In addition to these there is an is-it-necessary check, to see if the effects of the action have already been achieved, and a parent indicator which points to the parents of the action in the hierarchy, facilitating plan recognition.

Tutoring Session

It is envisaged that FITS will give the student a number of composite problems to solve and an environment to solve them in. In UNIX, an example problem would be to rearrange the file hierarchy and the problem-solving environment would be the UNIX operating system. The student's problem-solving process will then result in one or more actions in the CPN being activated as possible interpretations of his behavior. When errors occur guidance for the student can be derived both from these interpreting actions and from the student models. Guidance will only be given by FITS when the precise nature of the error has been established.

In the existing implementation the student is given a problem containing a single goal, and a simulated UNIX environment. If during the attempt to achieve this goal the student issues an incorrect command, FITS suggests a heuristically ordered list of correct commands which it is expecting.

Plan Recognition

The sequences of UNIX commands that the student enters can be interpreted by FITS as plans, where a plan is a series of actions which combine to achieve a desired goal. It is the recognition of these plans which enables FITS to perform error checking and provide guidance. The recognition process requires a considerable amount of constraint handling which is not discussed in this paper. There are two stages in plan recognition: plan activation and prediction generation. If there are no active plans in the database, the following algorithm will perform a bottom-up recognition process, matching the student's input and activating the action's parents in the CPN.

1. Match the observed UNIX input against the actions in the CPN, finding some action A.
2. Check that the following slots in A are satisfied: local constraints, prerequisites, is-it-necessary, and the expansion.
3. Make the FITS model of the student and the world consistent with the actions having been performed.
4. Perform a depth first search moving up the CPN from action A. For each action encountered, perform the checks described in 2. NOTE: when checking the expansion only check up to the point indicated by the parent pointer in the child.
5. Define as active any action that satisfies these checks. Active actions are possible interpretations of the student's input.

The prediction generation process involves the top-down generation of expected UNIX inputs from the active actions. The following algorithm is used:

1. Take each of the highest-level active actions.
2. Duplicate the actions according to the number of possible instantiations.
3. Remove the satisfied expansion elements.
4. Make predictions from the remaining elements in the expansion.
5. IF the prediction is a UNIX command
 THEN add it to the prediction list
 ELSE IF the element matches an active action
 THEN repeat 5. for the child
 ELSE perform the checks described in 2. of the previous algorithm on the child action
 and repeat from 2. in this algorithm [this has not been implemented]

Upon the completion of prediction generation there exists a list of expected inputs.

Heuristics

In order to provide guidance when errors occur it is necessary to heuristically order these predictions. This is achieved by applying heuristic rules to each level of the action trees grown by the prediction generation algorithm. Three types of heuristic rules have been used. The first

type assesses the student's knowledge of the action. The second assesses the appropriateness of the action relative to the specified goals, and the third assesses domain-specific attributes such as nearly complete plans being more likely than partially completed plans.

Future Developments

The FITS project is in its early stages and consequently there are many developments needed. The most immediate is to improve the efficiency of the plan recognition process. This will be achieved by the inclusion of a heuristically ordered agenda, possibly augmented by dependency-directed backtracking as used in POISE [Carver et al. 1984]. This will necessitate the definition of plan activation and prediction rules, based upon the beliefs and knowledge of the student.

The student models will be modified to incorporate erroneous information. This will add to their focussing power during plan recognition and enable a more sophisticated tutoring strategy to be used.

Finally, it is hoped to incorporate meta-level rules corresponding to the evolutionary relationships described in Goldstein (1979). This will allow FITS to recognize plans which have a meta-level relationship, such as analogy, with existing plans in the CPN and student models.

References

- Burton, R.R. and Brown, J.S., "An Investigation of Computer Coaching for Informal Learning Activities," *Int. J. Man-Machine Studies* 11, 5 (1979).
- Carver, N.F., Lesser, V.R., and McCue, D.L., "Focussing in Plan Recognition," *Proceedings AAAI*, 1984 42 (1984).
- Genesereth, M.R., "The Role of Plans in Automated Consultation." *Proceedings IJCAI 1977*, (1977).
- Goldstein, I.P., "The Genetic Graph: A Representation for the Evolution of Procedural Knowledge," *Int. J. Man-Machine Studies* 11, 51 (1979).
- McCalla, G., Peachey, D., and Ward, B., "An Architecture for the Design of Large Scale Intelligent Tutoring Systems," *Proceedings CSCI 1982*, 85 (1982).

Intelligent Help Systems

Charles Woodson
Institute of Cognitive Studies
University of California

Our research goal is to better understand how to build effective computer help systems which increase human effectiveness by meeting the help needs of human computer users. In this paper we describe the conceptual basis for help systems and illustrate these by reference to our own projects.

Help systems are based on:

1. A model of the user's problem,
2. A model of the user, and
3. An algorithm for providing assistance to the user. In most current systems these are all quite simple.

Our observation of users using UNIX, EMACS, vi in UNIX, nroff/troff with various printers, the Pascal compiler in UNIX, and the CMS system, indicate:

1. Users use only a small portion of the commands available,
2. Two individual users with similar objectives often use subsets of commands which are remarkable different,
3. Many commands are used only occasionally and with high error rates,
4. Users lack the concepts essential to understand some commands and there is almost no help available from the system to develop these concepts.

Passive help is usually available to aid the user in the use of commands which are understood. Passive help systems provide command options lists, command descriptions, index of help displays and error explanations. Less frequently, they provide on-line tutors, explanation of concepts, explanation of systems structure, referrals to other sources of instruction, and demonstrations.

Active help is needed to help users develop new concepts. For example:

1. Recognize a user's goal and suggest alternate, more effective approaches to it,
2. Recognize repeated attempts of improperly using the same command, and
3. Suggest concepts a user may need to learn.

I will illustrate the above by describing programs we are developing to aid UNIX users. These programs seek to help users by

1. Providing mental structure,
2. Displaying sensitivity to users' circumstances,
3. Supporting browsing, and
4. Supporting concept learning.

DRAFT