A Model of the Acquisition of Rule Knowledge with Visual Helps: The Operational Knowledge for a Functional, Visual Programming Language*

Olaf Schröder

University of Oldenburg
Department of Computational Science
D-2900 Oldenburg
F.R. Germany

Abstract

A simulation model of the acquisition of rule knowledge with visual helps is described in the domain of the operational knowledge for ABSYNT, a functional, visual programming language. The knowledge acquisition process is viewed as an iterative two-stage process:

a) acquiring new knowledge by making use of the supplied visual help material in response to difficulties: that is, in new situations for which the current knowledge is not sufficient;
b) improving existing knowledge by dealing with familiar types of situations.

The simulation model was developed by protocol analysis of one single subject. The model describes 60% of a continuous portion of the protocol. Some more coarse data from other subjects are analyzed in the light of predictions of the model. The model further suggests how to continuously change the help material in order to adapt it to the actual knowledge state of the learner during the knowledge acquisition process.

1. Introduction

This paper describes a simulation model of a help-guided knowledge acquisition process in the domain of the operational knowledge about the interpreter of ABSYNT. This is the knowledge about how the ABSYNT-interpreter works. ABSYNT (*Ab*stract *Syn*tax *T*rees) is a purely functional, visual programming language developed in our project [12, 18, 19, 21, 22, 23]. ABSYNT is aimed at supporting the acquisition of functional programming concepts up to recursion.

The goal of our project is to build an adaptive problem solving monitor for programming in ABSYNT. Currently the problem solving monitor analyzes the students´ blueprints, gives helps and error feedback, [11], [20], and [24], this volume. Our future work is aimed at tailoring analysis, helps, error-feedback and reason-giving closely to the actual knowledge state of the student. For this we persue modeling help-guided

* This work is sponsored by the Deutsche Forschungsgemeinschaft (DFG) in the SPP Psychology of Knowledge, Grant-no. MO 292/3-3.

knowledge acquisition processes of real learners. This is important within intelligent help or tutoring systems [14, 28] for constraining hypotheses about the actual knowledge state of a learner [9], and for improving the timing and content of helps [24] in order to enhance the learners´ construction of a mental model [13] of "how-it-works"-knowledge and "how-to-do-it"-knowledge [15].

As the domain for modeling help-guided knowledge acquisition, we chose the operational knowledge at first because a) it is less complex than the programming knowledge, and b) we developed visual help material for it (see below). But we plan to transfer aspects of the model to the domain of programming in ABSYNT.

2. The Domain

The model to be described deals with the hypothetical operational knowledge of a subject and the acquisition of this knowledge while the subject simulates the ABSYNT-interpreter at the terminal, having access to abstract, visual helps. This section serves to explain this activity.

An ABSYNT-program consists of a head tree and a body tree. Also, there is a start tree from which programs can be called. 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 (see the lower and right margin in figure 1) and connecting them. ABSYNT-programs are illustrated in [24], this volume.

Another ABSYNT-environment consists of a visual trace which was implemented according to the runnable specification [8] of the interpreter [23]. The trace makes every computational step of the interpreter visible. In the trace, computation goals (symbolized by question marks) and obtained values are represented within each node. Figure 1 shows an example.
Figures 2a to d show different states of the trace for a very simple ABSYNT-start tree without function calls.

Besides editor and trace, there is also a prediction environment where the learner may predict the single steps of the ABSYNT-interpreter by simulating it with the help of mouse and keybord.
In a fourth environment, the student can test hypotheses about the correctess of her/his program [20, 24].

Thus ABSYNT provides an iconic environment in the sense of [10]. An analysis of ABSYNT in terms of properties of visual languages is provided in [23]. We will briefly describe some relationships between the ABSYNT-environments and some other iconic programming environments:
The ABSYNT-environments differ from PICT/D [10], for example, in that ABSYNT is purely functional, and there are individual connected icons (nodes) for operators, parameters, and constants. In PICT/D, the icons contain operators together with their variable(s) and constants; the system is flowchart based.
BridgeTalk [4] differs from ABSYNT in that its icons are interrelated plan pieces, whereas ABSYNT-programs are actual code. Planning rules are used for analyzing and synthesizing ABSYNT-programs [24]. One of our efforts will be to convert these rules into visual planning aids.

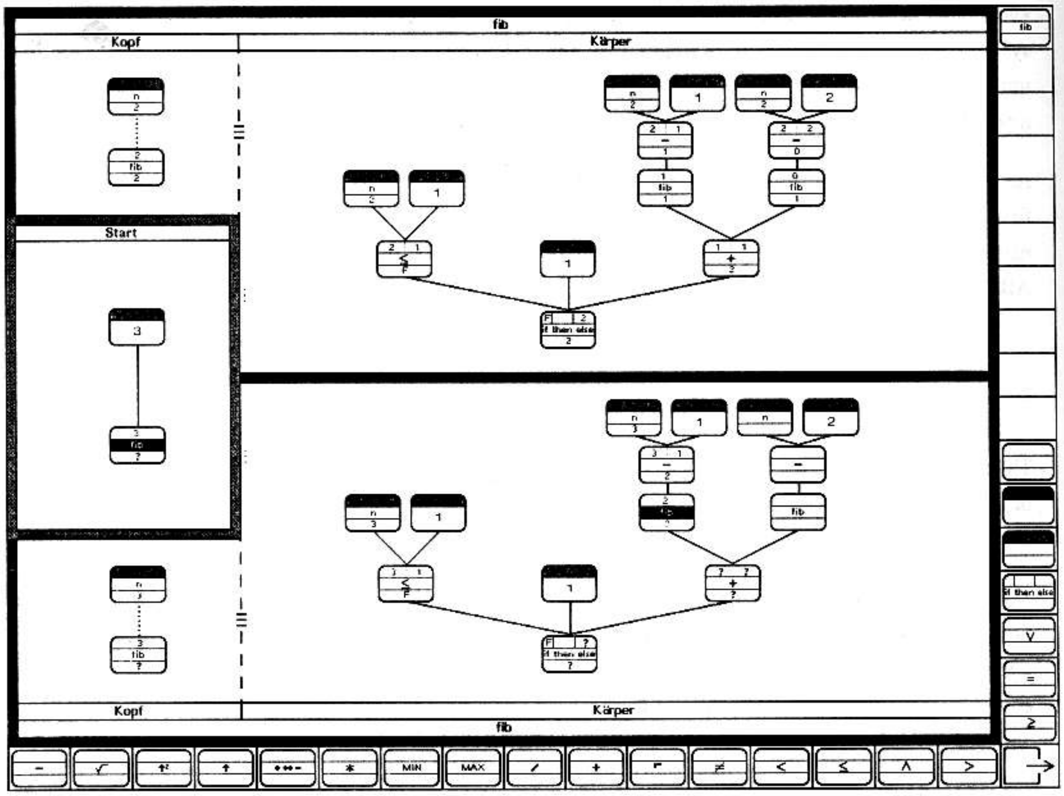


Figure 1: A snapshot of the visual trace of an ABSYNT-program

GIL (Graphical Instruction in LISP; [26]) differs from ABSYNT in that in GIL, program code and intermediate computational results are both represented in a program graph. ABSYNT-code does not contain computational results, computation is visualized (figures 1 and 2) in separate environments. The flexibility in GIL to work top-down, bottom-up or middle-out is also a property of ABSYNT.

