

Tutors, Instructions and Helps

Claus Möbus & Heinz-J.Thole

Project **ABSYNT**

University of Oldenburg

FB 10, Informatik

Unit on Tutoring and Learning Systems

D-2900 Oldenburg

W-Germany

Abstract

The goals of this paper are threefold. First we want to present a review of the literature on computer assisted instruction, second we want to discuss the quality of instructions in some texts and human-computer dialogs concerning computer programming. Third we want to demonstrate the cognitive-science-based development of our programming environment ABSYNT. This includes the construction of iconic instructions and helps which promise to be superior to verbal instructions and helps when properly designed.

This paper consists of three parts. In the first part (1.-3.) we give a short introduction to computer aided instruction (CAI), intelligent computer aided instruction (ICAI) and a special variant, namely intelligent tutoring systems (ITS).

One of the most underestimated problems in the development of an ITS is the proper design of instruction. We prefer to discuss this problem not on an abstract but on a rather concrete level. So the second part (4.) documents two examples of instructions and helps typically given in textbooks and intelligent tutoring systems concerning computer science education. These examples show that at present the construction of instructions is more art than science. They are an uncontrolled source for errors and misconceptions of the students. This causes severe problems in CAI and ICAI, especially for the design of help components.

In the third part (5.) we want to demonstrate how to derive instructions and helps for a problem solving monitor (PSM) presently under construction. This approach rests on production based learning theory (ANDERSON, 1983, 1987a; ANDERSON, GREENO, KLINE & NEVES, 1981; ANDERSON, KLINE & BEASLEY, 1980; EGAN & GREENO, 1974; SIMON & LEA, 1974; VanLEHN, 1987a,b). We used rule sets to formalize the knowledge about the operational semantics of a graphical virtual machine, which is driven by graphical programs in the ABSYNT language (ABSTRACT SYNTAX TREES). Abstract facts and rule sets, which can be conceived as a runnable specification of the virtual machine, are related to concrete counterparts: icons and iconic rules. Thus diagrammatic information forms the core of our instructional and help system.

1. Introduction

ICAI has two predecessors: machine based teaching and CAI. Machine based teaching was a direct consequence of principles developed under the influence of SKINNER's theory of operant conditioning. According to this behavioristic learning theory an operant is a unit of behavior which is conditioned by reinforcement. This reinforcement was not given by human teachers but by teaching machines. Instructional programs were at first implemented on nonelectronic equipment. But since the early 1960s (with the rise of electronic computers) the notion of CAI emerged. Many systems were developed in the hope of constructing tutors who could support human teachers or relieve them from routine work. But the development of this field showed some parallelism to machine translation and natural language processing. Excessively high expectations were nourished by researchers (e.g. KLING, 1979; TAUBER, 1980) but research showed only slow progress due to deficiencies of instructional theories and shortcomings in computational media and programming techniques. The disappointment of the nonscientific community led to a cut in funding. For instance, in the midseventies nearly all third party sponsored projects in Germany were cancelled.

Today the international and national scene has changed somehow, which is partly attributable to new developments in computer science (KAWAI, MIZOGUCHI, KAKUSHO & TOYODA, 1987) and partly to progress in cognitive science and cognitive psychology (SLEEMAN & BROWN 1982; MANDL & FISCHER, 1985; KEARSLEY, 1987; WENGER, 1987). Progress in the design of hardware (workstations with bitmapped high resolution displays) and software (object- and rulebased programming languages) made it seem possible to meet the challenge of designing intelligent tutor systems realizable. The performance of these systems should equal that of a human tutor (ANDERSON, 1987). Admittedly, progress is still not rapid and there are nearly no commercial systems available, which would justify the predicate "intelligent". KEARSLEY pointed out clearly that:

"ICAI is an emerging field that is ill-defined at present. The distinction between intelligent CAI systems and computer-based instruction programs cannot be sharply drawn. ICAI programs use AI programming techniques and are implemented in languages as LISP and PROLOG. Developers of ICAI systems focus on problems of knowledge representation, student misconceptions, and inferencing. By and large, they have ignored instructional theory and past research findings in computer-based instruction."

Another obstacle to fast progress in this field is due to the fact that a successful design, development and evaluation of an ICAI-System or ITS has to be a joint effort of researchers from cognitive psychology, cognitive, educational and computer science. Different backgrounds and goal structures can stimulate scientific discussions (BODEN, 1981; SCHEERER, in press), but usually do not provide a fertile environment for the development of software under time and budget constraints.

2. Early Teaching Machines, Computer Aided Instruction (CAI) and Intelligent Computer Aided Instruction (ICAI)

2.1 Behaviorism and Teaching Machines

Manually operated drill or teaching machines (PRESSEY, 1926, 1927; SKINNER, 1958) were the forerunners of today's CAI and ITS systems. They presented the instructional material as a linear sequence of frames (frames meaning pages here, not the MINSKYan knowledge structure (MINSKY, 1975)). Each frame required a response from the student. Even after an incorrect response the student had to move to the next problem. The theoretical position of this kind of teaching style was based on a theory of instruction which descended from operant learning theory (GALANTER, 1959; HOLLAND, 1960, 1964; SKINNER, 1954, 1958, 1968). From the introduction of programmed instruction researchers expected more optimal planning, scheduling and individualization of the learning process with immediate feedback of success or failure, and a more economical expenditure of teaching resources (WEINERT, 1967). These early efforts culminated in the book "Analysis of Behavior" (HOLLAND & SKINNER, 1961), which is an attempt to teach the underlying behavioristic learning theory with "linear programs". The instructions and a short excerpt of the program are included in **appendix A**. It is interesting to see that still today some computer scientists with a strong inclination to artificial intelligence are following the behavioristic tradition when teaching the state-of-art symbolic computer language SCHEME (FRIEDMAN & FELLEISEN, 1987). We included the pages 3 and 4 of their textbook in **appendix B**.

It was the pioneer PRESSEY (1963), who anticipated the failure of programmed instruction due to the atomization of knowledge structures and due to the lack of a supportive and friendly environment for selfpaced and autonomous explorative learning.

Today we know that because of its inherent inability to provide individualization and rich feedback programmed instruction could fulfill teaching purposes only when students were highly motivated. It is thus not useful for large scale applications with heterogenous student populations, but only for short-time "crash-courses".

2.2 Computer Based Instruction (CAI)

Soon it was realized that to provide the necessary individualization and feedback linear programs had to be abandoned in favor of branching programs which could provide remedial loops in case of student errors or jumps in case of student competence. These control structures could not be managed efficiently by mechanic devices. So researchers who had access to computer resources wrote branching programs in a computer language. This meant that to provide branches for individualized instruction the implementer and the courseware author had to anticipate every response or misconception of the student (ATKINSON & WILSON, 1969; BARR & ATKINSON, 1977). This is nearly impossible to achieve and led to the demand of ICAI, which should relieve courseware authors from the burden of explosive branching. Today we recognize a three phase sequence in the history of CAI.

2.2.1 Basic Research and Prototypes

The first phase was characterized by basic research for laboratory prototypes (SUPPES, JERMAN & BRIAN, 1968; SUPPES & MORNINGSTAR, 1972). Researchers had become more modest than SKINNER and his group, who could not imagine any complexity barriers for programmed instruction. To facilitate the development of learning aids and feedback helps, domains of teaching were confined to basic skills in mathematics, reading and computer programming (BARR & ATKINSON, 1977).

Branching programs were written in conventional imperative programming style with nested IF-statements or conditional jumps. So rather sophisticated decision structures evolved (ATKINSON, 1972; SMALLWOOD, 1962, 1970). But to achieve a high response-sensitivity to idiosyncratic patterns of student behavior, courseware authors had to face the well known problem of combinatorial explosion of alternative paths in the discrimination tree of system responses to student inputs.

In the same period fell some early developments concerning detailed behavioral assessment to model the learner and learning processes (BARR & ATKINSON, 1977; FLETCHER, 1975; SMITH & BLAINE, 1976; SMITH, GRAVES, BLAINE & MARINOV, 1975; SUPPES, 1981; SUPPES, FLETCHER & ZANOTTI, 1975, 1976;) and the use of personal(-ized) computers (DYWER, 1974).

2.2.2 Large Scale Applications and Commercial Systems

The second phase showed larger systems with author languages and selfguided primitive problem and item generators (UTTAL, ROGERS, HIERONYMOUS & PASICH, 1969; WEXLER, 1970; KOFFMAN & BLOUNT, 1975) to develop cost-effective teaching environments. Some of these systems (LEKAN, 1971) became familiar outside academic institutions. Two of them are even known today. These are PLATO (with author language TUTOR) and TICCIT (with language APT).

PLATO (Programmed Logic for Automatic Teaching Operation) was developed by CDC (Control Data Corporation) and the author language TUTOR by researchers at CERL (Computer-based Education Research Laboratory). Similar to TICCIT (Time-shared Interactive Computer Controlled Information Television) terminals with graphic capabilities were served by a mainframe computer. PLATO was used for lessons in genetics, elementary mathematics and game playing (DAVIS, DUGDALE, KIBBEY & WEAVER, 1977). The educational game WEST (RESNICK, 1975) for example, which became a famous paradigm in ICAI (BURTON & BROWN, 1979) was taught in PLATO lessons, too. Even experiments in the microworld style with direct manipulation of the user-interface could be conducted (SHNEIDERMAN, 1983, 1987). Contrary to PLATO TICCIT claimed to have an instructional design framework which led to some evaluation studies (BUNDERSON, 1974; ALDERMAN, 1977; MERRILL, SCHNEIDER & FLETCHER, 1980; BORK, 1981, 1986).

At the present moment we see a revival of author languages under the name of "stackware". This revival was started by bundling the personal computer MACINTOSH with the new object-orientated HYPERCARD database system from ATKINSON (GOODMAN, 1987; WILLIAMS, 1987).

2.2.3 The Demand for AI in CAI

In the third phase of CAI the shortcomings of the CAI systems became more and more apparent. On the one side software engineering techniques (requirement analysis, specifications, evaluations etc.) came into use to remedy design flaws. On the other hand, it was realized that tutoring devices which should show a similar competence as human tutors needed a stronger theoretical, methodological and empirical basis (CARBONELL, 1970; BROWN, 1977; FORD, 1986, 1987; SPADA & OPWIS, 1985). This requires to improve cognitive science theories on knowledge acquisition, retrieval and deduction and to implement separable knowledge components in the computer tutors to overcome "theoretical frontiers in building a machine tutor" (WOOLF, 1987). These frontiers are due to the complexity of the domain of discourse, to specific unexpected events in the tutoring situation and to idiosyncrasies of the student concerning his learning history, his knowledge state, his problem solutions, his failures and misconceptions. These insights led to the world-wide demand of "intelligent" computer aided instruction (ICAI) (TCHOGOVADZE, 1985; YAZDANI, 1986, 1987).

2.2.4 Summary

CAI systems are characterized by the goal of teaching a large body of knowledge. Their theoretical basis rests on classical psychological learning and instructional theory. Lessons are structured into sequences of frames. The system responses are largely prespecified. Instructional methodology ranges from direct instruction to discovery or exploratory learning. Domain knowledge is not explicitly represented in the system. Instead of that the design phase of the system is sometimes started with a careful task-analysis thus delivering a hierarchical organisation of subtasks and learning prerequisites (BLOOM, 1962, 1972; GAGNE, 1974). There is only rudimentary student modelling. Quantitative measures of student behavior such as test scores and solution times are used as a description of the learner personality. This is in accordance with psychological test theory (FRICKE, 1972; FISCHER, 1974; SPADA, 1976). Instructional formats show broad variation: "slide shows", drill & practice lessons with multiple choice answer capabilities, games and simulations of microworlds (FEURZEIG, 1987; LAWLER, 1984, 1987; PAPERT, 1987). The subject matters and discourse themes are not constrained to formalized knowledge domains. It is even tried to teach such difficult and fuzzy-structured domains as natural languages (LAWLER & LAWLER, 1987). Some computer based curricula were empirically evaluated according to well established statistical methods developed in educational psychology (GAGE, 1967; KLAUSMEIER, 1971; STRITTMATTER, 1973; FRICKE, 1974; ANDERSON, BALL, MURPHY & Associates, 1976; CRONBACH & SNOW, 1977; KRAUSE & SEEL, 1979). The systems were implemented mostly on general purpose hardware in general purpose or special author languages.

2.3 ICAI: Reactive Environments, Problem-solving Monitors and Intelligent Tutoring Systems

ICAI comes in various styles and forms. The main categories of ICAI systems are (1) reactive environments (BROWN, 1977) or reactive microworlds, (2) problem-solving monitors (SLEEMAN, 1975) and (3) intelligent tutoring and learning systems (CARBONELL, 1970). They differ with respect to goals, theoretical bases, instructional processes and principles, methods of knowledge representation, student modelling, subject matter area, evaluation designs and last not least

hard/software requirements. However, they have a common denominator, which is their ability to represent knowledge of various sources.

The most important thing for an ICAI system is to show adaptability with respect to idiosyncrasies of the learner and to his competence level. This is a function of metaknowledge concerning declarative, diagnostic and procedural knowledge. This means that the ICAI system knows what, whom, and how to teach (SELF, 1974; DEDE, 1986). There is a vast body of literature on how to implement this kind of metaknowledge. The most clear-cut structure is threefold: an expert, a teacher and a student module. The expert module represents and delivers the domain knowledge. The teacher module diagnoses errors, assesses knowledge states of the student, makes inferences concerning the plans of the problem solver, explains errors, gives helps and proposes new tasks. The student module represents the actual knowledge state and his misconceptions. The modules can be elaborated, so that they are models of experts, teachers and students, respectively. It is hoped that such models show a greater ability to explain their operations than simple modules do.

Their identification in current systems is difficult or even impossible to obtain because their knowledge bases are often not as clearly separated as theory postulates.

Intelligent tutoring systems will only become popular if they outperform old and well established educational technology. Their performance must not be inferior to books or classroom instruction and should equal that of individual lessons given by a human tutor. Both criteria require powerful workstations with bitmapped high resolution graphic displays (SMITH, IRBY, KIMBALL, VERPLANK & HARSLEM, 1982; WILLIAMS, G., 1983; WILLIAMS, G., 1984), direct manipulation facilities (HUTCHINS, HOLLAN & NORMAN, 1986; SHNEIDERMAN, 1983), sophisticated software tools and elaborated psychological guidelines (ANDERSON, BOYLE, FARRELL & REISER, 1987) concerning the optimization of the knowledge acquisition process. The software tools have to support object orientated programming for the development of courseware and rule-based programming for error diagnosis, student modelling and helps (KAWAI, MIZOGUCHI, KAKUSHO & TOYODA, 1987).

Reactive environments and problem monitoring systems (PSMs) do not instruct explicitly. They deliver a friendly and supporting problem solving environment (e.g. a structure editor), so that the working memory load of the problem solver is kept at a minimum and certain classes of errors (e.g. syntactic errors in writing computer programs) cannot occur. Instructions appear only in the form of helps after diagnosis of an error. If the diagnosis is based on plan diagnosis we will talk of a PSM. PROUST (JOHNSON, 1986; JOHNSON & SOLOWAY, 1985, 1987) and ABSYNT which is presently under construction (COLONIUS, FRANK, JANKE, KOHNERT, MÖBUS, SCHRÖDER, THOLE, 1987; JANKE & KOHNERT, 1988; KOHNERT & JANKE, 1988; MÖBUS, 1985, 1987) belong to this group of ICAI environments.

2.3.1 Some Classic ICAI Systems

There is a core of laboratory systems which are regarded as classics by various authors (BARR & FEIGENBAUM, 1982; PARK, PEREZ & SEIDEL, 1987; SHAPIRO, 1987; SLEEMAN & BROWN, 1982; WENGER, 1987; YAZDANI, 1986):

(1) the Problem Solving Monitor ACE (= Analyzer of Complex Explanations). The system accepts interpretations of nuclear magnetic resonance spectra. But more important is that it analyzes the explanations and justifications of students in natural language dialogs (SLEEMAN, 1975; SLEEMAN & HENDLEY, 1982). SLEEMAN & HENDLEY argue that this "teaching back" is because of the greater involvement on behalf of the student educationally much more valuable than simple "learning".

(2) the BASIC Instructional Program BIP (BARR, BEARD & ATKINSON, 1975, 1976), which used as a very early system the structuring of the curriculum in an information network (CIN) according to required skills and tasks. It was one of the first systems which used the notion of a student model, which consisted of a set of scores representing the mastery of a skill.

(3) the diagnostic modelling scheme BUGGY for the diagnosis and analysis of errors ("bugs") in the field of place-value subtraction (BROWN & BURTON, 1978; BURTON, 1982). The system was based on a skill-subskill lattice, which was derived from empirical error statistics. It showed that a careful analysis could demonstrate the nonrandomness of many errors which would otherwise have been regarded as random errors. It was even possible to use the occurrence of errors to derive the knowledge structure of the task domain, which was formalized as a skill lattice.

(4) the medical diagnosis tutor GUIDON (CLANCEY, 1982, 1983, 1987) which used knowledge of the expert systems MYCIN, EMYCIN and NEOMYCIN as a databasis (CLANCEY, 1986a). The goal was to develop and improve diagnostic skills of medical students in the domain of bacterial infections. The development of the tutor showed the inadequacy of the compiled knowledge of MYCIN for teaching purposes and the need for restructuring and decompiling the knowledge basis when it should be used for instructional purposes.

(5) the tutor INTEGRATE for symbolic integration (KIMBALL, 1982). The tutor communicated judgmental knowledge about to choose an approach or a heuristic for a given integration problem. Expert and student models are represented by probability matrices, which describe the preference of solution approaches (columns) in problem solving states (rows). So the matrices contain compiled knowledge in a very condensed form. Example problems are chosen in such a way that the discrepancies between expert and student model are minimized. The tutor is able to learn from the student, if the student's solution is superior to its own proposal. Student and expert models are changed by updating the entries of the underlying probability matrices according to bayesian methodology. Thus the tutor is able to improve its competence.

(6) the MACSYMA Advisor (GENESERETH, 1982) is conceptualized as an intelligent help system for MACSYMA users. MACSYMA is a large package for symbolic mathematics developed at MIT. Because users had some difficulties with MACSYMA's cryptic messages and syntax a friendly help system seemed to be a good idea. In reality the Advisor remained an experimental system. Nevertheless some interesting concepts were introduced which are still relevant.

So a plan generator MUSER, that should mimic the problem solving behavior of a MACSYMA user was constructed. Plan recognition was treated as the inverse of plan generation. It was treated as a parsing problem. Student's actions constitute the terminal vocabulary and the planning methods the rewrite rules of the planning grammar. The plan recognition procedure of the Advisor is working in a hybrid top-down, bottom-up methodology: from overt actions corresponding goals are inferred and from goals subgoal expectations are computed. The search space is controlled by propagation of dataflow and expectation constraints.

(7) the first artificial intelligence based tutor SCHOLAR. It was designed by CARBONELL (1970) to mark explicitly a change in paradigm from frame-based CAI to a knowledge-based ICAI approach. The teaching domain was chosen to be rather simple: South American geography. The new methodological concepts were a) a mixed-initiative instructional dialog on the basis of a case grammar, b) the representation of domain knowledge by the semantic network formalism, c) an agenda of conversational topics to allow the switching of the focus of the dialog and d) default inference strategies to cope with incomplete knowledge on the tutor and the learner side.

(8) the reactive learning environment SOPHIE (Sophisticated Instructional Environment) was dedicated to electronics trouble-shooting. The development period ranged from 1973 to 1979. Three systems were conceptualized and partly implemented: SOPHIE-I (BROWN & BURTON, 1975), SOPHIE-II (BROWN & BURTON, 1986) and SOPHIE-III (BROWN, BURTON & de KLEER, 1982).

The main goal was to provide a reactive learning environment which could in the course of the project be extended to an expert system useful for online repairs. But this was an unrealistic goal as the results of the project clearly showed. Both the pragmatic need for greater computing resources and the theoretical necessity of representing the qualitative and not only the quantitative reasoning processes of the students forced BROWN, BURTON and de KLEER to change their research priorities and SOPHIE to a premature stop.

SOPHIE-I consisted of an automatic laboratory instructor and a simulated laboratory workbench. This made it possible to run experiments to understand the workings of the faulted circuit. The student was to find the fault by taking measurements. A simple coach, which was made of procedural specialists, criticized the trouble-shooting behavior of the student. It checked the redundancy of the proposed measurements of the circuit, the optimality and consistency of the students' hypotheses. The coach even answered questions and hypothetical questions and worked out all hypotheses compatible with the set of measurements performed so far. To provide an easy interaction mode with SOPHIE a natural language interface based on a semantic grammar (BURTON & BROWN, 1979) was implemented as an ATN.

