Computers in Human Behavior, Vol. 6, pp. 31–49, 1990
Printed in the U.S.A.

0747-5632/90 $3.00 + .00

Instruction-Based Knowledge Acquisition and Modification:

The Operational Knowledge for a Functional, Visual Programming Language*

Olaf Schröder, Klaus-Dieter Frank, Klaus Kohnert, Claus Möbus, and Matthias Rauterberg

University of Oldenburg

Abstract— *This contribution deals with instruction-based knowledge acquisition in a fairly complex but well-defined domain. The domain is the operational knowledge about the interpreter of ABSYNT, a functional, visual programming language which was developed in our project. Runnable specifications of the ABSYNT-interpreter were translated into sets of visual rules, serving as instructional material for students to acquire the operational knowledge.*

We are concerned with the following questions:

1. *How do subjects acquire the operational knowledge while simulating the interpreter of ABSYNT with the help of the instructional material?*
2. *How can the operational knowledge gained by subjects be described? For example, does this knowledge differ from the instructional material?*

If the mental representation of the operational knowledge is isomorphic to the instructional material, then hypotheses about certain performance aspects can be stated. An experiment was conducted in which dyades of programming novices acquired the computational knowledge for ABSYNT by computing the value of ABSYNT-programs with the help of the instructions, thus simulating the interpreter. The hypotheses were disconfirmed. The results suggest that the mental representation of the operational knowledge consists of larger units than the instructional material, leading to the following hypotheses about the acquisition process and the mental representation of the operational knowledge:

1. *When faced with a difficulty, there will be problem solving with the help of the instructions. Thus new knowledge is acquired by failure-driven learning.*
2. *When faced with familiar situations, compound rules are built. Thus the existing knowledge is improved by success-driven learning.*

---

*The research was sponsored by the Deutsche Forschungsgemeinschaft (DFG) in the SPP Psychology of Knowledge under Grant No. MO 292/3-3.

Requests for reprints should be addressed to Olaf Schröder, Project ABSYNT, FB 10, Unit on Tutoring and Learning Systems, University of Oldenburg, D-2900 Oldenburg, Federal Republic of Germany.

A specification of the instruction-based acquisition of the operational knowledge for ABSYNT is currently developed, based on the detailed analysis of a portion of the data gathered. This specification and some of its implications are described.

GENERAL RESEARCH PROBLEM

This work is done in the project "Developing a Knowledge-Diagnosis and Error-Explanation System for the Acquisition of Programming Knowledge for ABSYNT" (Janke & Kohnert, 1989; Möbus, 1985; Möbus & Schröder, 1989; Möbus & Thole, 1989). ABSYNT (Abstract Syntax Trees) is a purely functional, visual programming language developed in the project, using ideas from the "calculation sheet machine" (Bauer & Goos, 1982). ABSYNT includes abstraction, recursion, and call-by-value-semantics.

The goal of the project is to build a problem solving monitor for programming in ABSYNT which analyzes the students' blueprints, gives comments and proposals (Sleeman & Hendley, 1982).

ABSYNT-programs are collections of trees. The nodes of the trees are constants, parameters, and primitive and self-defined operators. The connections between the nodes are the "pipelines" for control and data flow. Programs are edited by taking nodes with the mouse from a menu bar and connecting them. Figure 1 shows

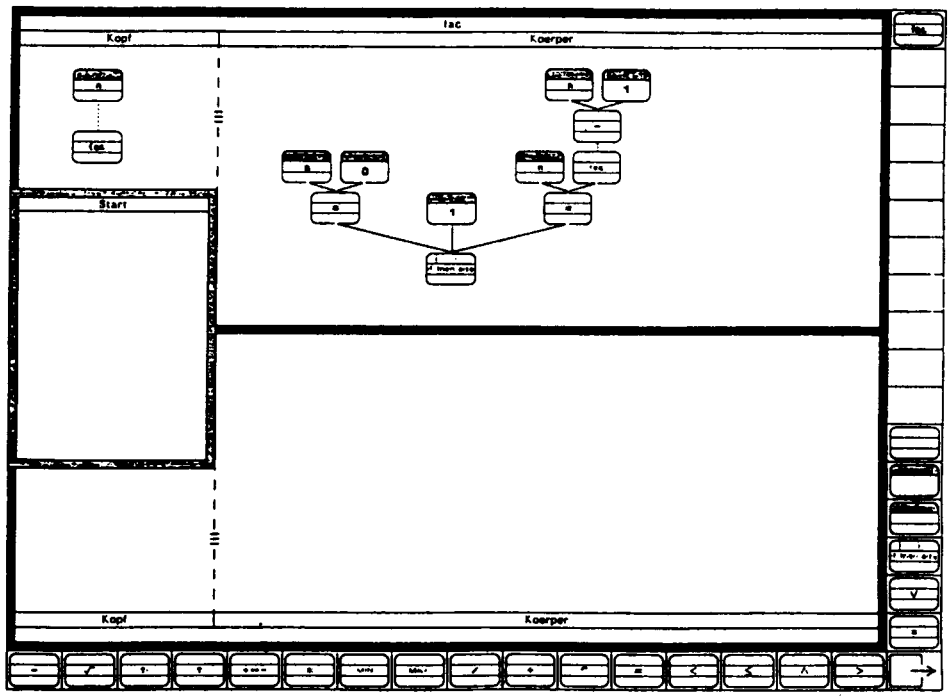

Figure 1. ABSYNT-program for the factorial.