In addition to the ABSYNT-environments, the runnable specifications of the ABSYNT-interpreter were translated into sets of visual rules (according to design principles explained in [23]). Each visual rule describes some set of computational steps of the ABSYNT-interpreter, including branching, function calls, abstraction, and recursion. The visual rules serve as the help material the learner may use while simulating the ABSYNT-interpreter in the ABSYNT-prediction environment.
For example, figures 3 to 5* show the visual rules which describe the computational steps depicted in figures 2a to d. One application of the visual rule in figure 3 to the situation depicted in figure 2a leads to figure 2b. Then, three applications of the visual rule in figure 4 lead to figure 2c. Finally, applying the visual rule in figure 5 leads to the situation shown in figure 2d.

* The visual rules were implemented in HYPERCARD by Klaus-Dieter Frank, but they are not so depicted here for space reasons. [22] contains HYPERCARD presentations of all visual rules.

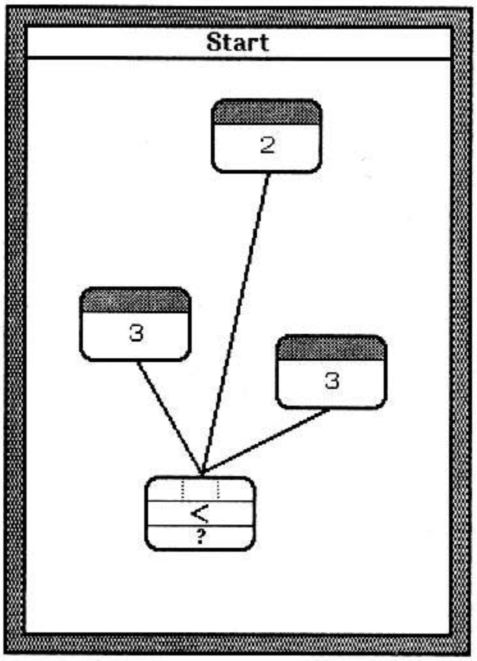


Figure 2a: before computation

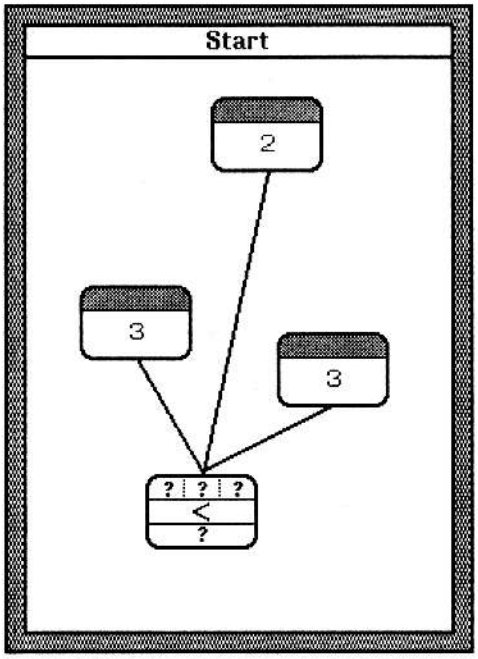


Figure 2b: after application of the visual rule in figure 3

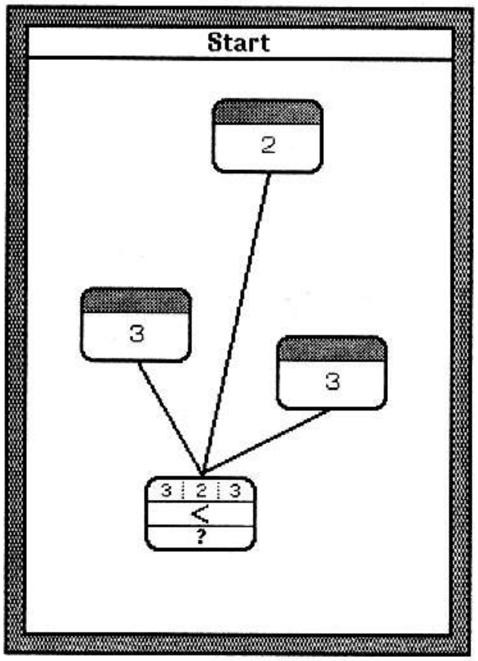


Figure 2c: after three applications of the visual rule in figure 4

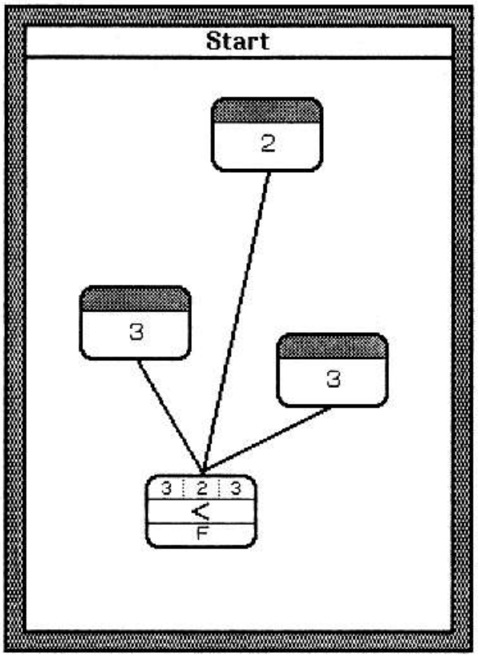


Figure 2d: after application of the visual rule in figure 5

Figure 2: ABSYNT start tree without function calls and with only one primitive operator node

Rule 1: Computing primitive operator node (no IF-THEN-ELSE-nod

Situation:

1. The output stripe of a primitive operator node contains a "?".
2. The primitive operator node is not an IF-THEN-ELSE-node.
3. The input stripe of the primitive operator node is empty.