SOPHIE-II is an attempt to improve the poor explanation capabilities of SOPHIE-I. So a trouble-shooting expert implemented as an annotated and parameterized decision tree and a referee module were added. The expert demonstrated diagnostic strategies given a fault proposed by the student. Expert reasoning was done in a qualitative mode, which could be better explained and verbalized thus giving causally meaningful information. The students' personal experimentations alternate with observations of expert behavior. In case of uncertain knowledge they consult the referee which runs the simulated laboratory of SOPHIE-I. Trouble-shooting was later wrapped into a gaming environment to stimulate the verbalization of thought processes.

SOPHIE-III should combine the support for student initiatives and the powerful inference strategy from SOPHIE-I with the quality of the explanation capability from SOPHIE-II. So the inference machine was redesigned according to empirical observations of the cognitive strategies used by experts and students. The architecture consisted of three major expert modules: the electronic expert, the trouble-shooter and the coach, which was only rudimentarily implemented. The electronic expert utilized general electronic knowledge and some circuit-specific knowledge. Information flows upwards through the layers: 1) propagation of constraints (of the form: IF $A = x$, THEN $B = y$) to reason upon the quantitative measurements and the circuit topology. These inferences are modified into qualitative assertions which are 2) used by a production system drawing inferences about the module behavior. This is 3) further analyzed on the background of circuit-specific knowledge. The trouble-shooting expert works on top of this three-layered system. It proposes new measurements according to the deductions of the electronics expert.

(9) the Structured Planning and Debugging Environment (SPADE) for elementary LOGO programming (MILLER, 1979, 1982). The system rests on the assumption that the problem solving process is describable as a hierarchically organized decision process (MILLER & GOLDSTEIN, 1976; 1977a,b). Only the leaves of this tree consist of the application of manifest observable problem solving operators. So the development of computer programs is to a great deal a planning and decision problem which could in principle be described by a planning grammar. Only in the end it is a coding problem. An intelligent help system should thus give support for choosing optimal planning steps.

SPADE-0 was constructed as a "limited didactic" system which was used as an explorative testbed for this grammatical approach. Its purpose could best be described as a plan-oriented programming editor. The editor is driven by a context-free problem solving grammar whose nonterminals stand for goals or planning decisions and whose terminal vocabulary consists of LOGO code. The program is developed in a liberal top-down mixed-initiative dialog: proposals of the system are discussed with the student, who has the freedom to defer decisions and to move freely between the nodes of the tree in the problem solving grammar.

(10) the tutor WHY (STEVENS & COLLINS, 1977, 1980 ; STEVENS, COLLINS & GOLDIN, 1979, 1982) which tried to help the student develop a causal model of a complex physical process: meteorological conditions for rainfall. The student should be enabled to answer questions, give explanations and make predictions. The knowledge domain of weather conditions was chosen to study a) tutorial dialogs between human teachers and students, b) to classify, explain and correct typical misconceptions (e.g. bugs concerning facts, relationships and rules: overgeneralization, overdifferentiation etc.) about the physical processes, c) to look for a control regime that is scheduling tutorial dialogs with the aim of achieving a socratic interaction style. This style is characterized by a mixed-initiative interaction stimulating on the learner side the autonomous development of hypotheses, the discovery of contradictions and the drawing of inferences by issuing from the tutor side successive questions, counterexamples, generalizations and specializations. To support the goal of controlling a socratic dialog the authors collected a large set of heuristics (COLLINS, WARNOCK, AIELLO & MILLER, 1975; COLLINS, 1976, 1985; COLLINS & STEVENS, 1982; 1983).

2.3.2 Actual Problems in ICAI: Student Models, Plan Diagnostics and Cognitive Design Principles

2.3.2.1 Student Modelling and Plan Recognition

As was mentioned earlier, the "intelligence" of an ICAI system depends on the metaknowledge what, whom, how to teach. There is general agreement among researchers and practitioners (CLANCEY, 1986b) that to this end we need information about the knowledge state of the student and his/her intentions. Thus, student modelling and plan recognition become important research goals.

Unfortunately the notion student model is used with three different meanings: as a model of the

(1) ideal student. This is a normative concept prescribing the knowledge to be acquired up to a specified time point. Deviations from this ideal path can be measured and remedial actions are to be derived. This approach called "model tracing" is chosen by ANDERSON (1987).

(2) typical student. The student is conceptualized as a collection of facts, rules, malfacts and malrules. The latter are sometimes called "bugs". This kind of modelling was mainly explored in the domain of simple arithmetic (BROWN & BURTON, 1978; BUNDY, 1983; BURTON, 1982; YOUNG & O'SHEA, 1981) algebraic skills (SLEEMAN, 1982, 1983, 1984, 1985; SLEEMAN & SMITH, 1981) and mathematical games (GOLDSTEIN, 1980, 1982) with probabilistic inferences.

The domains had to be simple because the collection of bug catalogues was a time-consuming enterprise as was shown by BROWN & BURTON (1978) and BURTON (1982). So student-modelling was confined to domains where procedural skills but not problem solving abilities were responsible for success.

(3) the concrete student. The idea behind this model is that student modelling should be done with the help of inductive learning processes concerning rules and concepts. There is a vast body of published research on this topic from the view of experimental psychology (BOURNE, 1966, 1974; BOURNE, EKSTRAND & DOMINOWSKI, 1971; EGAN & GREENO, 1974; GOEDE & KLIX, 1972; HAYGOOD & BOURNE, 1965; HUNT, MARTINE & STONE, 1966; MEDIN & SMITH, 1984; MEDIN, WATTENMAKER & MICHALSKI, 1987; SIMON & LEA, 1974). The same is true for mathematical systems theory (UNGER & WYSOTZKI, 1981) and artificial intelligence. Here research goes under the heading of inductive

machine learning (DIETTERICH, LONDON, CLARKSON, DROMEY, 1982; BUNDY, SILVER & PLUMMER, 1985; MICHALSKI, 1987; vanLEHN & BALL, 1987; CARBONELL & LANGLEY, 1987).

The application of basic results of research on human and machine learning to student modelling has just begun. So there are only a few papers on the inductive construction of learner models (GILMORE & SELF, 1988; KAWAI, MIZOGUCHI, KAKUSHO & TOYODA, 1987; SELF, 1986). Results based on implementations of learning algorithms have been reported by KAWAI et al. (1987), LANGLEY, OHLSSON & SAGE (1984) and SLEEMAN (1986).

Work on computational models of plan recognition started to be widely known with the article of SCHMIDT, SRIDHARAN & GOODSON (1978). In contrast to classical work on student modelling, the research on plan recognition was focused on problem solving. Here, we are assuming complex goal hierarchies. These could be used explicitly as a help guiding the problem solving process (MILLER, 1982; MILLER & GOLDSTEIN, 1976, 1977a,b). These authors showed that plan recognition and problem solving could be described by using the framework of parsing. The problem solving grammars for planning purposes is used in a generative way. The student is offered a goal tree. In dialogical situations the student chooses solution paths. Thus the parsing process is inverted to generate proposals and provide guidance in the problem solving process.

Plan recognition as a variation of parsing has a rather short tradition. Though in linguistics it is a standard approach to describe and explain verbal and written behavior by rule-based grammars, early attempts to extend the grammatical approach to nonverbal behavior and problem-solving (POHL, 1973; SKVORETZ, 1984; SKVORETZ & FARARO, 1980) are less known. Nevertheless various studies describing human-computer interaction by taskgrammars have become rather popular in recent years (REISNER, 1981, 1984; HOPPE, TAUBER & ZIEGLER, 1986; GREEN, SCHIELE & PAYNE, 1985; HOPPE, 1987a, b; PAYNE & GREEN, 1986). Similar ideas were applied to problem solving domains (DESMARAIS, LAROCHELLE & GIROUX, 1987; GILMORE & GREEN, 1987; JOHNSON, 1986; JOHNSON & SOLOWAY, 1984; ROSS & LEWIS, 1987).

At the present moment there are only a few attempts to integrate student modelling into plan recognition, though the benefits of an integrative approach are obvious (LLOYD, 1986). It supports the development of adaptive context-sensitive helps and the resolution of ambiguities during the diagnostic process. This includes for example the ability to explain an error by one of several alternative goal-action sequences. The selection of one path should be based on the diagnostic information gathered so far in the student model.

2.3.2.2 Cognitive Design Principles

Up to now we presented some design considerations mainly from an artificial intelligence point of view. This has to be completed by arguments from cognitive science or cognitive psychology. Because there is no single theory about human information processing we restrict ourselves to computational cognitive theories. The most general of them is ANDERSON'S ACT* - theory (ANDERSON, 1983).

On the basis of the ACT* (Adaptive Control of Thought) - theory ANDERSON, BOYLE, FARRELL & REISER (1987) propose eight design principles which are relevant for the construction of intelligent tutoring systems. Some of them have a rather solid empirical basis.

The underlying ACT*-theory (ANDERSON, 1983) can be subsumed under a few main principles:

- 1) Human cognition can be hypothesized to work as pattern-action sequences which can be modelled on a symbolic description level by goal-triggered rules (production systems).

- 2) The productions operate on the content of a working memory with limited capacity.
- 3) Procedural learning takes place as production strengthening and knowledge compilation (NEVES & ANDERSON, 1981; ANDERSON, 1986).

The later process can be subdivided into proceduralization and composition. Proceduralization is some kind of instantiation of parameters in production-rules thus producing specialized rules. Composition involves the building of macro-operators out of a number of successful single operators.

The derived principles are:

1. Represent the Student as a Production Set

Some arguments in favor of production systems are: 1) The ideal and the current student should be modelled by process models, so that at each instance in time the deviation from the desired state is measurable. 2) Each production is a package of knowledge that can be communicated easily. 3) Student misconceptions and bugs can be organized as production rules which are perturbation of correct rules.

2. Communicate the Goal Structure Underlying the Problem-Solving

According to ACT* and other cognitive theories successful problem-solving behavior is organized as AND/OR trees of goals and subgoals. ANDERSON stresses the observation that traditional textbooks do not provide sufficient information about the goal structure and the search processes necessary to solve the problem.

3. Provide Instruction in the Problem-Solving Context

ACT* hypothesizes that production compilation only takes place during problem solving. They cannot be learned in the abstract. Thus, instructions are more effective for the learning process, when the student is forming the productions. A similar line of argumentation was put forward by vanLEHN (1987). Instructions and helps are most effective, when they are provided at the time of impasse in the problem solving process. Another reason for the importance of the context is the human storage mechanism of packing and retrieving information in episodes (TULVING, 1983).

4. Promote an Abstract Understanding of the Problem-Solving Knowledge

The general finding is that students encode concrete knowledge far more easily than abstract knowledge. But in the case of generality the productions should be parameterized to have abstract principles at hand.

5. Minimize Working Memory Load

Working memory errors degrade the speed of learning and restrict the amount of learning. To a high percentage errors of novices can be explained as working memory errors.

6. Provide Immediate Feedback on Errors

This design principle is in accordance to classical learning theory (BILODEAU, 1969; SKINNER, 1958) and repair theory (BROWN & Van LEHN, 1980; Van LEHN, 1983a,b, 1987). According to repair theory an impasse occurs, when the problem-solving skills fail to propel the solution. If the application of general weak domain independent heuristics (repairs) also fails, the student needs help and feedback.

7. Adjust the Grain Size of Instruction with Learning

The productions in the student model should define the grain size of the instructions. Because of the compilation process, this grain size is changing permanently. This argument is also supported by a different production based learning theory which is based on the chunking concept (NEWELL & ROSENBLOOM, 1981; LAIRD, ROSENBLOOM & NEWELL, 1986).

8. Facilitate Successive Approximations to the Target Skill

As nobody gets an expert by solving one problem of the task domain a careful sequence of training lessons has to be designed, so that the production set can be compiled, augmented and tuned in a smooth way. The construction of a production-based training sequence was demonstrated by vanLEHN (1987a).

2.3.3 Summary

There is general agreement that it is metaknowledge that makes an tutor "intelligent". This metaknowledge schedules what, whom, how to teach. To include this knowledge we need expert and teacher knowledge and knowledge about the mental state of the student. The diagnosis of the student has been done through modelling procedural skills and goal hierarchies. Though the construction of a tutor has some engineering phases, design decisions have to be restricted by cognitive principles which have a rather strong empirical evidence. So the development of an ICAI system has to be the result of an interdisciplinary approach.

3. ICAI in the Domain of Computer Programming

In recent years many new ICAI systems appeared on the scene. The relevant american literature is compiled in KEARSLEY (1987), LAWLER & YAZDANI (1987) and WENGER (1987). European authors describe their work in MANDL & FISCHER (1985) and SELF (1988).

In the following we will focus our attention on the domain of computer programming which is a well established research area in ICAI. We only discuss systems which have a strong relevance for our own project ABSYNT.

3.1 Problem Solving Monitor: From PHENARETTE to The PROgram Understander for Students (PROUST)

We will now review some work which runs under the title of help systems or problem solving monitors. The aim of these systems is to give more specific, detailed and "intelligent" feedback on programmer's errors than an ordinary compiler or interpreter could do (BACKHOUSE, 1979, Ch.5,6; EFE, 1987).

One of these systems is WERTZ' PHENARETTE (WERTZ, 1982, 1985, 1987) which has become quite wellknown though it was originally implemented in France. The program debugs LISP code with the help of syntax and semantic "specialists", which are represented by heuristic rules. Though these heuristics contain general programming knowledge, no attempt is made to include knowledge about the problems or the intentions of the programmer. This is the reason why WERTZ (1982) calls PHENARETTE a stereotyped debugging aid which could in principle be a backend of every LISP interpreter or compiler. Thus depth of understanding programs was sacrificed in favor of general applicability of the system.

The next step in the direction of deeper understanding of programs was made by the development of PROUST (JOHNSON, 1986; JOHNSON & SOLOWAY, 1985, 1987). It was designed to help novice programmers find and understand bugs in PASCAL programs, which are usually written in an imperative style. PROUST does not teach directly, but gives feedback, so that students are encouraged to formulate ideas, see relationships, draw conclusions and discover their own misconceptions. This is in complete agreement with ANDERSON's design principles 3 and 6. In contrast to PHENARETTE PROUST finds only nonsyntactic bugs. These semantic bugs are partly due to working memory errors and partly due to

planning errors. It determines how the bug could be corrected and even suggests why the bug arose in the program and the problem context. To cope with various types of errors PROUST contains three knowledge bases: a description of the problem and its requirements, general programming knowledge (programming schemes or stereotyped plans) and situation-specific intentions of the programmer (goals). PROUST parses the syntactically correct blueprints of the student and generates from the requirement (goal) structure of the problem description a proposal for a plan structure which can be transformed to PASCAL code. The predicted code is matched against the actual code which was originated by the student. Discrepancies are explained with bug rules. Natural language explanations and recommendations are put forward after the diagnosis phase.

In contrast to PHENARETTE PROUST puts great emphasis on plans and intentions of the programmer. The diagnosis of both latent constructs seems to be very sophisticated because the authors report high error recovery rates on nonselected PASCAL programs. But because both systems are not teaching explicitly they are not very much concerned with teaching sequences and the building of knowledge structures. So they are lacking individualized explanations for errors, which can only be derived from the learning history of the person and their formalization in a background student model. Such a model could collect (like a special short term memory) data and hypotheses about the students previous problem solving episodes (WEBER, WALOSZEK & WENDER, 1988).

3.2 An ITS System: ANDERSON's LISP Tutor

The development of PHENARETTE and PROUST has shown how much work has to be conducted in developing a high quality debugging environment. This is absolutely indispensable for an ITS teaching a complete non-toy programming language. So only a few groups are known for taking such an endeavor.

In Germany there is only one ITS for a complete programming language (LISP) in progress (WALOSZEK, WEBER & WENDER, 1986; KÖHNE & WEBER, 1987; WEBER, WALOSZEK & WENDER, 1988; WALOSZEK, WEBER & WENDER, in press). In the United Kingdom the favourite language is PROLOG (ROSS, 1987; RAJAN, 1987).

Quite wellknown and even obtainable as a commercial system (BOYLE, 1986) is ANDERSON's LISP tutor (ANDERSON, 1987; ANDERSON, BOYLE, FARRELL & REISER, 1987; ANDERSON & REISER, 1985; ANDERSON & SKWARECKI, 1986). The tutor teaches a full semester course in LISP and is field tested at the Carnegie-Mellon University since 1984. Its theoretical foundations were ANDERSON's cognitive theory ACT* (ANDERSON, 1983a; ANDERSON, BOYLE, FARRELL & REISER, 1987) and several empirical psychological studies of LISP programming (ANDERSON, 1983b; ANDERSON, FARRELL & SAUERS, 1982, 1984). Though there exists much published work describing the tutor, the construction of the system is not as transparent as the design of PROUST. Therefore we do not want to reconstruct the architecture of the tutor. Instead of that we will analyze the flow of dialog communicating knowledge between the tutor and the user in chapter 4. We think that this is more important from a cognitive science perspective than the analysis of the software layers.

The tutor teaches 18 lessons ranging from 'Basic LISP Functions' (lesson 1) over 'Integer Based Recursion' (lesson 7) to 'Advanced Topic: Implementing Production Systems' (lesson 18). Concerning instructions ANDERSON (1985, p.164) writes:

"Each topic involves a small instructional booklet and many problems practicing the skills taught in that lesson. Our goal in designing these booklets was to keep the written instruction to a minimum. There is considerable evidence that written technical instruction is most effective when it is brief."

This instructional material which is used to build up the knowledge structure has been published in modified form as a textbook (ANDERSON, CORBETT & REISER, 1986).

4. Examples For Instructions and Helps in Teaching Programming

We agree with FORD (1987) and PEACHEY & McCALLA (1986) that the emphasis of ICAI research on student models has impeded the research on other more tractable goals. Our first goals should be concerned with making teaching strategies and tactics less ad hoc. According to FORD teaching strategies involve decisions which have to be made at each cycle of the student-computer interaction. Each decision determines the next communication and learning steps. The decisions have to be derived by educational "philosophies" like discovery learning or learning by instruction. Even student modelling (CLANCEY, 1987) should not be seen independent of the strategies of teaching which should be seen as a concept with a higher priority.

In ANDERSON's LISP tutor there seems to be nothing that could be called a model of the learning history of the individual student though ANDERSON et al. use the terms "current student", "ideal student" and "model tracing" (ANDERSON, BOYLE, FARRELL & REISER, 1987). The last concept subsumes the four features of his tutoring methodology (ANDERSON, 1987, p. 443):

1. The tutor constantly monitors the student's problem solving and provides direction whenever the student wanders off path.
2. The tutor tries to provide help with both the overt parts of the problem solution and the planning. However, to address the planning, a mechanism had to be introduced in the interface (in this case menus) to allow the student to communicate the steps of planning.
3. The interface tries to eliminate aspects like syntax checking, which are irrelevant to the problem-solving skill being tutored.
4. The interface is highly reactive in that it does make some response to every symbol the student enters.

This strategy is accomplished by a 1000 production rules. 40% of these rules model correct generation of LISP code (the expert model) and 60% model various bugs, which were collected by empirical observation of student errors. The first rough copy of a student's program is parsed by the production rule system. A tutorial rule is associated with each buggy or correct production. If it seems that the student has hypothetically "used" a production, the appropriate tutorial rule can be triggered. Also, if a student shows an impasse, the system can automatically determine the next step and an appropriate tutorial rule.

ANDERSON described some tutorial implications which resulted from his research. The more important ones for our work are the advices (ANDERSON, 1987, p. 454):

"In addition to basing instruction on a production-system analysis and emphasizing communication of goal structures, there are a number of other general recommendations to make about the design of instructional activities. It makes sense to provide the instruction in the context where it should be used to maximize the probability that the student will retrieve and try to use the knowledge. Also, it makes sense to provide that knowledge in a form that can be most easily used by weak methods. For instance, we try to fashion our instruction to take the form of rules for means-ends solutions or of examples for use by analogy."

We will now look at the realization of these statements. We quote an original dialog with the tutor. This dialog is published identically type-set in three independent articles (ANDERSON & REISER, 1985; ANDERSON & SKWARECKI, 1986; ANDERSON, 1987) so that the possibility of misprints is ruled out.

4.1 A Dialog with ANDERSON's LISP Tutor

The dialog represents (appendix C), as ANDERSON points out, the linearized teletype version of the original dialog. The original dialog takes place on a screen split into three dialog windows, which are reserved for instructions, code of the student and plan dialogs. The output of the tutor is given in normal type while the student's input is printed in bold characters.