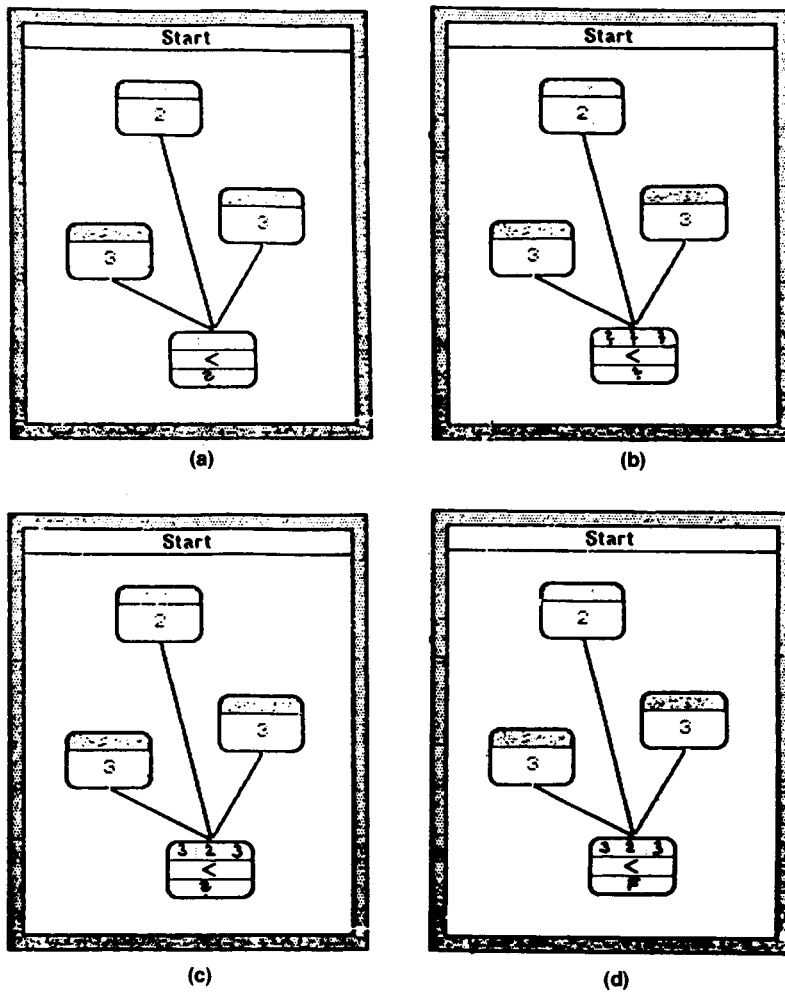

Figure 2. ABSYNT start tree without function calls, just one primitive operator node. (a) Before computation, (b) after application of the visual rule in Figure 3, (c) after 3 applications of the visual rule in Figure 4, (d) after application of the visual rule in Figure 5.

an ABSYNT-program for the factorial. A simple start tree without function calls is displayed in Figures 2a to d.

The ABSYNT-environment also has a visual trace which was implemented according to the runnable specification (Davis, 1982) of the interpreter. The trace makes every computational step visible. Thus ABSYNT is suited to support students in learning the basic functional programming concepts, and to facilitate studying these processes (Janke & Kohnert, 1989; Möbus & Thole, 1989).

We think that a programmer needs a precise understanding of the interpreter for successful programming. Evidence gathered so far is in accordance with this assumption: Subjects who had acquired the operational knowledge—that is, the knowledge about how the ABSYNT-interpreter works—made heavy and successful use of the visual trace while they were programming in ABSYNT.

The operational knowledge for ABSYNT can be acquired in the following way: Since we had developed two alternative runnable specifications of the ABSYNT-interpreter, we translated them into two alternative sets of visual rules, following design principles motivated by Larkin and Simon (1987) and Pomerantz (1985) (see Möbus & Thole, 1989). These visual rules serve as instruction and help material for the acquisition of the operational knowledge (Möbus & Schröder, 1989; Möbus & Thole, 1989).

Besides programming mode and trace mode, ABSYNT also has a prediction mode. In this mode, the student can simulate the interpreter by performing the computational steps of the ABSYNT-interpreter with the mouse and keyboard with the help of the instructional material. So the student can acquire the operational knowledge in an instruction-based way.

Here we deal with the instruction-based knowledge acquisition in the fairly complex, but well-defined domain of the operational knowledge about the interpreter of ABSYNT. We divide this issue into the following problems:

1. How can the acquired knowledge (the mental representation) be described? Does it differ from the instructions? If so, in what respects, and why (see also Newell & Simon, 1972, Chapter 3)? Why do certain difficulties, bugs, and misinterpretations arise while using the instructions?

2. How do subjects acquire the operational knowledge while simulating the interpreter of ABSYNT under guidance of the instructional material?

3. What do the answers to these questions imply for the design and improvement of the instructional material?

We think that these questions are also relevant to the issues of computer-assisted instruction and for the design of tutorial systems:

1. It is important to know what the student is being told, and what is left out (what is not mentioned) when acquiring a new knowledge domain. His/her concepts and misconceptions may originate in the instructions given. In the domain of the operational knowledge, we approached this problem by basing the instructions on runnable specifications. The instructional material could also be used for help and feedback in case of problems.

2. Understanding the process of instruction-based knowledge acquisition may motivate design principles. For instance, our investigations indicate that the instructions could be adapted to the student's current knowledge state by merging visual rules together. This would create visual compound rules. (An example will be given below.) So if there is a specification of the instruction-based knowledge acquisition process, then the instructions could be adapted step by step to the actual knowledge state of the learner. Thus it would be possible to provide help which is tailored to the subject.

TASK ANALYSIS, GENERAL ASSUMPTIONS, AND HYPOTHESES

There are two alternative versions of the instructional material: two alternative visual rule sets. They are completely displayed in Möbus and Schröder (in press). The first rule set consists of eight rules. It is "operator-centered," since each rule describes the complete computation of some ABSYNT-node. Most rules of this set consist of several subrules. The application of those rules consists of several steps:

(a) The first subrule is applied, (b) then the rule instantiation is suspended and stacked, and other rules have to be applied, until (c) the rule instantiation is retrieved, and the next subrule is applied.

In contrast, the other rule set consists of 16 rules. It is "state-centered," since each rule describes a specific change of a state of some node. This rule set has a flat structure. Like a production system, any applicable rule may be applied, and then the rule instantiation may be forgotten immediately. As an example, Figures 3 to 5 show rules of the state-centered rule set.

The visual rules shown in Figures 3 and 5 roughly correspond to the two subrules of the first visual rule of the operator-centered rule set.