Action:

Write a "?" into every input field of the primitive operator node.

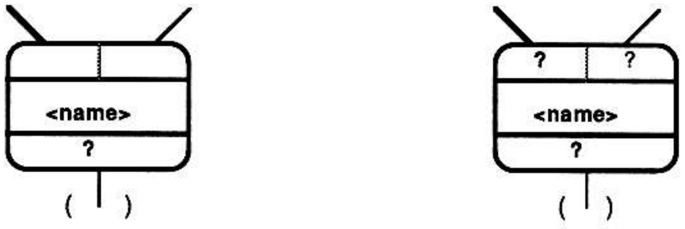


Figure 3: A visual rule of the help material

Rule 4: Fetching input value for operator node

Situation:

1. The output stripe of an operator node contains a "?".
2. Any input field of the operator node contains a "?"
3. The input field of the operator node is connected to another node which output stripe contains a value.

Action:

Write the output value of the node connected to the input field into that input field.

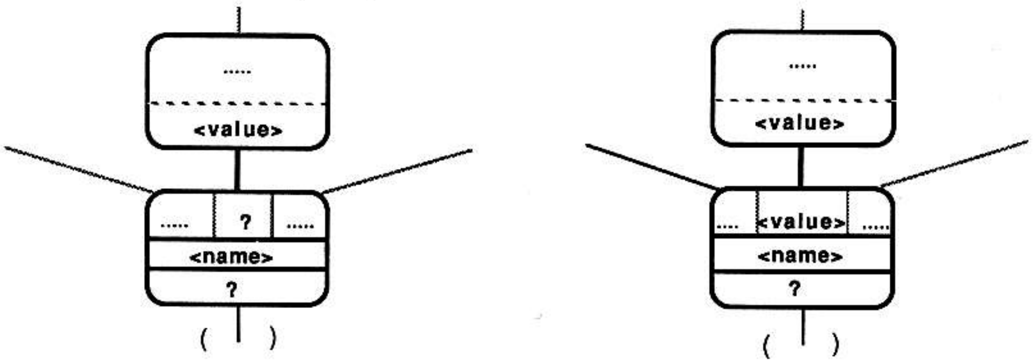


Figure 4: A visual rule of the help material

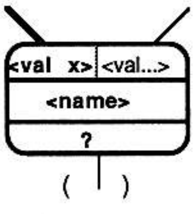
3. An Earlier Study as the Starting Point for the Model

In an earlier study [27], 12 programming novices simulated the ABSYNT-interpreter, provided with the visual rules as helps on paper. The subjects worked in dyades. Each dyade computed 33 ABSYNT-

Rule 2: Computing primitive operator node (no IF-THEN-ELSE-node)

Situation:

1. The output stripe of a primitive operator node contains a "?".
2. The primitive operator node is not an IF-THEN-ELSE-node.
3. The input stripe of the primitive operatornode contains values only.



Action:

1. Compute the primitive operator node.
2. Write the value into the output stripe of the primitive operator node.

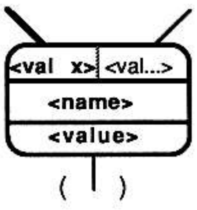


Figure 5: A visual rule of the help material

programs in this way at the terminal. The ABSYNT-programs were ordered in 7 sequences of equally difficult programs, that is, programs which computation is described by the same set of visual rules. From one sequence to the next, difficulty increased: The ABSYNT-programs contained some new concept (like branching, abstraction and, finally, recursion), and the supplied set of visual rules was augmented accordingly. The subjects were free to make or not make use of the helps (visual rules). In case of a mistake, feedback was provided by the experimenter that the last computational step performed was wrong. There were no time limits. The subjects were video- and audiotaped.

The subjects acquired the operational knowledge for ABSYNT and got skilled with this task. Typically, within a sequence of equally difficult ABSYNT-programs, the subjects start with much problem solving and make much use of the helps. Then the use of the help material decreases, and finally most of the computational steps are done quickly and without verbalizing, until a more difficult ABSYNT-program is presented (the first program of the next sequence. A computational step is every change of the content of an ABSYNT-node, like filling in a computation goal or a value, and creating and deleting an incarnation of an ABSYNT-program.) Within this new level of difficulty, the same pattern then starts again. There were only about 1 % mistakes.

So first there are situations causing problems. They lead to gathering of new information with the helps. Second, afterwards there are non-problematical situations, leading to quick performance where the help material is not needed any more.

Starting from this observation, we view the acquisition of the operational knowledge as an iterative two-stage-process, which is the framework for the model:

1. *Acquisition of new knowledge*. This is triggered by difficulties [16], or impasses, [5, 29, 30]. In order to overcome a difficulty, problem solving steps are performed by applying weak heuristics [16] with the use of the helps. The problem solving steps may lead to new information. Thus the acquisition of new knowledge is viewed als *impasse-driven learning*.

2. *Improvement of existing knowledge.* Due to practise, the existing knowledge is reorganized so it can be used more efficiently. Thus the improvement of existing knowledge is viewed als *success-driven learning*.

4. The Model

In developing the model, we performed the following steps:
1. A continuous part of the protocol of one single subject of a dyade was analyzed within the framework of impasse-driven and success-driven learning.
2. A set of hypotheses about the content and structure of the hypothetical domain-specific (operational) knowledge of the subject at each point in the knowledge acquisition process was stated.
3. Based on the categories of the protocol, three sets of hypotheses were stated which were aimed at wholly covering the analyzed part of the protocol, leading to a verbal model:
   - Hypotheses about the use of domain-specific knowledge: When are the helps needed / not needed? When do difficulties arise?
   - Hypotheses about problem solving steps (= application of weak heuristics) in response to difficulties, and about their results, including the acquisition of new domain-specific knowledge.
   - Hypotheses about when what domain-specific knowledge is improved.
4. Most aspects of the verbal model were implemented, obtaining the simulation model.

Thus the model incorporates the regularities extracted from the protocol within the framework of impasse-driven and success-driven learning. The knowledge acquisition process is viewed as a continuous change in structure and content of domain-specific knowledge, while the domain independent knowledge (i.e.: making use of and improving domain-specific knowledge; weak heuristics for dealing with domain-specific problems) remains unchanged. We will now take up the four steps listed above.