Before the first line appears some interaction has already taken place. The dialog starts with an instruction, how the factorial is to be computed. We want to criticize two things. First, all parentheses are left out so that the definition is wrong (!), and second, the instruction gives a definition which makes the student induce an iterative solution.

After the student read the instruction "Define the function fact", he typed into the machine " (defun ". The tutor responded with the template "(defun <name> <parameters> <body>)". Then the student filled the <name> slot with "fact" and the parameters slot with "(n)" and started to fill in the <body> slot with "(cond ((equal)". At this moment a new template is offered to the student. The rest of the template "<action>) <recursivecase>)" can be seen in episode 1. The student has forgotten to provide arguments for equal. At that moment the buggy production

"IF the goal is to test if a value is equal to zero

THEN use the function EQUAL and set as subgoals to code the value and zero"

matches with its right side the student's code (ANDERSON & SKWARECKI, 1986, p.845). The tutor issues the natural language comment "I assume that writing a predicate." which is attached to the buggy production. There is a production whose left side matches the same goals the buggy production matches, but which could generate correct code (ANDERSON, 1987, p.445):

"IF the goal is to test if *arg1* is zero

THEN use the LISP function ZEROP, and set a subgoal to code *arg1*."

We think that the student had the correct but somewhat inefficient idea to use the function equal. The explanation why zerop is the better choice is not sufficient.

In line episode 3 the tutor enters a planning dialog with the student. The tutor offers a menu with two distractors that are answer categories which are not to be considered correct. The student chose the third category which is correct (!) according to the instruction given before line 1. But the tutor squeezes the student to program a recursive solution. So a subdialog with

some concrete examples is entered. What we see here is a mixture of notations. The column headings of the EXAMPLES are written in the mathematical notation $fact(n)$, but the table entries are function calls in LISP! The same notational confusion can be seen in the definition of the factorial function. The language SCHEME shows how to avoid this kind of terminological mix up.

Later on in the "IF YOU WANT TO:" menu the tutor does not use the operator "minus" in phrases like "n minus 1" as he did before but introduces the predicate "less". The student is for instance offered option 2 "Multiply n by fact of one less than n". We know from the psychology of memory that if the concept LESS is primed, then similar concepts are activated. But, if the student would later use the LISP function LESS he would get an error message. The same is true when he uses the LISP function MINUS, which is used for multiplication with -1.

We do not want to pursue the dialog any further because it has become obvious that the information the tutor gives may be confusing to a novice. It shows that the instructional component of the tutor is suboptimal and does not show compatibility with ANDERSON's own eight cognitive design principles.

4.2 The Description of an Abstract Machine: the "Calculation Sheet" Machine

In this section we will present other pieces of instructional texts, whose purpose is to explain the operational semantics of an abstract machine. This machine is used in a modified form in our ABSYNT (Abstract Syntax Trees) project (COLONIUS, FRANK, JANKE, KOHNERT, MÖBUS, SCHRÖDER, THOLE, 1987). The notion "abstract syntax" is used to define the program structure relevant for translation, interpretation and transformation (AHO; SETHI & ULLMAN, 1986; DOSCH, 1984, p.154; PAGAN, 1981). Our goal is to implement the problem solving monitor ABSYNT as a functional visual programming language. In the rest of the paper we want to show how to use informal texts (BAUER & GOOS, 1982³, p. 110 - 116; BERGHAMMER, DOSCH & OBERMEIER, 1985, p.55; DOSCH, 1984, p.163f;) and diagrams (SCHMITT-WOHLFARTH, 1978; BAUER & GOOS, 1982, ch. 2) to develop visual instructions and a graphical help system. This will be used in our problem solving monitor according to ANDERSON's cognitive design principles and in agreement with BROWN & vanLEHN's repair theory when impasses in the problem solving behavior of the learner occur.

This means that we formalize the operational semantics of the machine as a set of production rules which are represented visually as iconic rules. We will show that the specification process of the iconic rules should be guided by principles derived from cognitive psychology and cognitive science.

There is a rather long tradition in using artificial or abstract machines and diagrams for the demonstration of logical and computational processes (GARDNER, 1982²). The idea to write the result of computations onto sheets of paper which show a spatial arrangement in form of trees was introduced in school books by SCHMITT-WOHLFARTH (1978). The idea of using trees for computations is ubiquitous in computer science (ABELSON, SUSSMAN & SUSSMAN, 1985; BAUER & WÖSSNER, 1981; 1984; KANTOROVIC, 1957; LUTZE, 1987) and even rather well known in psychology (EGAN & GREENO, 1974; PAVEL, MARCOVICI, SHERMAN & FALMAGNE, 1983). The abstract syntax of purely applicative expressions can graphically be represented by KANTOROVIC trees (DOSCH, 1984, p.160).

Our purely functional tree-like programming language is a toy language compared to SCHEME or PASCAL. In spite of this the language is powerful through its facilities which allow abstraction and chunking of thought processes (GREEN, 1980, 1983; GREEN & PAYNE, 1984). The reason for the preference of a "small" language is motivated by the intention to use the programming environment ABSYNT as a testbed for research in planning diagnostics, help messages, student modelling and learning research.

An English description of the "calculation sheet" machine appeared in DOSCH (1984, p.163f):

"The course of computation is - up to the freedom in the evaluation - fixed by the calculation sheet associated with an expression. If in expressions additionally calls of (recursive) routines occur, this computational model can be extended to the **calculation sheet machine**.

For every routine there is a supply of calculation sheets for the expression forming its body. Assuming a call-by-value semantics for routines, upon a call the argument values are entered into a new sheet. Every such form is called an incarnation of the routine. In general, of course, the next (recursive) routine call occurs before the calculation on the present sheet is finished; this leads to the notion of pending operations. When the computation terminates on a sheet, the result value is transferred back to the previous sheet, that is, to the dynamic predecessor. There pending operations are evaluated, and so on. In the sequential conditional, after calculating the condition the not chosen branch is truncated. Note that this leads to the termination of recursive routines."

The German description of the machine and various diagrams from the university level computer science textbook (BAUER & GOOS, 1982³) are included in **appendix D**.

Empirical studies (COLONIUS, FRANK, JANKE, KOHNERT, MÖBUS, SCHRÖDER & THOLE, 1987; SCHRÖDER, FRANK & COLONIUS, 1987) have shown that these informations have to be modified a great deal, if computer novices are to gain sufficient knowledge about the behavior of the machine so that they are able to predict it. It has to be assured that prediction errors are not attributable to misinterpretations of the instructional material.

5. The Design of our Reactive Graphical Programming Environment ABSYNT

5.1 Psychological Arguments in Favor of a Functional Visual Programming Language

The main research goal of ABSYNT is the construction of a problem solving monitor (PSM). Some PSM-relevant research has been reported about solving problems in simple arithmetic tasks (ATTISHA, 1984; ATTISHA & YAZDANI, 1983; BROWN & BURTON, 1978; BUNDY, 1983; BURTON, 1982; VanLEHN & BROWN, 1980; YOUNG & O'SHEA, 1981), in quadratic equations (O'SHEA, 1979, 1982), in simple algebra problems (SLEEMAN, 1982, 1983, 1984, 1985, 1986), in geometry (ANDERSON, 1983c; ANDERSON, BOYLE, FARRELL & REISER, 1987; ANDERSON, GREENO, KLINE & NEVES, 1981) and in computer programming (ANDERSON, 1983b, 1987; ANDERSON, FARRELL & SAUERS, 1982, 1984; ANDERSON & REISER, 1985; ANDERSON & SKWARECKI, 1986; JOHNSON, 1986; JOHNSON & SOLOWAY, 1985, 1987; SOLOWAY, 1986; WERTZ, 1982, 1985, 1987).

We chose the domain of computer programming because problem solving is the main activity of each programmer. Furthermore, errors can be diagnosed easily. We had to make some more design decisions. Because the PSM should mainly supervise the planning processes of the programmer, we decided to use a simple programming language, the syntax and semantics of which can be learned in a few hours. We decided to take a purely functional language. From the view of cognitive science functional languages have some beneficial characteristics. So less working memory load on the side of the programmer is obtainable by their properties, referential transparency and modularity (ABELSON, SUSSMAN & SUSSMAN, 1985; GHEZZI & JAZAYERI, 1987²; HENDERSON, 1980, 1986). Furthermore, there is some evidence that there is a strong correspondency between programmer's goals and use of functions (PENNINGTON, 1987; SOLOWAY,

1986; JOHNSON & SOLOWAY, 1985, 1987). So we avoid the difficult problem of interleaving plans in the code which show up in imperative programming languages because it makes the diagnosis of programmer's plans rather difficult (SOLOWAY, 1986). If we take for granted that a goal can be represented by a function, we can gain a great flexibility in the PSM concerning the programming style of the student. We can offer him facilities to program in a bottom-up, top-down or middle-out style. The strategy of building up a goal hierarchy can correspond to the development of the functional program.

There are some similar psychological reasons for the use of a visual programming language, too. There is some evidence that less working memory load is obtainable through the use of diagrams if they support encoding of information or if they can be used as an external memory (FITTER & GREEN, 1981; GREEN, SIME & FITTER, 1981; PAYNE, SIME & GREEN, 1984; LARKIN & SIMON, 1987). Especially if we demand the total visibility of control and data flow the diagrams can serve as external memories.

The diagrammatic structuring of information should also reduce the amount of verbal information which is known to produce a higher cognitive processing load than "good" diagrams (LARKIN & SIMON, 1987). "Good" diagrams produce automatic control of attention with the help of location objects. These are in our case object icons, which are made of two sorts: straight connection lines and convex objects. Iconic objects of these types are known to control perceptual grouping and simultaneous visual information processing (POMERANZ, 1985; CHASE, 1986).

5.2 Computer Science Based Design Guidelines for Visual Languages

Work on the design of visual languages has just started twenty years ago (e.g. LAKIN, 1980), but some results have been obtained which are not controversial among scientists. Visual programming languages can be described by a profile in a three-dimensional system according to 1) visual extent, 2) scope and 3) language level (SHU, 1986). A language has a high visual extent if graphics are not mere illustrations but play a central role in programming. They must be "executable graphics" (LAKIN, 1986). The scope of a language is a measure of the generality of their applicability. The language level gives a hint how abstract and hardware independent the constructs of the language are.

The design of a visual language has to be based on the concept of generalized icons (CHANG, 1987), which are dual representations of abstract and visual parts. The type of generalized icons can be divided into object icons and process icons. Object icons define the representation of static language constructs, whereas process icons specify the representation of dataflow and controlflow. We want to quote CHANG (1987, p.9f) on this subject:

An iconic system is a structured set of related icons. A complex icon can be composed from other icons in the iconic system, and therefore express a more complex visual concept. **An iconic sentence** ... is a spatial arrangement of icons from an iconic system. **A visual language** is a set of visual sentences constructed with given syntax and semantics. **Syntactic analysis of visual language** (spatial parsing) is the analysis of the spatial arrangement of icons (i.e. visual sentences) to determine the underlying syntactic structure. Finally, **semantic analysis of visual language** (spatial interpretation) is the interpretation of a visual sentence to determine its underlying meaning.

From the view point of system implementation, to design an iconic system and a visual language, two major software tools are required: a) an iconic editor to edit a generalized icon; and b) an icon interpreter to perform syntactic analysis and semantic analysis of the visual system."

Similar ideas stem from GLINERT & GONCZAROWSKI (1987) and GLINERT, GONCZAROWSKI & SMITH (1987). CHANG (1986) proposed a specification cycle in the language design (Figure 1). We used this cycle for our design.

The abstract parts of the language were specified in PROLOG according to some ideas of (PEREIRA, 1986) as a runnable specification (DAVIS, 1982). The corresponding visual parts were specified obeying results of our own empirical research or generalizing findings and standards from cognitive psychology and cognitive science.

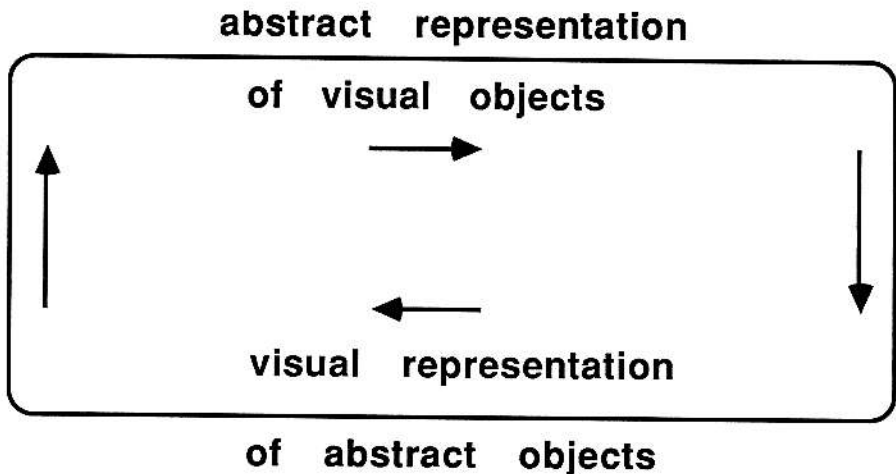


Figure 1: The Specification Cycle of a Visual Language (CHANG, 1986)

The complete programming environment is implemented in INTERLISP and the object-orientated language LOOPS (JANKE & KOHNERT, 1988; KOHNERT & JAHNKE, 1988) to have a system with direct manipulation capabilities which is an absolutely necessary prerequisite for our system (FÄHNRIK & ZIEGLER, 1985a,b; HUTCHINS, HOLLAN & NORMAN, 1986; SHNEIDERMAN, 1983, 1987). Following SHU's dimensional analysis, ABSYNT is a language with high visual extent, low scope and medium level.

5.3 Cognitive Science and Psychology Based Guidelines for the Design of Graphical Objects and Diagrams

Though through decades research in psychology, physiology and computer science has been devoted to human, animal and machine perception, results which could guide decisions in building tutorial applications have remained on a rather informal level of "gestalt laws" or phenomenological principles (BERTIN, 1981, 1983; CAMPBELL & ROSS, 1987; CLEVELAND, 1985; FITTER & GREEN, 1979; LUTZE, 1987; TUFTE, 1980; WOOD & WOOD, 1987).

In our work we relied on empirical studies partly done by others (e.g. WEBER & KOSSLYN, 1986) and partly conducted by ourselves. The former gave us hints concerning the synchronisation of the properties of graphics and the mental imagery system. The latter dealt with the memory representation of the tree programs (SCHRÖDER, COLONIUS & FRANK, 1987) and errors which resulted from misinterpretations of the syntax and the semantics of the original language as appeared in BAUER & GOOS (1982). The last version of the language which is used for ABSYNT was strongly influenced

by an empirical study of POMERANTZ (1985) and a theoretically orientated article by LARKIN & SIMON (1987). POMERANTZ made some careful studies about selective and divided attention information processing. One consequence for our design was that time-indexed information had to be spatial indexed by locations, too. Information with the same time index should have the same spatial index. This means that this information should appear in the same location. In our design a location is identical with a visual object. These insights were supported by the formal analysis of LARKIN & SIMON (1987). They showed under what circumstances a diagrammatic representation of information consumes less computational resources as an informational equivalent sentential representation.

5.4 The Iterative Specification Cycle for the Derivation of Iconic Objects and Iconic Rules Concerning the Operational Semantics of ABSYNT

Though our main research goals lie in the exploration and debugging of planning processes we have to deal with the computational knowledge of the programmer, too. It is our opinion that a user of our language should have sufficient knowledge about the interpreter so that he/she is able to predict the set of the successor states from knowledge of the current state. To get and maintain this expertise we have to implement an instructional component and a help system. The specification of the operational knowledge was made in an iterative specification cycle (MÖBUS, 1987a,b,c) (Figure 2).

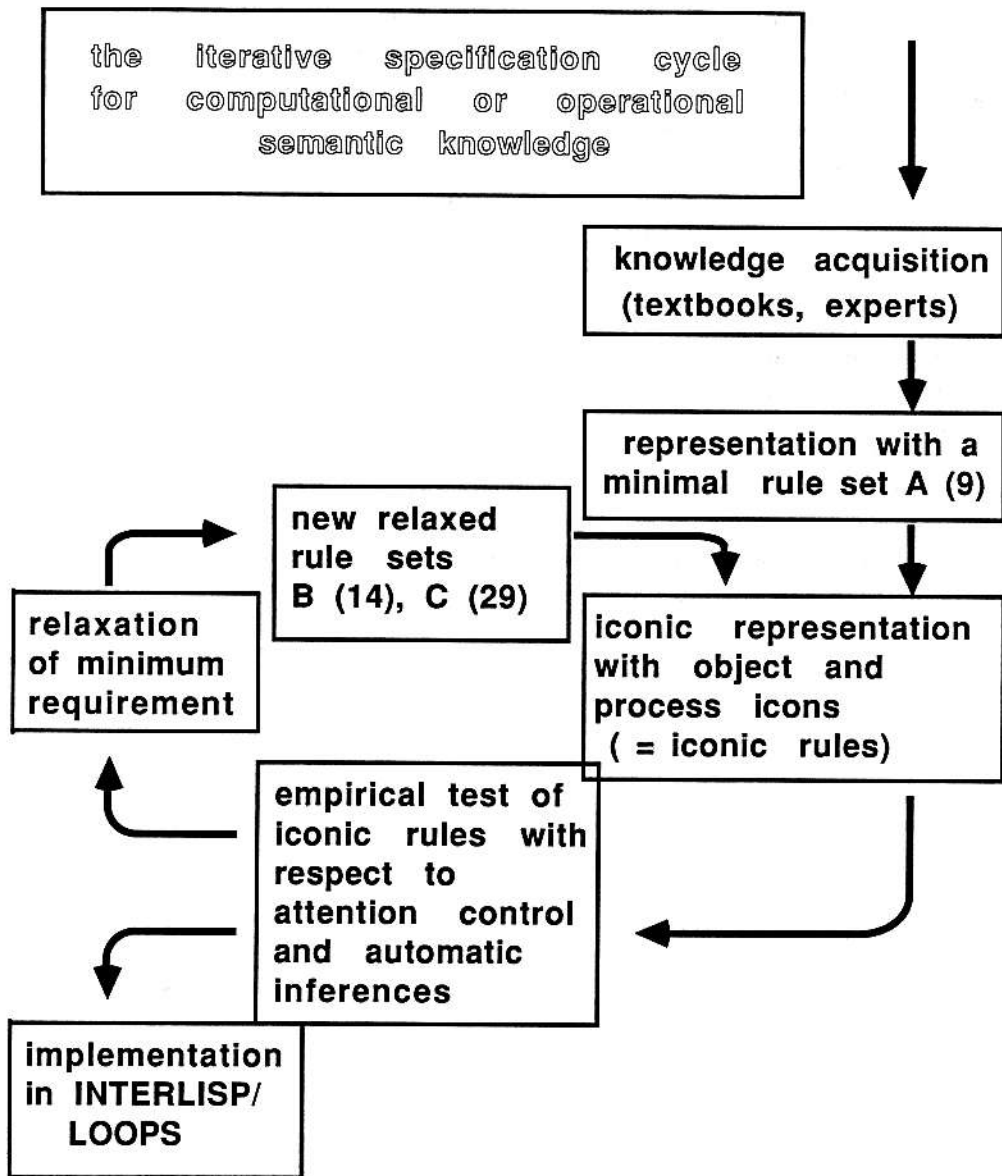


Figure 2: The Iterative Specification Cycle for Computational or Operational Semantic Knowledge

The first step consisted of the knowledge acquisition phase. The next step led to a ruleset A of 9 main Horn clauses (plus some operator-specific rules). The set contained the minimal abstract knowledge about the interpretation of ABSYNT programs. The abstract structure of a program was formalized by a set of PROLOG facts similar to an approach of GENESERETH & NILSSON (1987, ch. 2.5).

Then we tried an iconic representation of the facts and Horn clauses. Soon we realized that a visual representation according to the recommendations of LARKIN & SIMON (1987) was only possible, if we "enriched" the iconic structure. This means that we had to add iconic elements which were not present in the abstract structure. This led to an iconic structure which remained unchanged and was used as the interface of the programming environment (FIGURE 3).

Problems occurred if we wanted to keep the number of iconic objects fixed during a computation of a recursive program. The postulate of total visibility led to a visual trace with an information overload (FIGURE 6). Time indexed information was not location indexed. So selective attention according to POMERANTZ (1985) was not possible: computational errors were inevitable.

This forced us to relax our requirement to use only a minimal number of object icons. We came up with a relaxed rule set B with 14 main rules (plus operator-specific rules).