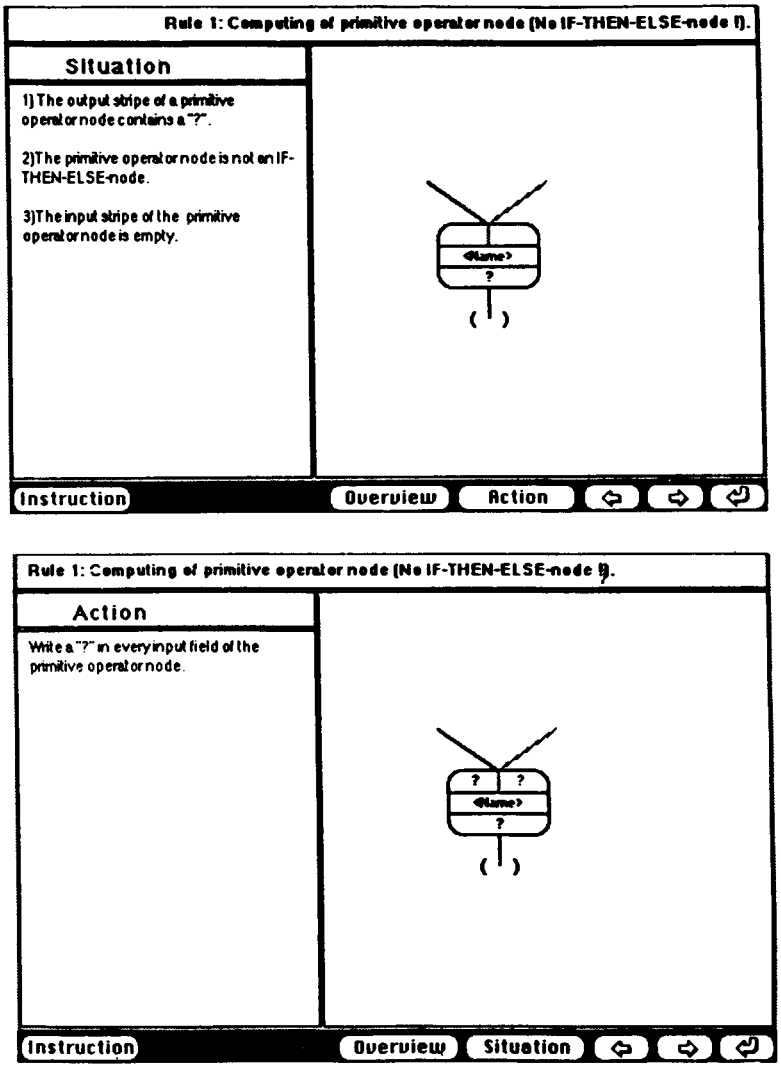

Figure 3. Rule of the state-centered rule set.

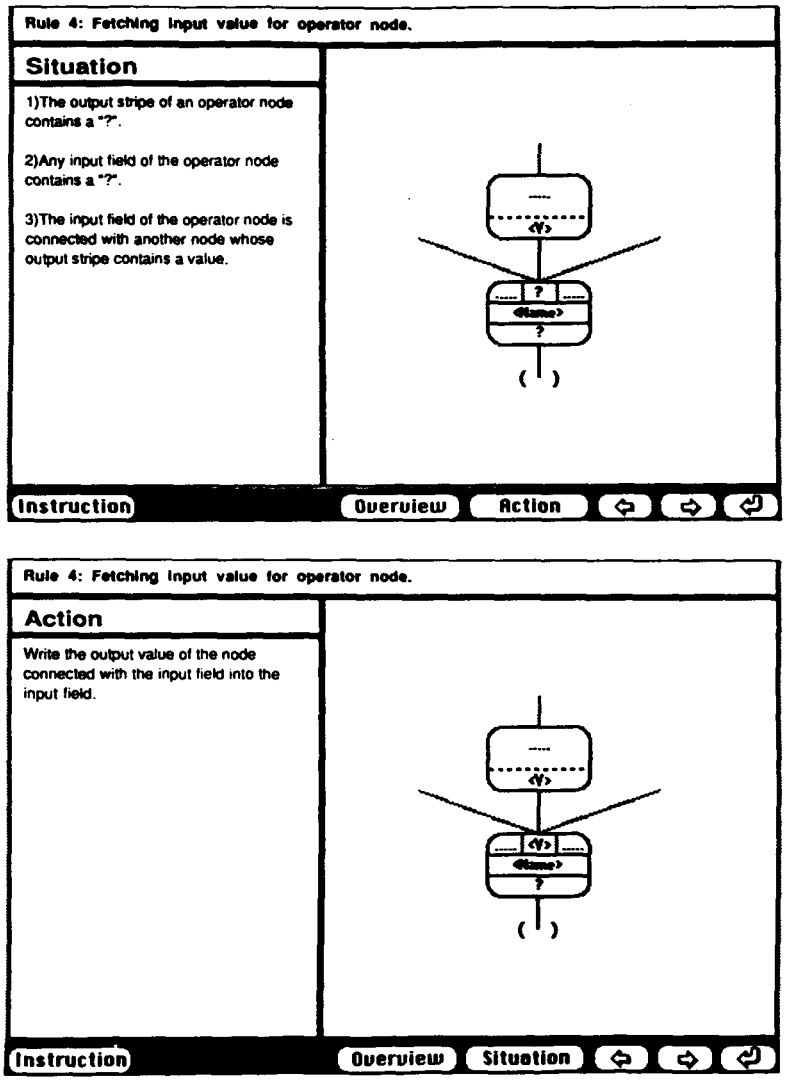

Figure 4. Rule of the state-centered rule set.

How might the operational knowledge built up by subjects with the help of the instructional material be described? Anderson, Greeno, Kline and Neves (1981), van Dijk and Kintsch (1983, Chapter 10), and Kintsch and Greeno (1985) seem to share the following view about the utilization of domain-specific knowledge: The units of the domain knowledge (for example, a geometry postulate: Anderson et al., knowledge about set relations and properties: Kintsch & Greeno) are represented as facts; at least as long as they are not proceduralized. So in order to make use of a knowledge unit, an instance of it has to be loaded into working memory and instantiated by the features of the current task. After the actions specified in this instantiation are executed, the instantiation is discarded. When the actions of the instantiation cannot be performed immediately, because intermediate, preparatory

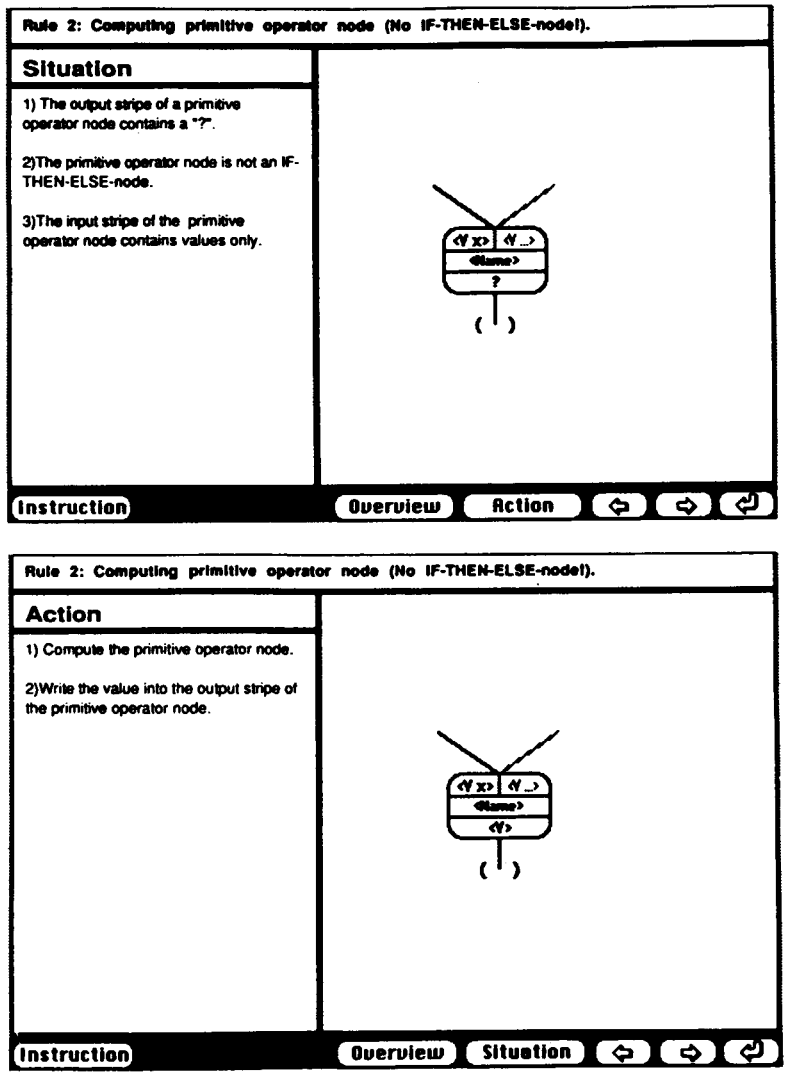

Figure 5. Rule of the state-centered rule set.

steps are necessary, then the instantiation must be held in working memory until the time for its continuation or completion has come.