4.1 Protocol Analysis

The analyzed protocol covers about four hours of continuous working of one single subject of a dyade. Several conventions were introduced in order to handle her interactions with the other subject of the dyade. The protocol analysis resulted in a set of protocol categories for coding. The main protocol categories are:

- seven different kinds of difficulties which the subject encounters. For example, three difficulties are:
  D1: Applicability of a rule is unclear. (The subject is uncertain whether the condition of some rule is satisfied or not.)
  D2: Rule condition is not satisfied, but satisfiable by some other rule. That is, the subject is aware of the fact that the condition of some rule is violated, but it is possible to satisfy this condition by applying some other rule first.
  D3: Same as D2, except that the rule condition is not satisfiable by any other rule. (For example, the rule in figure 3 is not applicable to an if-then-else-node)
  The other four difficulties are: no rule for a goal; tie; action unclear; and negative feedback.
- setting a goal to apply a rule with or without making use of the helps.

- rule applications with or without making use of the helps. Any continuous sequence of computational steps that can be described by one visual rule is called a rule application.

In actual coding, each protocol category carries additional information about a) the part of the screen it refers to, b) the change of this screen part as considered by the subject, c) the rule describing this change, and d) whether the subject makes use of the helps by looking at visual rules. For example, if the subject would say in the situation depicted in figure 2a: "Maybe we can put ´?´ into the input stripe of the ´<´-node" without using helps, then this would be coded as a difficulty D1, refering to a) the ´<´-node with an empty input stripe, which is b) considered to be filled with ´?´. This change is c) described by rule 1 (figure 3), but d) the actual visual rule is not looked at.

4.2 The Hypothetical Domain-specific Knowledge

The hypothetical operational knowledge acquired by the subject at any given point is represented as a set of internal rules: an internal compound rule and possibly additional internal simple rules.
An *internal simple rule* represents acquired, but not yet improved knowledge and is an abstract representation of the corresponding visual rule. The acquisition of new knowledge results in the creation of a new internal simple rule.
An *internal compound rule* represents improved knowledge. Knowledge improvement results in chunking an internal simple rule into the compound rule. There are two different forms of an internal compound rule:
As long as the individual action steps of the compound rule can be applied in the same order for all tasks of the current sequence of equally difficult ABSYNT-programs, the internal compound rule is represented as the *composite* [1, 2, 17, 25] of two or more internal simple rules. Figure 6 shows the visual representation of the composite built from abstract representations of the visual rules in figures 3 to 5. The application of this composite to the situation in figure 2a leads to the situation in figure 2d.
But if the action steps are to be applied in variable order (depending on the actual task at hand) for the tasks of the current sequence of equally difficult ABSYNT-programs, then the action part of the internal compound rule consists of subrules (a simple version of the proposal of [7]). This "*rule net*" allows for iterative action sequences [31], it describes the computation of ABSYNT-trees of varying size by a single compound rule. Figure 7 shows the visual representation of the rule net built from abstract representations of the visual rules in figures 3 to 5, plus another visual rule (visual rule 3) not depicted which passes computation goals between connected nodes within an ABSYNT-tree.
Internal simple rules may be merged as new subrules into the rule net. We view this as proceduralization [1, 2, 25], since we represent internal simple rules as facts, but subrules as rules.

In order to tie these hypothetical knowledge forms to the protocol, the following assignments were made:

- There is no internal representation of a rule never applied yet.
- If a rule application is performed which is described by a new visual rule never encountered before, then a corresponding internal simple rule is created.
- If a rule application is performed without a lookup in the helps for the first time, then the respective internal simple rule has been integrated into the internal compound rule (composite or rule net). Thus as long as applying a rule always requires the helps, the rule is viewed as an internal simple rule.

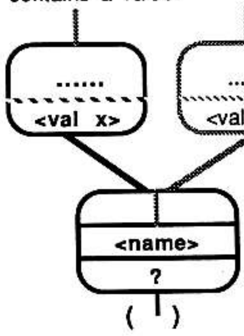
Composite of visual rules 1, 4+, 2

Situation:

1. The output stripe of a primitive operator node contains a "?".
2. The primitive operator node is not an IF-THEN-ELSE-node.
3. The input stripe of the primitive operator node is empty.
4. All input fields of the primitive operator node are connected to another node which output stripe contains a value.

Action:

1. Write a "?" into every input field of the primitive operator node.
2. Write the output values of the connected nodes into each connected input field of the primitive operator node.
3. Compute the primitive operator node.
4. Write the value into the output stripe of the primitive operator node.