The behavior of these rules led to a new visual trace. Time indexed information was now location indexed so that undesired perceptual grouping could not occur. But computational goals and intermediate results were kept visible only as long as they were absolutely necessary for the ongoing computation.

Empirical considerations showed that the programmer had to reconstruct former computations mentally, because their result disappeared from the screen. So we had to relax the minimum assumption a second time and introduce even more visual redundancy. This was e.g. in accordance with the third principle of FITTER & GREEN (1979).

But there were some other reasons which influenced the decision to modify the ruleset a third time. First, rules were still recursive. This leads in computations to pending rules. The derivation of instructions from recursive rules forces a higher working memory load because of the mental maintenance of a goal stack with return points. Second, if we had derived iconic rules from the abstract rule set B we would have gotten two disjunctive rules. But there is a fair amount of experimental evidence that for humans conjunctive rules are easier to process than disjunctive rules (BOURNE, 1974; HAYGOOD & BOURNE, 1965; MEDIN, WATTENMAKER & MICHALSKI, 1987). So we decided to avoid disjunctive iconic rules.

A third ruleset C was developed with 29 (plus operator specific) rules. Now there was even more redundant iconic information on the screen. This computational behavior was "frozen" in our INTERLIPS/LOOPS implementation.

5.5 The Programming Environment of ABSYNT

The programming environment (KOHNER & JANKE, 1988) as the result of our specification cycle is shown in figure 3. The screen is split into several regions. On the right and below we have a menu bar for nodes. A typical node is divided into three stripes: an input stripe (top), a name stripe (middle) and an output stripe (bottom). These nodes can be made to constants or variables (with black input stripe) or a language supplied primitive operators or user defined functions.

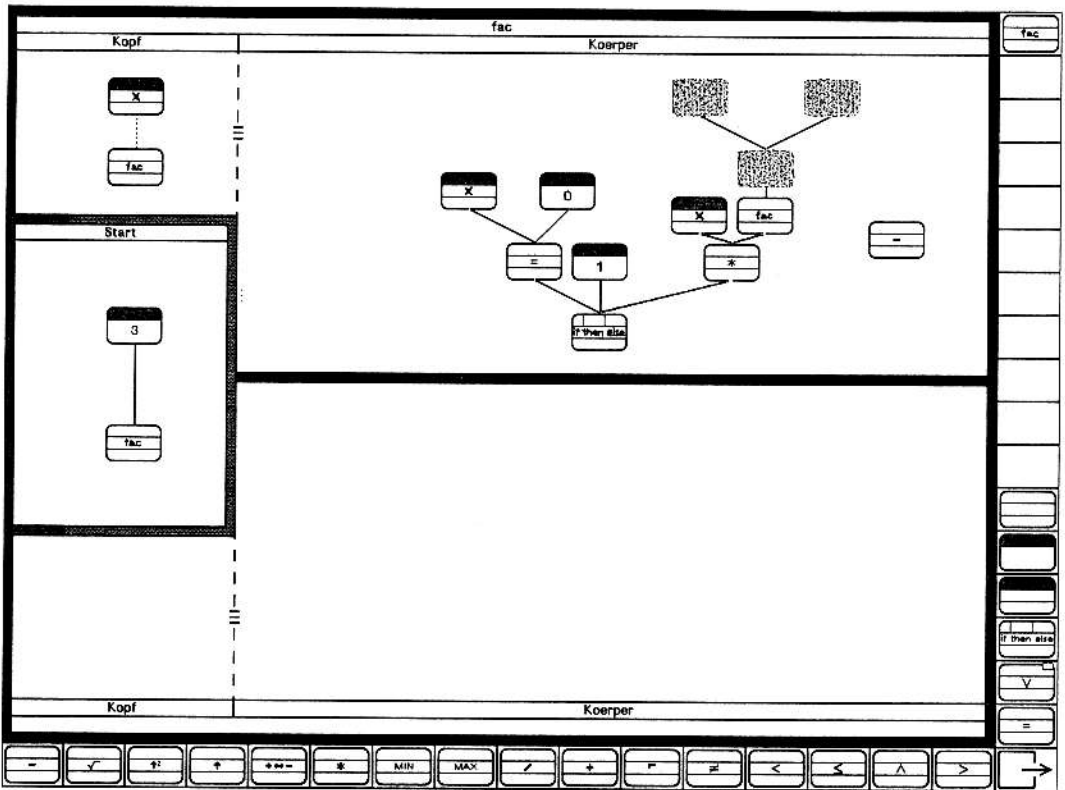


Figure 3: The programming Environment of ABSYNT

The programmer sees in the upper half of the screen the main worksheet and in the lower half another one. Each worksheet is called frame. The restriction on two visible worksheets led to corresponding psychologically motivated restrictions in the computational freedom of the interpreter (5.7.2). The frame is split into a left part: "head" (in german: "Kopf") and into a right part "body" (in german: "Körper"). The head contains the local environment with parameter-value bindings and the function name. The body contains the body of the function.

Programming is done by making up trees from nodes and links. The programmer enters the menu bar with the mouse, chooses one node and drags the node to the desired position in the frame. Beneath the frame is a covered grid which orders the arrangements of the nodes so that everything looks tidy. Connections between the nodes are drawn with the mouse. The connection lines are the "pipelines" for the control and data flow. If a node is missed the programmer is reminded with a phantom node that there is something missing. The editor warns with flashes if unsyntactic programs are going to be constructed: crossing of connections, hiding of nodes etc. The function name is entered by the programmer with the help of pop-up-menus in the root node of the head and the parameters in the leaves of the head.

If the function is syntactically correct, the name of the function appears in the frame title and in one of the nodes in the menu bar so that it can be used as a higher operator. When a problem has to be solved a computation has to be initialised by

the call of a function. This call is programmed into the "Start"-Tree. Initial numbers are entered by pop-up-menus in constant nodes in the start tree. This tree has a frame without a name, so that the iconic bars are consistent.

5.6 The integration of rules in the architecture of an PSM or ITS

A very crucial point concerning the "intelligence" of an PSM lies in the quality of the design for the feedback system. In literature two approaches have been proposed. One proposal is the explicit "debugging" approach (BURTON, 1982; VanLEHN, 1981): tracing an error with the help of a diagnostic procedure and an extensive bug collection back to underlying malrules or misconceptions. The other idea rests solely on the specified expert knowledge and a model of human learning (EGAN & GREENO, 1974; SIMON & LEA, 1974; ANDERSON, 1983; VanLEHN, 1987a,b). According to these rule-based theories of human skill acquisition a learner has to be aware of at least two types of information: the current goal within the problem and the conditions under which rules apply. McKENDREE (1987) could show in three experiments, that "goal" information is even more important than "condition" information in promoting learning of skill. This type of feedback design is more simple to implement than the "debugging" strategy. But there are still no experimental comparisons between the two methods.

Either way, we have to specify goals and rules an expert would use when predicting the computational behavior of the ABSYNT interpreter. So in the last part of our paper we show how we try to achieve the design of iconic rules and visual helps.

When should the tutor administer feedback? Our tutorial strategy is guided by "repair theory" (BROWN & VanLEHN, 1980) and follows the "minimalist design philosophy" (CARROLL, 1984a,b).

This means, that if the learner is given less (less to read, less overhead, less to get tangled in), the learner will achieve more. Explorative learning should be supported as long as there is preknowledge on the learner side. Only if an error occurs feedback becomes necessary and information should be given for error recovery.

According to repair theory an impasse occurs, when the student notices that his solution path shows no progress or is blocked. In that situation the person tries to make local patches in his problem solving strategy with general weak heuristics to "repair" the problem situation. In our tutorial strategy we plan to give feedback and helps only, when this repair leads to a second error.

5.7 The Genealogy of Rule Sets Concerning the Operational Semantics of ABSYNT

5.7.1 The First Rule Set A: a Minimal Interpreter in PROLOG for ABSYNT Programs

The specification cycle (Figure 2) led to the first ruleset A. The program is described abstractly by a set of nodes and a set of connections which are represented by PROLOG facts. The nodes possess the attributes frame-name, tree-type, instance-number, name and value. These attributes determine the location, the within structure and the value of the node.

The connections possess the attributes frame, tree, out-instance, in-instance and input-number. They link the outputfield of a node with the inputfield of another node.

Semantic knowledge is moulded into two types of rules. One consists only of one "input" rule and the other of several "output" rules. The "input" rule (Figure 4) contains the knowledge about the migration of computation goals and data between the nodes. The "output" rules contain the knowledge about computations within one node. Because the nodes have different meanings, we need different "output" rules. There is one for each primitive operator, one for the parameters in the tree "head", one for constant nodes, one for parameter nodes in the tree "body", one for the root in the tree "head" and one for the computation of higher (self defined) operators. In the last rule parameters are bound in a parallel fashion to their arguments (call by value) and the new leaves of the tree "head" are put onto the stack. Furthermore we have rules which contain the knowledge to generate roots and leafs or to check nodes with respect to their root or leaf status.

```

input(frame(Frame),tree(Tree),instance(Instance),inputno(Inputno),value(Value))
:-      connection(frame(Frame),tree(Tree),out_inst(Out_inst),in_inst(Instance),in_inst_no(Inputno)),
         output(frame(Frame),tree(Tree),instance(Out_inst),name(Name),value(Value)).

/*      IF      there is the goal to compute the value of the input with number Inputno in node Instance in the
              tree Tree in the frame Frame,
      THEN     there is a subgoal to look for a connection, which leads to this input from a yet unknown node
              Out-inst, which is the source of this connection
      AND      there is another subgoal to compute the value of the node Out-Inst
              (this value is then the value of the goal in the IF part of this rule).          */

```

FIGURE 4: The Abstract Input Rule

As a further example we include the "output"-rule for a higher operator (FIGURE 5). This rule describes the call-by-value mechanism.

```

output(frame(Frame),tree(Tree),instance(Instance),name(Name),value(Value))
:-   node_name(frame(Frame),tree(Tree),instance(Instance),name(Name)),
    findall(Argument,input(frame(Frame),tree(Tree),instance(Instance),
        inputno(Inputno),value(Argument)),List_of_arguments),
    set_of(Parameter,(leaf(frame(Name),tree(head),instance(Inst_leaf)),
        node_name(frame(Name),tree(head),instance(Inst_leaf),name(Parameter))),
        List_of_parameters),
    forall(parm_arg_pair(Parm,Arg,List_of_parameters,List_of_arguments),
        (node_name(frame(Name),tree(head),instance(Inst_parm),name(Parm)),
        asserta(node(frame(Name),tree(head),instance(Inst_parm),name(Parm),value(Arg))))),
    root(frame(Name),tree(head),instance(Inst_root_head)), ! ,
    output(frame(Name),tree(head),instance(Inst_root_head),name(Name),value(Value)),
    forall(parm_arg_pair(Parm,Arg,List_of_parameters,List_of_arguments),
        (node_name(frame(Name),tree(head),instance(Inst_parm),name(Parm)),
        retract(node(frame(Name),tree(head),instance(Inst_parm),name(Parm),value(Arg))))),!).

/*   IF   there is the goal to compute the output value of a higher operator node,
    THEN  the following subgoals have to be solved:
        - determine the node name
        - compute all input values of the node
        - determine all parameters of the frame whose name is identical to the node name
        - put the parameter-argument bindings into the new local environment
        - find the head root of the frame
        - compute the output value of the head root
          (this value is then the value of the goal in the IF part of this rule)
        - destroy the local environment      */

```

FIGURE 5: The Abstract Output Rule for a Higher Operator

As we wrote in 5.4 it is not possible to make a visual representation of facts and rules from set A. We "enriched" the iconic structure by adding some iconic elements. FIGURE 6 demonstrates how the computation of the well-known factorial would look like, if we keep the number of object icons to a minimum: there is only one frame for recursive computations and intermediate results and computation goals (represented by "?") disappear when no longer needed for the computation.

We see that value and goal stacks are collapsed into the various fields of a node. For the application of an operator we have to select all numbers with the same time index. POMERANTZ (1985) showed that this kind of selective attention is extremely difficult and not trainable. If the function gets more complicated like a tree recursive function, a diagrammatic information of this kind would be completely misleading.

5.7.2 The second rule set B

We had to modify rule set A because of the following reasons, which result from constraints in the human information processor:

- 1) any undesired perceptual grouping of information in operator nodes ,
- 2) iconic rules with disjunctive conditions , and
- 3) visual hiding of dynamic successor frames already put on the linear stack

This required various modifications of the abstract rules.

Iconic rules with disjunctive conditions require selective attention, which causes matching errors and longer processing time (BOURNE, 1974; HAYGOOD & BOURNE, 1965; MEDIN, WATTENMAKER & MICHALSKI, 1987).

Also the "output" rule for a higher operator had to be modified. When a higher operator is called, a fresh copy of the original frame is created. Because we wanted to avoid a only partly visible "spaghetti"-stack in the sense that from one frame several new successor frames could be opened by calling "higher" operators, we allowed only one call per frame at the moment. This results in a depth first search in the call tree. The copies of the frames are ordered by frame number and are put on a frame stack. The arguments are copied in parallel into the parameter leaves of the head. Nodes and connections get the new attribute frame number, too. This allows to location-index time-indexed information. The "output" rule for higher operators is split into two rules corresponding to the call location (start tree, body tree).

Because we used recursive rules, the control and data flow occurred through the parameters. An iconic representation would require that intermediate results should be visible only when they belong to a pending operation. So intermediate results "die" before the corresponding frame "dies". This is not optimal from a cognitive science point of view, because a programmer who wants to recapitulate the computation history has to reconstruct mentally the already obtained results. This leads to higher working memory load for the programmer.

5.7.3 The third set C: objects and rules

The third rule set was motivated by the postulate, that the extent of the intermediate result should not end before the life of a frame ends. This seemed to require only a few changes to the visual interface. But the abstract rules had to be rewritten completely. There is no "input" rule any longer. We have 18 "output" rules instead which all lost their parameters. Like production rules they manipulate the nodes directly via the databasis. Computation goals ("?) and input and output values are written into the nodes. For this purpose a new attribute input-stripe is added to the node description.

We have included examples for abstract parts of object icons in FIGURE 7 and examples for abstract rules in FIGURES 8 and 9. The PROLOG facts in FIGURE 7 describe two nodes and two connections in the incomplete program of FIGURE 3. Both nodes are in the root position of the head and the body of the program, respectively.

```
node(frame_name(fac),frame_no(0),tree_type(head),instance_no(2),
      input-stripe([empty]),name-stripe(fac),output-stripe(empty)).
node(frame_name(fac), frame_no(0), tree_type(body), instance_no(11),
      input-stripe([empty,empty,empty]),name-stripe(if), output-stripe(empty)).
```

FIGURE 7: An example for Abstract Nodes and Connections

```

connection(frame_name(fac),frame_no(0), tree_type(head), out_instance_no(1),
           in_instance_no(2), input_no(1)).
connection(frame_name(fac),frame_no(0), tree_type(body),out_instance(10),
           in_instance_no(11),input_no(3)).

```

FIGURE 7: An example for Abstract Nodes and Connections

output :-

```

node(frame_name(Frame_name),frame_no(Frame_no),tree_type(Tree_type),
     instance_no(Instance_no),input_stripe(Input_stripe),name_stripe(Name_stripe),
     output_stripe(Output_stripe)),
higher_operator(name(Name_stripe)),Tree_type = start,
not(inverted_name_stripe(frame_name(Frame_name),frame_no(Frame_no),
     tree_type(Tree_type),instance_no(Any_instance_no))),
Output_stripe = ? ,forall(on(Element,Input_stripe),value(Element)),
copy_frame_on_top(frame_name(Name_stripe),top_frame_no(Top_frame_no)),
assert(inverted_name_stripe(frame_name(Frame_name),frame_no(Frame_no),
     tree_type(Tree_type),instance_no(Instance_no))),
root(frame_name(Name_stripe),frame_no(Top_frame_no),tree_type(head),
     instance_no(Instance_no_root_head)),
modify(frame_name(Name_stripe),frame_no(Top_frame_no),tree_type(head),
     instance_no(Instance_no_root_head),input_stripe(Input_stripe)),
bind_parameter_of_top_frame(input_stripe(Input_stripe)),
modify(frame_name(Name_stripe),frame_no(Top_frame_no),tree_type(head),
     instance_no(Instance_no_root_head),output_stripe( ? )),

```

output.

```

/* IF there is a node which has the following features:
(1) The node name is a higher operator.
(2) The node is located in the start tree.
(3) The name stripe of the node is the only inverted one in the tree which contains the node
(4) The output_stripe of the node contains a "?".
(5) The input_stripe of the node contains all input values.
THEN create the frame with the operators name and place it on top of the frame stack.
Invert the name stripe of the node.
Determine it's head root, transfer the input_stripe of the node to the head root .
Bind the parameters and put a "?" into the output_stripe.of the head root.*/

```

FIGURE 8: Abstract Rule 8 (First part of Call-by-Value, call in start tree)

output :-

```
node(frame_name(Frame_name),frame_no(Frame_no),tree_type(Tree_type),
      instance_no(Instance_no),input_stripe(Input_stripe),name_stripe(Name_stripe),
      output_stripe(Output_stripe)),
higher_operator(name(Name_stripe)),Tree_type = start,
inverted_name_stripe(frame_name(Frame_name),frame_no(Frame_no),tree_type(Tree_type),
instance_no(Instance_no)),
Output_stripe = ? ,forall(on(Element,Input_stripe),value(Element)),
value_of_upper_visible_frame(Output_stripe_root_head),not_exist_lower_visible_frame,
modify(frame(Frame_name),frame_no(Frame_no),tree_type(Tree_type),
instance_no(Instance_no),output_stripe(Output_stripe_root_head)),
delete_frame_from_top,
retract(inverted_name_stripe(frame_name(Frame_name),frame_no(Frame_no),
tree_type(Tree_type),instance_no(Instance_no))),
```

output.

```
/* IF there is a node which has the following features:
(1) The node name is a higher operator.
(2) The node is located in the start tree.
(3) The name stripe of the node is inverted.
(4) The output_stripe of the node contains a "?".
(5) The input_stripe of the node contains all input values.
and the head root of the upper visible frame contains a value and there is no other
visible frame
THEN transfer this value into the output_stripe of the node.
Delete the upper visible frame.Turn the inversion of the name stripe of the node back. */
```

FIGURE 9: Abstract rule 9 (Second part of Call-by-Value, call in start tree)

5.8 Iconic Rules for the Instructional Component and a Help System

On the basis of rule set B and C we developed iconic rules to describe the operational behavior of the interpreter. Because of space restrictions we can only show the rules from set C (FIGURES 10, 11), which are representation of the production-like PROLOG rules of FIGURES 8 and 9. At the present moment these rules are not implemented in an instructional or help component. But they are used successfully in experiments where novices are requested to predict the computation steps of the interpreter. Each rule consists of a description of the triggering situation and a description of the situation after the rules has been applied. We tried to make the rules self-explanatory as much as possible. So we need only a short introduction to explain the syntax of the iconic rules.