Kintsch and Greeno (1985) found that problem difficulty can be related to the amount of information which must be simultaneously available in working memory. Information which is needed but not available in working memory at a given point in the problem-solving process has to be retrieved from an "episodic memory." This may cause errors. Thus performance on a problem is better if there is less information in working memory. Similarly, Egan and Greeno (1974) found in the domain of the Tower-of-Hanoi-puzzle that the number of goals that have to be remembered in order to perform a certain disc move is related to the number of errors made instead of this move.

Following these ideas and findings, we made the following assumptions concerning the mental representation of the operational knowledge constructed by subjects:

1. The operational knowledge acquired by subjects consists of units. Each unit is the mental representation of a visual rule of the instructional material. Thus each unit is an "internal rule" corresponding to an "external rule." This assumption seems particularly plausible in the domain at hand for two reasons: (a) It is easy to find subjects who will not have any preknowledge of the operational knowledge for a functional programming language, so the knowledge built up by these subjects would be mainly determined by the visual, external rules of the instructional material, (b) since the instructional material for the operational knowledge of the ABSYNT-interpreter rests on runnable specifications, we can be sure that it is precise, complete, and correct. So if one constructs a mental representation which is isomorphic to the instructions, then one could master any situation within the knowledge domain. Thus the operational knowledge can be acquired as presented in the instructions without doing any further inferences.

2. In order to apply an internal rule, this rule has to be loaded into working memory and instantiated with some part of the computational situation currently visible on the screen. Thus, the elements of the working memory are instantiated internal rules.

3. There is a limited working memory capacity. This means that (a) as more elements are in working memory, the performance decreases. Also, (b) the more new working memory elements are created after the creation of a certain working memory element, the more likely it is that this working memory element gets lost.

According to these assumptions, the mental representation of the operational knowledge constructed by subjects while simulating the interpreter with the help of the instructions is isomorphic to the instructional material—the visual rules (first assumption, see above). So the mental operations should reflect the structure of the used rule set (second assumption): Users of the operator-centered rule set would have to stack and to retrieve rule instantiations. In contrast, users of the state-centered rule set would not.

With respect to the assumed limited working memory capacity (third assumption), we expected that users of the operator-centered rule set would have difficulties with certain computation situations on the screen, where according to the operator-centered rule set (a) the number of rule instantiations on the stack is high, or (b) a rule instantiation that has been on the stack for a long time has to be retrieved. That is, there are many intermediate steps between stacking and retrieving a rule instantiation. In contrast, users of the state-centered rule set should not have special problems with the same situations because they do not have to maintain a stack.

The difficulties expected for the users of the operator-centered rule set in the two types of situations just mentioned should show up as more time needed in these situations because the relevant rule instantiation has to be retrieved, or, alternatively, reconstructed with the help of the external, visual rules. Additionally, it should be more likely to make an error when there are many rule instantiations in working memory (when the stack is large).

These hypotheses are depicted in Table 1. The number of rule instantiations on the stack is subsequently referred to as "s." The number of intermediate steps between stacking and retrieving a rule instantiation will be referred to as "i."

Table 1. Overview of the hypotheses

	Computational situations with the following features according to the operator-centered rule set:	
	s and/or *i* high	*s* and *i* low
users of the operator-centered rule set	• make more mistakes • need more time	• make less mistakes • need less time
users of the state-centered rule set	• make less mistakes • need less time	• make less mistakes • need less time

THE EXPERIMENT

Overview

The experiment had two parts. In part one, 12 subjects computed correct ABSYNT-programs, thus simulating the ABSYNT-interpreter in the prediction mode. The subjects worked in dyades (Miyake, 1986). Three dyades were supplied with the operator-centered rule set, and the other three dyades were supplied with the state-centered rule set.

In part two, the subjects worked individually. Partially computed ABSYNT-programs were presented to them. The task was to decide whether these partial computations were correct. Part two served the following purpose: If one has to judge if a partially computed ABSYNT-program is correct, s/he has to mentally trace the computational steps leading to the displayed situation. For each computational step in this sequence, there is a certain *s* and *i* according to the operator-centered rule set. So if there are effects of *s* and *i*, as postulated in the hypotheses (Table 1), then these effects sum up across all the mentally traced computational steps. Therefore the effects should show up very clearly.

Material

For part one, the material consisted of the two visual rule sets (operator-centered vs. state-centered), of a glossary, of 33 ABSYNT-programs to be computed, and of instructions about the subjects' task to compute the programs. The glossary contained an explanation of the basic concepts mentioned in the rules, a legend for the visual rules, and a description of how to handle mouse and keyboard. The ABSYNT-programs were grouped into sets of increasing difficulty. They were partially ordered by the number of visual rules sufficient for their computation. Function calls, branching, abstraction, recursion, and combinations of these concepts were introduced step by step. Each concept was exemplified by up to six programs, making up the total of 33 programs.

For part two, the material consisted of 15 partially computed ABSYNT-programs (depicted on paper). There were three computational situations exemplifying *s* low, *i* low; three more for *s* high, *i* low, another three for *s* low, *i* high, and six wrong computational situations. The latter served for (a) making sure that each program is checked completely, and (b) minimizing practice effects with this task: The first three computational situations presented were wrong. The other ones were randomized.

Experimental Design

For part one of the experiment, the independent variables were:

1. The supplied rule set (operator- vs. state-centered)
2. Four categories of difficulties of computational situations, according to the operator-centered rule set: (a) s low, i low ($s + i < 5$). Examples are the first steps in the computation of a program. (b) s low, i high ($s < 4$; $i > 9$). Examples are the last steps in the computation of a program. (c) s high, i low ($s > 5$; $i = 0$). An example is binding the body parameters. (d) s high, i high ($s > 5$; $i > 20$). An example is passing a value from one frame (incarnation of an ABSYNT-program) back to the calling operator node.

The dependent variables were: (a) number of errors (wrong computational steps), and (b) the time needed for each computational step.

For part two, the independent variables were:

1. Preknowledge (training with the operator- vs. state-centered rule set in part one)
2. Three categories of difficulties of computational situations, according to the operator-centered rule set: (a) s low, i low ($s + i < 4$); (b) s high, i low ($s > 6$; $i < 5$); (c) s low; i high ($s < 5$; $i > 20$).