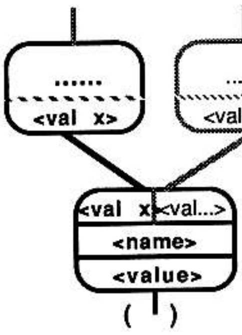


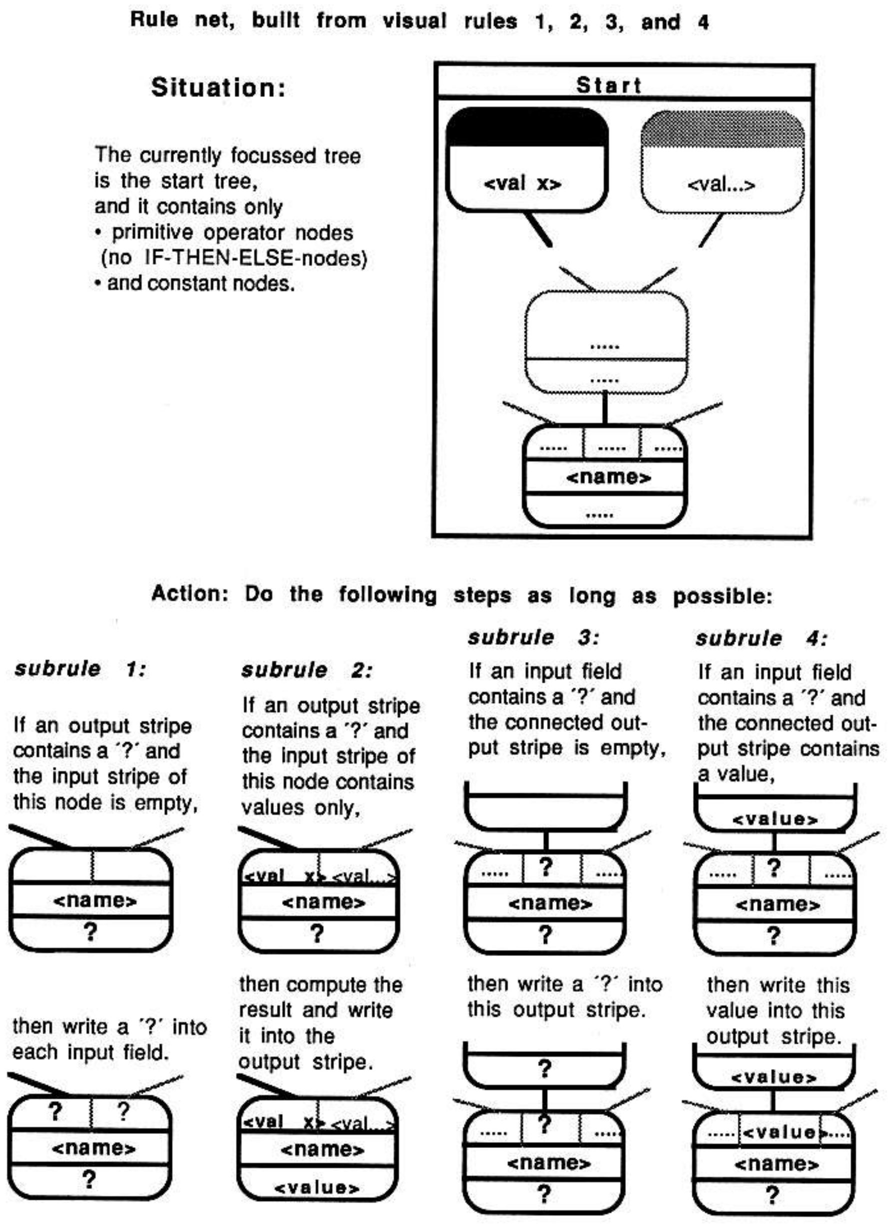
Figure 6: Visual representation of the composite built from abstract representations of the visual rules in figures 3 to 5: Rules 1, 4+, and 2. ("+" is an abbreviation for one or more rule firings.)

4.3 The Verbal Model

As mentioned, three sets of hypotheses about the knowledge acquisition process were extracted. Table 1 shows an example from each set, stated as if-then-rules. For each hypothesis shown, the number of supporting and disconfirming cases ([6], Ch. 5) in the protocol is given. Altogether, there are 20 such hypotheses, supported by 397 of 474 cases.

The first set of hypotheses (6 "*application hypotheses*") is concerned with the activity just after a rule application: Depending on the actual hypothetical operational knowledge, does another rule application occur with or without making use of the helps? Does a difficulty arise?

The second set of hypotheses (8 "*problem solving hypotheses*") is concerned with problem solving steps and their results in response to different difficulties. We hypothesized problem solving steps as "connectors" between certain protocol categories. For example, if a difficulty D2 is followed by another difficulty or by setting a goal which is concerned with satisfying the violated condition, then the problem solving step "working backward" was assigned. A problem solving step may remove a difficulty, resulting in a goal to apply a certain rule to a certain part of the screen. Additionally, new knowledge may be acquired (see the "knowledge acquisition hypothesis" below). But a problem solving step may also lead to yet another difficulty.



Rule net, built from visual rules 1, 2, 3, and 4

Situation:

The currently focussed tree
is the start tree,
and it contains only
• primitive operator nodes
(no IF-THEN-ELSE-nodes)
• and constant nodes.

Action: Do the following steps as long as possible:

subrule 1:

If an output stripe contains a '?' and the input stripe of this node is empty,

then write a '?' into each input field.

subrule 2:

If an output stripe contains a '?' and the input stripe of this node contains values only,

then compute the result and write it into the output stripe.

subrule 3:

If an input field contains a '?' and the connected output stripe is empty,

then write a '?' into this output stripe.

subrule 4:

If an input field contains a '?' and the connected output stripe contains a value,

then write this value into this output stripe.

Figure 7: Visual representation of the rule net built from abstract representations of the visual rules in figures 3 to 5: Rules 1, 4, 2, plus the additional rule 3 (not depicted)

Table 1: An Application, Problem solving and Knowledge Improvement Hypothesis

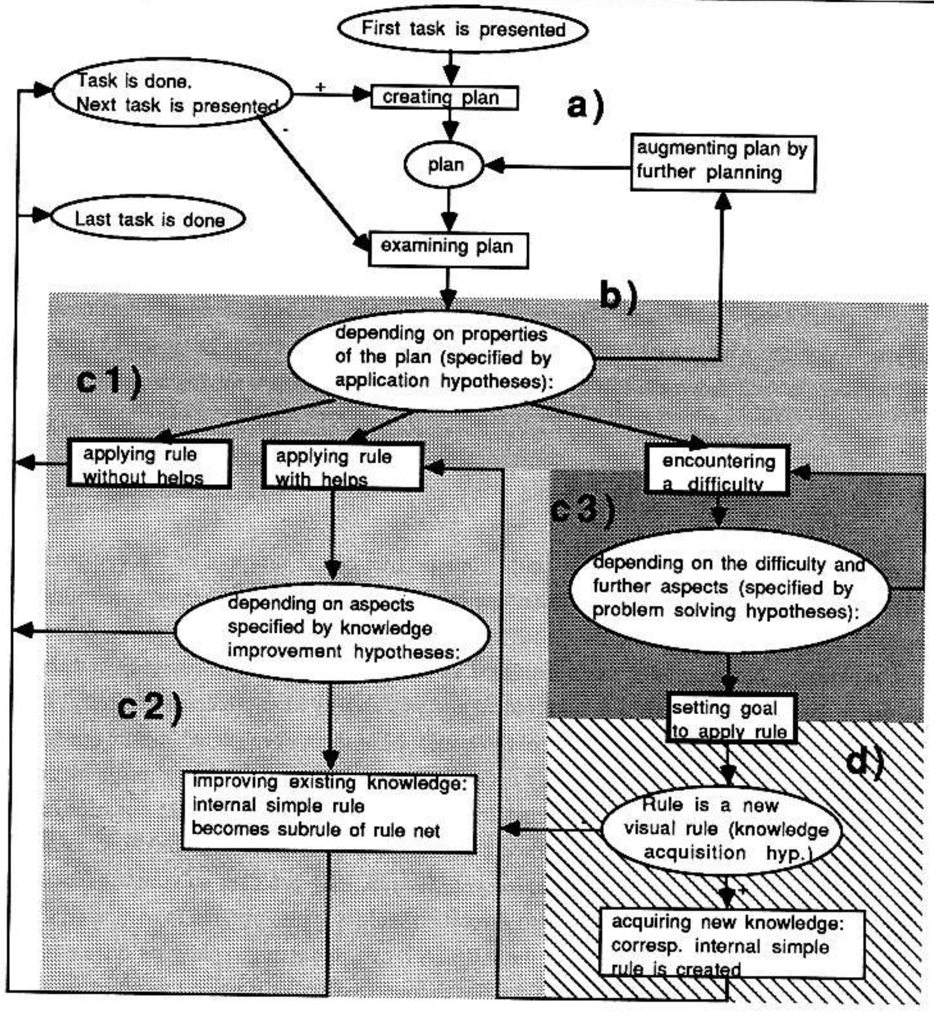
	If ...	then ...	supp.	disc.
An application hypothesis:	If the internal compound rule allows for a sequence of two or more rule applications,	then the next rule application will be performed without helps.	191	21
A problem solving hypothesis:	If there is a difficulty D2: a rule condition is not satisfied but will be satisfied if some other rule is applied first,	then the problem solving step "work backward" is applied in order to find a rule satisfying the violated condition, resulting either in setting a goal or in a new difficulty D2 or D3 for this other rule.	18	3
A Knowledge improvement hypothesis:	If an internal simple rule has just been created (a new, not yet encountered visual rule has just been applied), and another rule differing only in the required type of ABSYNT-node or tree is already part of the internal compound rule,	then the new internal simple rule becomes also part of the compound rule (next application of this rule without using helps).	8	1