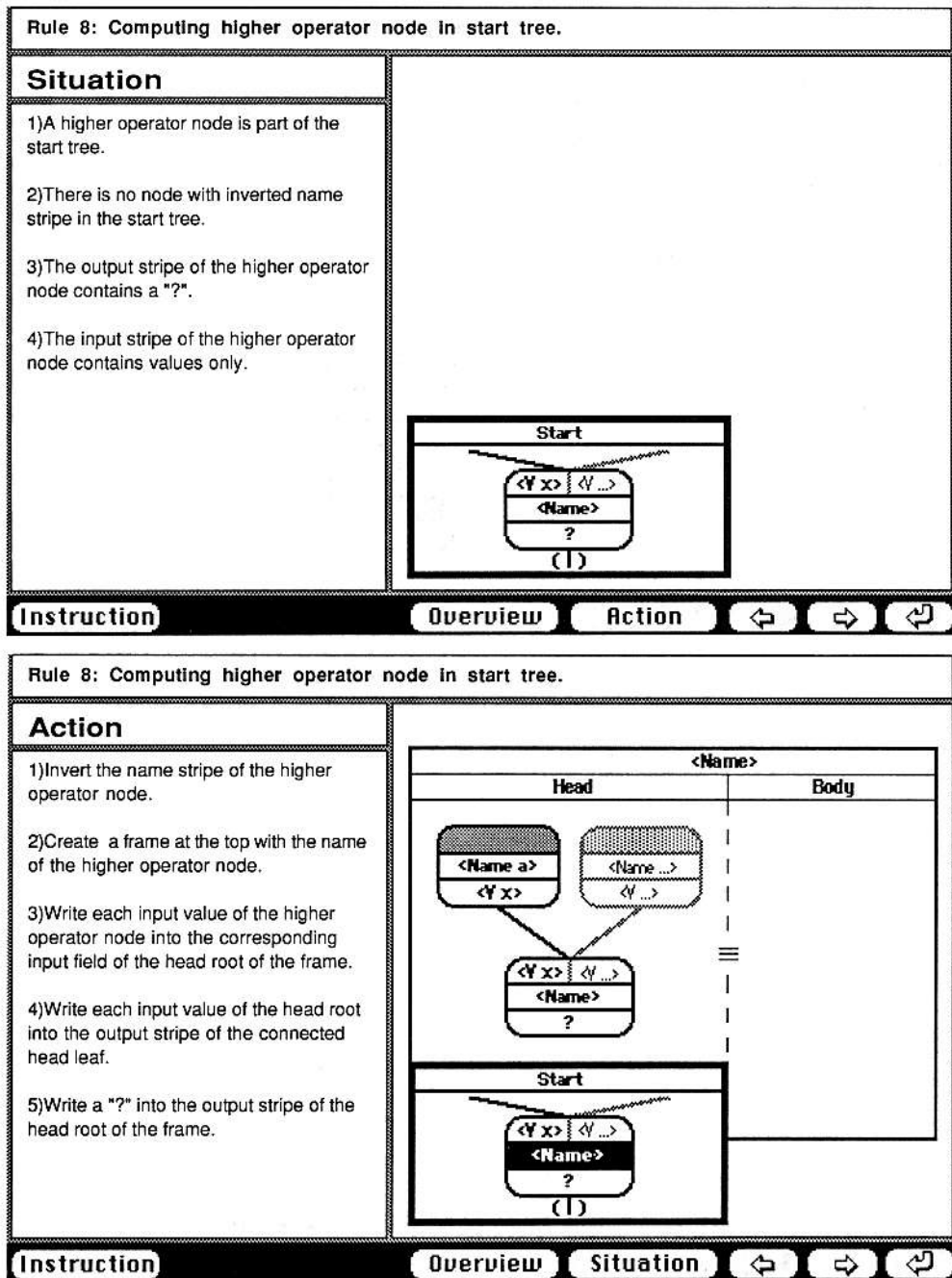


FIGURE 10: Iconic Rule on the Basis of Abstract Rule 8 in FIGURE 8

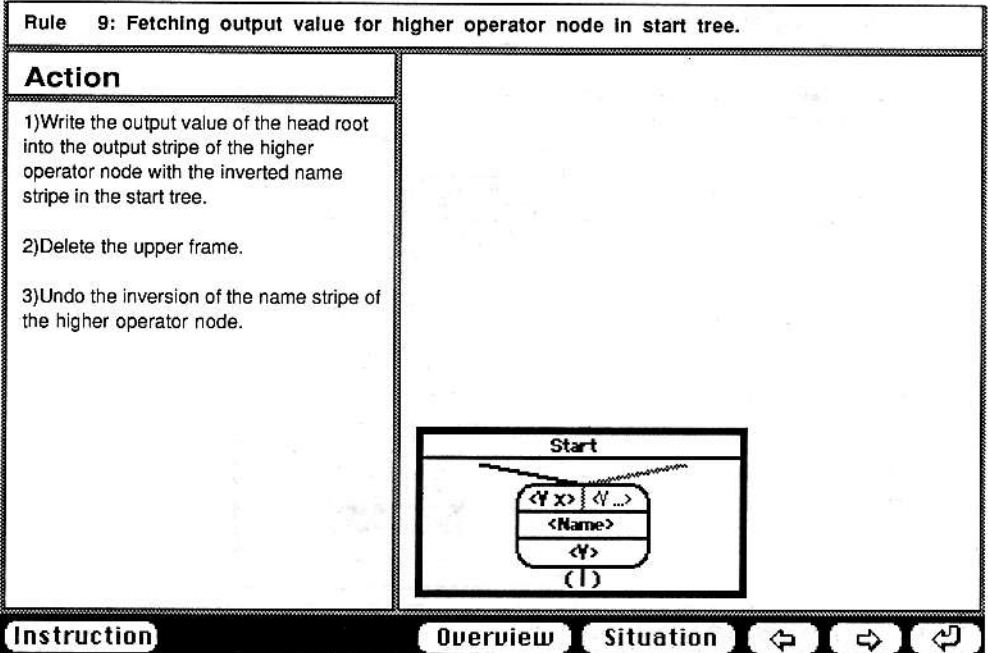
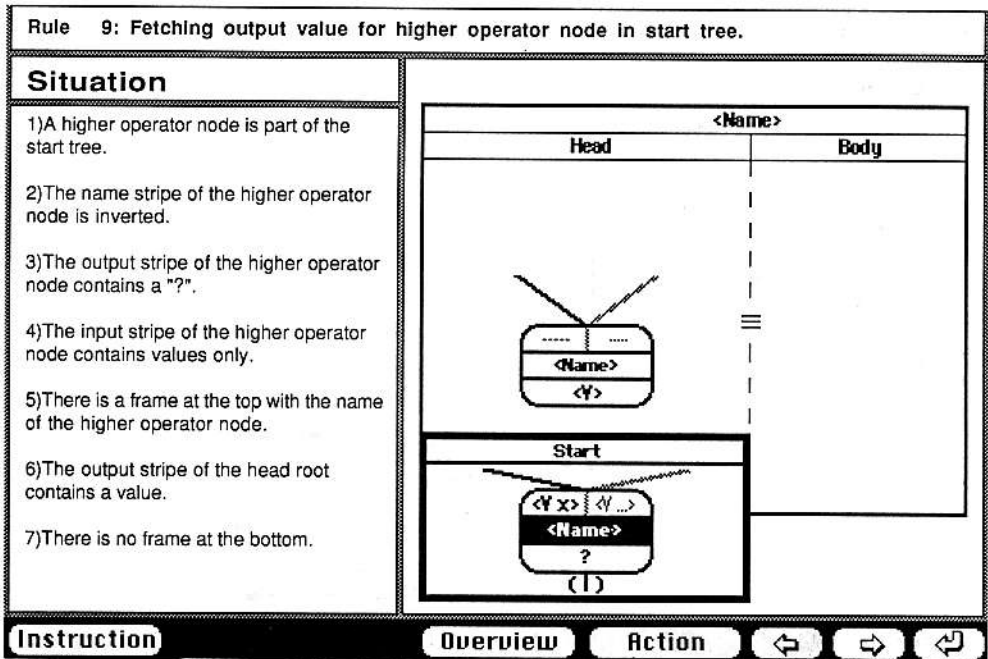


FIGURE 11: Iconic Rule on the Basis of Abstract Rule 9 in FIGURE 9

We found that the sentential information is used only in situations when an impasse in the computational process occurs. Novices were able to predict the interpreter in less than five hours learning time.

The next step is to implement the rules for instructional purposes so that the interpreter becomes selfexplaining. This situation can arise when the student is uncertain about the calculation process of the machine.

6. Summary

We reviewed the literature on CAI and ICAI systems and discussed some design principles from a cognitive science view. We discussed student models and plan recognition as important research goals but also stressed the importance of a rather neglected topic: design of the "student input". We need very careful and knowledge-crafted instructions, interfaces and helps so that our plan diagnosis and student modelling components get the opportunity to work satisfactorily. As an example we introduced iconic rules which transmit information in a diagrammatic form. Only if no phase of the design process is omitted we will achieve true "intelligent" CAI.

7. References

- ABELSON, H., SUSSMAN, G.J. & SUSSMAN, J., *Structure and Interpretation of Computer Programs*, Cambridge, Massachusetts: MIT Press, 1985
- AHO, A.V., SETHI, R. & ULLMAN, J.D., *Compilers: Principles, Techniques & Tools*, Reading, Massachusetts: Addison-Wesley Publ. Co., 1986
- ALDERMAN, D.L., Evaluation of the TICCIT Computer-assisted instructional system in the community college, *SIGCUE Bulletin*, 1979, 13, 5-17
- ANDERSON, J.R., *The Architecture of Cognition*, Cambridge, Mass.: Harvard University Press, 1983a
- ANDERSON, J.R., *Learning to Program*, Proceedings of the Eighth International Joint Conference on Artificial Intelligence, Los Altos, California: Morgan Kaufman, 1983b
- ANDERSON, J.R., *Acquisition of Proof Skills in Geometry*, in: R.S. MICHALSKI, J.G. CARBONELL, T.M. MITCHELL (eds), *Machine Learning: An Artificial Intelligence Approach*, Plao Alto: Tioga Publ. Co., 1983c, 191 - 219
- ANDERSON, J.R., *Knowledge Compilation: The General Learning Mechanism*, in: R.S. MICHALSKI, J.G. CARBONELL, T.M. MITCHELL (eds), *Machine Learning: An Artificial Intelligence Approach*, Vol. II, Los Altos, California: Morgan Kaufman Publ., 1986, 289 - 310
- ANDERSON, J.R., *Production Systems, Learning, and Tutoring*, in: D.KLAHR, P.LANGLEY & R.NECHES (eds), *Production System Models of Learning and Development*, Cambridge, Mass.: MIT Press, 1987, 437-458
- ANDERSON, S.B., BALL, S., MURPHY, R.T. & Associates, *Encyclopedia of Educational Evaluation*, San Francisco: Jossey-Bass Publ., 1976
- ANDERSON, J.R., BOYLE, C.F., FARRELL, R. & REISER, B.J., *Cognitive Principles in the Design of Computer Tutors*, in: P.MORRIS (ed), *Modelling Cognition*, Chichester, Sussex: J.Wiley, 1987, 93-133
- ANDERSON, J.R., CORBETT, A.T. & REISER, B.J., *Essential LISP*, Reading, Mass.: Addison-Wesley Publ.Co., 1986
- ANDERSON, J.R., FARRELL, R. & SAUERS, R., *Learning to Plan in LISP*, Technical Report ONR-82-2, Department of Psychology, Carnegie-Mellon University, Pittsburgh, PA, 1982
- ANDERSON, J.R., FARRELL, R. & SAUERS, R., *Learning to Program in LISP*, *Cognitive Science*, 1984, 8, 87-129
- ANDERSON, J.R., GREENO, J.G., KLINE, P.J. & NEVES, D.M., *Acquisition of Problem-Solving Skill*, in: J.R.ANDERSON (ed), *Cognitive Skills and their Acquisition*, Hillsdale, New Jersey: Erlbaum Ass., 1981, 191-230
- ANDERSON, J.R., KLINE, P.J. & BEASLEY, C.M., *Complex Learning Processes*, in: R.E.SNOW, P.A.FEDERICO & W.E. MONTAGUE (eds), *Aptitude, Learning and Instruction: Cognitive Process Analyses of Learning and Problem Solving*, Hillsdale, New Jersey: Erlbaum Associates, 1980, 199-235
- ANDERSON, J.R. & REISER, B.J., *The LISP Tutor*, *BYTE*, 1985, 4, 159-175
- ANDERSON, J.R. & SKWARECKI, E., *The Automated Tutoring of Introductory Computer Programming*, *Communications of the ACM*, 1986, 29(9), 842-849
- ATKINSON, R.C., *Ingredients for a Theory of Instruction*, *American Psychologist*, 1972, 27, 921-931
- ATKINSON, R.C. & WILSON, H.A. (eds), *Computer-assisted Instruction*, New York: Academic Press, 1969
- ATTISHA, M., *Non-borrow Subtraction Algorithm*, Working Paper W-119, Computer Science Dept., University of Exeter, 1984

- ATTISHA, M. & YAZDANI, M., An Expert System for Diagnosing Children's Multiplication Errors, Research Report R-117, Computer Science Dept., University of Exeter, 1983
- BACKHOUSE, R.C., Syntax of Programming Languages, Englewood Cliffs, New Jersey: Prentice Hall, 1979
- BARR, A. & ATKINSON, R.C., Adaptive Instructional Strategies, in: H. SPADA & W.F.KEMPF (eds), Structural Models of Thinking and Learning, 83 - 112, Bern: Hans Huber Publ., 1977
- BARR, A., BEARD, M. & ATKINSON, R.C., A Rationale and Description of a CAI Program to Teach the BASIC Programming Language, Instructional Science, 1975, 4, 1-31
- BARR, A., BEARD, M. & ATKINSON, R.C., The Computer as a Tutorial Laboratory: The Stanford BIP Project, International Journal of Man-Machine Studies, 1976, 8, 567-596
- BARR, A. & FEIGENBAUM, E.A. (eds), The Handbook of Artificial Intelligence, Vol. II, Los Altos, Calif.: W.Kauffman, Inc., 1982
- BAUER, F.L. & GOOS, G., Informatik, Bd. I, Berlin: Springer Verlag, 1982; Bd. II, Berlin: Springer Verlag, 1984
- BAUER, F.L. & WÖSSNER, H., Algorithmische Sprache und Programmentwicklung, Berlin: Springer Verlag, 1981
- BAUER, F.L. & WÖSSNER, H., Algorithmic Language and Program Development, Berlin: Springer Publ., 1984
- BERGHAMMER, R., DOSCH, W. & OBERMEIER, R., CIP-LS: Pascal Variante (Übersicht über Sprache, Übersetzer und Formulardmaschine), München: Institut für Informatik der TU München, Dezember 1985
- BERTIN, J., Graphics and Graphics Information Processing, Berlin: Walter de Gruyter, 1981
- BERTIN, J., The Semiology of Graphics, Madison, Wisconsin: The University of Wisconsin Press, 1983
- BLOOM, B.S., Taxonomie von Lernzielen im kognitiven Bereich, Weinheim: Beltz, 1972
- BODEN, M., Minds and Mechanism: Philosophical Psychology and Computational Models, Brighton, Sussex: The Harvester Press, 1981
- BORK, A., Learning with Computers, Bedford, Massachusetts: Digital Press, 1981
- BORK, A., Learning with Personal Computers, New York: Harper & Row, 1986
- BOURNE, L.E., Human Conceptual Behavior, Boston: Allyn & Bacon, 1966
- BOURNE, L.E., An Inference Model of Conceptual Rule Learning, in: R. SOLSO (ed), Theories in Cognitive Psychology, Washington, DC.: LAWRENCE ERLBAUM, 1974, 231 - 256
- BOURNE, L.E., EKSTRAND, B.R. & DOMINOWSKI, R.L., The Psychology of Thinking, Englewood Cliffs, N.J.: Prentice Hall, 1971
- BOYLE, C.F., 1986, Advanced Computer Tutoring, Inc., 701 Amberson Avenue, Pittsburgh, PA 15232, USA
- BROWN, J.S., Uses of Artificial Intelligence and Advanced Computer Technology in Education, in: R.J. SEIDEL & M. RUBIN (eds), Computers and Communications: Implications for Education, New York: Academic Press, 1977, 253-270
- BROWN, J.S. & BURTON, R.R., Multiple Representation of Knowledge for Tutorial Reasoning, in D. BOBROW & A. COLLINS (eds), Representation and Understanding: Studies in Cognitive Science, New York: Academic Press, 1975
- BROWN, J.S. & BURTON, R.R., Diagnostic Models for Procedural Bugs in Basic Mathematical Skills, Cognitive Science, 1978, 2, 155-192
- BROWN, J.S. & BURTON, R.R., Reactive Learning Environments for Teaching Environments for Teaching Electronic Troubleshooting, in W.B. ROUSE (ed), Advances in Man-Machine Systems Research, Greenwich, Connecticut: JAI Press, 1986
- BROWN, J.S., BURTON, R.R. & BELL, A.G., SOPHIE: A Step Towards a Reactive Learning Environment, International Journal of Man Machine Studies, 1975, 7, 675-696
- BROWN, J.S., BURTON, R.R. & de KLEER, J., Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE I, II and III, in: D.SLEEMAN & J.S.BROWN (eds), Intelligent Tutoring Systems, New York: Academic Press, 1982
- BROWN, J.S. & Van LEHN, K., Repair Theory: A Generative Theory of Bugs in Procedural Skills, Cognitive Science, 1980, 4, 379 - 426
- BUNDERSON, C.V., The Design and Production of Learner-controlled Courseware for the TICCAT System, International Journal of Man Machine Studies, 1974, 6, 479-491
- BUNDY, A., Computer Modelling of Mathematical Reasoning, New York: Academic Press, 1983
- BUNDY, A., SILVER, B. & PLUMMER, D., An Analytical Comparison of Some Rule Learning Programs, Artificial Intelligence, 1985, 27, 137 - 181
- BURTON, R.R., Diagnosing Bugs in a Simple Procedural Skill, in: D.SLEEMAN & J.S.BROWN (eds), Intelligent Tutoring Systems, New York: Academic Press, 1982, 157-183
- BURTON, R.R. & BROWN, J.S., A Tutoring and Student Modelling Paradigm for Gaming Environments, SIGCSE Bulletin, 1976, 8, 236-246
- BURTON, R.R. & BROWN, J.S., Toward a Natural Language Capability for Computer-assisted Instruction, in: H. O'NEIL (ed), Procedures for Instructional Systems Development, New York: Academic Press, 1979a
- BURTON, R.R. & BROWN, J.S., An Investigation of Computer Coaching for Informal Learning Activities, International Journal of Man-Machine Studies, 1979b, 11, 5-24
- CAMPBELL, J.A. & ROSS, S.P., Issues in Computer-assisted Interpretation of Graphs and Quantitative Information, in: G.SALVENDY (ed), Cognitive Engineering in the Design of Human-Computer Interaction and Expert Systems, Amsterdam: Elsevier Science Publ., 1987, 473-480
- CARBONELL, J.R., AI in CAI: An Artificial Intelligence Approach to Computer-assisted Instruction, IEEE Transactions on Man-Machine Systems, 1970, 11, 190-202
- CARBONELL, J. & LANGLEY, P., Machine Learning, in: St.C. SHAPIRO (ed), Encyclopedia of Artificial Intelligence, Vol.1, 464 - 488, 1987
- CARROLL, J.M., Minimalist Design for Active Users, in: B.SHACKLE (ed), Interact '84, First IFIP Conference on Human-Computer Interaction, Amsterdam: Elsevier/North-Holland, 1984a