The dependent variables were: (a) number of wrong judgements about the correctness of the presented computational situations, and (b) the decision time needed for each presented computational situation.

Subjects

Subjects were 12 students of psychology at the University of Oldenburg. All subjects were programming novices. They were paid for participation.

Procedure

Each dyade was introduced to the basic concepts of ABSYNT used in the visual rules, and was familiarized with mouse and keyboard. Additionally, they answered questions about their programming experience.

Then the subjects started with part one: computation of the ABSYNT-programs. They were asked not to consult the experimenter, but to make use of the supplied material in case of problems. The material was presented in a folder. The subjects were free to use it whenever they wanted.

Each dyade computed the same ABSYNT-programs, which were presented in fixed order. Each time a new concept was introduced, the supplied visual rules were augmented with additionally needed visual rules. So the subjects never had more visual rules than actually needed.

During the first computation of each ABSYNT-program, there was no feedback by the experimenter. In case of correct computation, the next ABSYNT-program was presented. In case of a mistake, the program was presented again. This time, feedback was given immediately in case of another mistake. (The subjects were told that the last computational step was wrong.)

The actions of the subjects and their verbalizations were video and audiotaped. Since each dyade computed 33 ABSYNT-programs, there were about 2500 computational steps performed by each dyade. Each dyade worked about 12 hours in 6 sessions.

For part two, each subject was supplied with the 15 computation situations, and with written instructions concerning the task. If a situation was considered as wrong, the next situation had to be taken. Otherwise, the immediately next computational step had to be filled in with pencil. For each situation the time was recorded.

Results

The latency times for part one and part two were evaluated by two Brown-Mood-tests (Lienert, 1973, p. 205 f.) in order to test for the interactions (see Table 1), because there were multiple measurements for each pair of subjects within each cell, and the latency times were not normally distributed and with unequal variances. For part one, those computational steps were discarded which were done in situations not belonging to one of the four categories of difficulties of computational situations. Secondly, only correct computational steps were included for the analysis of the time latencies. Finally, in order to obtain equally sized cells, some randomly selected data of part one were dropped.

The interactions between supplied rule set and categories of difficulties of computational situations, based on latency times, were not significant: $\text{Chi}^2 = 3.86$; d.f. = 3; $p > 0.2$ for part one ($n = 165$ data points per cell), and $\text{Chi}^2 = 0.99$; d.f. = 2; $p > 0.5$ for part two ($n = 18$ data points per cell).

The error frequencies were not analyzed statistically, since there were only few errors. For part one, there were 33 wrong computational steps (= 0.75%) altogether in the computational situations belonging to one of the four categories of difficulties of computational situations. The users of the operator-centered rule set made 1.3%, 0.7%, 2.4%, and 0.9% errors in the computational situations *s* low, *i* low; *s* high, *i* low; *s* low, *i* high; *s* high, *i* high, respectively. The users of the state-centered rule set made 0.9%, 0.1%, 3.0%, and 0.9% errors in the same situations. For part two, there were 17 (or 15%) wrong judgements of computational situations: 10 mistakes (28%) for the combination "operator-centered rule set; *s* and/or *i* high"; 7 mistakes (10%) for all other combinations.

Discussion

The main result is that the different structure of the two rule sets did not show up in the subjects' performance. One could hypothesize that the subjects did not make much use of the instructions but used their own ideas instead, so that instructions did not have a strong influence on the knowledge acquired. But this is not plausible since, among other results, in 98% of the cases where the subjects were faced with a situation covered by a new rule, they looked this rule up in the instructions. Additionally, the total error rate was 0.71% (users of the operator-centered rule set) and 1.19% (users of the state-centered rule set), although the subjects did not have any functional programming knowledge. Without much use of the instructions, there would certainly have been more errors.

So the hypotheses were disconfirmed. With respect to the assumptions stated above, we see the following possibilities for this outcome:

1. There is no one-to-one-mapping between the individual visual rules and the units of the mental representation of the operational knowledge for ABSYNT. That is, the internal rules are not isomorphic to the external rules. That is, instead of constructing an isomorphic representation, the subjects used the instructions as a base for constructing something different.

2. There is no need to load the internal rules (the acquired knowledge) into working memory because this knowledge is proceduralized. Thus there are no problems of limited working memory capacity with this knowledge (Anderson et al., 1981).

We favor the first possibility for the following reasons: The computational steps made by the subjects can be classified into four categories:

(a) Correct computational steps which are done quickly and without any verbalizations. Sequences of such steps seem to be produced by knowledge units which correspond to more than one external, visual rule.

(b) Correct computational steps which are preceded by a verbalization and possibly by a short lookup of the corresponding visual rule.

(c) Correct computational steps which are preceded by problems: In these situations, the subjects initially do not know what to do, and they make heavy use of the instructions. These problematical situations are not only situations which require the application of a new, not yet encountered rule, but also situations requiring the application of an already known rule in a new situation (that is, a situation on the screen which contains some new concept, for instance branching, for which there are one or more new, not yet encountered visual rules).

(d) Wrong computational steps: Wrong applications of new, not yet encountered visual rules, but also the inappropriate use of already known rules in new situations.

Problems or errors (= categories c and d) with already known rules in new situations occur often. This was the case in 52% of the situations where an already known rule had to be applied in a new situation. In contrast, problems or errors occurred in only 7% of the computational steps altogether. This "Einstellung" effect is clearly not in accordance with the assumption that the operational knowledge acquired by the subjects consists of internal rules that are just isomorphs to the external, visual rules. Instead, our interpretation is that independently of the supplied visual rule set, the subjects constructed larger knowledge units which were tuned to the particular type of ABSYNT-programs currently encountered. When the type of ABSYNT-programs was changed by introducing a new concept (and correspondingly, one or more new visual rules), these knowledge units were not applicable any more, leading to problems or even errors with already known rules.

Thus we view the acquisition of the operational knowledge as a two-stage-process:

1. Acquisition of new knowledge in response to difficulties with the help of the instructional material, that is, impasse or failure-driven learning.

2. Improvement of existing knowledge by creation of compound rules, that is, success-driven learning.

In order to make these ideas concrete, a specification of the instruction-based acquisition of the operational knowledge for ABSYNT is currently developed. In the rest of this paper, we will describe this specification and some of its implications.

SPECIFICATION OF THE ACQUISITION OF THE OPERATIONAL KNOWLEDGE

Purpose of the Specification

The specification of the processes of the instruction-based acquisition of the operational knowledge has the following objectives:

1. To achieve an integrated description of the data gathered. By this, we investigate the following questions: How can the knowledge acquired with the help of the instructions be described? How can the acquisition of this knowledge be described? Does this knowledge differ from the instructions? For instance, do the subjects reorganize or reinterpret the instructions? When do the subjects not follow the instructions? If so, what might be the reasons for this?
2. To generate predictions about performance aspects in certain computational situations.
3. To generate further design criteria concerning the instructional material; for instance, whether it would be feasible and sensible to adapt the instructional material to the actual knowledge state of the learner.