The third set of hypotheses (6 "*knowledge improvement hypotheses*") is concerned with the question when an internal simple rule, supposed to require a lookup in the helps, is made part of the internal compound rule, so the next time it is applied without making use of the helps.
Finally, another hypothesis ("*knowledge acquisition hypothesis*") states that if a goal has been set to apply a new visual rule of the helps never encountered before, then this rule will become an internal simple rule.

4.4 The Implementation

Although a composition method has been implemented, the simulation model (in PROLOG) currently contains only the rule net. The model predicts different difficulties, set goals and rule applications with or without helps, based on the hypothetical actual knowledge and the application, problem solving, knowledge improvement and knowledge acquisition hypotheses. The model proceeds roughly as follows (see figure 8):

a) A plan is created (or augmented, if already existing). It consists of a sequence of planned rule applications, containing internal simple rules, subrules of the rule net, or new visual rules.
b) Depending on the properties of the plan, the next planned rule application is performed without or with helps, or a difficulty arises, as specified by the application hypotheses. There might also be more planning.
c1) After a rule application without helps, the model resumes with step a) or b) (depending on whether computation of the current ABSYNT-program is finished).
c2) If a rule is applied with helps, the rule may be improved by proceduralization (= become a subrule of the rule net), as specified by the knowledge improvement hypotheses. Then the model continues as in step c1).
c3) A difficulty triggers a problem solving step (e.g. "working backward", "trial and error"), as specified by the problem solving hypotheses. This results in setting a goal (step d) or in yet another difficulty.


First task is presented
Task is done. Next task is presented
+
-
creating plan
a)
plan
augmenting plan by further planning
Last task is done
examining plan
b)
depending on properties of the plan (specified by application hypotheses):
c1)
applying rule without helps
applying rule with helps
encountering a difficulty
c3)
depending on the difficulty and further aspects (specified by problem solving hypotheses):
depending on aspects specified by knowledge improvement hypotheses:
c2)
setting goal to apply rule
d)
improving existing knowledge: internal simple rule becomes subrule of rule net
Rule is a new visual rule (knowledge acquisition hyp.)
acquiring new knowledge: corresp. internal simple rule is created

⬭	States: properties of current hypothetical knowledge and of task-specific information, including the computational state currently visible on the screen
□	Processes / activities for which there are no protocol categories
■	Processes / activities corresponding to protocol categories

▨	Knowledge acqusistion hypothesis	▒	Application hypotheses
░	Knowledge improvement hypotheses	▓	Problem solving hypotheses

The labels a), b), c1), ... refer to the text.

Figure 8: Basic processing cycles of the simulation model

d) If a goal refers to a new, not yet encountered visual rule, then the knowledge acquisition hypothesis applies, creating a corresponding internal simple rule applied with helps. So the model continues with step c2), leading to possible improvement of the newly acquired internal simple rule.

Thus, during computation of the ABSYNT-program, the hypothetical operational knowledge changes because new visual rules will become internal simple rules, and internal simple rules will become subrules of the rule net. So computation of the next ABSYNT-program of a given level will involve fewer difficulties and fewer use of the helps. There will be more rule applications without helps.
Another feature of the model is that if the rule net is augmented by a new subrule, then the condition of the rule net is generalized if necessary. There is also a memory for certain difficulties encountered in the past. Thus knowledge about what *not* to do is gained, allowing the model to avoid certain difficulties later on.
For illustration, table 2 shows the PROLOG representation of the problem solving hypothesis of table 1.

5. Evaluation of Aspects of the Model

We compared a run of the model to a continuous portion of the protocol of the computation of two ABSYNT-programs including function calls. The programs were the first two programs within a sequence of equally difficult programs, and many difficulties and problem solving steps arose. A prediction of a protocol category by the model was considered as correct if the protocol category and the additional information about it (see the section on protocol analysis above) were predicted correctly. The analyzed portion of the protocol contained 96 protocol categories. 22 of them were excluded from this consideration, since they were not predicted correctly only because of an immediately preceding misprediction. 45 of the remaining 74 categories (61%) were predicted, leaving 4 commission errors and 25 omission errors.
The deviations of the model from the protocol are due to mainly three reasons. First, as noted, not all aspects of the verbal model are implemented yet. Second, long problem solving sequences where the subject struggles for some time, running into several difficulties, cause trouble. In particular, the model has trouble if the subject "jumps around" in the ABSYNT-tree, focussing rapidly on different parts of the screen. Third, the issue of a memory for difficulties encountered earlier is not yet resolved.

We also assessed the generalizeability of aspects of the model over subjects by evaluating some model predictions with more coarse data from different subjects of the investigation. Since the application hypotheses predict difficulties and rule applications with / without helps, we expected that

- in a situation where an application hypothesis applies which predicts a difficulty, much time is needed (following [5], p. 411). Also, the helps are used.
- in a situation where an application hypothesis applies which predicts a rule application with or without helps, less time is needed. The helps are used or not used, respectively.