- CARROLL, J.M., *Minimalist Training*, Datamation, 1984b, 125 - 136
- CAWSEY, A., *Bugs in Decimal Addition: Model, Applications and Explanations*, *Artificial Intelligence and Simulation of Behavior Quarterly*, 1986, 59, 12-13
- CHANG, S.K., *Visual Languages: A Tutorial and Survey*, in: P. GORNY & M.J. TAUBER (eds), *Visualization in Programming*, Lecture Notes in Computer Science, Nr. 282, Berlin: Springer, 1987, 1-23
- CHASE, W. G., *Visual Information Processing*, in: K.R. BOFF, L. KAUFMAN & J.P. THOMAS (eds), *Handbook of Perception and Human Performance*, Vol. II, *Cognitive Processes and Performance*, New York: Wiley, 1986, 28-1 - 28-71
- CLANCEY, W.J., *Dialogue Management for Rule-Based Tutorials*, *IJCAI*, 1979, 6, 155-161
- CLANCEY, W.J., *Methodology for Building an Intelligent Tutoring System*, in: W.KINTSCH, J.R. MILLER & P.G. POLSON (eds), *Methods and Tactics in Cognitive Science*, Hillsdale, N.J.: Lawrence Erlbaum Ass., 1984
- CLANCEY, W.J., *Tutoring Rules for Guiding a Case Method Dialogue*, in: D.SLEEMAN & J.S.BROWN (eds), *Intelligent Tutoring Systems*, New York: Academic Press, 1982, 201-225
- CLANCEY, W.J., *GUIDON*, *Journal of Computer-based Instruction*, 10, 8-14, 1983
- CLANCEY, W.J., *From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons: ORN Final Report*, 1979 - 1985, *AI Magazine*, 1986a, 7(3), 40 - 60 & 187
- CLANCEY, W.J., *Qualitative Student Models*, in: J.F.TRAUB (ed), *Annual Review of Computer Science*, 1986b, 1, 381-450
- CLANCEY, W.J., *Knowledge-based Tutoring: The GUIDON Program*, Cambridge, Mass.: 1987
- CLEVELAND, W.S., *The Elements of Graphing Data*, Belmont, California: Wadsworth, 1985
- COLLINS, A., *Processes in Acquiring Knowledge*, in: R.C.ANDERSON, R.J.SPIRO & W. MONTAGUE (eds), *Schooling and the Acquisition of Knowledge*, Hillsdale, N.J.: Lawrence Erlbaum Press, 1976
- COLLINS, A., *Teaching Reasoning Skills*, in: S.F. CHIPMAN, J.W. SEGAL & R. GLASER (eds), *Thinking and Learning Skills: Research and Open Questions*, Hillsdale, N.J.: Lawrence Erlbaum Ass., 1985
- COLLINS, A. & STEVENS, A.L., *Goals and Strategies for Inquiry Teachers*, in: R. GLASER (ed), *Advances in Instructional Psychology*, II, Hillsdale, N.J.: Lawrence Erlbaum Ass., 1982
- COLLINS, A. & STEVENS, A.L., *A Cognitive Theory of Interactive Teaching*, in: C.M. REIGELUTH (ed), *Instructional Design Theories and Models: An Overview*, Hillsdale, N.J.: Lawrence Erlbaum Ass., 1983
- COLLINS, A., WARNOCK, E.H., AIELLO, N. & MILLER, M.L., *Reasoning from Incomplete Knowledge*, in: BOBROW, D.G. & COLLINS, A. (eds), *Representation and Understanding*, New York: Academic Press, 1975
- COLONIUS, H., FRANK, K.D., JANKE, G., KOHNERT, K., MÖBUS, C., SCHRÖDER, O. & THOLE, H.J., *Entwicklung einer Wissensdiagnostik- und Fehlererklärungskomponente beim Erwerb von Programmierwissen für ABSYNT*, paper presented on the workshop "Intelligente Lernsysteme", Tübingen, DIFF, 1987
- COLONIUS, H., FRANK, K.D., JANKE, G., KOHNERT, K., MÖBUS, C., SCHRÖDER, O. & THOLE, H.J., *Syntaktische und semantische Fehler in funktionalen graphischen Programmen*, ABSYNT-Report 2/87, Projekt ABSYNT, FB 10, Arbeitsgruppe Lehr-Lernsysteme, Universität Oldenburg, 1987
- DAVIS, R.E., *Runnable Specification as a Design Tool*, in: K.L. CLARK & S.A. TÄRNLUND (eds), *Logic Programming*, New York: Academic Press, 1982, 141 - 149
- DAVIS, R.B., DUGDALE, S., KIBBEY, D. & WEAVER, Ch., *Representing Knowledge about Mathematics for Computer-Aided Teaching, Part II - The Diversity of Roles that a Computer Can Play in Assisting Learning*, in: E.W.ELCOCK & D.MICHIE (eds), *Machine Intelligence 8*, Chichester, Sussex: Ellis Horwood Ltd, 1977, 387-421
- DEDE, C., *A Review and Synthesis of Recent Research in Intelligent Computer-assisted Instruction*, *International Journal of Man-Machine Studies*, 1986, 24, 329-353
- DESMARAI, M.C., LAROCHELLE, S. & GIROUX, L., *The Diagnosis of User Strategies*, in: H.J. BULLINGER & B. SHACKEL (eds), *INTERACT '87*, Amsterdam: Elsevier Science Publ., 1987, 185 - 189
- DIETTERICH, T.G., LONDON, B., CLARKSON, K., DROMEY, G., *Learning and Inductive Inference*, in: P.R. COHEN & E.A. FEIGENBAUM (eds), *The Handbook of Artificial Intelligence*, Vol.3, 323 - 511, London: Pitman Books Ltd., 1982
- DOSCH, W., *New Prospects of Teaching Programming Languages*, in: F.B.LOVIS & E.D.TAGG (eds), *Informatics Education for all Students at University Level*, IFIP, Amsterdam: Elsevier Science Publishers, 1984, 153 -169
- DUGDALE, S. & KIBBEY, D., *Elementary Mathematics with PLATO*, Urbana, Illinois: Computer-based Education Laboratory (op.cit.in: O'SHEA, 1982)
- DYWER, T.A., *Heuristic Strategies for Using Computers to Enrich Education*, *International Journal of Man-Machine Studies*, 1974, 6, 137-154
- EFE, K., *A Proposed Solution to the Problem of Levels in Error-message Generation*, *Communications of the ACM*, 1987, 30(11), 948 - 955
- EGAN, D.E. & GREENO, J.G., *Theory of Rule Induction: Knowledge Acquired in Concept Learning, Serial Pattern Learning, and Problem Solving*, in: L.W.GREGG (ed), *Knowledge and Cognition*, Potomac, Maryland: L.Erlbaum Ass.Publ., 1974, 43-103
- FÄHNRIICH, K.P. & ZIEGLER, J., *Workstation Using Direct Manipulation as Interaction Mode*, in: *Proceedings of INTERACT '84*, Vol.II, 1985a, 203 - 208 (in german: *Direkte Manipulation als Interaktionsform an Arbeitsplatzrechnern*, in: H.J. BULLINGER (Hrsgb), *Software-Ergonomie '85 -Mensch-Computer- Interaktion*, Stuttgart: Teubner, 1985, 75 - 85
- FEURZEIG, W., *Algebra Slaves and Agents in a LOGO-based Mathematics Curriculum*, in: R.W. LAWLER & M. YAZDANI (eds), *Artificial Intelligence and Education: Learning Environments and Tutoring Systems*, 27 - 54, Norwood, N.J.: Ablex Publ. Co., 1987
- FISCHER, G., *Einführung in die Theorie psychologischer Tests*, Bern: Hans Huber, 1974

- FITTER, M. & GREEN, T.R.G., When Do Diagrams Make Good Computer Languages?, *International Journal of Man-Machine Studies*, 1979, 11, 235-261 and in: M.J. COOMBS & J.L. ALTY (eds), *Computing Skills and the User Interface*, New York: Academic Press, 1981, 253 - 287
- FLETCHER, J.D., Modeling the Learner in Computer-assisted Instruction, *Journal of Computer-based Instruction*, 1975, 1, 118-126
- FORD, L., *Intelligent Computer Aided Instruction*, Research Report, R.121, Computer Science Dept., University of Exeter, 1986
- FORD, L., Teaching Strategies and Tactics in Intelligent Computer Aided Instruction, *Artificial Intelligence Review*, 1987, 1, 201-215
- FRICKE, R., *Über Meßmodelle in der Schulleistungsdiagnostik*, Düsseldorf: Pädagogischer Verlag Schwann, 1972
- FRICKE, R., *Kriteriumsorientierte Leistungsmessung*, Stuttgart: W.Kohlhammer, 1974
- FRIEDMAN, D.P. & FELLEISEN, M., *The Little LISPer*, Cambridge, Massachusetts: MIT Press, 1987
- GAGE, N.L.(ed), *Handbook of Research on Teaching*, Chicago: Rand McNally & Co, 1967⁵
- GAGNE, R.M., The Acquisition of Knowledge, *Psychological Review*, 1962, 69, 355-365 (german translation in: M.HOFER & F.E.WEINERT (eds), *Pädagogische Psychologie*, Bd.2, Lernen und Instruktion, 1973, 106-123
- GAGNE, R.M., Task Analysis - Its Relation to Content Analysis, *Educational Psychology*, 1974, 11, 11-18
- GALANTER, E.H. (ed.), *Automatic Teaching: the State of Art*, N.Y.: Wiley, 1959
- GENESERETH, M.R., The Role of Plans in Intelligent Teaching Systems, in: D.SLEEMAN & J.S.BROWN (eds), *Intelligent Tutoring Systems*, New York: Academic Press, 1982, 137 - 155
- GENESERETH, M.R. & NILSSON, N.J., *Logical Foundations of Artificial Intelligence*, Los Altos, California: Morgan Kaufman Publ., 1987
- GHEZZI, C. & JAZAYERI, M., *Programming Language Concepts 2/E*, New York: Wiley, 1987
- GILMORE, D.J. & GREEN, T.R.G., Are "programming plans" psychological real - outside PASCAL?, in: H.J. BULLINGER & B. SHACKEL (eds), *INTERACT '87*, Amsterdam: Elsevier Science Publ., 1987, 497 - 503
- GILMORE, D. & SELF, J.A., The Application of Machine Learning to Intelligent Tutoring Systems, in: J.A.SELF (ed), *Intelligent Computer-Aided Instruction*, London: Chapman & Hall (in press)
- GLINERT, E.P. & GONCZAROWSKI, J., A (Formal) Model for (Iconic) Programming Environments, in H.J. BULLINGER & B. SHACKEL (eds), *Human - Computer Interaction - INTERACT '87*, Amsterdam: Elsevier Science Publ., 1987, 283 - 290
- GLINERT, E.P. & GONCZAROWSKI, J. & SMITH, C.D., An Integrated Approach to Solving Visual Programming's Problems, in: G. SALVENDY (ed), *Cognitive Engineering in the Design of Human - Computer Interaction and Expert Systems*, Amsterdam: Elsevier Science Publ., 1987, 341 - 348
- GOEDE, K. & KLIX, F., Lernabhängige Strategien der Merkmalsgewinnung und der Klassenbildung beim Menschen, in: F.KLIX, W.KRAUSE & H.SYDOW (Hrsgb), *Kybernetik-Forschung*, H.1, Zeichenerkennungs- und Klassifikationsprozesse bei biologischen und technischen Systemen, VEB Deutscher Verlag der Wissenschaften, Berlin 1972
- GOLDSTEIN, I.P., The Genetic Epistemology of Rule Systems, *International Journal of Man-Machine Studies*, 1979, 11, 51-77
- GOLDSTEIN, I.P., Developing a Computational Representation for Problem-Solving Skills, in: D.T.TUMA & F.REIF (eds), *Problem Solving and Education: Issues in Teaching and Research*, Hillsdale, N.J.: Erlbaum Ass. Publ., 1980, 53-79
- GOLDSTEIN, I.P., The Genetic Graph: A Representation for the Evolution of Procedural Knowledge, in: D.SLEEMAN & J.S.BROWN (eds), *Intelligent Tutoring Systems*, New York: Academic Press, 1982
- GOODMAN, D., *The Complete HyperCard Handbook*, Toronto: Bantam Books, 1987
- GREEN, T.R.G., Programming as a Cognitive Activity, in: H.T. SMITH & T.R.G. GREEN (eds), *Human Interaction with Computers*, New York: Academic Press, 1980, 271 - 319
- GREEN, T.R.G., Learning Big and Little Programming Languages, in: A.C. WILKINSON (ed), *Classroom Computers and Cognitive Science*, New York: Academic Press, 1983, 71 - 93
- GREEN, T.R.G. & PAYNE, S.J., Organization and Learnability in Computer Languages, *International Journal of Man-Machine Studies*, 1984, 21, 7 - 18
- GREEN, T.R.G., SCHIELE, F. & PAYNE, S.J., Formalizable Models of User Knowledge in Human Computer Interaction, 1985, to appear in: GREEN, HOC, MURRAY & VEER (eds), *Theory and Outcomes in Human Computer Interaction*, London: Academic Press (in press)
- GREEN, T.R.G., SIME, M.E. & FITTER, M.J., The Art of Notation, in: M.J. COOMBS & J.L. ALTY (eds), *Computing Skills and the User Interface*, New York: Academic Press, 1981, 221 - 251
- HAYGOOD, R.C. & BOURNE, L.E., Attribute- and Rule-learning Aspects of Conceptual Behaviour, *Psychological Review*, 1965, 72, 175 - 195
- HENDERSON, P., *Functional Programming: Application and Implementation*, Englewood Cliffs, N.J.: Prentice Hall, 1980
- HENDERSON, P., Functional Programming, Formal Specification and Rapid Prototyping, *IEEE Transactions on Software Engineering* SE-12 2, 1986, 241 - 250
- HOLLAN, J.D., HUTCHINS, E.L. & WEITERMAN, L.M., STEAMER: An Interactive, Inspectable, Simulation-based Training System, in: G.KEARSLEY (ed), *Artificial Intelligence and Instruction: Applications and Methods*, Reading, Mass.: Addison-Wesley, 1987, 113-134
- HOLLAND, J.G., Teaching Machines: an Application of Principles from the Laboratory, *Journal of Experimental Analysis of Behavior*, 1960, 3, 275 - 286 (german translation in: W.CORRELL(ed), *Programmiertes Lernen und Lehrmaschinen*, Braunschweig, 1965)
- HOLLAND, J.G., Response Contingencies in Teaching Machine Programs, *Journal of Programmed Instruction*, 1964, 3, 1-8

- HOLLAND, J.G. & SKINNER, B.F. *The Analysis of Behavior*, New York: McGraw Hill, 1961 (german translation: *Analyse des Verhaltens*, München: Urban & Schwarzenberg, 1974)
- HOPPE, H.U., A Grammar-based Approach to Unifying Task-oriented and System-oriented Interface Descriptions, in: D. ACKERMAN & M.TAUBER (eds), *Mental Models and Computer Interaction*, Amsterdam: North-Holland (in press)
- HOPPE, H.U., *Task-oriented Parsing - A Diagnostic Method to be Used by Adaptive Systems*, working paper, GMD, 1987
- HOPPE, H.U., TAUBER, M. & ZIEGLER, J.E., A Survey of Models and Formal Description Methods in HCI with Example Applications, ESPRIT Project 385, HUFIT Report B.3.2.a, Fraunhofer Institute (FHG-IAO), 1986
- HUNT, E.B., MARTIN, J. & STONE, P.J., *Experiments in Induction*, New York: Academic Press, 1966
- HUTCHINS, E.L., HOLLAN, J.D. & NORMAN, D.A., *Direct Manipulation Interfaces*, in: D.A.NORMAN & S.W.DRAPER (eds), *User Centered System Design - New Perspectives on Human Computer Interaction*, Hillsdale, N.J.: Lawrence Erlbaum Ass., 1986, 87-124
- JANKE, G. & KOHNERT, K., *Interface Design of a Visual Programming Language: Evaluating Runnable Specifications According to Psychological Criteria*, paper to be presented at MACINTER, 1988, Berlin/GDR
- JOHNSON, W.L., *Intention-Based Diagnosis of Novice Programming Errors*, Los Altos, California: Morgan Kaufman Publ., 1986
- JOHNSON, W.L. & SOLOWAY, E., *Intention-based Diagnosis of Programming Errors*, Proceedings of the AAAI-84, 1984, 162-168
- JOHNSON, W.L. & SOLOWAY, E., PROUST: An Automatic Debugger for Pascal Programs, BYTE, 1985, April, 179-190 and in: G.P.KEARSLEY (ed), *Artificial Intelligence & Instruction*, Reading, Mass.: Addison Wesley Publ.Co., 1987, 49-67
- KAHNEY, H., *The Behaviour of Novice and Expert Problem Solvers*, *Artificial Intelligence and Simulation of Behaviour Quarterly*, 1983, No.48, 20 - 24
- KANTOROVIC, L.V., *Ob odnoi matematischeskoi cimbolike, udobnoi pri provedenii witschiclenii na maschinach* (On a Mathematical Symbolism Convenient for Performing Machine Calculations), *Doklady Akademii Nauk CCCP*, 1957, 113, 738 -741
- KASS, R., *The Role of User Modelling in Intelligent Tutoring Systems*, Department of Computer and Information Science, School of Engineering and Applied Science, Philadelphia, PA., LINC LAB 41, MS-CIS-86-58, 1987
- KAWAI, K., MIZOGUCHI, R., KAKUSHO, O. & TOYODA, J., *A Framework for ICAI Systems Based on Inductive Inference and Logic Programming*, *New Generation Computing*, 1987, 5, 115 - 129
- KEARSLEY, G.P. (ed), *Artificial Intelligence & Instruction*, Reading, Mass.: Addison-Wesley, 1987
- KIMBALL, R., *A Self Improving Tutor for Symbolic Integration*, in: D.SLEEMAN & J.S.BROWN (eds), *Intelligent Tutoring Systems*, New York: Academic Press, 1982
- KLAUSMEIER, H.J., *Learning and Human Abilities*, New York: Harper & Row, 1971
- KLING, U., *Kognitive Aspekte bei Mensch/Maschine-Interaktionsformen im Bereich des Lernens und Problemlösens*, in: H.UECKERT & D.RHENIUS (eds), *Komplexe menschliche Informationsverarbeitung*, Bern: Hans Huber, 1979
- KÖHNE, A. & WEBER, G., STRUEDI: A Lisp-Structure Editor for Novice Programmers, in: H.J.BULLINGER & B.SHACKEL (eds), Elsevier Science Publ., 1987, 125-129
- KOFFMAN, E.B. & BLOUNT, S.E., *Artificial Intelligence and Automatic Programming in CAI*, *Artificial Intelligence*, 1975, 6, 215-234
- KOHNERT, K. & JANKE, G., *The Object-oriented Implementation of the ABSYNT Environments*, ABSYNT-Report 4/88, FB 10 Informatik, Arbeitsgruppe Lehr-Lernsysteme, University of Oldenburg
- KRAUSE, M.U. & SEEL, B.R. (eds), *Lernerfolgsmessung: Beiträge zur computerunterstützten Auswertung von Lernerfolgsdaten*, München: Oldenbourg Verlag, 1979
- LAIRD, J.E., ROSENBLOOM, P.S. & NEWELL, A., *Chunking in SOAR: The Anatomy of a General Learning Mechanism*, *Machine Learning*, 1986, 1, 11 - 46
- LAKIN, F.H., *Computing with Text-Graphic Forms*, in: J. ALLEN (ed), *Records of the LISP Conference*, 1980, 100 - 105
- LAKIN, F.H., *Spatial Parsing for Visual Languages*, in: S. CHANG, T. ICHIKAWA & P.A. LIGOMENIDES (eds), *Visual Languages*, New York: Plenum Press, 1986
- LANGLEY, P., OHLSSON, S. & SAGE, St., *A Machine Learning Approach to Student Modelling*, Pittsburgh, PA.: Carnegie-Mellon University, The Robotics Institute, Tech.Rep., CMU-RI-TR-84-7, 1984
- LARKIN, J.H. & SIMON, H.A., *Why a Diagram is (Sometimes) Worth Ten Thousand Words*, *Cognitive Science*, 1987, 11, 65 - 99
- LAWLER, R.W., *Designing Computer-based Microworlds*, in: M. YAZDANI (ed), *New Horizons in Educational Computing*, Chichester, England: Ellis Horwood, Ltd., New York: John Wiley, 1984
- LAWLER, R.W., *Learning Environments: Now, Then, and Someday*, in: R.W. LAWLER & M. YAZDANI (eds), *Artificial Intelligence and Education*, Norwood, N.J.: Ablex Publ. Co., 1987
- LAWLER, R.W. & LAWLER, G.P., *Computer Microworlds and Reading: An Analysis for their Systematic Application*, in: R.W. LAWLER & M. YAZDANI (eds), *Artificial Intelligence and Education*, Norwood, N.J.: Ablex Publ. Co., 1987, 95 - 115
- LAWLER, R.W. & M. YAZDANI (eds), *Artificial Intelligence and Education: Learning Environments and Tutoring Systems*, Norwood, N.J.: Ablex Publ. Co., 1987
- LEKAN, H.A., *Index to Computer-assisted Instruction*, New York: Harcourt Brace, 1971
- LEVESQUE, H.J., *Knowledge Representation and Reasoning*, *Annual Review of Computer Science*, 1986, 1, 255-287
- LLOYD, C.J., *Integrating Plan Recognition and User Modelling*, Centre for Research on Computers and Learning, Department of Computing, The University of Lancaster, 1986
- LUTZE, R., *The Gestalt Analysis of Programs*, in: P. GORNY & M.J. TAUBER (eds), *Visualization in Programming*, 5th Interdisciplinary Workshop in Informatics and Psychology, Schärding, Austria, May 1986, Berlin: Springer-Verlag, 1987, 24 - 36