Current State of the Specification

The operational knowledge for ABSYNT acquired by the subjects is represented as a rule net. The rule net is continuously changed by acquisition of new knowledge due to problem solving, and by improvement of existing knowledge due to practice.

Acquisition of new knowledge is triggered by difficulties (Laird, Rosenbloom, & Newell, 1986), or impasses (Brown & van Lehn, 1980; van Lehn, 1987, 1988). In response to a difficulty, there are problem-solving steps with the help of the instructional material, the external, visual rules. If successful, the problem-solving steps lead to the generation of new information. This information is then used to augment the rule net. Thus the rule net is changed in order to cope with new situations.

Improvement of existing knowledge is triggered by practice with several instances of the same type of ABSYNT program. During practice, rules of the rule net are merged into compound rules. Thus the rule net is changed in order to handle a given type of situation more efficiently.

The current state of the specification was developed in the light of these guidelines and by protocol analysis of a portion of the data (see below).

The Rule Net

We distinguish internal rules (the rules of the rule net) and external rules (the rules of the instructional material). In the rule net (Figure 6), a rule consists of a directed labeled link with two nodes (connected by an and-node) below it. The link is the condition (C) of the rule. The first node is the action (A). The second node is a recursive call of the top node of the rule net. Figure 6a depicts a rule net abstractly. If the rule net specifies the knowledge of a user of the operator-centered rule set, then the rules of the rule net are (a) the hypothetical internal representations of the *subrules* of the external, visual rules of the operator-centered rule set, or (b) compound rules built from such subrules. If the rule net specifies the knowledge of a user of the state-centered rule set, then the rules of the rule net are (a) the hypothetical internal representations of the external, visual *rules* of the state-centered rule set, or (b) compound rules built from such rules. Thus the structure of the rule net is the same for the knowledge acquired by the operator- and state-centered rule set.

For example, C1 in Figure 6a is the internal representation of the situation description of the visual rule of Figure 3. A1 is the internal representation of the

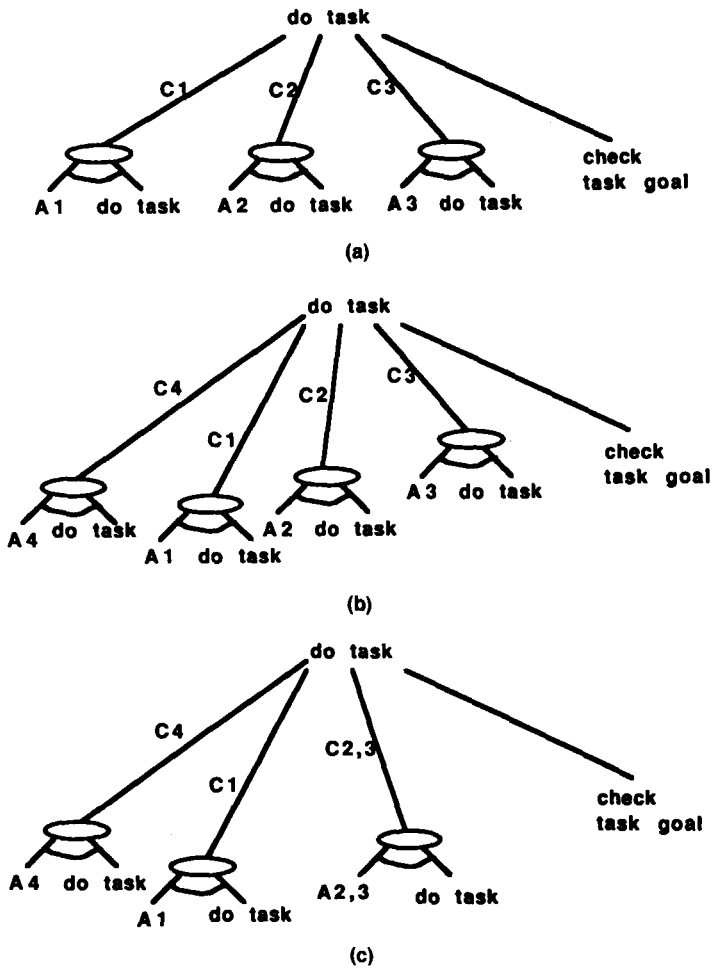

Figure 6. (a) A rule net, (b) knowledge acquisition by failure-driven learning, (c) knowledge improvement by success-driven learning.

action description of this visual rule. The "task goal" is the goal given to the subjects: The output stripe of the root node of the start tree must contain a value (see Figure 2d).

Acquisition of New Knowledge

New knowledge is acquired in response to a difficulty. Currently there are two main types of difficulties:

D1: No condition leaving the top node is satisfied by the situation currently visible on the screen, and the task goal is not yet fulfilled.

D2: There is feedback that the last computational step was wrong.

In response to a difficulty, problem solving is starting. There are the following main categories of problem solving steps:

P1: "Working backward": In case of D1 there might be a rule in the rule net which action description does not make use of information obtained by the binding of the condition of this rule. For example, it is always possible to fill an empty input stripe with computation goals ("?") (the action description of the visual rule in Figure 3), even if the conditions in which this is allowed (the situation description of the visual rule in Figure 3) are not true. So if there is such a rule in the rule net, it serves as the starting point for "working backward" with the help of the visual rules: There will be an attempt to fulfill the unfulfilled condition. For example, the output stripe of an ABSYNT-node might be empty. Thus the unfulfilled condition for filling its input stripe with computation goals is that the output stripe must contain a computation goal (see Figure 3). So a goal is set to put a computation goal into the output stripe of this node. Then a visual rule is looked for which achieves this.

P2: "Trial and error": Also in case of D1, there might be just an attempt to find an applicable visual rule. Thus the visual rules are scanned until an applicable rule is found. This is also done if there is no unfulfilled but currently "applicable" rule in the rule net.

P3: Identification of an unfulfilled condition: In case of D2, there is an attempt to find the difference between the computational situation currently visible on the screen and the visual rule whose action corresponds to the computational step just applied. That is, it is tried to find the reason for the mistake. As the result, an instance of D1 arises.

So after P1, P2, or P3, there is new information: With the help of the visual rules, an applicable rule is found, or an unfulfilled condition element of a rule already part of the rule net is identified. This new information is then used to augment the rule net with the new rule (Figure 6b), or to augment the condition of an existing rule with the identified condition element.