Table 3 shows some aggregated results. They match with these predictions.
A similar analysis was done with the knowledge improvement hypotheses. They predict that certain rules are applied without helps very soon (see the knowledge improvement hypothesis of table 1, which predicts positive transfer, based on similarity of a new and an already improved rule), while other rules require the helps for a long time. Across subjects, for 14 of 22 rules the data corresponded to these expectations.

Table 2: PROLOG-representation of the problem solving hypothesis of table 1

```
PROLOG-Code:                                        Comments:

situation(d2,[Node1,Rule,Node2]) :-                 If a difficulty D2 has been noticed
                                                    (applying "Rule" to "Node1" would lead to
                                                    "Node2", but "Node1" does not satisfy "Rule"),
      subgoal(Node1,Rule,Node2,Subgoal),            - then set as subgoal a modified version
                                                    of "Node1" (= "Subgoal") such that "Subgoal"
                                                    will satisfy the Condition of "Rule";
      rule(rule_name(R),condition(C),action(A)),    - find some visual rule,
      on(change_to(Node_before_change,              which in its action part changes some
                   Node_after_change),A),           ABSYNT-node such that
      compare(Node1,Node_before_change,[]),         this action is applicable to "Node 1"
      compare(Subgoal,Node_after_change,[]),        and "Subgoal" can then be reached;
      test_condition(Node1,                         - and if such a visual rule has been found,
                   rule(rule_name(R),condition(C),  then test its condition part, obtaining a
                   action(A)),Difference),          difference list between condition and "Node1",
      result_of_problem_solving(d2,                 and this problem solving step may lead
            [Node1,Rule,Node2],                     to some further result, depending on the
            [Node1,rule(IS,R),Node2],               nature of the difference.
            Difference).                            (see "result_of_problem_solving(...)").

result_of_problem_solving(Difficulty,Old,           If there is no difference ([]) between a rule
            [Node1,rule(V,R_name),Kn2],[]) :-       condition and an ABSYNT-node,
      change(difficulty_or_goal(Difficulty,Old),    then replace the current difficulty by
            difficulty_or_goal(goal,                setting a goal to apply this rule to this node,
            [Node1,rule(V,R_name),Node2])),
      generalize_cond_of_rule_net(V,R_name),        and generalize the condition of the rule net ,
      situation(goal,[Node1,rule(V,R_name),Node2]). and persue the goal just set.

result_of_problem_solving(D,Old,New,Difference) :-  If there is a difference between a rule
      on([X,node_type(Y,Z)],Difference),            condition and an ABSYNT-node, such that the
                                                    type of the node (e.g., IF-THEN-ELSE-node)
                                                    cannot possibly satisfy the rule condition,
      change(difficulty_or_goal(D,Old),             then replace the current difficulty
            difficulty_or_goal(d3,New)),            by another difficulty, D3,
      situation(d3,New).                            and persue this difficulty.

result_of_problem_solving(D,Old,New,Difference) :-  If there is a difference between a rule
                                                    condition and an ABSYNT-node,
      content_of_field(Difference,Binding_list),    such that some content of the node does not
      on([Element,X],Binding_list),                 fit the rule condition (the rule is satisfiable
      on(Element,[empty,question_mark,value]),      by obtaining the required content,
                                                    applying some other rule),
      change(difficulty_or_goal(D,Old),             then replace the current difficulty
            difficulty_or_goal(d2,New)),            by another difficulty, D2,
      situation(d2,New).                            and persue this difficulty.
```

Table 3: Predictions of the application hypotheses on a coarse level of analysis (difficulty; rule application with / without helps), and corresponding results (helps used; time needed)

		number of cases analyzed	helps used in these cases (%)	time needed (median)
application	difficulty	67	80%	114 sec.
hypothesis	rule application with helps	87	57%	26.4 sec.
predicts...	rule application without helps	158	18%	13.3 sec.

6. Discussion

The simulation model describes a continuous knowledge acquisition and problem solving process of a single subject in terms of impasse-driven and success-driven learning. Comparing the model to a portion of the protocol showed the degree of fit, but also the current weaknesses. A coarse analysis of the generalizeability of the model across subjects led to results which seem to be encouraging both for continuing the work on the model, and for experimentally testing aspects of the model.

How does the model relate to some other work? Concerning impasse-driven learning, the main similarity to Brown & van Lehns repair theory [5] is that difficulties ("impasses") lead to the application of weak methods which are aimed at overcoming the difficulty (see also [16, 30]), followed by some overt action as well as a secondary impasse. Brown & van Lehn do not directly adress the acquisition of new knowledge (but [29]) and knowledge improvement. But repair theory is constrained by more general principles.
Concerning success-driven learning, we tried to incorparate aspects of knowledge compilation [1, 2, 3, 25]: composition, proceduralization and generalization. Proceduralization differs from Andersons conceptualization due to the rule net in our model. Our model does not contain rule strengths, as proposed by Anderson. But some of the mispredictions of the model suggest that this is useful.

A further implication of the model is that it suggests how the help material might be improved. If the learner is not sure about applying the compound rule in a new situation, then it would seem appropriate if the helps are adapted to his / her current knowledge state. This would mean to augment the helps by a visual compound rules as the ones depicted in figures 6 and 7. Thus it is possible to tailor the help material to the knowledge state of the learner, and to empirically test whether tailored helps are really more helpful.
Another implication comes from the idea that the acquisition of new knowledge is difficulty-driven. If there is a hypothesis about the knowledge state of a subject at a given point (for example, whether a difficulty is encountered), then it can be predicted whether the subject will accept or even actively search for new information [20, 24, 30]. This is important for timing helps in tutorial dialogues.