- MANDL, H. & FISCHER, P.M. (eds), *Lernen im Dialog mit dem Computer*, München: Urban & Schwarzenberg Publ., 1985
- MATZ, M., Towards a Process Model for High School Algebra Errors, in D. SLEEMAN & J.S. BROWN (eds), *Intelligent Tutoring Systems*, New York: Academic Press, 1982, 25 - 50
- McKENDREE, Jean, Feedback Content During Complex Skill Acquisition, 181-188, in: G.SALVENDY, S.L.SAUTER & J.J.HURRELL (eds), *Social, Ergonomic and Stress Aspects of Work with Computers*, Amsterdam: Elsevier Science Publ., 1987
- MEDIN, D.L. & SMITH, E.E., Concepts and Concept Formation, *Annual Review of Psychology*, 1984, 35, 113 - 138
- MEDIN, D.L., WATTENMAKER, W.D. & MICHALSKI, R.S., Constraints and Preferences in Inductive Learning: An Experimental Study of Human and Machine Performance, *Cognitive Science*, 1987, 11, 299 - 339
- MERRILL, M.D., SCHNEIDER, E.W. & FLETCHER, K.A., TICCIT, EnglewoodCliffs,N.J.: Educational Technology,1980
- MICHALSKI, R.S., Learning Strategies and Automated Knowledge Acquisition: An Overview, in: L.BOLC (ed), *Computational Models of Learning*, Springer: Berlin, 1987, 1 - 19
- MILLER, M.L., A Structural Planning and Debugging Environment for Elementary Programming, *International Journal of Man Machine Studies*, 1979, 11, 79 - 95 and in: D.SLEEMAN & J.S. BROWN (eds), *Intelligent Tutoring Systems*, New York: Academic Press, 1982, 119 - 135
- MILLER, M.L. & GOLDSTEIN, I.P., SPADE: A Grammar Based Editor for Planning and Debugging Programs, *AI Memo 386*, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Dec. 1976
- MILLER, M.L. & GOLDSTEIN, I.P., Problem Solving Grammars as Formal Tools for Intelligent CAI, *Proceedings of the ACM Conference*, 1977a, 220 - 226
- MILLER, M.L. & GOLDSTEIN, I.P., Structured Planning and Debugging, *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI)*, 1977b, 773 - 779
- MINSKY, M., A Framework for Representing Knowledge, in: P.WINSTON (ed), *The Psychology of Computer Vision*, New York: McGrawHill, 1975, 211-277
- MITRE Corporation, An Overview of the TICCIT Program, McLean, Virginia: MITRE Corporation, 1974 (op.cit.in: O'SHEA, 1982)
- MÖBUS, C., Die Entwicklung zum Programmierexperten durch das Problemlösen mit Automaten, in: H.MANDL & P.M.FISCHER (eds), *Lernen im Dialog mit dem Computer*, München: Urban & Schwarzenberg Publ., 1985, 140-154
- MÖBUS, C., Knowledge Specification and Instructions for a Visual Computer Language, paper presented on the "Workshop on Knowledge Representation and Information Processing", Institute of Cybernetics and Information Processes, Academy of Sciences, Berlin/GDR, 24.-28. June, 1987a
- MÖBUS, C., Logic Programs as a Specification and Description Tool in Designing an Intelligent Tutoring System,in: *Abridged Proceedings of the HCI International Conference on Human-Computer Interaction*, Honolulu, Hawaii 1987b, p.119f
- MÖBUS, C., Specifications of Instructions and Helps for an ICAI-System in the Field of Graphical Programming, paper presented at the First European Seminar on Intelligent Tutoring Systems, Commission of the European Communities, Rottenburg, 25.-31. October, 1987c
- NEVES, D.M. & ANDERSON, J.R., Knowledge Compilation: Mechanisms for the Automatization of Cognitive Skills, in: J.R. ANDERSON (ed), *Cognitive Skills and their Acquisition*, Hillsdale, N.J.: Lawrence Erlbaum Ass., 1981, 57 - 84
- NEWELL, A. & ROSENBLUM, P.S., Mechanisms of Skill Acquisition and the Law of Practice, in J.R. ANDERSON (ed), *Cognitive Skills and their Acquisition*, Hillsdale, N.J.: Lawrence Erlbaum, 1981
- O'SHEA, T., *Self Improving Teaching Systems*, Basel: Birkhäuser Verlag, 1979
- O'SHEA, T., A Self-improving Quadratic Tutor, *International Journal of Man-Machine Studies*, 1979a, 11, 97-124 and in: D.SLEEMAN & J.S.BROWN (eds), *Intelligent Tutoring Systems*, New York: Academic Press, 1982, 309 - 336
- O'SHEA, T., *Intelligent Systems in Education*, in: D.MICHIE (ed), *Introductory Readings in Expert Systems*, New York: Gordon & Breach Science Publ., 19##, 147-176
- PAGAN, F.G., *Formal Specification of Programming Languages: A Panoramic Primer*, Englewood Cliffs: Prentice Hall, 1981
- PAPERT, S., *Microworlds: Transforming Education*, in: R.W. LAWLER & M. YAZDANI (eds), *Artificial Intelligence and Education*, Norwood, N.J.: Ablex Publ. Co., 1987, 79 - 94
- PARK, O.C., PEREZ, R.S. & SEIDEL, R.J., Intelligent CAI: Old Wine in New Bottles or a New Vintage? in: G.P. KEARSLEY (eds), *Artificial Intelligence & Instruction*, Reading, Mass.: Addison-Wesley, 1987, 11-45
- PAVEL, M., MARCOVICI, S., SHERMAN, A. & FALMAGNE, J.C., ARIS: A Computer-assisted Instruction System, *Behaviour Research Methods and Instrumentation*, 1983, 15, 138-141
- PAYNE, S.J. & GREEN, T.R.G., Task-action Grammars: A Model of the Mental Representation of Task Languages, *Human Computer Interaction*, 1986, 2, 93 - 133
- PAYNE, S.J., SIME, M.E. & GREEN, T.R.G., Perceptual Structure Cueing in a Simple Command Language, *International Journal of Man-Machine Studies*, 1984, 21, 19 - 29
- PEACHEY, D.R. & McCALLA, G.I., Using Planning Techniques in Intelligent Tutoring Systems, *International Journal of Man-Machine Studies*, 1986, 24, 77-98
- PENNINGTON, N., Stimulus Structures and Mental Representations in Expert Comprehension of Computer Programs, *Cognitive Psychology*, 1987, 19, 295 - 341
- PEREIRA, F.C.N., Can Drawing Be Liberated from the von NEUMANN Style?, in: M.V. CANEGHEM & D.H.D. WARREN (eds), *Logic Programming and its Applications*, Norwood, N.J.: Ablex Publ., 1986, 175 - 187
- POHL, I., Syntactic Models of Cognitive Behavior, in: A. ELITHORN & D. JONES (ed), *Artificial and Human Thinking*, Amsterdam: Elsevier Scientific Publ. Co., 1973, 34 - 44
- POMERANTZ, J.R., Perceptual Organization in Information Processing, in: A.M. AITKENHEAD & J.M. SLACK (eds), *Issues in Cognitive Modelling*, Hillsdale, N.J.: LAWRENCE ERLBAUM Ass., 1985, 127 - 158

- PRESSEY, S.L., A Simple Apparatus Which Gives Tests and Scores and Teaches, *School and Society*, 1926, 23 (german translation in: W.CORRELL (ed.), *Programmiertes Lernen und Lehrmaschinen*, Braunschweig, 1965)
- PRESSEY, S.L., A Machine for Automatic Teaching of Drill Material, *School and Society*, 1927, 25, 1-14 (german translation in: W.CORRELL (ed.), *Programmiertes Lernen und Lehrmaschinen*, Braunschweig, 1965)
- PRESSEY, S.L., Teaching Machine (and Learning Theory) Crisis, *Journal of Applied Psychology*, 1963, 47, 1-6 (german translation in: F.WEINERT (ed.), *Pädagogische Psychologie*, Köln: Kiepenheuer & Witsch, 1961)
- RAJAN, T., APT: A Principled Design of an Animated View of Program Execution for Novice Programmers, in: H.J. BULLINGER & B. SHACKEL (eds), *Human - Computer Interaction - INTERACT '87*, Amsterdam: Elsevier Science Publishers, 1987, 291 - 296
- RESNICK, C.A., Computational Models of Learners for Computer-assisted Learning, Doctorial Dissertation, University of Illinois, Urbana-Champaign, Illinois, 1975
- REISNER, P., Formal Grammar as a Tool for Analyzing Ease of Use: Some Fundamental Concepts, in: J.C. THOMAS & M.L. SCHNEIDER (eds), *Human Factors in Computer Systems*, Norwood, N.J.: Ablex Publ. Co., 1984
- REISNER, P., Formal Grammar and Human Factors Design of Interactive Graphics System, *IEEE Transactions on Software Engineering*, Vol. SE-7, No.2, 229 - 240, 1981
- ROSS, P., Some Thoughts on the Design of an Intelligent Teaching System for PROLOG, *Artificial Intelligence and Simulation of Behavior Quarterly*, 1987, No. 62, 6 - 10
- ROSS, P. & LEWIS, J., Plan Recognition and Chart Parsing, Department of Artificial Intelligence, University of Edinburgh, DAI- Research Paper, No. 309, 1987
- SCHEEERER, E., Notes Toward a History of Cognitive Science, *International Social Science Journal*, in press
- SCHMIDT, C.F., SRIDHARAN, N.S. & GOODSON, J.L., The Plan Recognition Problem: An Intersection of Psychology and Artificial Intelligence, *Artificial Intelligence*, 1978, 11, 45-83
- SCHMITT, H. & WOHLFARTH, P., *Mathematikbuch 5N*, München: Bayerischer Schulbuchverlag, 1978
- SCHRÖDER, O., FRANK, K.D. & COLONIUS, H., Gedächtnisrepräsentation funktionaler, graphischer Programme, ABSYNT-Report 1/87, Projekt ABSYNT, FB 10, Arbeitsgruppe Lehr-Lernsysteme, Universität Oldenburg, 1987
- SELF, J.A., Student Models in Computer-aided Instruction, *International Journal of Man-Machine Studies*, 1974, 6, 261-276
- SELF, J.A., The Application of Machine Learning to Student Modelling, *Instructional Science*, 1986, 14, 327 - 338
- SELF, J.A., *Artificial Intelligence and Human Learning: Intelligent Computer-aided Instruction*, London: Chapman & Hall, 1988
- SHAPIRO, S.L.C. (ed), *Encyclopedia of Artificial Intelligence*, New York: John Wiley, 1987
- SHNEIDERMAN, B., Direct Manipulation: A Step Beyond Programming Languages, *IEEE Computer*, 1983, 16(8), 57 - 69
- SHEIDERMAN, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Reading, Mass.: Addison-Wesley, 1987
- SHU, N.C., Visual Programming Languages: A Perspective and a Dimensional Analysis, in: S. CHANG, T. ICHIKAWA & P.A. LIGOMENIDES (eds), *Visual Languages*, New York: Plenum Press, 1986, 11 - 34
- SIMON, H.A. & LEA, G., Problem Solving and Rule Induction: A Unified View, in: L.W.GREGG (ed), *Knowledge and Cognition*, Potomac, Maryland: L.Erlbaum Ass.Publ., 1974, 105-127
- SKINNER, B.F., The Science of Learning and the Art of Teaching, *Harvard Educational Review*, 1954, 24, 86-97 (german translation in: F.WEINERT (ed.), *Pädagogische Psychologie*, 247-258, Köln: Kiepenheuer & Witsch, 1967
- SKINNER, B.F., *Teaching Machines*, Science, 1958, 128, 969-977 (german translation in: W.CORRELL (ed), *Programmiertes Lernen und Lehrmaschinen*, Braunschweig, 1965)
- SKINNER, B.F., *The Technology of Teaching*, N.Y.: Appleton Century Crofts, 1968
- SKVORETZ, J., Languages and Grammars of Action and Interaction: Some Further Results, *Behavioral Science*, 1984, 29, 81-97
- SKVORETZ, J. & FARARO, Th.J., Languages and Grammars of Action and Interaction: A Contribution to the Formal Theory of Action, *Behavioral Science*, 1980, 25, 9 - 22
- SLEEMAN, D.H., A Problem Solving Monitor for a Deductive Reasoning Task, *International Journal of Man-Machine Studies*, 1975, 7, 183-211
- SLEEMAN, D.H., Assessing Aspects of Competence in Basic Algebra, in: D.SLEEMAN & J.S. BROWN (eds), *Intelligent Tutoring Systems*, New York: Academic Press, 1982
- SLEEMAN, D.H., Inferring Student Models for Intelligent Computer-aided Instruction, in: R.S. MICHALSKI, J.G.CARBONELL, T.M. MITCHELL (eds), *Machine Learning: An Artificial Intelligence Approach*, Plao Alto: Tioga Publ.Co., 1983, 483-510
- SLEEMAN, D.H., An Attempt to Understand Student's Understanding of Basic Algebra, *Cognitive Science*, 1984, 8, 387-412
- SLEEMAN, D.H., Basic Algebra Revisited: A Study with 14-year-olds, *International Journal of Man-Machine-Studies*, 1985, 22, 127-149
- SLEEMAN, D.H., Inferring (Ma)rules from Pupil's Protocols, in: L.STEELS & J.A. CAMPBELL (eds), *Progress in Artificial Intelligence*, Chichester, Sussex: Ellis Horwood Ltd, 1986
- SLEEMAN, D.H. & BROWN, J.S. (eds), *Intelligent Tutoring Systems*, New York: Academic Press, 1982
- SLEEMAN, D. & HENDLEY, R.J., ACE: A System which Analyses Complex Explanations, in: D. SLEEMAN & J.S. BROWN (eds), *Intelligent Tutoring Systems*, New York: Academic Press, 1982, 99 - 118
- SLEEMAN, D.H. & SMITH, M.J., Modelling Pupil's Problem Solving, *Artificial Intelligence*, 1981, 16, 171 - 187
- SMALLWOOD, R.D., A Decision Structure for Teaching Machines, Cambridge, Massachusetts: MIT Press, 1962
- SMALLWOOD, R.D., Optimal Policy Regions for Computer-directed Teaching Systems, in: W.H.HOLTZMAN (ed), *Computer-assisted Instruction, Testing and Guidance*, New York: Harper & Row, 1970
- SMITH, D.C., IRBY, C., KIMBALL, R., VERPLANK, B., & HARSLEM, B., Designing the STAR User Interface, *BYTE*, 1982, 7(4), 242-282

- SMITH, R.L. & BLAINE, L.H., A Generalized System for University Mathematics Instruction, SIGCUE Bulletin, 1976, 1, 280-288
- SMITH, R.L., GRAVES, W.H., BLAINE, L.H. & MARINOV, V.G., Computer-assisted Axiomatic Mathematics: Informal Rigor, in: O.LACARME & R.LEWIS (eds), Computers in Education, IFIPS (Pt.2), Amsterdam: North-Holland, 1975, 803-809
- SOLOWAY, E., Learning to Program = Learning to Construct Mechanisms and Explorations, Communications of the ACM, 1986, 29(9), 850-858
- SOLOWAY, E., I Can't Tell What in the Code Implements What in the Specs, in: G.SALVENDY (ed), Cognitive Engineering in the Design of Human-Computer Interaction and Expert Systems, Amsterdam: Elsevier Science Publ., 1987, 317-328
- SPADA, H. Modelle des Denkens und Lernens: Ihre Theorie, empirische Untersuchung und Anwendung in der Unterrichtsforschung, Bern: Hans Huber, 1976
- SPADA, H. & OPWIS, K., Intelligente tutorielle Systeme aus psychologischer Sicht, in: H.MANDL & P.M.FISCHER (eds), Lernen im Dialog mit dem Computer, München: Urban & Schwarzenberg, 1985, 13 - 23
- STEVENS, A.L. & COLLINS, A., The Goal Structure of a Socratic Tutor, Proceedings of the ACM Conference, Seattle, Washington, New York: Association for Computing Machinery, 1977, 256 - 263
- STEVENS, A.L. & COLLINS, A., Multiple Conceptual Models of a Complex System, in: R.E. SNOW, P.A. FEDERICO & W.E. MONTAGUE (eds), Aptitude, Learning and Instruction, Vol. 2: Cognitive Process Analyses of Learning and Problem Solving, Hillsdale, NJ.: 1980, 177 - 197
- STEVENS, A.L., COLLINS, A. & GOLDIN, S.E., Misconceptions in Students' Understanding, International Journal of Man-Machine Studies, 1979, 11, 145 - 156 and in: D. SLEEMAN & J.S. BROWN (eds), Intelligent Tutoring Systems, New York: Academic Press, 1982, 13 - 24
- STRITTMATTER, P.(ed), Lernzielorientierte Leistungsmessung, Weinheim: Beltz Verlag, 1973
- SUPPES, P., University-level Computer-assisted Instruction at Stanford: 1968-1980, Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1981
- SUPPES, P., FLETCHER, J.D. & ZANOTTI, M., Performance Models of American Indian Students on Computer-assisted Instruction in Elementary Mathematics, Instructional Science, 1975, 4, 303-313
- SUPPES, P., FLETCHER, J.D. & ZANOTTI, M., Models of Individual Trajectories in Computer-assisted Instruction for Deaf Students, Journal of Educational Psychology, 1976, 68, 117 - 127
- SUPPES, P., JERMAN, M. & BRIAND, D., Computer Assisted Instruction: The 1965-66 Stanford Arithmetic Program, N.Y.: Academic Press, 1968
- SUPPES, P. & MORNINGSTAR, M., Computer-Assisted Instruction at Stanford 1966-68: Data, Models and Evaluation of Arithmetic Programs, N.Y.: Academic Press, 1972
- TAUBER, M., A Computer-aided Management System in Distance Education, European Journal of Education, 15, 285 - 297, 1980
- TCHOGOVDADZE, G.G., Some Steps Towards Intelligent Computer Tutoring Systems, Microprocessing and Microprogramming, 1985, 16, 1-5
- TUFTE, E., The Visual Display of Quantitative Information, Cheshire, Connecticut: Graphics Press, 1985⁵
- TULVING, E., Elements of Episodic Memory, London: Oxford University Press, 1983
- UNGER, S. & WYSOTZKI, F., Lernfähige Klassifizierungssysteme, Berlin: Akademie-Verlag, 1981
- UTTAL, W.R., ROGERS, M. HIERONYMOUS, R. & PASICH, T. Generative Computer Assisted Instruction in Analytic Geometry, Newburyport, MA.: Entelek, Inc., 1969
- VanLEHN, K., Bugs are not Enough: Empirical Studies of Bugs, Impasses and Repairs in Procedural Skills, XEROX Parc, Cognitive and Instructional Sciences Group, 1981, CIS-11 (SSL-81-2) and Journal of Mathematical Behavior, 1982, 3, 3 - 72
- VanLEHN, K., On the Representation of Procedures in Repair Theory, in: H.P. GINSBURG (ed), The Development of Mathematical Thinking, New York: Academic Press, 1983, 197 -252
- VanLEHN, K., Learning One Subprocedure per Lesson, Artificial Intelligence, 1987a, 31, 1-40
- VanLEHN, K., Towards a Theory of Impasse-driven Learning, ONR, Techn.Rep., CMU-University, Pittsburgh, USA, 1987b
- VanLEHN, K. & BALL, W., A Version Space Approach to Learning Context-free Grammars, Machine Learning, 1987c, 2, 39 - 74
- VanLEHN, K. & BROWN, J.S., Planning Nets: A Representation for Formalizing Analogies and Semantic Models of Procedural Skills, in: R.E. SNOW, P.-A. FEDERICO & W.E. MONTAGUE (eds), Aptitude, Learning and Instruction, Vol. II, Cognitive Process Analyses of Learning and Problem Solving, Hillsdale, NJ.: Lawrence Erlbaum Ass., 1980, 95 - 137
- WALOSZEK, G., WEBER, G. & WENDER, K.F., Entwicklung eines intelligenten LISP-Tutors, Institut für Psychologie, 1986/2, Technische Universität Braunschweig
- WALOSZEK, G., WEBER, G. & WENDER, K.F., Probleme der Wissensrepräsentation in einem intelligenten LISP-Tutor, in: HEYER & KREMS (eds), Wissensarten und ihre Darstellung, Informatik Fachberichte, Heidelberg: Springer (in press)
- WEBER, R.J. & KOSSLYN, S.M., Computer Graphics and Mental Imagery, in: S. CHANG, T. ICHIKAWA & P.A. LIGOMENIDES (eds), Visual Languages, New York: Plenum Press, 1986, 305 - 324
- WEBER, G., WALOSZEK, G. & WENDER, K.F., The Role of Episodic Memory in an Intelligent Tutoring System, in: J.A. SELF (ed), Artificial Intelligence and Human Learning: Intelligent Computer-aided Instruction, London: Chapman & Hall, 1988
- WEINERT, F. (ed), Pädagogische Psychologie, Köln: Kiepenheuer & Witsch, 1967

- WENGER, E., *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*, Los Altos: Morgan Kaufman Publishers, Inc., 1987
- WERTZ, H., *Stereotyped Program Debugging: An Aid for Novice Programmers*, *International Journal of Man-Machine Studies*, 1982, 16, 379-392
- WERTZ, H., *Intelligence Artificielle: Application à l'Analyse de Programmes*, Paris: Masson, 1985
- WERTZ, H., *Automatic Correction and Improvement of Programs*, Chichester, West Sussex: Ellis Horwood Ltd, 1987
- WEXLER, J.D., *Information Networks in Generative Computer-assisted Instruction*, *IEEE Transactions on Man-Machine Systems*, 1970, 11, 181-190
- WILLIAMS, G., *The LISA Computer System*, *BYTE*, 1983, 8(2), 33-50
- WILLIAMS, G., *The Apple MACINTOSH Computer*, *BYTE*, 1984, 9(2), 30-54
- WILLIAMS, G., *HyperCard*, *BYTE*, Vol.12, 109 - 117, 1987
- WILLIAMS, M., HOLLAN, J. & STEVENS, A., *Human Reasoning About a Simple Physical System*, in: D.GENTNER & A.STEVENS (eds), *Mental Models*, Hillsdale, N.J.: Erlbaum Press, 1983
- WOOD, W.T & WOOD, S.K., *Icons in Everyday Life*, in: G.SALVENDY, S.L.SAUTER & J.J.HURRELL (eds), *Social, Ergonomic and Stress Aspects of Work with Computers*, Amsterdam: Elsevier Science Publ., 1987, 97-104
- WOOLF, B.P., *Theoretical Frontiers in Building a Machine Tutor*, in: G.P.KEARSLEY (ed), *Artificial Intelligence & Instruction*, Reading: Mass., 1987, 229-267
- YAZDANI, M., *Intelligent Tutoring Systems Survey*, *Artificial Intelligence Review*, 1986,1, 43-52
- YAZDANI, M., *Intelligent Tutoring Systems: An Overview*, in: R.W. LAWLER & M. YAZDANI (eds), *Artificial Intelligence and Education*, Vol. 1, 183 - 201, 1987
- YOB, G., *Hunt the Wumpus*, *Creative Computing*, Sept./Oct., 1975, 51 - 54
- YOUNG, R.M. & O'SHEA, T., *Errors in Children's Subtraction*, *Cognitive Science*, 1981, 5, 153-177

8. Appendices

8.1 **Appendix A:** Instructions for reading the book "Analysis of Behavior" (HOLLAND & SKINNER, 1961,p.viif., p.1f.) with an excerpt of part I:

To the Student

With this book the student should be able to instruct himself in that substantial part of psychology which deals with the analysis of behavior - in particular the explicit prediction and control of behavior of people. The practical importance of such a science scarcely needs to be pointed out, but understanding and effective use of the science require fairly detailed knowledge. This program is designed to present the basic terms and principles of the science. It is also designed to reveal the inadequacy of popular explanations of behavior and to prepare the student for rapidly expanding extensions into such diverse fields as social behavior and psychopharmacology, space flight and child care, education and psychotherapy. This book is itself one application of the science.