Improvement of Existing Knowledge

If no difficulties arise, the rule net is improved by building compound rules. The result is depicted abstractly in Figure 6c: Two rules leaving the top node ("do task") are merged into one in a way that tries to use ideas of composition (Anderson, 1983, 1986; Anderson et al., 1981; Neves & Anderson, 1981). The compound rule is formed by inspection of the trace of rule applications. For example, C2 → A2 and C3 → A3 of Figure 6b might be composed into C2,3 → A2,3 of Figure 6c. In order to illustrate how this might proceed, it is assumed that (a) C2 → A2 of Figure 6b is the internal representation of the visual rule in Figure 4; (b) C3 → A3 of Figure 6b is the internal representation of the visual rule in Figure 5; (c) C2,3 → A2,3 of Figure 6c is the internal representation of the rule in Figure 7. (This is not a visual rule of the instructional material; see below.)

The trace of the rule applications consists of three applications of the visual rule in Figure 4 to the situation depicted in Figure 2b (this leads to the situation depicted in Figure 2c) and one application of the rule in Figure 5 to the situation depicted in Figure 2c (this yields the situation depicted in Figure 2d).

1. The three input fields in the three instantiations of the rule of Figure 4 are generalized to "all input fields" (cf. Benjamin, 1987). These "all input fields" are identical to the "input stripe" in the condition of the instantiation of the rule in Figure 5. Therefore, in the condition and action part of the new compound rule (Fig-

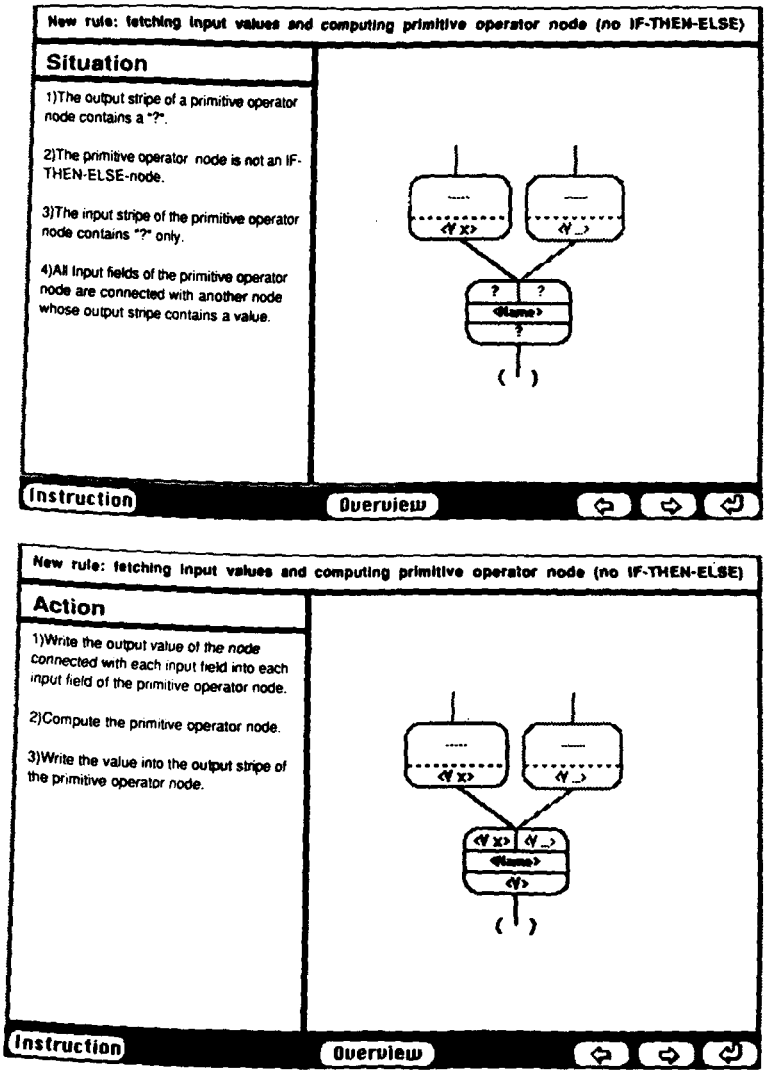

Figure 7. Example of a "visual compound rule" based on the visual rules in Figures 4 and 5.

ure 7) "all input fields" are specified by all the conditions and the actions for input fields mentioned in the rule of Figure 4. But the condition for these input fields required by the rule of Figure 5 (namely, that they contain values) is not included in the condition of the new rule, because this is the result of the three applications of the rule in Figure 4, which are followed here by the application of the rule in Figure 5.

2. The operator node in the three bindings of the rule in Figure 4 is the same as the primitive operator node (no if node) in the binding of the rule in Figure 5 (namely, the "<" node in Figure 2). Therefore, only "primitive operator node (no if node)" is included in the rule in Figure 7.

3. The action of the rule in Figure 5 is included in the action of the new rule.

The resulting new rule is an improvement because it reduces matching. For example, the sequence of the three applications of the rule in Figure 4 and the one application of the rule in Figure 5 test the "<" node of Figure 2 four times. Applying the new rule instead would test this node only once.

But if now a computational situation is encountered where the original rules are needed in combination with new rules, then a difficulty will arise again, since the original rules are not available as single rules any more; they have become part of a compound rule. (For example, the original rules C2 → A2 and C3 → A3 are not contained in the rule net of Figure 6c). So there will be problem solving again, and the original rules are added again to the rule net.

EMPIRICAL BASE

Some Questions of Method

The data of one dyade are used for the derivation of the specification. The other data are used for validation (Card, Moran, & Newell, 1983, Ch. 5).

The relevant data are the computational steps performed by the subjects (including bugs and the sequencing of the computational steps), and verbalizations. Only the verbalizations of one (the more active) subject of the dyade are analyzed. Therefore the specification hypothesizes knowledge states and acquisition for one subject, not for a dyade. The advantage of letting subjects work in dyades is that the verbalizations are richer and more natural (Miyake, 1986). The disadvantage is that we had to find ways of how to deal with the verbalizations of the other subject. A convention is: If B (the subject whose verbalizations are not regarded) objects to or proposes some rule or computational step, and A (the subject whose verbalizations are of interest) accepts this, then the objection/proposal is treated as if made by A. Otherwise, B's objection/proposal and A's reply to it are removed from the protocol.

Protocol Analysis

The aim of the protocol analysis is to operationalize the activities prior to each computational step: difficulties, problem solving steps, applications of rules of the rule net, and applications of the action steps of compound rules.

About four hours of computing ABSYNT-programs by one pair of subjects were protocol analyzed. The segments of the protocol were assigned to coding categories. Then sequences of coding categories were aggregated into types of difficulties and into types of problem solving steps. Table 2 is a simplified list of some of the operationalizations.

We try to base the assignment of coding categories on key words (Miyake, 1986) as far as possible. For instance, key words for "state sequence of computational steps" are "first . . ., then . . .". Key words for "notice that a rule is not applicable" are "but," "wrong," etc., "find internal rule" is assigned if, for instance, a rule number is mentioned. The corresponding visual rule might subsequently be checked, but critical to the assignment of this coding category is that the idea is expressed first. In contrast, "find external rule" is assigned if the visual rules are consulted first. "Find internal or external rule" is a superordinate category for "find internal rule" and "find external rule."