7. References

[1] Anderson, J.R., The Architecture of Cognition. Harvard University Press, Cambridge, 1983

[2] Anderson, J.R., Knowledge Compilation: The General Learning Mechanism. In: Michalski, R.S.; Carbonell, J.G.; Mitchell, T.M.: Machine Learning, Vol. II. Los Altos: Kaufman, 1986, 289-310

[3] Anderson, J.R.; Greeno, J.G.; Kline, P.J.; Neves, D.M.: Acquisition of Problem-Solving Skill. In: Anderson, J.R. (ed): Cognitive Skills and their Acquisition. Hillsdale, Erlbaum, 1981, 191-230

[4] Bonar, J.G., Liffick, B,W., A Visual Programming Language for Novices. In: Chang, S.K. (ed): Principles of Visual Programming Systems. Englewood Cliffs: Prentice Hall, 1990, 326-366

[5] Brown, J.S.; van Lehn, K., Repair Theory: A Generative Theory of Bugs in Procedural Skills. Cognitive Science, 1980, 4, 379-426

[6] Card, S.K., Moran T.P., Newell, A., The Psychology of Human-Computer Interaction. Hillsdale: Erlbaum, 1983

[7] Cheng, P., Carbonell, J.G., The FERMI System: Inducing Iterative Macro-operators from Experience. Proceedings of the fifth National Conference on Artificial Intelligence. Cambridge: Morgan Kaufman, 1986, 490-495

[8] Davis, R.E., Runnable Specifications as a Design Tool. In: Clark, K.L.; Tärnlund, S.A. (eds), Logic Programming, New York, Academic Press, 1982, 141-149

[9] Goldstein, I.P., The Genetic Graph: A Representation for the Evolution of Procedural Knowledge. In: Sleeman, D, Brown, J.S. (eds): Intelligent Tutoring Systems. New York: Academic Press, 1982, 51-77

[10] Glinert, E.P., Nontextual Programming Environments. In: Chang, S.K. (ed): Principles of Visual Programming Systems. Englewood Cliffs: Prentice Hall, 1990, 144-230

[11] Janke, G., Möbus, C., Thole, H.-J., Konzeptualisierung eines problemlösezentrierten Hilfesystems. In: Stetter, F., Brauer, W. (ed), Informatik und Schule, Informatik-Fachberichte 220, Berlin, Springer, 1989, 44-55

[12] Janke, G., Kohnert, K., Interface Design of a Visual Programming Language: Evaluating Runnable Specifications. In: F. Klix, N.A. Streitz, Y. Waern & N. Wandke (eds), Man-Computer-Interaction Research MACINTER-II, Proceedings of the Second Network Seminar of MACINTER held in Berlin/GDR, March 21 - 25, 1988, Amsterdam: Elsevier, 1989, 567-581

[13] Johnson-Laird, P.N., Mental Models. In Aitkenhead, A.M., Slack, J.M. (eds): Issues in Cognitive Modeling. Hillsdale: Erlbaum, 1985, 81-99

[14] Kearsley, G., Online Help Systems. Norwood: Ablex, 1988

[15] Kieras, D, Polson, P.G., An Approach to the Formal Analysis of User Complexity. Int. Journal of Man-Machine Studies, 1985, 22, 365-394

[16] Laird, J., Rosenbloom, P.S. & Newell, A. (eds), Universal Subgoaling and Chunking, Boston: Kluwer Academic Publ. 1986

[17] Lewis, C., Composition of Productions, In: Klahr, D., Langley, P., Neches, R. (eds), Production System Models of Learning and Development, Cambridge, MIT press, 1987, 329-358

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

[19] Möbus, C., Logic Programs as a Specification and Description Tool in the Design Processes of an Intelligent Tutoring System, in Salvendy, G. (ed), Abridged Proceedings of the HCI International 87, 1987, 119

[20] Möbus, C., Toward the Design of Adaptive Instructions and Helps for Knowledge Communication with the Problem Solving Monitor ABSYNT. To appear in: Stepankova, O., Marik, V. (eds): Proceedings of the CEPES UNESCO Workshop "The Advent of Higher Education", Prague, Oct. 23-25, 1989, Berlin-Heidelberg-New York: Springer (in press)

[21] Möbus, C.; Schröder, O., Knowledge Specification and Instruction for a Visual Computer Language. In: Klix, F.; Wandke, H; Streitz, N.A.; Waern, Y. (eds): Man-Computer-Interaction Research MACINTER-II , 1989, 535-565

[22] Möbus, C.; Schröder, O., Representing Semantic Knowledge with 2-Dimensional Rules in the Domain of Functional Programming. In: Tauber, M; Gorny, P. (eds), Visualization in Human-Computer Interaction, Heidelberg, Springer Computer Science Lecture Series, in press

[23] Möbus, C.; Thole, H.J., Tutors, Instructions, and Helps. In: Christaller, Th. (ed), Künstliche Intelligenz, KIFS87, Proceedings, Informatik-Fachberichte 202, Heidelberg: Springer, 1989, 336-385

[24] Möbus, C, Thole, H.J., Interactive Support for Planning Visual Programs in the Problem Solving Monitor ABSYNT: Giving Feedback to User Hypotheses on the Language Level (this volume)

[25] Neves, D.M.; Anderson, J.R., Knowledge Compilation: Mechanisms for the Automatization of Cognitive Skills. In: Anderson, J.R. (ed): Cognitive Skills and their Acquisition. Hillsdale, Erlbaum, 1981, 57-84

[26] Reiser, B.J., Ranney, M., Lovett, M.C., Kimberg, D.Y., Facilitationg Students´ Reasoning with Causal Explanations and Visual Representations. In: Bierman, D., Breuker, J., Sandberg, J. (eds), Artificial Intelligence and Education. Amsterdam: IOS, 1989, 228-235

[27] Schröder, O., Frank, K.D., Kohnert, K., Möbus, C., Rauterberg, M., Instruction-Based Knowledge Acquisition and Modification: The Operational Knowledge for a Functional, Visual Programming Language, Computers in Human Behavior, in press

[28] Sleeman, D., Brown, J.S., Intelligent Tutoring Systems, New York: Academic Press, 1982

[29] van Lehn, K., Learning One Subprocedure per Lesson, Artificial Intelligence, 1987, 31, 1 - 40

[30] van Lehn, K., Towards a Theory of Impasse-Driven Learning. In: Mandl, H; Lesgold, A.: Learning Issues for Intelligent Tutoring Systems. Springer, New York, 1988, 19-41

[31] Wolff, J.G., Cognitive Development as Optimization, In: Bolc, L. (ed), Computational Models of Learning, Heidelberg: Springer, 1987, 161-205

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

438

D.H. Norrie H.-W. Six (Eds.)

Computer Assisted Learning

3rd International Conference, ICCAL '90
Hagen, FRG, June 1990
Proceedings

Springer-Verlag

[illegible]

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

438

D.H. Norrie H.-W. Six (Eds.)

Computer Assisted Learning

3rd International Conference, ICCAL '90
Hagen, FRG, June 11–13, 1990
Proceedings

Springer-Verlag
New York Berlin Heidelberg London Paris Tokyo Hong Kong