How to Use the Book


The material was designed for use in a teaching machine. The teaching machine presents each item automatically. The student writes his response on a strip of paper revealed through a window in the machine. He the operates the machine to make his written response inaccessible, though visible, and to uncover the correct response for comparison.

Where machines are not available, a programmed textbook such as this may be used. The correct response to each item appears on the following page, along with the next item in the sequence. Read each item, write your response on a separate sheet of paper, and then turn the page to see whether your answer is correct. If it is incorrect, mark an "x" beside it. Then read and answer the next question, and turn the page again to check your answer.

Writing out the answer is essential. It is also essential to write it *before* looking at the correct answer. When the student, though well-intentioned, glances ahead without first putting down an answer of his own, he commits himself to only a vague and poorly formulated guess. This is not effective and in the long run makes the total task more difficult.

It is important to do each item in its proper turn. The sequence has been carefully designed, and occasional apparent repetitions or redundancies are there for good reason. Do not skip. If you have undue difficulty with a set, repeat it before going on to the text. A good rule is to repeat any set in which you answer more than 10 per cent of the items incorrectly. Avoid careless answers. If you begin to make mistakes because you are tired or not looking at the material carefully, take a break. If you are not able to work on the material for a period of several days, it may be advisable to review the last set completed.

The review sets will help you to find your weaknesses. When you miss an item in a review set, jot down the set number given in the answer space and review that set after you have completed the review set.

Set 1	PART I Reflex Behavior Simple Reflexes Estimated time: 23 minutes Turn to next page and begin		A doctor taps your knee (patellar tendon) with a rubber hammer to test your _____.
stimulus (tap on the knee) 1-7	Technically speaking, a reflex involves an eliciting stimulus in a process called elicitation. A stimulus _____ a response. 1-8	elicits 1-8	To avoid unwanted nuances of meaning in popular words, we do not say that a stimulus "triggers," "stimulates," or "causes" a response, but that it _____ a response. 1-9
threshold 1-15	The fraction of a second which elapses between "brushing the eye" and "blink" is the _____ of the reflex. 1-16	latency 1-16	In the patellar-tendon reflex, a forceful tap elicits a strong kick; a tap barely above the threshold elicits a weak kick. Magnitude of response thus depends on the intensity of the _____. 1-17
threshold 1-23	The greater the concentration of onion juice (stimulus), the _____ the magnitude of the response. 1-24	greater (higher larger) 1-24	Onion juice elicits the secretion of tears by the lachrymal gland. This causal sequence of events is a(n) _____. 1-25
elicit 1-31	In the pupillar reflex, a very bright flash of light elicits a response of greater _____ than a weak flash of light. 1-32	magnitude (intensity) 1-32	A response and its eliciting stimulus comprise a(n) _____.
latency 1-39	A solution of lemon juice will not elicit salivation if the stimulus is _____ the threshold. 1-40	below (less than, sub-) 1-40	The latency of a reflex is the (1) _____ between onset of (2) _____ and _____. 1-41
(1) magnitude (2) latency 1-47	Presentation of a stimulus is the "cause" of a response. The two from a(n) _____. 1-48 Page 1	reflex 1-48	The layman frequently explains behavior as the operation of "mind" or "free will." He seldom does this for reflex behavior, however, because the _____ is an adequate explanation of the response. 1-49 Page 2

Is it true that this is an <i>atom</i> ?	Yes, because <i>atom</i> is a string of characters beginning with the letter <i>a</i> .	No, since this is actually two <i>S-expressions</i> not enclosed by parentheses. The first one is a list containing two atoms, and the second one is an atom.
Is it true that this is an atom? <i>turkey</i>	Yes, because <i>turkey</i> is a string of characters beginning with a letter.	Yes, because the two <i>S-expressions</i> are now enclosed by parentheses.
Is it true that this is an atom? 1492	Yes, since 1492 is a string of characters beginning with a digit.	Yes, because all atoms are <i>S-expressions</i> .
Is it true that this is an atom? <i>3turkeys</i>	Yes, since <i>3turkeys</i> is a string of characters beginning with a digit.	Yes, because it is a list.
Is it true that this is an atom? <i>u</i>	Yes, because <i>u</i> is a string of one character beginning with a letter or digit.	Yes, because all lists are <i>S-expressions</i> .
Is it true that this is an atom? *abc\$	Yes, because *abc\$ is a string of characters beginning with a letter, digit, or special character other than a left "(" or right ")" parenthesis.	Yes, because it is a collection of <i>S-expressions</i> enclosed by parentheses.
Is it true that this is a <i>list</i> ? (atom)	Yes, because (atom) is an atom enclosed by parentheses.	Six, how, are, you, doing, so, and far.
Is it true that this is a list? (atom turkey or)	Yes, because it is a collection of atoms enclosed by parentheses.	Yes, because it is a collection of <i>S-expressions</i> enclosed by parentheses.

8.3 Appendix C: A dialog with ANDERSON's LISP tutor

Define the function fact. It takes a number argument, n , and computes the factorial of n , which is the result of multiplying $n \times n - 1 \times n - 2 \dots \times 1$. For instance, $(\text{fact } 3) = 3 \times 2 \times 1 = 6$. by definition, the factorial of 0 is 1.

```
1. (defun fact (n)
  (cond ((equal) <ACTION>))
  <RECURSIVECASE>))
```

I assume that you are trying to test if the number argument is equal to the number zero. There is a special function for testing equivalence to zero. Try to use that function instead of EQUAL.

What code can you write to test if the condition for terminating case is 'true'? Try writing a predicate.

```
2. (defun fact (n)
  (cond ((zero <ACTION>))
  <RECURSIVECASE>))
Did you mean zero? (y or n)? y
```

```
3. (defun fact (n)
  (cond ((zerop n) 1)
  <RECURSIVECASE>))
```

Is the value of fact in the recursive cases calculated by:

PRESS: IF YOU WANT TO:

- 1 Calling fact recursively with n .
- 2 Multiplying n by fact of n minus 1.
- 3 Getting $1 \ 2 \dots n$ and then multiplying the results.
- 4 have the tutor choose

Menu choice: 3

That is a reasonable way to think of doing factorials but it is not a plan for a recursive function. Since you seem to be having trouble with the recursive cases, let us work through some examples and figure out the conditions and actions for each of these cases.

The recursive relationship is the relationship between fact called

with n and fact called with

PRESS: IF YOU WANT TO

- 1 n minus 1
- 2 n equal to 0
- 3 n plus 1
- 4 have the tutor choose

Menu choice: 1

We will consider some concrete examples. We will characterize the recursive relationship between fact called with 1 and fact called with 0

What is the value of (fact 1)?

Answer: 1

What is the value of (fact 0)?

Answer: 1

EXAMPLES

```
fact (n)
A. (fact 1) = 1 (fact 0) = 1
```

We will consider some more concrete examples. We will characterize the recursive relationship between fact called with 3 and fact called with 2

What is the value of (fact 3)?

Answer: 6

What is the value of (fact 2)?

Answer: 2

EXAMPLES

```
fact (n)
fact (n - 1)
A. (fact 1) = 1 (fact 0) = 1
B. (fact 3) = 6 (fact 2) = 2
```

Since there is only one recursive case characterized by the examples you have just worked out the condition in this case can be just coded as T. We will just focus on characterizing the action in this case.

In examples A and B what do you have to do to get the result of fact called with n ?

PRESS:

IF YOU WANT TO:

- 1 Multiply n by one less than n .
- 2 Multiply n by fact of one less than n .
- 3 Add n to the result of fact called with one less than n
- 4 have the tutor choose

Menu choice: 2

We have now worked through a series of examples that illustrate the recursive relation. We will now return to the goal of trying to produce LISP code that specifies the recursive relation.

```
4. (defun fact (n)
  (cond ((zerop n) 1)
  (t (fact))))
```

You will have to use fact at some point but right now you want to multiply.

```
5. (defun fact (n)
  (code ((zerop n) 1)
  (t (times <NUM> <NUM2>))))
```

What do you want to do first?

PRESS:

- IF YOU WANT TO:
- 1 Write code to get the number argument
 - 2 Write code to get the recursive part

Menu choice: 1

```
(defun fact (n)
  (cond ((zerop n) 1)
  (t (times n (fact n)))))
```

You cannot just code n at this point. What you want to do is subtract one from n .

```
7. (defun fact (n)
  (cond ((zerop n) 1)
  (t (times n (fact (difference))))))
```

Difference would work in this case but it is more appropriate to use the function SUB1 when you are subtracting 1 from a number.

Figure 9.1

Figure 9.1 continued

(ANDERSON & REISER, 1985; ANDERSON & SKWARECKI, 1986; ANDERSON, 1987)

```

8. (defun fact (n)
    (cond ((zerop n) 1)
          (t (times n (fact (sub1 n))))))
... YOU ARE DONE. TYPE NEXT TO GO ON AFTER ...
... TESTING THE FUNCTIONS YOU HAVE DEFINED ...
THE LISP WINDOW

=> (trace fact)
(fact)
=> (fact 3)
1 <Enter> fact (3)
2 <Enter> fact (2)
3 <Enter> fact (1)
4 <Enter> fact (0)
4 <EXIT> fact 1
3 >EXIT> fact 1
2 <EXIT> fact 2
1 <EXIT> fact 6
6

```

Figure 9.1 continued

(ANDERSON & REISER, 1985; ANDERSON & SKWARECKI, 1986; ANDERSON, 1987)

8.4 Appendix D: A verbal description of the operational semantics of the recursive "calculation sheet" machine, an example of a diagrammatic form program and the corresponding trace

8.4.1 A verbal instruction

2.3.3 Die (rekursive) Formularmaschine

Der Gang der Berechnung einer Rechenvorschrift ist bis auf Kollateralität durch ein zugehöriges Formular festgelegt. Kommt im Formular selbst wieder eine Rechenvorschrift als Operation vor⁴⁴, so ist ein Formular dieser Rechenvorschrift anzulegen („Aufruf“) und deren Ergebnis schließlich rückzuübertragen. Dies gilt auch für eine rekursiv definierte Rechenvorschrift – mit der Besonderheit, daß im Lauf der Berechnung entsprechend den rekursiven Aufrufen weitere Exemplare des Formulars eben dieser Rechenvorschrift benötigt werden.

Zu jedem Aufruf werden in ein neues Exemplar des Formulars zunächst linksseitig die jeweiligen Argumentwerte eingetragen (*'call by value'*). Man nennt jedes solche Exemplar eine **Inkarnation** der Rechenvorschrift; um die Übersicht zu behalten, kann man die Inkarnationen und die entsprechenden Aufrufe im Verlauf der Berechnung durchnummerieren.

Für den rekursiven Fall ist es nun besonders bedeutsam, daß die Fallunterscheidung eine arbeitssparende Auswahl trifft: nachdem die Parameterbezeichnungen durch die linksseitig festgestellten Argumentwerte ersetzt sind, werden daher auf dem Urformular und allen folgenden Inkarnationen möglichst zuerst die Bedingungen ausgewertet und sodann die unzulässigen Zweige gekappt. Die Rekursion endet mit Inkarnationen, in denen kein Zweig mehr verbleibt, der einen rekursiven Aufruf enthält. Die ganze Berechnung **terminiert** (für einen bestimmten Parametersatz), wenn sie nur endlich viele Inkarnationen benötigt.

Die Tätigkeit eines Menschen, der auf diese Weise mit Formularen arbeitet, kann in einsichtiger Weise auch mechanisiert werden. Man gelangt so zum Begriff einer rekursiven (Gedanken-)Maschine, der **Formularmaschine**, in der die volle Freiheit der Berechnung noch erhalten ist. Man beachte, daß ein neues Exemplar eines Formulars auch dann angelegt wird, wenn die gleichen Argumente schon einmal aufgetreten sind: die Formularmaschine macht (auf der hier geschilderten Stufe) von einer möglichen Mehrfachverwendung eines Ergebnisses keinen Gebrauch.

Das oben erwähnte Kappen von Zweigen ist insbesondere dann ohne weiteres möglich, wenn in den Bedingungen keine rekursiven Aufrufe vorkommen. Noch übersichtlicher ist der Fall der **linearen Rekursion**, bei der außerdem in den einzelnen Zweigen der Fallunterscheidung höchstens ein rekursiver Aufruf vorkommt; dann wird nämlich in jeder Inkarnation höchstens eine neue Inkarnation angestoßen. Fast alle bisher behandelten Beispiele fallen übrigens in diese Klasse.

Für *fac* von 2.3.2 arbeitet eine Formularmaschine wie in Abb. 59 angeben. Typisch für die Rekursion ist das ‚Nachklappern‘ der Berechnung: Erst wenn die Rekursion mit der Inkarnation $fac^{(3)}$ geendet hat, werden die zurückgestellten Berechnungen in $fac^{(2)}$, $fac^{(1)}$ und $fac^{(0)}$ durchführbar und auch durchgeführt⁴⁵; das Urformular $fac^{(0)}$ liefert schließlich das Endergebnis. Das Nachklappern kann in besonders gelagerten Fällen von Aufrufen zu einem bloßen Rückübertragen der Ergebnisse der einzelnen Inkarnationen degenerieren, wie Abb. 60 für das Beispiel *gcd1* (15, 9), vgl. 2.3.2 zeigt. Ein solcher Aufruf heißt **schlicht**. Wenn in linearer Rekursion ausschließlich schlichte Aufrufe vorliegen, spricht man von **repetitiver Rekursion**.

Bei linear rekursiven Rechenvorschriften ist – abgesehen von der sonstigen Kollateralität des Formulars – die Reihenfolge, in der die benötigten Inkarnationen angestoßen werden, eindeutig bestimmt. Dies ist nicht notwendig so im allgemeinen Fall: wenn in einem Zweig mehrere Aufrufe vorkommen, so erlaubt die Kollateralität unter Umständen verschiedene Reihenfolgen und sogar Parallelarbeit.

⁴⁴ Für primitive, d. h. den zugrundeliegenden Rechenstrukturen entstammende Operationen ist kein Formular erforderlich.

8.4.2 A "calculation sheet" program

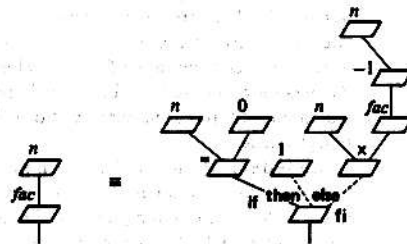
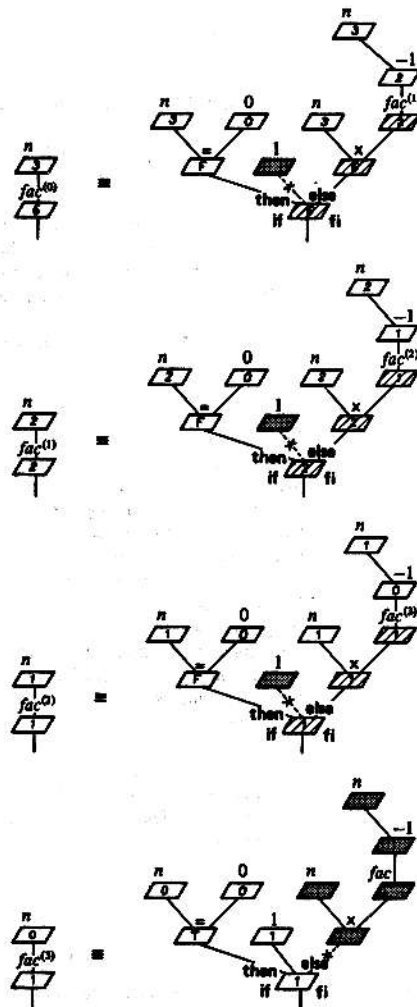


Abb. 54. Formular von fac

This program is shown on page 104 of BAUER & GOOS (1982).

8.4.2 A trace in computing the factorial(3)

Abb. 59. Arbeitsweise der Formelmaschine am Beispiel $fac(3)$

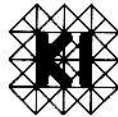
This trace can be found on page 112 of BAUER & GOOS (1982).

Thomas Christaller (Hrsg.)

Künstliche Intelligenz

5. Frühjahrsschule, KIFS-87
Günne, 28. März – 5. April 1987

Proceedings



Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo

Herausgeber

Thomas Christaller
Forschungsgruppe Expertensysteme
Gesellschaft für Mathematik und Datenverarbeitung
Postfach 12 40, D-5205 St. Augustin

CR Subject Classification (1987): I.2.3-6

ISBN 3-540-50884-8 Springer-Verlag Berlin Heidelberg NewYork
ISBN 0-387-50884-8 Springer-Verlag NewYork Berlin Heidelberg

CIP-Titelaufnahme der Deutschen Bibliothek.

Künstliche Intelligenz: ... Frühjahrsschule; proceedings / KIFS ... - Berlin; Heidelberg;
New York; London; Paris; Tokyo: Springer.

Teilw. mit d. Erscheinungsorten Berlin, Heidelberg, New York. -

Teilw. mit d. Erscheinungsorten Berlin, Heidelberg, New York, Tokyo. -

Bis 1985/86 ohne Kongressbenennung

NE: KIFS

5. 1987. Güinne, 28. März - 5. April 1987. - 1989

(Informatik-Fachberichte; 202 : Subreihe künstliche Intelligenz)

ISBN 3-540-50884-8 (Berlin ...) brosch.

ISBN 0-387-50884-8 (New York ...) brosch.

NE: GT

Dieses Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachdrucks, des Vortrags, der Entnahme von Abbildungen und Tabellen, der Funksendung, der Mikroverfilmung oder der Vervielfältigung auf anderen Wegen und der Speicherung in Datenverarbeitungsanlagen, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Eine Vervielfältigung dieses Werkes oder von Teilen dieses Werkes ist auch im Einzelfall nur in den Grenzen der gesetzlichen Bestimmungen des Urheberrechtsgesetzes der Bundesrepublik Deutschland vom 9. September 1965 in der Fassung vom 24. Juni 1985 zulässig. Sie ist grundsätzlich vergütungspflichtig. Zuwiderhandlungen unterliegen den Strafbestimmungen des Urheberrechtsgesetzes.

© by Springer-Verlag Berlin Heidelberg 1989

Printed in Germany

Druck- und Bindearbeiten: Weihert-Druck GmbH, Darmstadt
2145/3140 - 543210 - Gedruckt auf säurefreiem Papier