Table 2. Some examples for the operationalization of activities prior to computational steps in terms of coding categories of the protocol

Activities prior to comp. step	Coding categories of the protocol
Difficulty D1	"notice that a rule is not applicable"
Difficulty D2	negative feedback by the experimenter
Working backward (P1)	"state a goal" + "find internal or external rule" + "state sequence of computational steps"
Binding rule of rule net	"find internal rule"
Binding next action step of compound rule	no verbalization

SOME IMPLICATIONS OF THE SPECIFICATION

Some Predictions

One of the next steps will be to evaluate the specification. For example, there are the following predictions:

1. Prediction of situations causing difficulties because there is no rule in the rule net that handles the current situation.
2. Prediction of computational steps preceded by "find internal rule" verbalizations because they are generated by a rule of the rule net.
3. Prediction of computational steps performed silently because they are generated by the action steps of compound rules.
4. Prediction of specific problem solving steps in response to specific difficulties.

Implications for Further Improvement of the Instructional Material

There are also suggestions about how the instructions might be improved. If the subject is not sure about applying a compound rule in a new situation, then it would seem appropriate if the instructions are adapted to the current knowledge acquired by the learner. This would mean to augment the instructions by adding visual compound rules. So the instructional material could be tailored step by step to the knowledge acquisition process of the learner. Figure 7 provides an example of a visual compound rule.

Open Questions; Future Research

The next steps will be:

1. To evaluate the specification for the derivation data. This involves comparing predicted and observed difficulties, problem solving steps, applications of rules of the rule net, and of the action steps of compound rules.
2. To evaluate the specification for the validation data in the same way.
3. To continue the implementation of the specification. An implementation of the specification is necessary for (a) a full and detailed evaluation, and (b) the generation of not yet encountered hypotheses.

One concern for future is the extension of the specification to other task domains (i.e., the acquisition of programming knowledge for ABSYNT). This is another topic within our project. It will necessitate an augmentation towards the inclusion of elements of inductive learning.

REFERENCES

Anderson, J.R. (1983). *The Architecture of cognition.* Cambridge, MA: Harvard University Press.

Anderson, J.R. (1986). Knowledge compilation: The general learning mechanism. In R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.), *Machine learning, Vol. II.* pp. 289–310. Los Altos: Kaufman.

Anderson, J.R., Greeno, J.G., Kline, P.J., & Neves, D.M. (1981). Acquisition of problem-solving skill. In J.R. Anderson (Ed), *Cognitive skills and their acquisition.* pp. 191–230, Hillsdale, NJ: Erlbaum.

Bauer, F.L., & Goos, G. (1982). *Informatik.* 1. Teil. Berlin, Springer, (3. Auflage).

Benjamin, D.P. (1987). Learning strategies by reasoning about rules. *10th Int. Joint Conf. on Artificial Intelligence*, Mailand, 1987.

Brown, J.S., & van Lehn, K. (1980). Repair theory: A generative theory of bugs in procedural skills. *Cognitive Science*, **4**, 379–426.

Card, S.K., Moran, T.P., & Newell, A. (1983). *The psychology of human-computer interaction.* Hillsdale, N.J.: Erlbaum.

Davis, E.R. (1982). Runnable specifications as a design tool. In K.L. Clark, & S.A. Tärnlund (Eds.), *Logic programming.* pp. 141–149. New York: Academic Press.

Egan, D.E., & Greeno, J.G. (1974). Theory of rule induction: Knowledge acquired in concept learning, serial pattern learning, and problem solving. In L.W. Gregg (Ed.), *Knowledge and Cognition.* pp. 43–103. Potomac: Erlbaum.

Janke, G., & Kohnert, K. (1989). Interface design of a visual programming language: Evaluating runnable specifications. In F. Klix, N.A. Streitz, Y. Waern & N. Wandke (Eds.), pp. 567–581. *MACINTER-II Man-computer-interaction research, Proceedings of the Second Network Seminar of MACINTER.* Berlin/GDR, March 21–25, 1988, Amsterdam: Elsevier.

Kintsch, W., & Greeno, J.G. (1985). Understanding and solving word arithmetic problems. *Psych. Review*, **92**, 109–129.

Laird, J., Rosenbloom, P.S., & Newell, A. (Eds.) (1986). *Universal subgoaling and chunking.* Boston, MA: Kluwer Academic Publ, 135–199.

Larkin, J.H., & Simon, H.A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, **11**, 65–99.

Lienert, G.A. (1973). Verteilungsfreie Methoden in der Biostatistik, Band 1. Meisenheim: Hain.

Miyake, N. (1986). Constructive interaction an the iterative process of understanding. *Cognitive Science*, **10**, 151–177.

Möbus, C. (1985). Die Entwicklung zum Programmierexperten durch das Problemlösen mit Automaten. In Mandl, H.; Fischer, P.M. (Hg): *Lernen im Dialog mit dem Computer.* 140–154. München: Urban & Schwarzenberg.

Möbus, C., & Schröder, O., Knowledge specification and instruction for a visual computer language. In F. Klix, H. Wandke, N.A. Streitz, & Y. Waern (Eds.): *Man-computer interaction research, MACINTER II,* pp. 535–565.

Möbus, C., & Schröder, O. (in press). Representing semantic knowledge with 2-dimensional rules in the domain of functional programming. In M. Tauber & P. Gorny (Eds.), *Visualization in human-computer interaction,* Heidelberg, Springer Computer Science Lecture Series.

Möbus, C., & Thole, H.J. (1989). Tutors, instructions, and helps. In Th. Christaller (Ed.), *Künstliche Intelligenz,* pp. 336–385. KIFS87, Heidelberg, Springer Computer Science Lecture Series.

Neves, D.M., & Anderson, J.R. (1981). Knowledge compilation: Mechanisms for the automatization of cognitive skills. In J.R. Anderson (Ed.), *Cognitive skills and their acquisition.* 57–84. Hillsdale, NJ: Erlbaum.

Newell, A., & Simon, H.A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice Hall.

Pomerantz, J.R. (1985). Perceptual organization in information processing. In A.M. Aitkenhead, J.M. Slack, (Eds.), *Issues in cognitive modeling.* 127–158. Hillsdale, NJ: Erlbaum.

Sleeman, D.H., & Hendley, R.J. (1982). ACE: A system which analyses complex explanations. In D. Sleeman & J.S. Brown, *Intelligent tutoring systems* (pp. 99–118). New York: Academic Press.

van Dijk, T.A., & Kintsch, W. (1983). *Strategies of discourse processing.* New York: Academic Press.

van Lehn, K., (1987). Learning one subprocedure per lesson. *Artificial Intelligence.* **31**, 1–40.

van Lehn, K. (1988). Towards a theory of impasse-driven learning. In Mandl H., & Lesgold, A.: *Learning issues for intelligent tutoring systems.* 19–41. Springer, New York.